HYBRID GENETIC ALGORITHM WITH MULTI-PARENTS RECOMBINATION FOR JOB SHOP SCHEDULING PROBLEMS

ONG CHUNG SIN

FACULTY OF SCIENCE
UNIVERSITY OF MALAYA
KUALA LUMPUR

2013

HYBRID GENETIC ALGORITHM WITH MULTI-PARENTS RECOMBINATION FOR JOB SHOP SCHEDULING PROBLEMS

ONG CHUNG SIN

DISSERTATION SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

INSTITUTE OF MATHEMATICAL SCIENCES
FACULTY OF SCIENCE
UNIVERSITY OF MALAYA
KUALA LUMPUR

2013

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **ONG CHUNG SIN** I.C/Passport No: **840214-08-5145**

Registration/Matric No.: **SGP 110004**

Name of Degree: **MASTER OF SCIENCE**

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

"HYBRID GENETIC ALGORITHM WITH MULTI-PARENTS RECOMBINATION FOR JOB SHOP SCHEDULING PROBLEMS"

Field of Study: **OPERATIONAL RESEARCH**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date 1.8.2013

Name: **PROFESSOR DR MOHD BIN OMAR**
Designation:

PROFESOR DR MOHD OMAR
Institut Sains Matematik
Fakulti Sains
Universiti Malaya

Witness's Signature Date 1/8/2013

Name: **ASSOC. PROF. DR NOOR HASNAH MOIN**
Designation:

Prof. Madya Dr. Noor Hasnah Moin
Institut Sains Matematik
Universiti Malaya

ABSTRACT

Job Shop Scheduling Problem (JSSP) is one of the well-known hard combinatorial scheduling problems and one of the most computationally difficult combinatorial optimization problems considered to date. This intractability is one of the reasons why the problem has been so widely studied. The problem was initially tackled by "exact methods" such as the branch and bound method, which is based on the exhaustive enumeration of a restricted region of solutions containing exact optimal solutions. Exact methods are theoretically important and have been successfully applied to benchmark problems, but sometimes they, in general are very time consuming even for moderate-scale problems. Metaheuristic is one of the "approximation methods" that is able to find practically acceptable solutions especially for large-scale problems within a limited amount of time. Genetic Algorithms (GA) which is based on biological evolution is one of the metaheuristics that has been successfully applied to JSSP.

In this study an indirect representation incorporating a schedule builder that performs a simple local search to decode the chromosome into legal schedule called active schedule is proposed. The chromosomes are decoded into active schedules thus increasing the probability of obtaining near or optimal solution significantly.

Crossover between two parents is traditionally adopted in GA while multi-parents crossover (more than two parents) technique is still lacking. This research proposes extended precedence preservative crossover (EPPX) which uses multi-parents for recombination in the GA. This crossover operator attempts to recombine the good features in the multi-parents into a single offspring with the hope that the offspring

fitness is better than all its parents. EPPX can be suitably modified and implemented with, in principal, unlimited number of parents.

JSSP generates a huge search space. An iterative forward-backward pass which reduces search space has been shown to produce significant improvement in reducing makespan in other field of scheduling problem. The iterative forward-backward pass is applied on the schedules generated to rearrange their operation sequences to seek possible improvements in minimizing the total makespan.

Reduction of the search space does not guarantee the optimal solution will be found. Therefore, a neighborhood search is embedded in the structure of GA and it acts as intensification mechanism that exploits a potential solution. This mechanism is restricted to search the possible solutions in a critical path. Modification on the path by using neighborhood search significantly reduces the total length of the makespan.

The hybrid GA is tested on a set of benchmarks problems selected from literatures and compared with other approaches to ensure the sustainability of the proposed method in solving JSSP. The new proposed hybrid GA is able to produce 10 better or comparable solutions when compared to similar GA algorithms that employ two-parent crossover. In general this algorithm produces less than 6% deviation when compared to the best known solutions, especially in larger problems consisting of 20 jobs and 15 machines.

ABSTRAK

Kerja kedai penjadualan masalah (JSSP) adalah salah satu masalah penjadualan kombinasi yang terkenal dan merupakan salah satu masalah yang paling sukar dalam pengoptimuman kombinasi. Ciri kesukaran JSSP adalah salah satu sebab masalah ini dikaji secara meluas. Kaedah penyelesaian untuk JSSP pada mulanya menggunakan "kaedah tepat" seperti kaedah cabang dan batas yang berdasarkan penghitungan lengkap rantau penyelesaian yang terhad yang mengandungi penyelesaian optimum. Dari segi teori, kaedah tepat ini adalah amat penting dan telah berjaya digunakan untuk sesetengah masalah "benchmark", tetapi ia memerlukan masa komputasi yang amat panjang walaupun untuk penyelasaian masalah yang bersaiz sederhana. Metaheuristik adalah salah satu "kaedah penghampiran" yang mampu mendapatkan penyelesaian yang boleh diterima (penyelesaian hampir optimum) secara praktikal terutamanya bagi masalah yang bersaiz besar dalam jumlah masa yang terhad. Algoritma Genetik (GA) yang berdasarkan evolusi biologi adalah salah satu metaheuristik yang telah berjaya digunakan untuk JSSP.

Kajian ini mencadangkan penggabungan perwakilan secara tidak langsung dengan pembina jadual yang melaksanakan kaedah carian tempatan mudah untuk menyahkodkan kromosom ke dalam jadual yang dinamakan jadual aktif. Kromosom yang dinyahkod ke dalam jadual aktif akan meningkatkan kebarangkalian untuk mendapatkan penyelesaian yang hampir atau optimum.

Secara tradisinya, persilangan ini biasanya melibatkan dua ibubapa induk sahaja manakala teknik persilangan berbilang induk (lebih daripada dua induk) masih kurang digunakan dalam bidang GA. Kajian ini mencadangkan persilangan pengekalan

keutamaan lanjutan (EPPX) yang menggunakan induk berbilang untuk penggabungan semula dalam GA. Operator persilangan ini akan cuba menggabungkan ciri-ciri yang baik daripada berbilang induk untuk menghasilkan individu yang lebih baik. EPPX boleh diubahsuai dan dilaksanakan tanpa mengehadkan jumlah induk yang terlibat.

JSSP menjana ruang carian yang luas. Kaedah lelaran "forward-backward pass" yang mengurangkan ruang carian telah terbukti menghasilkan peningkatan yang ketara dalam mengurangkan pengurangan masa siap (makespan) dalam bidang masalah penjadualan yang lain. Kaedah lelaran forward-backward pass digunakan dalam pembinaan jadual dengan menyusun semula urutan operasi untuk mendapatkan penambahbaikan serta meminimumkan jumlah masa siap.

Pengurangan ruang carian tidak menjamin akan menemui penyelesaian optimum. Oleh sebab itu, carian kejiranan yang dimasukkan ke dalam struktur GA akan bertindak sebagai mekanisme intensifikasi untuk mengeksploitasi penyelesaian yang berpotensi. Mekanisme ini dihadkan untuk mencari penyelesaian dalam laluan kritikal. Pengubahsuaian ke atas laluan tersebut dengan menggunakan carian kejiranan boleh mengurangkan jumlah masa siap tersebut.

Hibrid GA diuji ke atas set masalah "benchmark" yang dipilih dari kesusasteraan dan dibandingkan dengan pendekatan lain untuk memastikan kemampanan dalam kaedah yang dicadangkan dalam menyelesaikan JSSP. Hibrid GA baru yang dicadangkan mampu menghasilkan 10 keputusan lebih baik atau setanding berbanding dengan algoritma GA seumpamanya yang menggunakan dua ibubapa induk sahaja. Secara umum, algoritma ini menghasilkan sisihan kurang daripada 6% berbanding dengan penyelesaian yang paling baik, terutamanya menonjol dalam

mencari penyelesaian di dalam masalah lebih rumit yang mempunyai 20 kerja dan 15 mesin.

ACKNOWLEDGEMENTS

I am greatly indebted and thankful to my supervisor Associate Prof. Dr. Noor Hasnah Moin and Prof. Dr. Mohd Omar for giving me the opportunity to explore my talents within and for being supportive. Their immense encouragement and ideas contributed greatly to the successful completion of my dissertation.

Also, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

Last but not the least; I would like to thank my parents who drive my courage and supporting me spiritually throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

In the current competitive economy, manufacturing industries have to shorten their production time significantly in order to meet customer demands and requirements, and survive in the market. Effective scheduling plays an important role in reducing the production processing time. Without incurring additional costs, such as machines or labor in the production line, effective scheduling aids in reduction of cost (time), increase of resource utilization and output. When a new product has been introduced into the production line, rearrangement of the process activities become a major factor in influencing the overall performance of the production rate, because the new product has its own process sequences. In order to fit them into the production line, the process activities that are assigned to the resources need to be relocated.

Optimization strategy of assigning a set of processing activities for products (jobs) into the resources has been studied intensively (Jones, 1999). The difficulty of the assignment is increased when the production line is producing variable products. Poor scheduling in this kind of production line are not time efficient because of ineffective resource allocation. This phenomenon is perennially seen in the manufacturing industries, especially in small and medium sized manufacturing companies which lack specialized personnel or effective tools for proper production scheduling optimization. Such inefficiencies in production scheduling result in an increased production time and diminished production rate.

Job Shop Scheduling Problem (JSSP) is one of the well-known hard combinatorial scheduling problems which is appropriate for addressing the practical problems related to production scheduling. It becomes complicated to solve when the size of the problems increases. The size of the problems refers to the total number of operation tasks and the total number of machines that are involved in the process. This condition simulates practical production scheduling when the new products and the associated new resources are introduced into the production line increasing the complexity of the task arrangement.

Since JSSP is a practical problem related to production scheduling, it has received a lot of attention from researchers. There are many different strategies ranging from mathematical programming (exact algorithms) to metaheuristics (especially Genetic Algorithm (GA)) to solve the problems (Jones, 1999). Käschel et al. (1999) compares the different methods for GA and concludes that the performance of GA is only average on many test cases, but GA is still considered as a powerful instrument because of its ability to adapt to new problem types. Due to the high capability of GA, a lot of studies and research have been conducted to investigate how GA could be effectively applied to JSSP (Cheng et al., 1996).

In recent years, since the first application of GA based algorithms to solve JSSP proposed by Davis (1985), GA has attracted the efforts of many researchers to make improvements in the algorithm to better solve the scheduling problems. GA does not always find the optimal solution; therefore, various GA strategies have been introduced to increase the efficiency of GA in finding the optimal or near optimal solutions for JSSP.

JSSP generates a huge search space. Reduction of search space has been shown to produce significant improvement in reducing makespan in JSSP. Therefore, search methods that focus on active schedule are introduced into GA to reduce the search space. The methods include GT algorithm (Giffler and Thompson, 1960) and active-decoding process (Wang and Zheng, 2001), which are used to generate active schedules. Recombination applied on these schedules shows significant improvement in generating new solutions.

In the GA strategies, hybridization of GA with other methods or local search methods provided good results in solving problems. Such strategies capitalize on the strength of GA incorporating local search options for locating the optimal or near optimal solutions. Specifically, the local search procedure of Nowicki and Smutnicki (1996) is embedded into GA because of its effectiveness and it has been shown to increase the performance of GA (Gonçalves et al., 2005; Zhang et al. 2008). Besides this, combination of metaheuristics algorithms with GA has also been proposed and the ability of such hybrid methods has also been tested for solving problems.

Additionally, the structure of the GA can be modified and enhanced to reduce problems often encountered in GA optimization. Park et al. (2003) retard the premature convergence in GA by using parallelization of GA (PGA) to find the near optimal solutions. Watanabe et al. (2005) proposed a GA with search area adaption and a modified crossover operator for adapting to the structure of the solutions space. Ripon et al. (2011) embedded heuristic method into crossover functions to reduce the tail redundancy of chromosomes when implementing crossover operations.

Throughout the literature survey it is observed that the GA's abilities are increased by modifying the structure of the GA. All these researches show that GA is not restricted to a single procedure and performs well when its structure is modified or hybridization is implemented with local search to increase the accuracy of identifying solutions. Such inherent flexibility in its structure has encouraged researchers to use and test GA in combination with different strategies. The framework of GA also allows for some modifications to be made accordingly to suit the problem at hand, including: selection of several parents (more than two parents) for the recombination operation, also known aptly as multi-parents crossover.

In solving combinatorial scheduling problems, to the best of our knowledge, only limited number of multi-parents crossover has been proposed and none is in JSSP. Therefore, the basic ideas and behaviors of the multi-parents recombination approach need to be understood before the method is applied in GA.

The application of multi-parents recombination can be found in different research areas. Mühlenbein and Voigt (1995) proposed Gene Pool Recombination (GPR) in solving discrete domain problems. Eiben and Kemenade (1997) introduced the diagonal crossover as the generalization of uniform crossover and one-point crossover in GA for numerical optimization problems. Wu et al. (2009) proposed multi-parents orthogonal recombination to determine the identity of an unknown image contour. Tsutsui and Jain (1998) proposed multi-cut and seed crossover for binary coded representation and Tsutsui et al. (1999) proposed simplex crossover for real coded GA. The multi-parents crossover operators have shown the good search ability of the operator but they are very problem dependent.

The above literatures indicated the ascendency of multi-parents crossover over two parents' crossover. Although multi-parents crossover has been used in different fields, to the best of our knowledge, only limited numbers are applied to combinatorial scheduling problems. In particular, Eiben et al. (1994) proposed multi-parents for the adjacency based crossover and Ting et al. (2010) developed Multi-Parents Extension of Partially Mapped Crossover (MPPMX) for the Travelling Salesman Problems (TSP). Although the experimental results point out that adjacency based crossover of multi-parents has no tangible benefit, MPPMX show significant improvement in the use of multi-parents in crossover. In other words, one would expect that by biasing the recombination operator the performance of the GA would improve.

Based on the literature reviews about multi-parents recombination approach it is found that some of the crossover operators are extended from the two parents' recombination method. They are modified to make it possible to adopt multi-parents into the operators. This means that the representation that is used for the two parents' recombination can also be reused in the multi-parents recombination technique to solve the problems, instead of being limited to using two parents only. As a result, some of these operators perform well compared to the two parents' recombination with the same recombination method.

In this study, we propose Extended Precedence Preservative Crossover (EPPX) as a multi-parents recombination method. EPPX is built based on the precedence preservative crossover (PPX) approach proposed by Bierwirth et al. (1996). PPX is used as our recombination references because of its capability to preserve the phenotypical properties of the schedules. Therefore, EPPX as a crossover operator will retain this advantage in the GA. EPPX is used to solve JSSP in conjunction with local search.

Furthermore, the large solution search space problem encountered by the GA is reduced by applying an iterative scheduling method. The simulations' results show the sustainability of this GA in solving JSSP.

1.2 Problem Statement

Previous studies show that two parents' crossover is commonly used in solving JSSP and there are rare applications of multi-parents crossover in GA optimizations. In this study, a new approach of multi-parents crossover EPPX is adapted in GA.

GA often encounters problems such as large search space and premature convergence. In the large search space, there always exist poor quality solutions. Therefore, we introduce the iterative forward-backward scheduling which had been used by Lova et al. (2000) in the multi-project scheduling problem to reduce the search space.

Neighborhood search embedded in GA has been proven to help improve the solutions of GA in solving JSSP. Hence, neighborhood search is applied in our algorithm to handle the problem of premature convergence and to escape from the local optima in order to find better solutions. Neighborhood searches for better solutions through the restricted movement of the jobs on the critical paths in the schedule.

These methods are tested on a set of benchmarks for JSSP. The results are compared with other methods to measure the capabilities of the proposed hybrid GA.

1.3 Objectives of the Study

The objectives of this research are:

- To propose multi-parents crossover in GA as crossover operator.
  - Suitable parameters for the multi-parents crossover are tested.
  - Diversification of the recombination methods by introducing multi-parents recombination instead of two parents.

- To hybridize GA with local search to increase the efficiency of GA in searching for the optimal solutions. The methods include:
  - Scheduling method which is employed from other areas and applied to GA to increase its efficiency.
  - Neighborhood search procedure on critical path in schedule that acts as an exploitation mechanism in the search for the best solutions.

- To evaluate the capability of both algorithms in reducing the total makespan time of the jobs using job shop scheduling problems benchmarks as references. Results are compared with other JSSP strategies as well.

1.4 Outline of the Dissertation

This dissertation is devoted to JSSP based on GA using multi-parents crossover as recombination operator and the hybridization with local search and scheduling methods to increase the performance of the GA.

In Chapter 2, the JSSP is introduced. Notation and the precedence constraints of JSSP is defined by formulating the objective functions. The main focus of JSSP is to find the minimum makespan for the scheduling (C_{max}). The different methods for feasible scheduling are explained. The local searches embedded in the GA are introduced. This chapter also contains the reviews of related literature for different multi-parents strategy and its capability in solving a manifold of problems. Hybridization methods that have already been applied to JSSP are explained with special focus on the effect of hybridization of GA in solving such problems. The benchmarks that are commonly used are introduced and their levels of difficulties are described in great details.

In Chapter 3, the methodology of the GA is explained. The framework of the GA, which is built on the hybridization approach with other methods, is described in this chapter. EPPX is proposed and the algorithm is explained in details. An iterative forward-backward scheduling adapted from other scheduling problems is applied to reduce large search space and the neighborhood search on critical path acts as exploitation mechanism to reduce the makespan.

In Chapter 4, suitable parameters for the crossover and mutation rates are examined before the algorithm is adapted to solve problems. The simulations are performed on a set of benchmarks from the literatures and the results are compared to ensure the sustainability of multi-parents recombination in solving the JSSP. The outcome of the comparison is discussed and analyzed in this chapter.

In Chapter 5, the research is summarized and concluded. Further works and directions are suggested for future studies.

CHAPTER 2

THE JOB SHOP SCHEDULING PROBLEM

2.1 Introduction

In this chapter, the background of the job shop scheduling problems is introduced. Job Shop Scheduling Problem is represented as JSSP and the terminology of manufacturing such as job, operation, machine, processing time, and task are used to express the conditions and requirements for the problem.

This chapter is divided into several sections. In Section 2.2, the details of JSSP are explained, including the scheduling methods for JSSP. Section 2.3 discusses the different metaheuristics and their methodologies that are used to solve the JSSP, especially in the last part of this section; the focus is dedicated to GA which is the foundation of this study. Section 2.4 introduces the multi-parents recombination operator with different strategies and Section 2.5 explains the concept of hybrid GA for JSSP. The testing of an algorithm's effectiveness is usually done on a set of benchmarks, which are described in Section 2.6. Finally, Section 2.7 concludes with discussions of the propose GA for JSSP.

2.2 Descriptions of the Job Shop Scheduling Problem

In production, scheduling may be described as sequencing in order to arrange the activities into a schedule. Kumar and Suresh (2009) classified the production systems which include the job shop problem in scheduling and controlling production

activities. Entities which pass through the shop are called jobs (products) and the work conducted on them on a machine (resource) is called an operation (task). Where it is applicable, the required technological ordering of the operations on each job is called a routing. To encompass the scheduling theory, Graves (1981) classifies the production scheduling problems by using the following dimensions:

1. *Requirement generation*

A manufacturing processing can be classified into an open shop or a closed shop. In an open shop, no inventory is stocked and the production orders are by customer requests. In a closed shop, a customer's order is retrieved from the current inventory. The open shop scheduling problem is also called job shop scheduling problem.

2. *Processing complexity*

It refers to the number of processing steps and resources that are associated with the production process. The types of this dimension are grouped as follows:

a. One stage, one processor.

b. One stage, multiple processors.

c. Multistage, flow shop.

d. Multistage, job shop.

One stage in a processor or multiple processors refers to a job that requires one processing step to be done in a machine or multiple machines, respectively. Multistage for flow shop indicates that several operations in the job that are required to be processed by distinct machines and there is a common route for all jobs. Multistage for

job shop refers to the alternative routes and resources which can be chosen and there is no restriction on the processing steps.

3. *Scheduling criteria*

Scheduling criteria is set by referring to the objectives in the schedule that need to be met. Mellor (1966) listed 27 objectives that need to be met in the scheduling criteria. In JSSP, the main objectives can be summarized as follows:

a. Minimum makespan problem

The first operation in the production needs to be started and the last operation needs to be finished as soon as possible. Therefore, the sum of completion times should be minimized. It can be done by utilizing the usage of the resources (reduce the idle time of the machine).

b. Due date problem

Efforts need to be taken in reducing the total delay time and the penalty due to the tardiness by rescheduling.

c. Multi objective scheduling problem

Consideration focuses several objectives and compromises the alternative ways to achieve the objectives.

2.2.1 Problem Definition

JSSP can be defined as a set of n jobs which needs to be processed on a set of m machines. A job consists of a set of operations J, where O_{ij}, represents the $j^{th}(1 \leq j \leq J)$ operation of the $i^{th}(1 \leq i \leq n)$ job. The technological requirements for each

operation processing time is denoted as p_{ij} and a set of machines is denoted by $M_k (1 \leq k \leq m)$.

Precedence constraint of the JSSP is defined as (Cheng et al., 1996):

- Operation j^{th} must finish before operation $j^{th} + 1$ in the job.
- A job can visit a machine once and only once.
- Only one operation can be processed in the machine at a time for one time.
- The delay time for the job transfer machine will be neglected and operation allocation for machine will be predefined.
- Preemption of operations is not allowed.
- There are no precedence constraints among the operations of different jobs.
- Neither release times nor due dates are specified.

2.2.2 Objectives Function

The main objective of JSSP is to find the minimum makespan for the scheduling. The finish time of job i and operation processing time are represented by F_{iJ} and p_{ij} repectively. The completion of the whole schedule or the makespan is also the maximum finish time in the set of the jobs i. Therefore, the makespan is denoted by C_{max} is expressed as follow:

$$C_{max} = max(F_{iJ}) \tag{1.1}$$

Let $G(k)$ be the set of operations being processed in machine k, and let

$$X_{O_{ij},k} = \begin{cases} 1 & \text{if } O_{ij} \text{ has been assigned to machine } k \\ 0 & \text{otherwise} \end{cases}$$

The conceptual model of the JSSP can be expressed as follows (Gonçalves et al., 2005):

$$\text{Minimize } \{(F_{iJ})\} \tag{1.2}$$

$$F_{ij} \leq F_{ij+1} - p_{ij+1}, \qquad j = 1,2,\ldots J, \text{for all } i \tag{1.3}$$

$$\sum_{O_{ij} \epsilon G(k)} X_{O_{ij},k} \leq 1, \qquad \text{for all } k \tag{1.4}$$

The objective function represented by Eq. (1.2) minimizes the maximum finish time in the set of the jobs i, therefore it minimizes the makespan. Eq. (1.3) satisfies precedence relationships between operations and Eq. (1.4) imposes that an operation can only be assigned to a machine at a time. The problem is to determine a schedule that minimizes the makespan, that is, to minimize the time required to complete all jobs.

An example of 3 jobs and their sequences are given in Table 2.1.

Table 2.1: Example of 3 Job and 3 Machine Problem

	Job	**Operation routing**		
		1	**2**	**3**
Processing time	1	3	3	2
	2	1	5	3
	3	3	2	3
Machine sequence	1	M1	M2	M3
	2	M1	M3	M2
	3	M2	M1	M3

The problem can also be represented in the processing time matrix (p) (Figure 2.1) and machine sequences matrix (M) (Figure 2.2) such as below:

$$p = \begin{bmatrix} 3 & 3 & 2 \\ 1 & 5 & 3 \\ 3 & 2 & 3 \end{bmatrix}$$

Figure 2.1: Processing Time

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

Figure 2.2: Machine Sequence

The rows of matrices represent the jobs and the columns represent the operations routing.

2.2.3 Scheduling

(a) Gantt Chart

In the project scheduling problem, Gantt chart is commonly used to illustrate the schedule of the process. It makes describing the JSSP solution more simple and the makespan of the schedule can be easily visualized. Researchers use the Gantt chart to illustrate their methods because the Gantt chart is able to illustrate the arrangement of the procedures of operation in the schedule (Porter, 1968). Gantt chart consist of blocks which are constituted by the operation O_{ij}. The Gantt chart's vertical axis shows a set of machines that are involved in the processing and the horizontal axis shows the accumulation of the processing time for the operations. In Figure 2.3, the Gantt chart shows that the minimum makespan can be found by referring to the maximum finish time ($C_{max} = 17$) in the last operation in the chart, $max\left(F_{ij}\right)$.

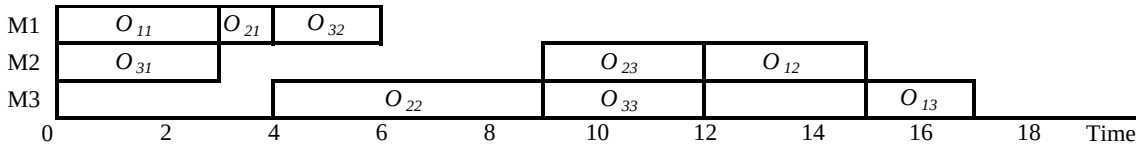


Figure 2.3: Gantt Chart

The sequence of the operation in the machine is presented in Figure 2.4. The matrix rows represent the machines.

$$S = \begin{bmatrix} O_{11} & O_{21} & O_{32} \\ O_{31} & O_{23} & O_{12} \\ O_{22} & O_{33} & O_{13} \end{bmatrix}$$

Figure 2.4: Operation Sequence

In addition disjunctive graph can also be used to calculate the makespan time for the JSSP.

(b) Disjunctive Graph

A disjunctive graph (Balas, 1969) is a graphical structure that can be viewed as one kind of job pair relation-based representation. In JSSP, these are frequently used in problem solving methods to illustrate the relationship between the operations and the machines. Yamada and Nakano (1997) described that a disjunctive graph can be written as $G = (N, A, E)$ where N denotes a set of operations with additional two tasks: a source and a sink. A represents the connection arc of the consecutive operations in the same job, and E contains the arcs that connects the operations which are processes in the same machine. The length of the makespan can be calculated by finding the longest path from the source to the sink. This can be done by summing all the consecutive arcs which are connected continuously in the graph. Figure 2.5 illustrates a disjunctive graph for the example given in Table 2.1.

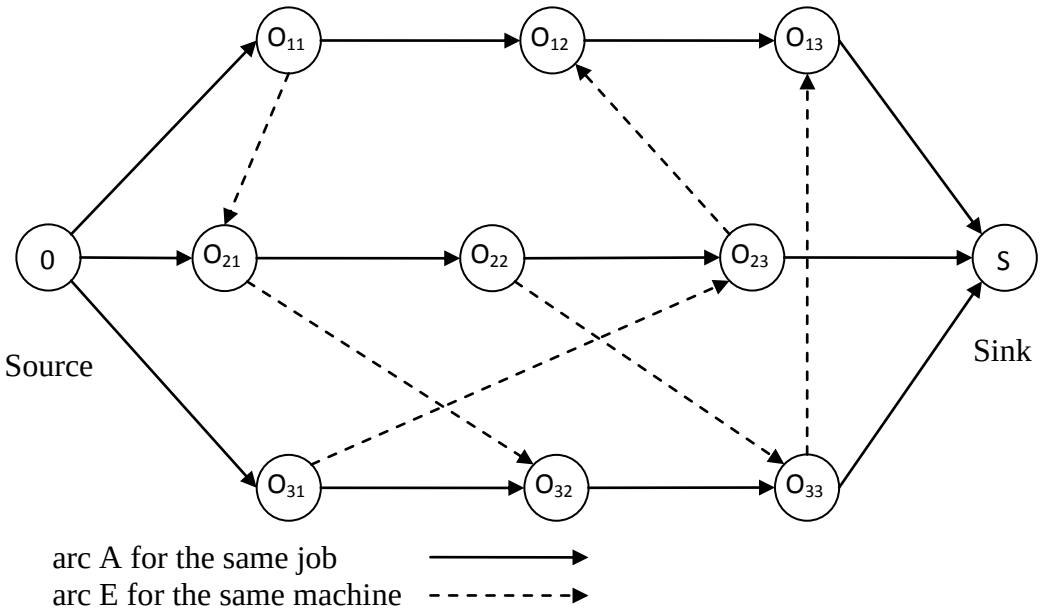


Figure 2.5: Disjunctive Graph

2.2.4 Critical Path

The critical path is the longest path in the schedule that the operation process passes through with respect to the individual operations' interdependencies (Gen et al., 2008). It is the shortest time in the schedule that starts from first operation until the last operation to complete the schedule. Any delay of any operation on the critical path will delay the makespan. The critical path can be identified in a schedule by determining the parameter of each operation (Kelly and Walker, 1959):

Earliest start time (ES): The earliest time at which the operation can start given that its precedent activities must be completed first.

Earliest completion time (EF): The sum of the earliest start time for the activity and the time required to complete the operation.

Latest start time (LS): The latest time at which the operation can be completed without delaying the project.

Latest completion time (LF): The latest finish time minus the time required to complete the operation.

The slack time for an operation is the difference between the ES and LS or EF and LF. An operation which is in the critical path is called a critical operation and can be identified if it contains zero slack time, i.e. $ES = LS$ and $EF = LF$. The critical path in the Gantt chart is illustrated in Figure 2.6.

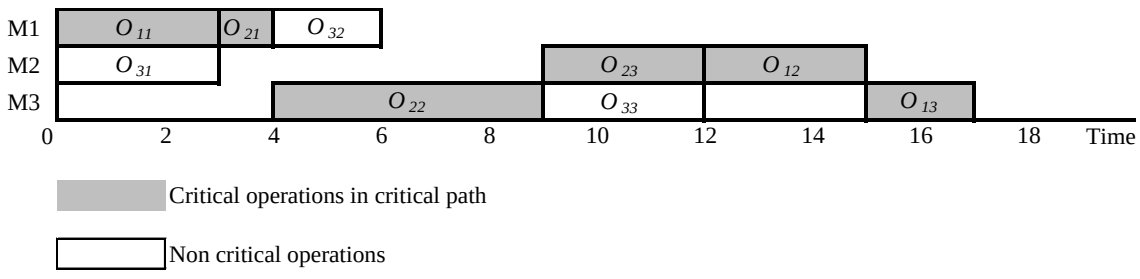


Figure 2.6: Critical Path in Gantt Chart

Figure 2.7 presents an example of non critical operations. Note that without changing the operation sequence in the machines, the operations O_{31}, O_{32}, and O_{33} can start latest without delaying the schedule time $ES \neq LS$ and $EF \neq LF$, therefore they are not critical operations.

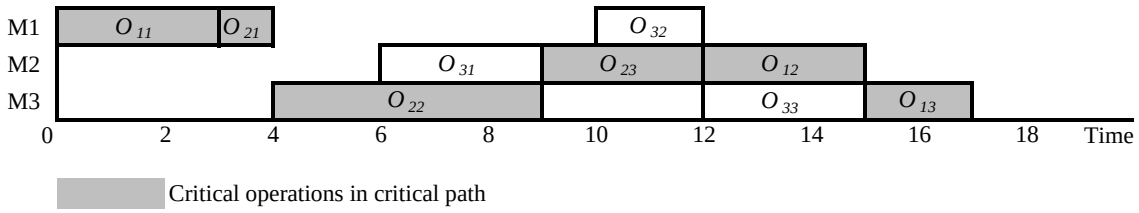


Figure 2.7: Non Critical Operations

The critical path also can be represented in the disjunctive graph (Figure 2.8). The longest path in the network is defined as that path which is connected consecutively forms a critical path.

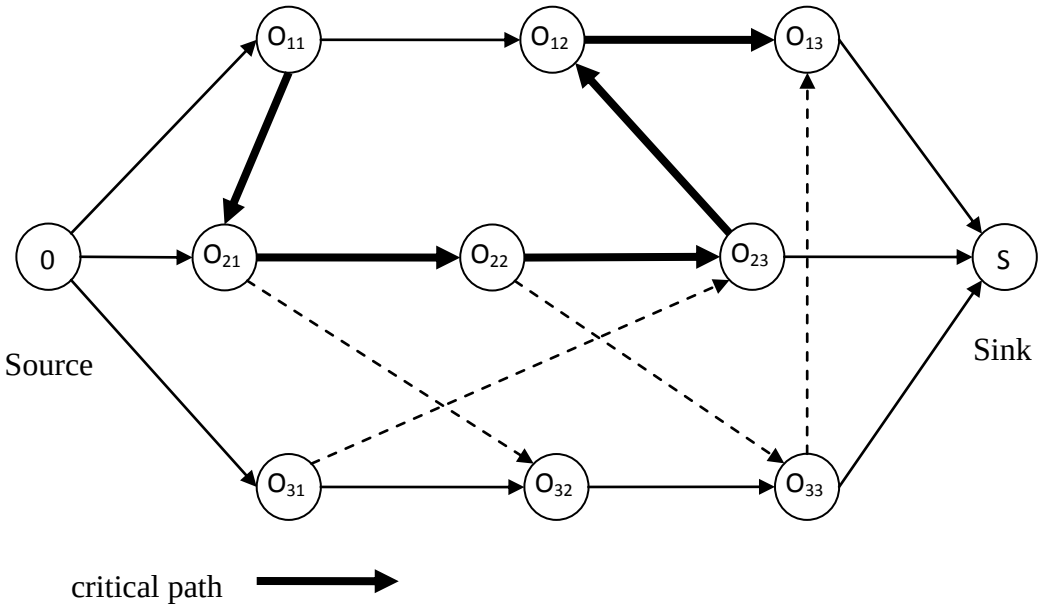


Figure 2.8: Critical Path in Disjunctive Graph

2.2.5 Type of Schedules

In the JSSP, the total solutions for all possible schedules are $(n!)^m$ for n jobs and m machines (Cai et al., 2011). Clearly, it is hard to find all the solutions and compare them with each other. Even for the easy problems, with 6 jobs and 6 machines (FT06) (Jain and Meeran, 1999), the total solutions consist of about $1.36\text{x}10^{17}$ schedules. Even in this case it is unreasonable to calculate all possible solutions. The total number of solutions comprises of feasible and infeasible schedules.

Feasible solutions consist of three types of schedules: semi-active, active and non-delay schedule (Sprecher et al., 1995). These distinctions of schedules narrow down the finding of optimal solutions that is located in the search space. Besides that, Baker (1974) defined that an operation can be left shifted without delaying any other operation in the schedule as a global left shift. This is used to differentiate the types of schedules.

The details of the types of schedules are described below:

Semi-active schedule: A feasible non-preemptive schedule is called active if it is not possible to construct another schedule by changing the order of processing on the machines and having at least one job/operation finishing earlier and no job/operation finishing later. Global left shift is possible in this type of schedule.

Active schedule: A feasible non-preemptive schedule is called semi-active if no job/operation can be finishing earlier without changing the order of processing on any one of the machines and global left shift is not possible. Active schedules the sub set of the semi-active schedules.

Non- delay schedule: A feasible schedule is called a non-delay schedule if no machine is kept idle while a job/an operation is waiting for processing. This schedule is also an active and semi-active schedule.

Optimal solution of the scheduling always lies in the active schedule (Gen and Cheng, 1997). Therefore, we only need to find the optimal solution in the set of active schedules. Figure 2.9 illustrates the relationship of the schedules.

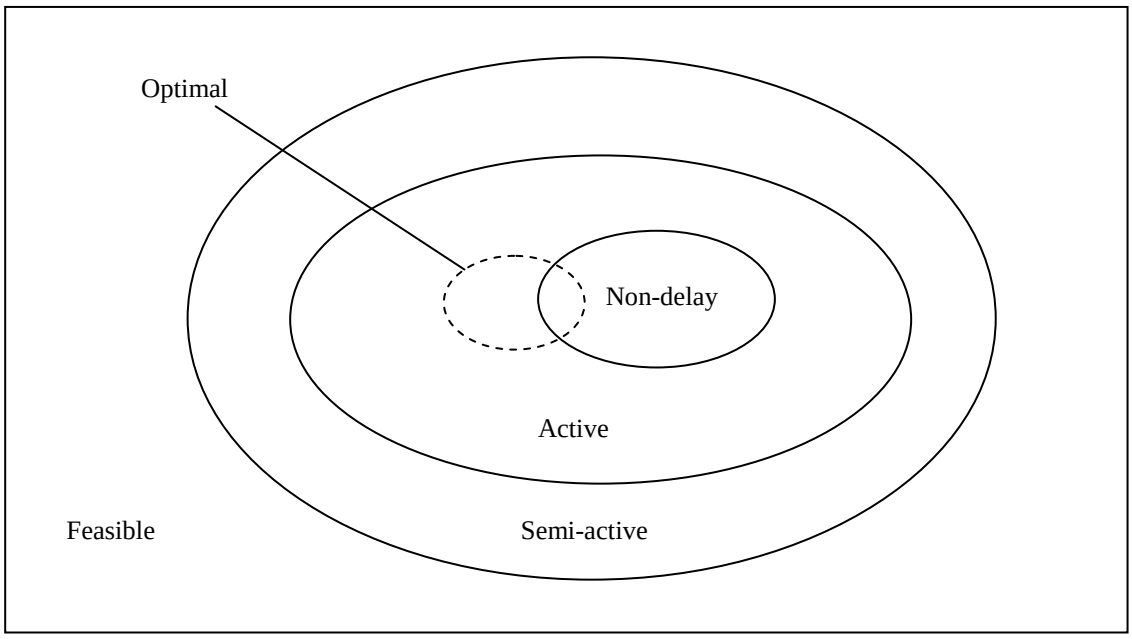


Figure 2.9: Relationship of Semi-Active, Active, and Non-Delay Schedules

2.2.6 Active Schedule Generation

2.2.6.1 Giffler and Thompson Algorithm (GT Algorithm)

In JSSP, the scheduling algorithm that has been proposed by Giffler and Thompson (1960) (GT algorithm) is the famous example representing the generation of active schedule. GT algorithm has been used widely by other researchers to generate active schedules that fit their algorithm. Bierwirth and Mattfeld (1999) combined GT algorithm and non-delay schedule that they had defined to find the performance in

generating the production scheduling solution. Yamada and Nakano (1997) used the GT algorithm and modified it into the form that was compatible with their algorithm. As a result, it shows significant improvement in solving tougher larger sized JSSP.

Below are the steps to obtain the active schedule by using GT algorithm:

Step 1: Let C be the a set of tasks that are not schedule yet

Step 2: Let t be the earliest completion time of the operation which is calculated for all the operations

Step 3: Let G denote the set of all operations that are processed in the machine m with the $time < t$

Step 4: Select an operation from G and insert it into the schedule

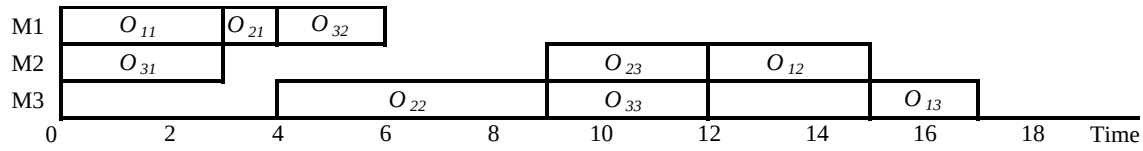
Step 5: Update the sets C and G

Step 6: Repeat the *Step* 1 – *Step* 5 until all operation is scheduled.

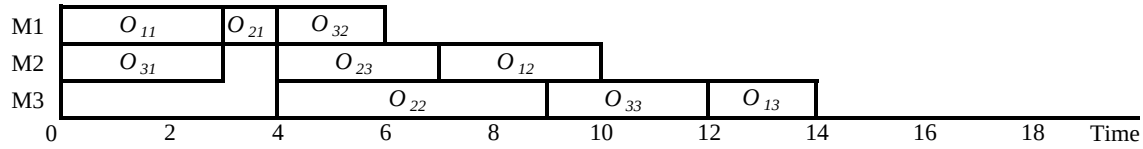
The schedule that is generated using this algorithm always produces the active schedule.

2.2.6.2 Active-Decoding Process

An active schedule can be obtained by shifting the operations to the left of a semi-active schedule without delaying other jobs, such reassigning, is called a permissible left shift, and a schedule with no more permissible left shifts is called an active schedule. This condition enables one to convert the semi-active schedule to an active schedule by using an active-decoding process that was introduced by Wang and Zheng (2001). Each process that is assigned is always shifted to the left until time equals to zero or inserted into empty time interval between operations to find the earliest completion time. The process is repeated until all operations are scheduled. A schedule generated by this procedure can be guaranteed to be an active schedule (Baker, 1974).



(a) Semi-active Schedule



(b) Active Schedule after Active-Decoding Process

Figure 2.10: Active-Decoding Process in Gantt Chart

Figure 2.10 illustrates the transformation of semi-active schedule into active schedule. The operations are shifted to the left in the semi-active schedule and this may decrease the makespan time.

2.3 Metaheuristics

Metaheuristics are designed to tackle complex optimization problems where other optimization methods have failed to be either effective or efficient (Ólafsson, 2006) in solving problems. The term "meta heuristic" was first used by Glover (1986). Osman and Laporte (1996) defined that metaheuristic is an iterative generation process which guides subordinate heuristics by combining different concepts and learning strategies that efficiently lead to near-optimal solutions. Blum and Roli (2003) summarize that metaheuristics are high level strategies for exploring search space by using different methods. The added search flexibility makes the algorithm attempt to find all the possible best solutions in the search space of an optimization problem. The advantage of metaheuristics is that it usually finds solutions quickly and the disadvantage is that the quality of the solution is generally unknown (Taha, 2011).

The common procedure of the metaheuristic is the application of an iterative procedure that is continuously operated and terminates when certain criterion is met. Examples of the terminations are (Taha, 2011):

- The search iteration number reach is a specified number.
- The frequently number of the best solution found that exceed a specified number.
- The optimal solution is found or the current best quality solution is acceptable.

One of the commonly used iterative search procedures in metaheuristics is called local search. Local search does not have consistent definition (Zäpfel et al., 2010). It is dependent on how the algorithm searches the result locally in the current solution. When a solution obtained is slightly different from the original solution, it is regarded as a neighbor. If it receives a set of neighboring solutions, it is called "neighborhood". In the iteration, the current solution tries to move to the best solutions within the neighborhood in hope of getting the optimal solution with the hill climbing. When there are no improvements present in the neighborhood, local search is stuck at local optimum. Then the algorithm has to restart (Lourenço et al., 2003).

In the next section, the metaheuristics that is applied on the JSSP is introduced. The three prominent metaheuristics introduced are tabu search, simulated annealing, and emphasizing on genetic algorithms which is the focus of this study.

2.3.1 Simulated Annealing (SA)

Kirkpatrick et al. (1983) and Cemy (1985) independently introduced the concept of SA in the combinatorial problem. This concept is based on the thermal process for

obtaining low energy of a solid in a heat bath which increases the heat until the maximum value is reached and then the temperature is slowly decreased to allow the particles to rearrange their own positions.

The main structure of the SA is almost the same as the local search but the difference is that SA does not specify the neighborhood but rather specifies an approach to accepting or rejecting solutions that allows the method to escape local optima (Zäpfelet al., 2010). SA from this point of view is using temperature control mechanism which affects the process of solution acceptance as illustrated in Figure 2.11. The acceptance criterion of the solution in the SA may be proposed based on the problem requirements, for example, Van Laarhoven et al. (1992) proposed the acceptance criterion based on statistical properties of the cost for SA in JSSP.

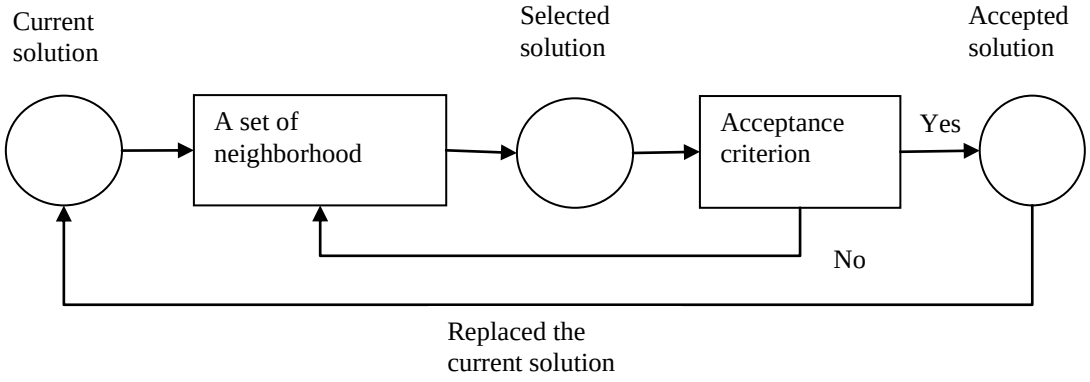


Figure 2.11: Simulated Annealing (SA)

SA has been applied to JSSP earlier, e.g., Van Laarhoven et al. (1992) had been applied SA to JSSP and performed a complexity analysis of their heuristics which are designed to minimize the makespan. Steinhöfel et al. (1999) analyze a neighborhood function which involves a non-uniform generation probability by using SA to search the results for JSSP.

2.3.2 Tabu Search (TS)

TS which was originally developed by Glover (1986), has been widely used in solving combinatorial problems. TS is a general framework for iterative local search strategy for problem optimization. TS, which extended from local search, uses the concept of memory to control the algorithm execution via a tabu list for the forbidden. Glover (1986) introduced the short-term memory to prevent the recent moves and longer-term frequency memory to reinforce attractive components.When TS encounters a local optimum, it will allow moves from the previous tabu list (see Algorithm 2.1).

Algorithm 2.1: Simple Tabu search

```
Initialize solution s

Initialize tabu list T

while termination criterion = false do

Determine a set of move, neighborhood N of current solution s;
Best non-tabu solution is chosen s0 from N;
Replace s by s0;
Update tabu list T and best found solution;

End while

Best solution is found
```

Tabu Search Algorithm with Back Jump Tracking (TSAB) proposed by Nowicki and Smutnicki (1996) is considered as one of the most restricted search in the TS. In the TSAB, the search focuses on the critical path. The critical path is divided into blocks which are called critical blocks that contain a maximum adjacent critical operation which require the same machine.

Through the finding, a good solution may be found by swapping the operations at the border of the block instead of swapping the operations inside the block. Given b blocks, if $1 < g < b$, then swap only the first two and the last two block operations.

Otherwise, if $g = 1$ (b), swap only the last (first) two block operations (see Figure 2.12). In the case where the first and/or the last block contain only two operations, these operations are swapped. If a block contains only one operation then no swap is made.

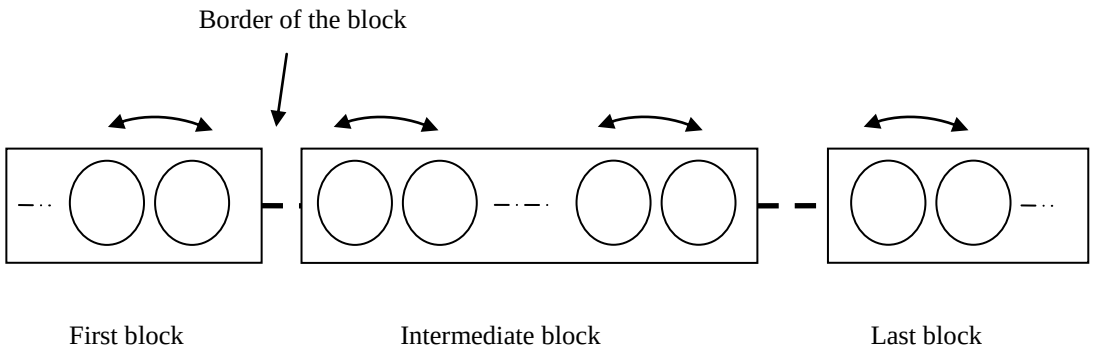


Figure 2.12: Swapping in the Critical Blocks

The possible swap is predetermine and the best swap that provides the best solution is used for the next solution and swapped operations is updated in the tabu list. When the tabu list reaches a certain memory, the forbidden moves are eliminated from the list and reused for the next search. There is an aspiration criterion in which if the swap is able to reduce to the makespan, it is accepted and cancelled from the tabu list (Zäpfel et al., 2010).

Dell'Amico et al. (1993) applies the tabu search technique to the JSSP and show that implementation of this method dominates both a previous approach with TS and the other heuristics based on iterative improvements. Recent results that use TS algorithm embedded within their algorithms includes Gonçalves et al., 2005 and Cai et al. (2011) in solving JSSP. In particular Zhang et al., (2008) propose a combination of SA and TS and their paper produces some of the best known results to date.

2.3.3 Genetic Algorithm (GA)

In recent years, since the first use of GA based algorithm to solve the JSSP proposed by Davis (1985), GA has attracted many researchers to improve efficiency of the scheduling method and frequently used to solve scheduling problem. Various GA strategies are introduced to increase the efficiency of GA to find the optimal or near optimal solutions for JSSP (Cheng et al., 1996; Cheng et al. 1999).

GA is a heuristic based search which mimics the evolutionary processes in biological systems. Evolutionary processes such as reproduction, selection, crossover, and mutation, which are inspired by natural evolution, are used to generate solutions for optimization problems (see Algorithm 2.2). Those techniques are translated into the form of computer simulations. GA begins with a population, which represents a set of potential solutions in the search space. It then attempts to combine the good features in each individual in the population using random search information exchange in order to construct individuals who are better suited than those in the previous generation(s). Through the process of evolution, individuals who are poor or unfit tend to be replaced by fitter individuals to generate a new and better population. In this way, GA usually converges to the estimation for a desired optimal solution.

Algorithm 2.2: A Standard Genetic Algorithm

```
Initialize population

Evaluation

while termination criterion=false do

        Selection
        Crossover
        Mutation
        Evaluation
        Reinsertion

End while
```

2.3.3.1 Representation

GA is an iterative and stochastic process that operates on a set of individuals (population). Each individual represents a potential solution to the problem. This solution is obtained by encoding and decoding an individual called chromosome (Taha, 2011). The illegality of the chromosomes refers to the phenomenon of whether a particular chromosome represents a solution or not (Cheng et al., 1996). An illegal chromosome needs to go through the legalization process to generate a feasible solution.

In the survey by Cheng et al. (1996), chromosome representation in JSSP was divided into two approaches: direct and indirect. The difference between direct and indirect approach depends on whether a solution is directly encoded into the chromosome. As an example: direct approach encoded a schedule directly into a binary string to evolve and find a better solution. Indirect approach requires a schedule builder to encode integer representations for the jobs into the chromosome.

Abdelmaguid (2010) classified the GA into two main categories, model based and algorithm based. The model based category enables chromosomes to be directly interpreted into feasible or infeasible solution. Algorithm based is used to store the information in order to generate feasible solution. The author points out that the different representations of JSSP affects the quality of the solution found and the calculation time.

Therefore, simplification of the representation is important in the steps related to encoding and decoding of a chromosome. One of the representations proposed by Gen et al. (1994) called operation based representation by using permutation with repetition integers that are able to encode a schedule according to the sequences into chromosome

without violating the technological constraint. Figure 2.13 presents examples of binary and integer with repetition to encode a chromosome.

Chromosome= [1 1 1 0 0 1 0 1 0]

a) Binary representation

Chromosome= [1 2 3 2 1 3 3 1 2]

b) Operation based representation

Figure 2.13: Examples of Representations for 3 Job and 3 Machine Problem

2.3.3.2 Initialize Population

A genetic algorithm work starts by building a population which contains a number of individuals; a set of possible solutions for the optimization problem. Each individual is called a chromosome. These individuals are evaluated by assigning value or fitness function to measure their quality in achieving the problem's solutions. Individuals are selected based on the fitness function to breed a new generation through the recombination process.

The two important aspects of population in GA are:

1) Initialization of population generation
2) Population size

Initialization of population generation

The population is normally generated randomly to achieve a set of solutions for breeding. However, Park et al. (2003) mentioned from their research that the initial solution plays a critical role in determining the quality of the final solution. Therefore, they generated the population using GT algorithm to acquire a set of active schedule chromosomes.

Population size

Goldberg et al. (1991) had shown that with a population size which is larger, it is easy to explore the search space. The disadvantages of the larger population size are that it demands more computational cost, memory, and time; so normally 100 individuals is a common population size selected in solving the GA problem (Sivanandam and Deepa, 2008).

Some problems have very large solution spaces which contain many variables and large ranges of permissible values for solutions. Therefore, a fixed population is probably not enough because it simply does not represent a large enough space sample for the solution space. The number of individuals can be changed due to machine capabilities in terms of time and memory, and the result qualities can be compared. For example, the number of individuals in the population generated by Gonçalves et al. (2005) is calculated based on twice the number of total operations in the different structures of JSSP.

2.3.3.3 Termination Criterion

Termination is the criterion by which the genetic algorithm decides whether to continue searching or stop the search. Each of the enabled termination criterion is checked after each generation to see if it is time to stop. The termination criteria in the JSSP are based on the maximum number of generations or the stage when the optimal solution is found.

2.3.3.4 Selection

Selection is a process of choosing the parents for recombination operations. It is a method to pick the parents according the parents' fitness. The fitness of an individual

is based on the evaluation of the objective function of the problem. In the JSSP, each job has a different finish time due to different schedules of operation time. C_{max} will be the maximum time for completion in the scheduling (please refer to Eq. (1.1)). The objective of the evaluation is to determine the ranking of the chromosome, which is used in the process of selection. Each chromosome competes with the others and the selected chromosome will survive to the next generation based on the objective function (fitness value). A chromosome with greater fitness means that it has a greater probability for survival. The highest ranking chromosome in a population is considered as the best solution. It is noted that the lower makespan is given the highest ranking in JSSP. This selection pressure of GA forces the population to improve its fitness over continuing generations (Sivanandam and Deepa, 2008).

The common use of the selection methods in GA are:

a) Roulette wheel selection

b) Stochastic universal sampling (Baker, 1987)

c) Tournament selection (Miller and Goldberg, 1996)

a) Roulette Wheel Selection

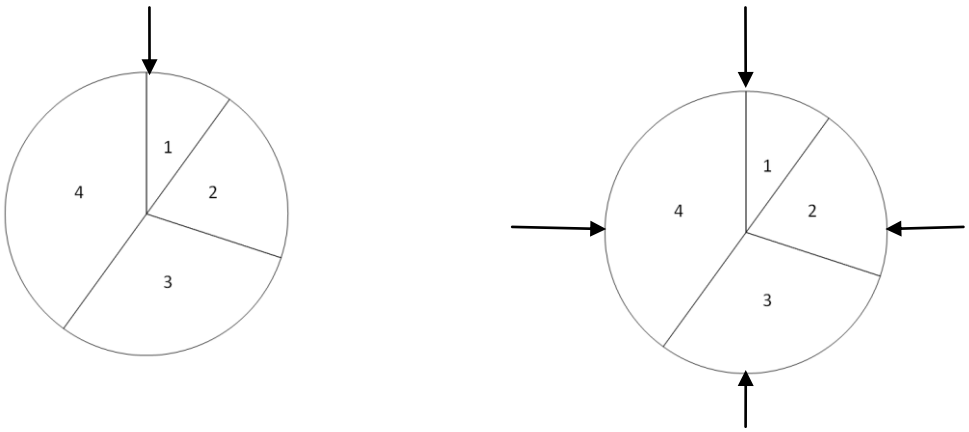
Roulette wheel selection selects the parents according to their proportional fitness (Zäpfel et al., 2010).The fitness of an individual is represented as a proportionate slice of the roulette wheel. The wheel is then spun and the slice underneath the wheel, when it stops, determines which individual becomes a parent. With high fitness value, there is a higher chance that the particular individual is selected (Eq. (2.1)).

$$p_i = \frac{f_i}{\sum f_i} \tag{2.1}$$

p_i = probability that individual i will be selected,
f_i = fitness of the individual i, and
Σf_i = sum of all the fitness values of the individuals within the population.

b) Stochastic Universal Sampling (SUS)

This fitness based proportionate selection, which was proposed by Baker (1987), selects and classifies the chromosomes into a recombination process with minimum spread and zero bias. Instead of the single selection pointer employed in roulette wheel methods, SUS uses N equally spaced pins on the wheel, where N is the number of selections required. The population is shuffled randomly and a single random number in the range $[\sum f_i/N]$ is generated. The difference between the roulette wheel selection and stochastic universal sampling can be illustrated in Figure 2.14.



a) Roulette Wheel Selection

b) Stochastic Universal Sampling

Figure 2.14: The Fitness Proportional Selection

c) Tournament Selection

Tournament selection is one of the important selection mechanisms for GA (Miller and Goldberg, 1996). In this selection scheme, a small number of individuals from the population are chosen randomly. These individuals then compete with each other and the winner of the competition is then inserted back into the mating pool. This tournament process is repeated until the mating pool is filled to generate offspring. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes. Selection pressure is easily adjusted by changing the

tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Among these selection techniques, stochastic universal sampling and tournament selection are often used in practice because both selections have less stochastic noise, or are fast, easy to implement, and have a constant selection pressure (Blickle and Thiele, 1996).

2.3.3.5 Crossover

Crossover is a solution combination method that combines the selected solutions to yield a new solution (Zäpfelet al., 2010). The crossover operator is applied on the selected parents for mating purposes to create a better offspring. The offspring that is generated by crossover may exist in one or more combined solutions.

The processes of crossover are done by three steps (Sivanandam and Deepa, 2008):
Step 1: The reproduction operator selects at random some parents for the mating.
Step 2: Cross point(s) along the chromosome is determined
Step 3: The position values are swapped between the parents following the cross point(s)

Different crossover strategies have been introduced in the literatures for JSSP. Yamada and Nakano (1992) proposed modified GT algorithm as a crossover operator. The crossover selected active schedule chromosome as parents to generate the new offspring that also is in the active schedule. Such recombination of active schedules produces good results.

Partial-mapped crossover (PMX) was proposed by Goldberg and Lingle (1985) is a variation of the two-cut-point crossover. This kind of crossover may generate an illegal offspring. By incorporating the algorithm with a special repairing procedure, possible illegitimacy can be solved. PMX can be divided into four major steps to generate new children. They consist of: selection of substring, exchange of substring, mapping of substring and legalization of the offspring.

Bierwirth (1995) proposed the crossover method based on the permutation crossover operator to preserve the phenotypical properties in the schedules. The chromosome represented in the form of permutation with repetition that is used for recombination. Figure 2.15 is an example of the precedence preservative crossover (PPX) proposed by Bierwirth et al. (1996). The vector is generated randomly with the element set{1,2}. The vector will define genes that are drawn from parent 1 or parent 2. After a gene is drawn from one parent, another parent with the same number at the left most side is also deleted. This process is continued until the end of the vector.

Parent 1 : 3 3 1 1 2 1 2 2 3
Parent 2 : 3 2 2 1 1 1 3 3 2
Vector : 1 1 2 1 2 1 2 1 2
Child : 3 3 2 1 2 1 1 2 3

Figure 2.15: Precedence Preservative Crossover (PPX)

In the literature (Bierwirth (1995); Bierwirth et al. (1996); Gonçalves et al. (2005); Park et al. (2003); Ripon et al. (2011); Wang and Zheng (2001); Yamada and Nakano (1992)), the crossovers are applied on the active schedule chromosomes and the solutions generated are in comparable ranges. These show that the active schedule chromosome and the crossover are interrelated in generating good solutions.

2.3.3.6 Mutation

Mutation is a genetic operator, analogous to the biological mutation, which is used to maintain genetic diversity from one generation in a population of chromosomes to the next. The purpose of mutation in GA is to diversify, thus allowing the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping the evolution. This reasoning also explains the fact that most GA systems tend to avoid taking only the fittest of the population when generating the next chromosome but rather select a random contingent from the population (or pseudo-random with a weighting towards those that are fitter).

The main idea of mutation in JSSP is generally followed by changing the gene position in the chromosome to generate new offspring. For example, a Forward Insertion Mutation (FIM) and a Backward Insertion Mutation (BIM), which were proposed by Cai et al. (2011), will place a chosen gene into selected positions.

In the evolutionary process, crossover and mutation operators are very popular for research endeavors. The reason for their preference is that the different rates for both operators influence the result of the solution. The operator with high rate will be the major operator in the process or vice versa. Typically, the crossover rate is set at the highest value and mutation rate is usually much smaller (Langdon et al., 2010) but some of the researchers prefer that the mutation rate is at a high value to ensure that the population is diversified enough (Ochoa et al., 1999). Therefore, there is further possibility of modifying the relative proportions of crossover and mutation as the search progresses (Reeves, 2003).

2.4 Multi-Parents Crossover

The multi-parents recombination or multi-parents crossover can be defined as using more than two parents in the crossover operator to perform the recombination process (Eiben, 2003). In the general GA, the crossover operator uses two parents for recombination. It is very typical to select multi-parents for recombination in a search protocol that mimics nature, since in nature there are only two types of reproduction (recombination), asexual (one parent) and bisexual (two parents) reproduction. However, in the computational mathematics, there is no restriction on the number of parents to use as long as the multi-parents crossover can be logically implemented in the GA.

Multi-parents recombination is not a new idea and has been used in research involving disparate fields of study. In testing multi-parents recombination affected on the representation, Tsutsui and Jain (1998) proposed multi-cut and seed crossover for binary coded representation. Additionally, Tsutsui et al. (1999) proposed simplex crossover for real coded GA. The crossover operators that are used in these two areas show good search ability of the operator but are very problem dependent.

In solving discrete domain problems, Mühlenbein and Voigt (1995) proposed gene pool recombination (GPR). In GPR, the genes for crossover are selected from the gene pool, which consists of several pre-selected parents instead of two parents. The authors conclude that GPR is mathematically more tractable and able to search more reasonably than two parents' recombination.

In the other field, Wu et al. (2009) proposed multi-parents orthogonal recombination to determine the identity of an unknown image contour. This

recombination is used to rearrange the genes by dividing the genes and gathering the information from the genes of different parents selected for the recombination. One of the major enhancements of the method is that the performance is more stable, consistent, and insensitive to the nature of the input contour.

Multi-parents recombination can produce one child or multiple children. This can be done by one of the multi-parents crossover techniques, called diagonal crossover, proposed by Eiben and Kemenade (1997). The crossover is based on the ratio using uniform crossover to create r children from r parents by selecting $(r-1)$ crossover points in the parents and then composing them into chromosome. The offspring will include the characteristics from the different parents after recombination. The process can be illustrated as in Figure 2.16.

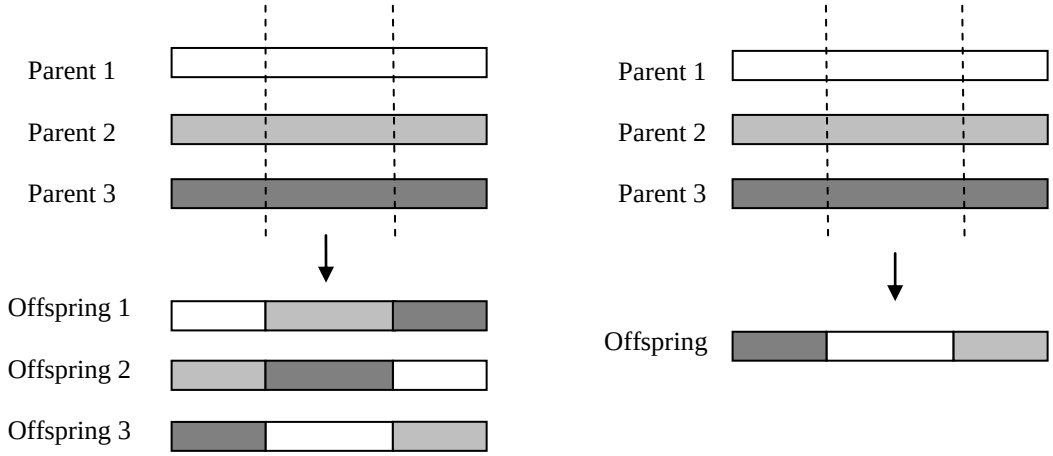


Figure 2.16: Diagonal Crossover with different Number of Offspring Generation

Besides creating new multi-parents crossover operators, the crossover operator can also be extend from the current crossover operator. Tsutsui and Jain (1998), Wu et al. (2009), and Ting et al. (2010) extended their multi-parents crossover technique from two parent crossover operator.

2.4.1 Occurrence Based Adjacency Based Crossover

In the combinatorial scheduling problem, the position or sequences in the chromosome is relatively important because it represents the arrangement of the actual schedule.

Occurrence based adjacency based crossover (OB-ABC) is specifically designed from Eiben et al. (1994) for solving the TSP, which is one of the hard combinatorial scheduling problem. The first gene value in the child is always inherited from the first gene value in the first parent. Then, for each parent its marker is set to the first successor of the previously selected value which does not already occur in the child (each individual must be seen as a cycle in order for this to work). The value to be inherited by the child is chosen based on which value occurs most frequently in the parents. If no value is in the majority, the marked value in the first parent is chosen to inherit. Figure 2.17 illustrates occurrence based adjacency based crossover.

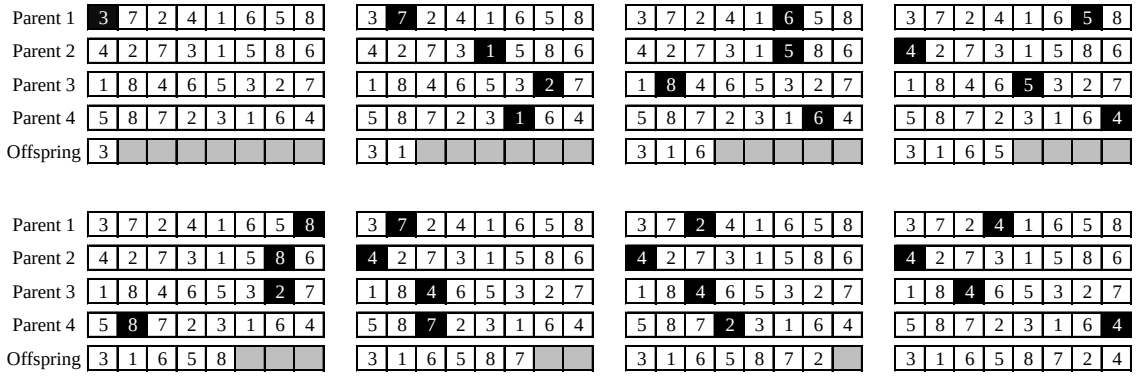


Figure 2.17: OB-ABC

2.4.2 Multi-Parent Extension of Partially Mapped Crossover (MPPMX)

MPPMX crossover is originated from the partially mapped crossover PMX method that was used by Ting et al. (2010) in the TSP. The difference between PPX and MPPMX is that they use multi-parents for recombination. In this way, Ting et al. (2010) proposed the suitable methods to legalize the chromosome into feasible solution.

Their crossover can be done in four steps:

Step 1 : Selection substring - Cut the parents into two substrings.

Step 2 : Substring exchange - Exchange the selected substrings.

Step 3 : Mapping list determination- Determine mapping relationship on selected substring.

Step 4 : Offspring legalization - Legalize the offspring into feasible solution.

As a result, the MPPMX test shows significant improvement in results compared to the PMX when applied to solve the same problem. The best solutions appear in the different number of parents for different problems.

2.5 Hybrid GA

In the GA strategies, hybridization of GA with other methods or local search methods provides good results in solving the problems. In such hybridization, the GA capitalizes on the strength of the local search method in locating the optimal or near optimal solutions.

Application of GA will be limited in application for problems when the problem size increases (Sivanandam and Deepa, 2008). For example, GA will encounter premature convergence when the complexity of the problem increases. This is because high complexity in JSSP will be lead to the high search space and solution pool will be dominated by certain individuals before the best result can be reached. Hence, modifications made to the structure or hybridization of the GA with other methods will make the resultant GA more capable in finding solutions.

Complex JSSP contains very large search space, this increases the computation cost as it takes a longer time to finish an iteration, which is proportional to the population size. Cantú-Paz (1998) pioneered the concept of parallel GA, which divides a task into smaller chunks and solves the chunks simultaneously by using multi-processor. The PGA subdivides the population into subpopulations to decrease the time of computation and the best individuals are shared between the subpopulations through migration. Yusof et al. (2011) harnessed the power of PGA by isolating the subpopulations from each other and running them in the GA by using different computers to reduce the time of computation.

The research of Park et al. (2003) proposed another idea, the Island-parallel GA. The GA maintains distinct subpopulations which act as single GAs. Some individuals can migrate from one subpopulation to another at certain intervals. The migration among subpopulations can retard premature convergence and may be allowed to evolve independently.

Sels et al. (2011) used the scatter search algorithm that had been proposed by Glover (1998) to split the single population into a diverse and high quality set in order

to exchange information between the individuals in a controlled way. The extension of splitting a single to a dual population acts as a stimulator to add diversity in the search process.

The extracted behavior of the methods, Watanabe et al. (2005) proposed the use of crossover search phase into the GA with search area adaption. This modified GA has capacity for adapting to the structure of the solutions space.

In the representation of the job shop scheduling, chromosomes that contain a sequence of all operations that decoded to the real schedule according to the gene sequences will have high redundancy at the tail of the chromosome and little significance of rear genes on the overall schedule quality. To solve these problems, Song et al. (2000) applied the heuristic method on the tail of the chromosome to reduce the redundancy. The method was also used by Ripon et al. (2011) in proposing a new crossover operator called improved precedence preservation crossover (IPPX). In this crossover operator the PPX crossover will be modified by adding the heuristic method. The crossover will perform PPX at the early gene in the chromosomes then follow it by the heuristic method. The method shows improvement in time reduction compared to the original PPX operator.

2.5.1 Hybridization with Local Search

GA has its own limitation in finding the global local optimum and identifying the local optima. Therefore, GA needs to be coupled with a local search technique. The configuration of this hybrid GA is not straightforward and may vary by adopting different local search techniques. The idea of combining the GA with local search is not

new and it has been studied intensively. Various methods of hybridization have been investigated extensively to test their ability to adapt to the problems in JSSP.

In the GA strategies, hybridization of GA with local search methods provided good results in solving the problems, where GA capitalized on the strength of the local search in identifying the optimal or near optimal solutions. For example, Gonçalves et al. (2005) and Zhang et al. (2008) embedded the local search procedure of Nowicki and Smutnicki (1996) into GA due to the effectiveness of this particular local search which increases the performance of GA.

Hasan et al. (2007) proposed the use of heuristic job ordering within a genetic algorithm. The heuristic ordering guides the individuals to a global optimum instead of conventional GA which may lead to convergence to local minima. It is done by using the heuristic information in the machine's sequences for each job. The highest priority machine in a schedule will be chosen first to be incorporated into the reproduction process. Algorithm 2.3 illustrates hybrid GA proposed by Hasan et al. (2007).

Algorithm 2.3: Hybrid GA

```
Initialize population

Evaluation

While termination criterion=false do

        Selection
        Heuristic ordering ← insert
        Crossover
        Mutation
        Evaluation
        Reinsertion

End

Best solution is found
```

Besides that, there are various ways to implement the combination of GA with SA to build a hybrid GA. The first one is by using parallel evolution structure. This framework which combines the GA and SA is called GASA by Wang and Zheng (2001) and can be described as below:

Step1: GA provided a population for SA to perform that using Metropolis structure sample for each solution until equilibrium condition is reach.

Step 2: Solution from SA is used by GA to continue parallel which means that the individual created from the SA is used to perform reproduction process in crossover and mutation operators.

Step 3: Result for the operator is used to search locally by SA for the current solution to achieve better solution.

Step 4: The procedure is repeated until termination.

Another effort that can be presented, for the hybrid relationship between GA and SA, is SA is used to replace the mutation operator in GA and becomes an operator in GA (Wang and Zheng, 2002).

Local search such as TS, provide the intensification for the solution, while GA provides diversification in the total solutions. Intensification tends to search for the optimal solution in the current solution; meanwhile diversification is an algorithmic mechanism that functions by forcing the search into previously unexplored areas of the search space (Zäpfel et al., 2010). The advantage of adding the TS with other methods is that it will outperform other optimization methods. For example, the hybrid SA with TS (TSSA), proposed by Zhang et al. (2008), is able to get the best solution for certain unsolved problems.

Application of tabu search in GA was implemented by Ombuki and Ventresca (2004) for the purpose of finding possible solutions for JSSP. This hybrid strategy using the genetic algorithm reinforced with a tabu search. In this hybrid GA, TS technique is applied on a given set of chromosomes for a number of generations to exploit for better solution. In this case, intensification is performed by tabu search and diversification is performed by GA.

2.6 Benchmarks Problems

In current JSSP benchmark problems, the FT problem is the oldest benchmark problem which has been referred by many researchers in JSSP area. In the benchmarks, the problem size can be as small as 6 jobs, 6 machines, which denotes as 6x6, and can be as large as 100x20.

The possible solutions for the problem can be calculated by $(n!)^m$, where n denotes the number of jobs and m denotes the number of machines. Hence, there is a large range from small problem to big problem. Based on Table 2.2, the solution spaces of LA31-LA35 (30x10) are bigger than the FT10 (10x10) in the calculation. Logically, it may lead to the consideration that the LA31-LA35 is harder than FT10. But according to the data acquired from Jain and Meeran (1999), LA31-LA35 is considered as an easy problem while FT10 is thought to be a difficult one. This shows that the problem structures have the most significant influence on the problem difficulty.

The measurements of the hard problems are summarized by Jain and Meeran (1999) as: $N\ (total\ operation) \geq 200,$ where $n \geq 15, m \geq 10,$ and $n < 2.5m$ which

satisfy $2SET$ principle, $k = 2$ which $2SET$ principle, $k = 2$ may refer to the Demirkol et al. (1998). The hard problems in the benchmarks are mostly used to test the researchers' algorithms.

Table 2.2: Benchmarks for JSSP

Benchmarks	Problem size (Jobs x Machines)			Proposed by
FT06	6	x	6	
FT10	10	x	10	Fisher and Thompson (1963)
FT20	20	x	5	
LA01 - LA05	10	x	5	
LA05 - LA10	15	x	5	
LA11 - LA15	20	x	5	
LA16 - LA20	10	x	10	Lawrence (1984)
LA21 - LA25	15	x	10	
LA26 - LA30	20	x	10	
LA31 - LA35	30	x	10	
LA36 - LA40	15	x	15	
ABZ5 - ABZ6	10	x	10	Adams et al. (1988)
ABZ7 - ABZ9	20	x	15	
ORB01 - ORB10	10	x	10	Applegate and Cook (1991)
SWV01 - SWV05	20	x	10	
SWV06 - SWV10	20	x	15	Storer et al. (1992)
SWV11 - SWV20	50	x	10	
YN1 - YN4	20	x	20	Yamada and Nakano (1992)
TA01 - TA10	15	x	15	
TA11 - TA20	20	x	15	
TA21 - TA30	20	x	20	
TA31 - TA40	30	x	15	Taillard (1993)
TA41 - TA50	30	x	20	
TA51 - TA60	50	x	15	
TA61 - TA70	50	x	20	
TA71 - TA80	100	x	20	

2.7 Conclusion

Efficiency of GA in adapting different problem size can be increased by hybridizing GA with other methods. In previous studies, GA has been well implemented in solving the JSSP. GA is able to perform as a powerful tool in solving the problem when combined with local search methods.

The results from the researches show that GA is not restricted to a single procedure and performs better when its structure is modified or it is hybridized with other methods to increase the accuracy of searching solutions. Hence, GA can be modified accordingly to suit the problem at hand, including selecting several parents for the crossover operation.

CHAPTER 3

GENETIC ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM

3.1 Introduction

GA structure can be modified to exhibit its capability for solving different types of problems. Various stages of GA can be modified easily to adapt to different applications. In particular, the application of GA is not restricted to the use of two parents for crossover, rather multi-parents which is a combination of more than two parents can also be performed.

In the past, GA had been studied intensively to measure its performance on different problems, capabilities and adaptations required to adapt it for the specific problem, including JSSP. Various GA strategies have been developed to determine the most suitable and problem specific approach in solving a particular problem. However many previous researches, most of the GAs' crossovers are based on the two parent crossover method. Multi-parents are still rarely utilized especially for solving JSSP.

In this study, we propose the extended precedence preservative crossover (EPPX). This crossover operator originated from the precedence preservative crossover (PPX) method first proposed by Bierwirth et al. (1996). The advantage of this crossover technique is that it is able to maintain the phynetopical of the chromosome. This crossover operator is extended from two parents to multi-parents by using the same approach. Operation based is used because it can be easily interpreted into feasible solution.

In order to increase the efficiency of this modified GA, hybrid GA is introduced. The idea of combining GA and some local search techniques for solving optimization problems was discussed in Chapter 2 (Section 2.5). In our proposed hybrid GA, a neighborhood search is added into the GA structure. This neighborhood search adapts the swapping operations proposed by Nowicki and Smutnicki (1996).

The set of active schedules generated by the local search procedure usually contains a very large search space and poor quality in terms of makespan, because of the fact that the solution space consists of many high delay times for the concerned operations. In order to reduce the size of the search space we used the concept of iterative forward-backward pass to reduce or eliminate poor solutions and increase the quality of the overall search space.

This study aims to propose the new multi-parents crossover and hybridization of GA for the JSSP. This hybrid GA is tested on different benchmark sets of JSSP to assess its performance and is discussed in latter part of this chapter. The algorithm is evaluated by the efficiency of the GA in searching for the optimal or near optimal solutions. The flow chart describing every step of the research methodology is shown in Figure 3.1.

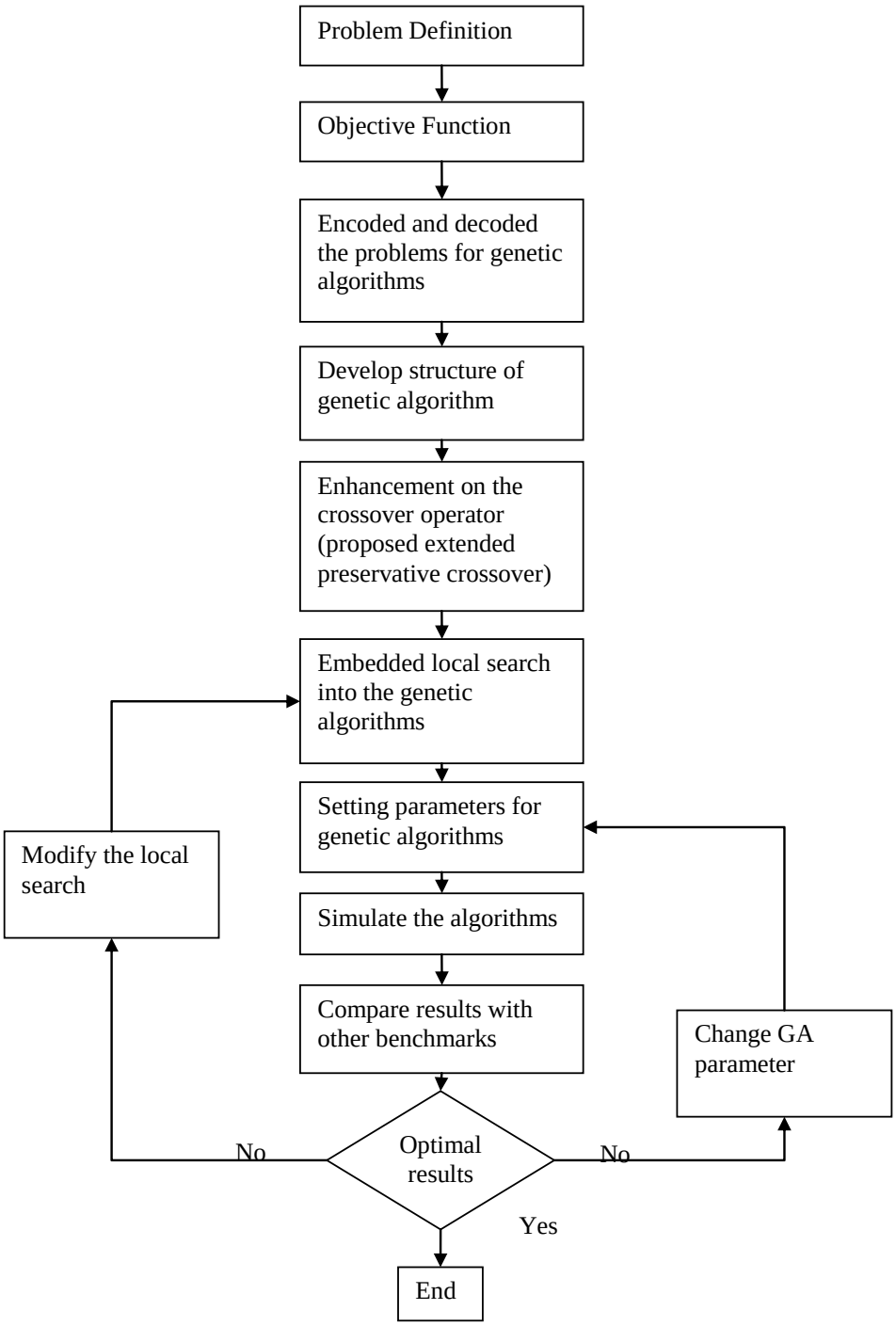


Figure 3.1: Flow Chart of Research Methodology

3.2 Representation

A suitable representation is vital in any GA algorithm. The chromosome in this study is represented as a permutation integer with repetition; a strategy proposed by Bierwirth (1995). This representation is called an operation based representation (Cheng et al., 1996) where integers in the chromosome represent the sequences of the jobs in the schedule. In this representation, number i where $i = 1, 2, 3$ ... represents the number of jobs and i is repeated according to the number of operations required. Figure 3.2 illustrates the representation of 3 jobs and 3 operations/machines. The chromosome is represented as $[1\ 2\ 3\ 3\ 2\ 2\ 3\ 1\ 1]$, where numbers 1, 2, and 3 in the chromosome represent job 1, 2, and 3 respectively. Each job consists of three operations so it is repeated three times in the chromosome. The chromosome is scanned from left to right with the j^{th} occurrence of a job number referring to the j^{th} operation in the technological sequence of this job. The chromosome created is always feasible and legal. For this type of representation, the total feasible solutions can be calculated as $(n!)^m$.

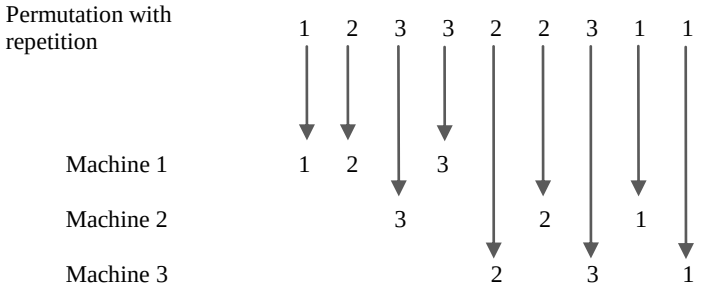


Figure 3.2: Permutation with Repetition Representation for 3 Jobs 3 Machines

3.3 Decoding

A scheduling can be built by decoding the genes of the chromosome from left to right to a list of ordered operations. The first operation in the list is scheduled first, then the second operation, and so on. While placing the job in the schedule, it must meet the technological requirement and precedence constraints. Referring to the chromosome representation given in Figure 3.2, the technological requirement for the chromosome is based on Table 3.1.

When a job (gene) is placed into a schedule, there are two considerations:

1) Finish time of the predecessor operation

2) Finish time of the last operation in the same machine

The job is placed based on the possible earliest start time. If time of $1 > 2$, the operation will start by referring the finish time as 1. On the other hand, if time of $2 > 1$, the operation will start by using the finish time of 2. Based on this operational arrangement, the operation inserted will always be at the last phase in the operation of the machine. Figure 3.3 illustrates that the schedule is built by decoding the genes starting from left to the right in the chromosome.

Table 3.1: Example of 3 Job and 3 Machine Problem

	Job	**Operation routing**		
		1	**2**	**3**
Processing time	1	3	3	2
	2	1	5	3
	3	3	2	3
Machine sequence	1	M1	M2	M3
	2	M1	M3	M2
	3	M2	M1	M3

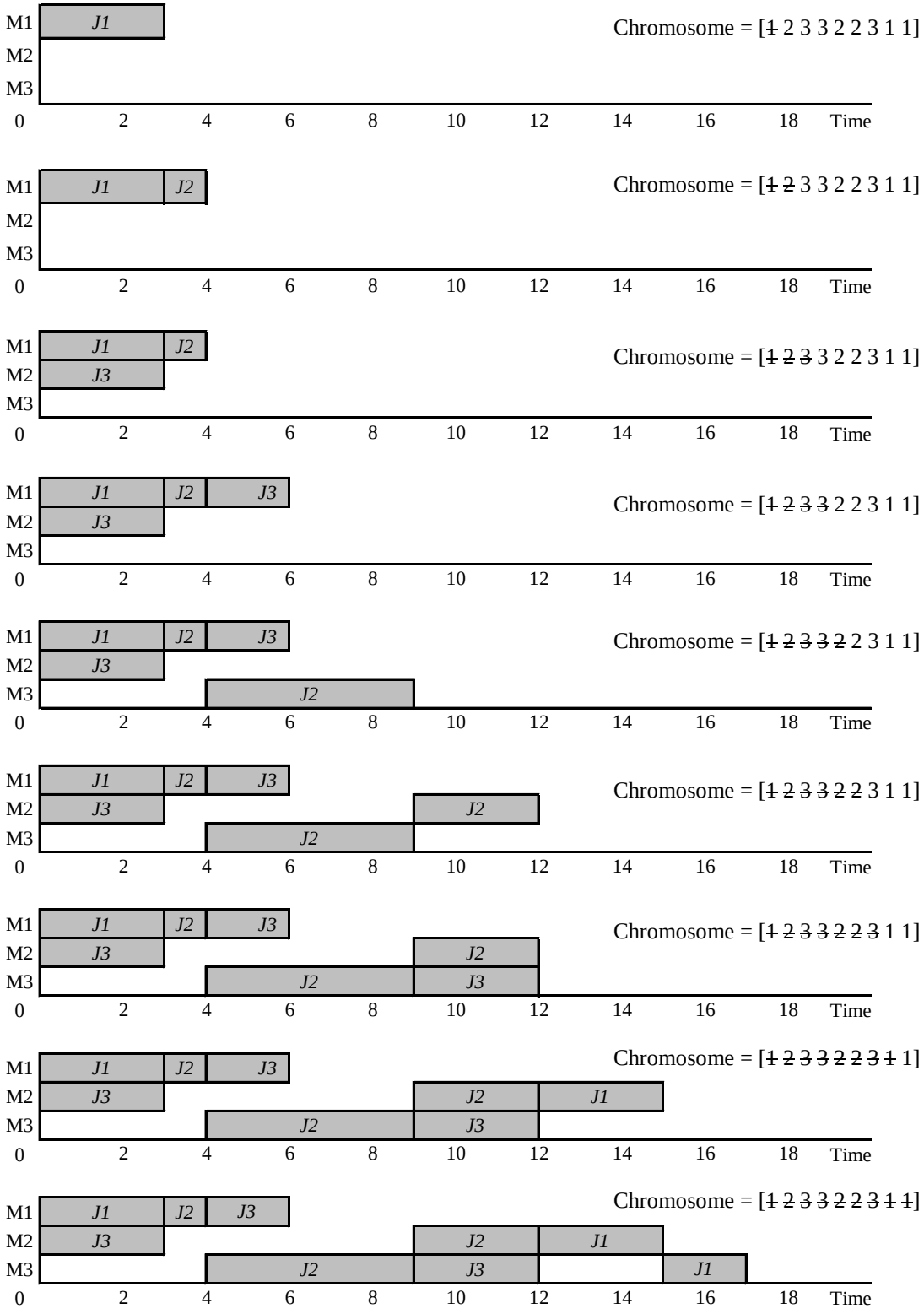
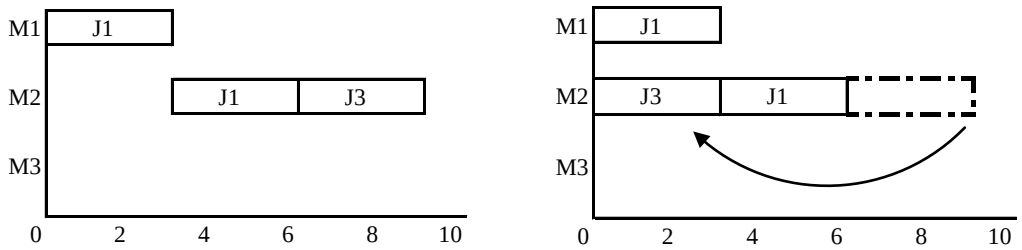


Figure 3.3: Schedule for JSSP

3.3.1 Active Schedule Builder

Optimal solution of the scheduling always lies in the active schedule with no permissible left shift in the schedule being possible. Recombination of such active schedule chromosomes produces good solutions (Yamada and Nakano, 1992). In order to build the active schedule chromosome, the chromosome needs to be decoded into a feasible schedule. This is achieved by constructing a schedule builder that performs a simple local search.

An active schedule can be built by selecting the gene (job) of the chromosome from left to right and inserting it into a schedule with an active schedule builder and then deleting it from the chromosome (Gen et al., 1994). The job always finds the earliest completion time to be inserted by using a simple local search. Figure 3.4(a) illustrates the scheduling without local search and the job will be placed by following the sequences encoded in the chromosome [1 1 3 ...]. Applying the local search enables the job to find the possible vacant time interval before appending an operation at the last position (Figure 3.4(b)) and the chromosome encoded becomes [1 3 1 ...].



(a) Semi-active schedule without simple (b) Active schedule after simple local search

Figure 3.4: Local Search Procedure

When the schedule is finished, it is encoded into the chromosome and the arrangement in the genes is the reference for the start time sequences in the schedule. The gene that is placed earlier into a schedule is forced to the left in the chromosome according to its earliest start time in the schedule.

3.4 Proposed Hybrid GA Structure

In the hybrid GA structure, the EEPX crossover is used as the crossover operator. This recombination operator attempts to combine the best features of each individual to get the best solutions. Besides that, the local search and the search space reduction method will also aid the flow of GA. The hybrid GA algorithm is represented by Algorithm 3.1, as illustrated below.

Algorithm 3.1: Genetic Algorithm

```
Initialize population

while termination criterion

       Selection
       Crossover
       Mutation
       Iterative forward-backward pass
       Neighborhood search on critical path
       Reinsertion

End while

Best Solution
```

In this hybridization, the intensification and diversification are executed by different operators. For intensification, the local search (neighborhood search on critical path) exploits the best possible solution in an individual. GA structure performs

diversification by providing different individuals for the local search. This interrelated behavior makes the search more efficient.

The quality of the offspring generated by crossover and mutation is generally unknown. When the offspring reaches an iterative forward-backward pass, the offspring is evaluated and the quality of the offspring is upgraded by rearranging the genes in the offspring in this scheduling method.

During the current research, this hybrid GA is modified from time to time. If the result of the simulation does not reach the optimal or deviates too far from the best known solutions, the first consideration of modification focuses on local search operator. Several methods of local search have been tested and it was found that the local search performed on the critical path has the highest impact in generating a schedule. Therefore, the use of neighborhood search in the critical path is proposed. After that, if the performance of the GA reaches an acceptable condition, the parameters in GA, such as crossover and mutation rates are adjusted to optimize its functionality.

3.5 Initial Population

The population generates potential solutions for GA to search in solution space. The individuals that are generated randomly from inheritance must be presented in the form of operation based representation. At the initial stage of the population, the individuals generated normally have very poor quality in terms of makespan. These poor quality individuals go through the reproduction process to recombine and become better solutions. New population is generated after the recombination process and

reused for the next process, finally the iteration stopped when the termination criterion is reached.

3.6 Termination Criteria

Termination criteria are set to stop the GA from running in the unlimited iteration mode. Termination of the searching procedure is active when GA has achieved the optimal solution (if there exists one) or reaches the maximum number of generations. If the number of generations is set at a high value, it is time consuming and ineffective because the potential solutions at the end of the generation are converging into a single solution (because all the chromosomes are similar to each other). Thus in JSSP, the maximum number of generations is set based on the population. For example, if the population size is small, the maximum number of generations is also small.

In the multi-parents crossover we proposed, the parents are recombined to generate one child. Therefore, there exists different numbers of parents for recombination with different total number of offspring (solutions). The total solutions can be defined as:

$$total\ solutions = \frac{MAXGen \times population\ \ size}{number\ \ of\ parents} \tag{3.1}$$

$MAXGen$ = maximum number of generations
$population\ size$ = total number of individuals in the search space
$number\ of\ parents$ = parents that are used for the recombination process

Referring to Eq. (3.1), if the number of parents selected is increased, the total number of solutions generated is reduced. When comparisons are needed to be made between different numbers of parents, it is not comparable because the total solutions

generated are different from each other. To preserve consistency, the total solutions will be fixed so different numbers of parents are able to generate approximately the same number of solutions for comparison.

In Eq. (3.1), the population size is fixed to control the total individuals involved in evolutionary process. The only variables that can be adjusted are maximum number of generations and number of parents. The total number of generations is adjusted to make sure that different number of parents for recombination generates approximately the same number of total solutions. Maximum number of generation is calculated by referring to the Eq. (3.2). The maximum number of generations, $MAXGen$, is adjusted as follows:

$$MAXGen = \frac{total\ \ solutions \times number\ \ of\ \ parents}{population\ \ \ size} \tag{3.2}$$

3.7 Selection

In selection operator, we use Stochastic Universal Sampling (SUS). This fitness based proportionate selection technique chooses the chromosomes for recombination process with minimum spread and zero bias. This ensures that even poor quality individuals have a chance to participate in the solution process. Unlike the tournament search and the roulette wheel selection, the selection probabilities for good quality individuals are very high and the chance to select other individuals is low resulting in those solutions dominating the population (Goldberg, 1989). Sometimes there are some good features in the poor quality individuals which combined with other individuals may produce good result because solutions from recombination between individuals are

unpredictable. So, this is the principal reason of using the SUS as the selection parameter in the GA.

3.8 Reinsertion

Elitism strategy is used to maintain a good solution for the population. Elitism strategy is applied to maintain the best fitness of the population, thus ensuring that the good individual is propagated to the next generation.

In the reinsertion procedure, some of the new offspring replace the bad individuals in the previous population to generate new population. The selected fittest individuals that are used to replace the bad individuals are the same proportion in order to maintain the size of the population. Under this selection pressure, the new population generated is expected to be better than previous.

3.9 Mutation

Mutation operator acts as a mechanism to diversify the individuals in order to escape from the local optima. In this study, the mutation is applied by swapping two genes which correspond to different jobs, in two different positions in the same chromosome. The process is repeated if two genes are selected are at the same position or represent the same job. Figure 3.5 illustrates the swapping of the two genes in the chromosome.

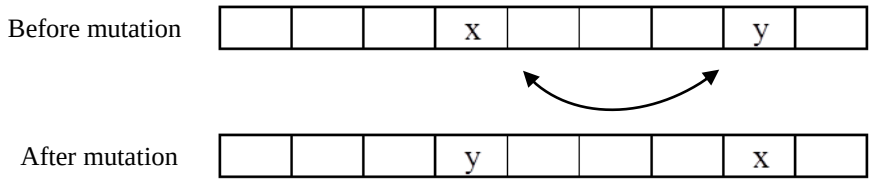


Figure 3.5: Mutation by Swapping Two Genes in the Chromosome

This mutation operator does not consider the restrictions of precedence constraints and operation sequences and it is just implemented by swapping the different genes. As a result of the mutation, the sequences of the operations in the machine are changed and this may affect the whole quality of the offspring.

3.10 Proposed Extended Precedence Preservative Crossover (EPPX)

In GA, there are no limitations that the recombination process needs only two parents, rather multi-parents consisting of more than two parents are also acceptable. Some of the multi-parents crossover operators are extended from the two parents' crossover operators for recombination process (Tsutsui and Jain, 1998; Tsutsui et al., 1999; Wu et al., 2009; Ting et al., 2010). In this hybrid GA, multi-parents crossover, EPPX, is an extension from PPX.

A crossover mask in the form of a vector is generated randomly to determine the genes in which parent, specified in the mask, to be selected for recombination. The multi-parents recombine into a single offspring (Figure 3.6 (a)). Starting from the first element on the mask vector, the first gene in that parent 1 is selected. The selected job (job 3) is eliminated in the other parents (Figure 3.6 (b, c)). The second element in the

mask indicates that the first element (after deletion) is to be selected also from parent 1 (Figure 3.6 (c)). The third element in the mask shows that the first element in parent 3 is selected (Figure 3.6(d)). The process continues until all the elements in the mask have been examined.

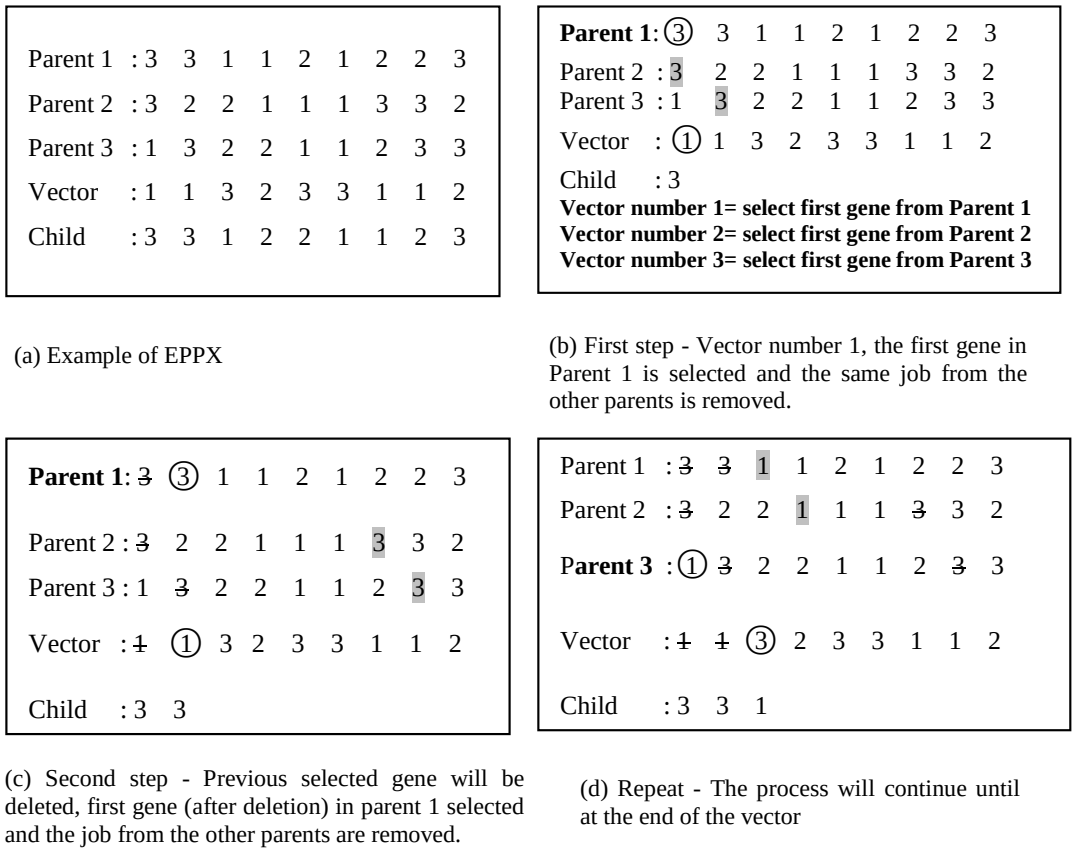


Figure 3.6: EPPX

The offspring created contains the elements from the parents with the hope that the offspring is better than the parents. This crossover always generates feasible solutions due to the offspring that are created are always legal; therefore legalization of the offspring which is very time consuming is eliminated. Higher number of parents can be easily adapted in the crossover for multi-parents recombination. Pseudo code for EPPX is presented as in Algorithm 3.2.

Algorithm 3.2: Pseudo Code for EPPX (3 Parents)

```
Crossover vector generated randomly

Three parents selected ->S1, S2, and S3

for    k=1 to length of the chromosome do


       Select vector number by position k-th starting from the left in  vector

       case    vector number of

              Vector number 1:
                     Choose first gene at left most S1
                     Search same job number at left most in S2 and S3
                     Remove the first gene in S1
                     Remove the gene searched in S2 and S3

              Vector number 2:
                     Choose first gene at left most S2
                     Search same job number at left most in S1 and S3
                     Remove the first gene in S2
                     Remove the gene searched in S1 and S3

              Vector number 3:
                     Choose first gene at left most S3
                     Search same job number at left most in S1 and S2
                     Remove the first gene selected in S3
                     Remove the gene searched in S1 and S2
              end case

       Selected gene insert to new chromosome by sequence from left to  right

end for
```

3.11 Iterative Forward-Backward Pass

The set of active schedules generated by the shifted left operations usually contain very large search space and are poor quality in terms of makespan because it consists of many high delay times of the operations. In order to reduce the size of search space and to reduce makespan, we used the concept of iterative forward-backward pass.

Lova et al. (2000) applied the iterative forward-backward pass into their multi-project scheduling which is similar with JSSP in which it also has precedence constraints to generate a schedule. The authors use this iterative method to reduce the

makespan time in their projects thus reduce the cost of the project and it is claimed that the method shows some improvement when compared with other methods.

The iterative forward-backward pass approach is applied in hybrid GA because of its capability to reduce makespan time. This approach is inserted in the structure of GA as an operator. An individual that passes through this operator is rescheduled to reduce the makespan time and then new individual is produced to a higher quality. This method consists of two types of scheduling methods that are used iteratively: Forward Pass and Backward Pass.

A Forward Pass is a process of shifting left the operations in a schedule, starting from beginning of the schedule until the end of the schedule, $max\left(F_{ij}\right)$ and the operations are able to be shifted left until the time equals to zero. In Backward Pass, the process starts from the end of the schedule, $max\left(F_{ij}\right)$ and ends at the beginning of the schedule, $min\ (F_{i1})$ in which the operations is shifted right until the time equals to the $max\left(F_{ij}\right)$. The Forward Pass is similar to the local search illustrated in Figure 3.4 and the Backward Pass is presented in Figure 3.7.

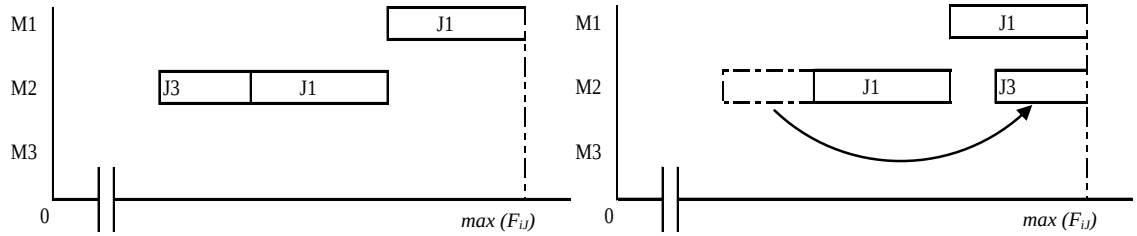
When applied, the iterative forward-backward pass approach is able to shorten the makespan time of the schedule. The iterative forward-backward can be described in the following steps:

Step 1: Chromosome is scanned from left to right to generate an active schedule by Forward Pass (see example Figure 3.4). Next, the new schedule is decode into chromosome with maximum makespan, $max\left(F_{ij}\right)$.

Step 2: New chromosome from the *Step 1* is used by Backward Pass. The chromosome is scanned from right to left with start time $max\left(F_{iJ}\right)$. A new chromosome is generated and the makespan of this schedule represented as below:

$$BC_{max} = max\left(F_{iJ}\right) - min(F_{i1}) \tag{3.3}$$

Step3: If the makespan $BC_{max} < max\left(F_{iJ}\right)$, there is improvement of the schedule and the chromosome generated by Backward Pass is used in *Step 1*. *Step 1* and *Step 2* are repeated until there is no more improvement on the schedule.



Chromosome = [… **3** 1 1]

(a) Before Backward Pass

Chromosome = [… 1 **3** 1]

(a) After Backward Pass

Figure 3.7: Backward Pass

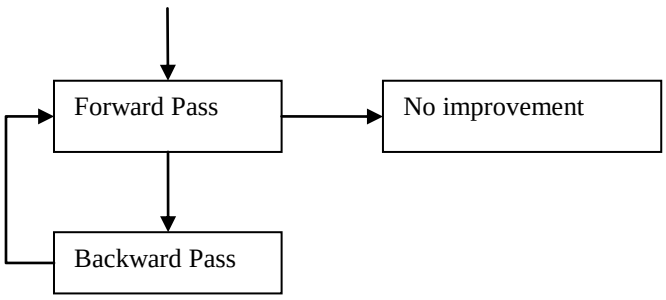


Figure 3.8: Iterative Forward-Backward Pass

Figure 3.8 illustrates the Iterative Forward-Backward Pass carried out in this study. It is noted that in this iterative function, the makespan of the both processes is

mutually restricted hence the makespan of new solution generated is either lesser or remain unchanged. The search space is reduced hence the overall of the quality of the chromosomes is improved, increasing the possibility of getting the optimal or near optimal solutions.

3.12 Neighborhood Search

Reduction of the search space does not always guarantee that the optimal solution will be found. Kelly and Walker (1959) noted that one of the effective ways to change and modify the scheduling time length is by changing the operations sequence in the critical path, because critical path determines the length of the process to be finished. Therefore, we use neighborhood search as exploitation mechanism to decrease the makespan and the neighborhood search starts with the identification of the critical path in the schedule generated by the scheduling process.

Critical path in the schedule is determined by using the Critical Path Method (Kelly and Walker, 1959). Operations on the critical path are called critical operations and a critical block consists of a maximal sequence of adjacent critical operations that are processed on the same machine (Nowicki and Smutnicki,1996). Figure 3.9 illustrates example of critical blocks consisting of several critical operations.

The neighborhood is defined as the random swap between two jobs in a critical block that contains two or more operations. If the critical block contains only one operation, no swap is made. All possible moves of the operations will be predetermined as illustrated in Figure 3.9. A swap of the operations is accepted if it improves the

makespan from its present state. Otherwise, the swap is undone if all of the possible swaps do not improve the makespan and the original solution is maintained. Once the swap is accepted, a new critical path is identified. The procedure is repeated and stops if there are no swaps that can improve the makespan. In this neighborhood search, the process is iterative and the iteration is terminated if no improvement is found. Algorithm 3.3 presents the pseudo code for neighborhood search.

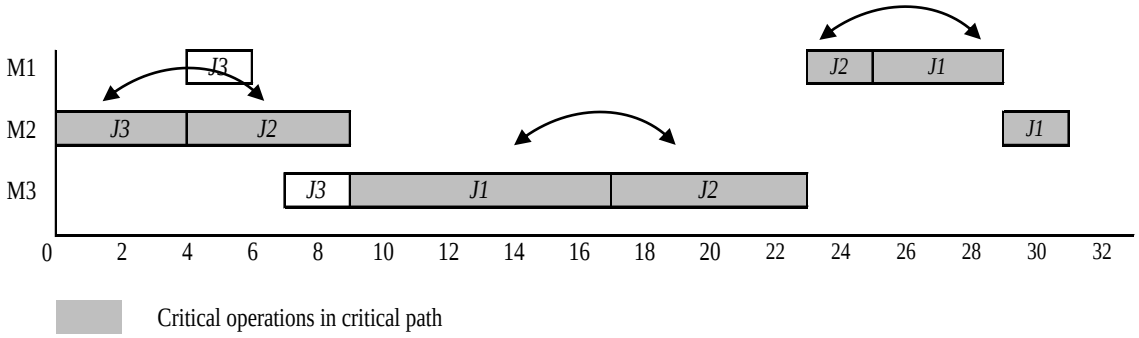


Figure 3.9: Critical Path, Critical Operations and Possible Operations Swaps

Algorithm 3.3: Pseudo Code for Neighborhood Search

```
while      New solution  accepted = true

New solutions accepted = false
Determine the critical path, critical block in New schedule
List out the possible swaps of the operations

           for     k=1 to total of possible swaps do
                   Swap a pair of operations
                   New schedule generated and makepsan recalculated (New
                   makespan)

                   if      New makespan< Current makespan
                           Current makespan = New Makespan
                           New solution accepted = true

                   end if

           end for

 end while
```

Local search in this hybrid GA tries to find the best solution that is attainable by using the single offspring. It exploits the best possible solution by using neighborhood search method.

3.13 Conclusion

This chapter presents a hybrid genetic algorithm with multi-parents crossover, EPPX, for the job shops scheduling problem. EPPX is a variation of the precedence preservative crossover (PPX) which is one of the crossovers that perform well to find the solutions for the JSSP. EPPX is based on a vector to determine the gene selected in recombination for the next generation. Legalization of children (offspring) can be eliminated due to the JSSP representation encoded by using permutation with repetition that guarantees the feasibility of chromosomes.

The hybrid GA combines with neighborhood search in which GA performs the exploration of the population and the neighborhood search performs the exploitation around individuals. The chromosome represented by operation-based representation is used to generate an active schedule through iterative forward-backward pass which can further reduce the search space.

The simulations are performed on a set of benchmarks from the literatures and the results are compared in the following chapter to ensure the sustainability of multi-parents recombination in solving the JSSP.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

The hybrid GA is developed by using MATLAB 7.11 R2010b and the simulations are run on workstation Intel Xeon CPU E5620 12GB RAM. The source of the benchmarks are from the OR library (Beasley, 1990). Selected benchmarks that are used to test the hybrid GA are the FT, ABZ, and ORB problems. These benchmarks were chosen because they are considered as hard problems and often used by other researchers for comparison and testing their algorithms in solving the JSSP.

4.2 Data Set – Benchmarks Problems

4.2.1 FT Problem

The FT problem which was developed by Fisher and Thompson (1963) has been widely applied in different algorithms. The FT10 and FT20 are considered as difficult computational problems especially the FT10 problem which is referred as a "notorious" problem because it remained unsolved for 20 years and now is no longer computationally challenging as most of the algorithms managed to attain optimal solution.

In Table 4.1, the first column shows the names of the instances which are followed by the total job and machines in the problem. The last column records the optimal solutions for the FT that has been solved optimally in past literatures. There are

only three types of problem sizes in the FT problem. They are written as $jobs$ x $machine$, e.g. 6x6 for FT06, 10x10 for FT10, and 20x5 for FT20.

Table 4.1: Instances for FT Problem

Instances	No. of Jobs	No. of Machines	Optimal
FT 06	6	6	55
FT 10	10	10	930
FT 20	20	5	1165

4.2.2 ABZ Problem

The ABZ problem proposed by Adams et al. (1988) contains the problem that is more difficult than the FT10 especially the instances of ABZ8 and ABZ9 which are still open problems. Table 4.2 shows the selected instances from the library with their best known solutions (BKS). Note that instances with asterisks are part of the ten tough problems (proposed by Applegate and Cook (1991)) which are more difficult than the FT10 problem.

Table 4.2: Instances for ABZ Problem

Instances	No. of Jobs	No. of Machines	BKS
ABZ5	10	10	1234
ABZ6	10	10	943
ABZ7*	20	15	656
ABZ8*	20	15	665
ABZ9*	20	15	678

4.2.3 ORB Problem

The ORB problem proposed by Applegate and Cook (1991) consists of instances with the same size problems of the FT10. Applegate and Cook (1991) collected the instances from different authors then renamed them as ORB.

Table 4.3: Instances for ORB Problem

Instances	No. of Jobs	No. of Machines	BKS
ORB01	10	10	1059
ORB02	10	10	888
ORB03	10	10	1005
ORB04	10	10	1005
ORB05	10	10	887
ORB06	10	10	1010
ORB07	10	10	397
ORB08	10	10	899
ORB09	10	10	934
ORB10	10	10	944

4.3 Hybrid GA Parameters

In this hybrid GA, the parameters that need to be set are:

- Population size
- Maximum number of generation
- Reinsertion rate
- Crossover rate
- Mutation rate

These parameters varied in the GA when applied in different fields of the problems. Thus, the parameters in our algorithm need to be set first before being applied to the JSSP. The population size and reinsertion rate portion refer to the mostly used value in the GAs and the maximum number of generation is adjusted based on the number of parents. Parameters such as the crossover rate and mutation rate need to be tested before being applied to other problems.

A population is set with 100 individuals for the problems FT06, FT10, ABZ5, ABZ6, and ORB01-ORB10. This is because this size of population is frequently used in other GAs (Sivanandam and Deepa, 2008). For the problems that contain large problems sizes and variables (FT20, ABZ7-ABZ9), their population sizes are increased to 150 to acquire more chances of obtaining optimal or near optimal solutions. After the recombination process, elitism strategy is applied and 10% of the best fitness new offspring replace the 10% of the worst individuals in the previous population to generate a new population.

Chapter 3 (Section 3.5) explains how the different numbers of parents for recombination generate different numbers of total solutions. To ensure the solutions from crossovers with different numbers of parents are fair and comparable, the maximum numbers of generations are adjusted based on the number of parents (Eq. (3.2)). In calculating the maximum number of generations, the $MAXGen$, $total\ solutions$ for instances FT06, FT10, ABZ5, ABZ6, and ORB01-ORB10 are fixed at 5000 schedules and the problems FT20 and ABZ7-ABZ9 are fixed at 10,000 schedules.

Table 4.4: Maximum Number of Generation

Maximum number of generation	Number of parents							
	3	4	5	6	7	8	9	10
100 initial individuals	150	200	250	300	350	400	450	500
150 initial individuals	200	267	333	400	467	533	600	667

Table 4.5: Total Solutions Generated

Generated solutions	Number of parents							
	3	4	5	6	7	8	9	10
100 initial individuals	4950	5000	5000	4800	4900	4800	4950	5000
150 initial individuals	10000	9879	9990	10000	9807	9594	10000	10005

Table 4.4 and Table 4.5 show the maximum number of generations for different numbers of parents and the total solutions generated respectively. The total solutions generated are slightly different from the original values because the total individuals in a population are divided by the number of parents before being rounded toward zero. With these numbers of generations, the results are more comparable and fair.

4.4 Parameters Testing for Hybrid GA

The parameters setting of crossover rate and mutation rate for the GA are very problem dependent. In the JSSP, there are no specific rates for these parameter values. Therefore, we use the instance FT10 to identify suitable parameters.

In testing the crossover rate and mutation rate, we consider three different cases:

Case 1:The crossover rate is set as a static value (0.9 to 0.5 with a decrement of 0.1) whilst the mutation rates are represented by the following equations:

$$p_m = 0.1 + \frac{gen}{maxgen}(0.4) \tag{4.1}$$

$$p_m = 0.1 + \frac{gen}{maxgen}(0.9) \tag{4.2}$$

p_m = mutation rate
gen = current number of generation
$maxgen$ = maximum number of generation

a) Mutation rate increases from 0.1 to 0.5 when the maximum number of generation increases (Eq. (4.1)).

b) Mutation rate increases from 0.1 to 1.0 when the maximum number of generation increases (Eq. (4.2)).

Table 4.6 shows the results of the different mutation rate for Case 1 with different crossover rate. a and b are represented Case 1(a) and Case 1(b) respectively. The lowest average solution obtains by Case 1(a) and Case 1(b) are using crossover rate at 0.6 and 0.8 respectively.

Table 4.6: Case 1 Results

	Case 1	Crossover rate				
		0.9	0.8	0.7	0.6	0.5
Average Solutions	a	979.23	978.87	972.23	968.10	976.40
	b	973.03	970.77	973.80	972.67	978.07

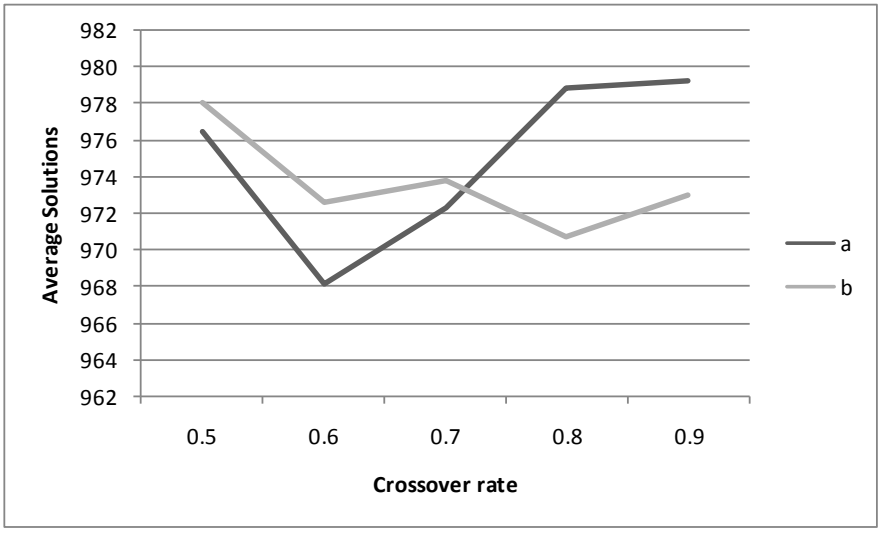


Figure 4.1: Graph for Case 1

The results in Table 4.6 are plotted into a line graph (Figure 4.1). In Figure 4.1, line-a shows that when the crossover rate increases, the average solutions (makespan) decreases by a different value of mutation rate. Line-b reflects that when the crossover rates increase, the average solutions are also increased. This graph shows that different conditions of mutation rate affects the crossover rate to attain best average solutions and the best average solutions for line-a and line-b are 0.6 and 0.8 respectively.

Case 2: Mutation rate is fixed and it varies from 0.1 to 1.0 with an increment of 0.1 and crossover rate decreases from 0.9 to 0.5 by Eq. (4.3).

$$p_c = 0.9 - \frac{gen}{maxgen}(0.4) \qquad (4.3)$$

p_c = crossover rate
gen = current number of generation
$maxgen$ = maximum number of generation

Table 4.7: Case 2 Results

Mutation rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average solutions	981.70	979.20	976.70	985.17	969.67	970.73	971.00	963.93	968.80	967.97

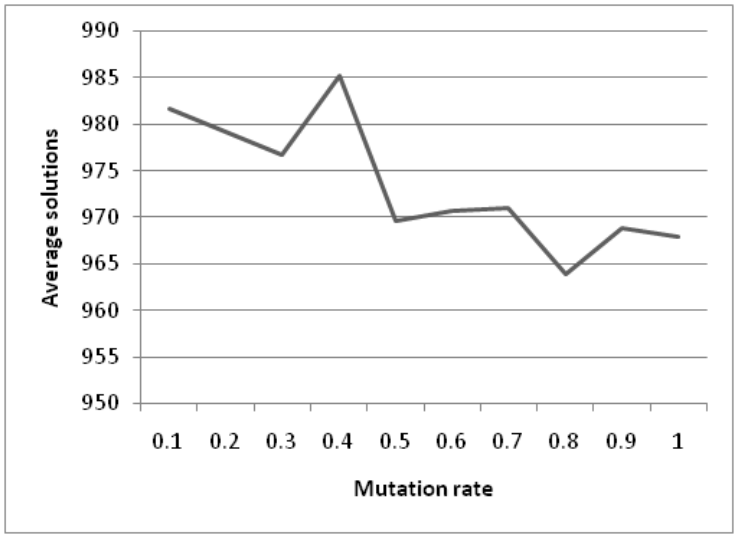


Figure 4.2: Graph for Case 2

When the mutation rate increases (Figure 4.2), the average value is in a decreasing trend and this signifies that higher mutation rates are able to obtain better average solutions.

Case 3: Adaptive crossover rate that has reverse hyperbolic relation with the mutation rate (decreases from 0.9 to 0.5 using Eq. (4.3)) and

a) Mutation rate varies from 0.1 to 1.0 by using Eq. (4.2)

b) Mutation rate varies from 0.1 to 0.5 by using Eq. (4.1)

Table 4.8: Case 3 Results

	Case 3a	Case 3b
Average solutions	970.5	979.71

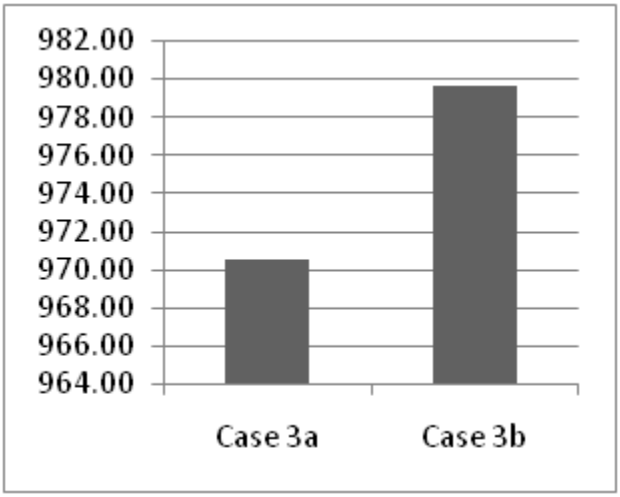


Figure 4.3: Bar Chart for Case 3

Table 4.8 displays the average solutions obtained by different cases and plotted in Figure 4.3 as bar chart. Figure 4.3 shows that with high mutation rate, the results will be better.

It can be concluded from Figure 4.1 that the crossover rate is best set in between 0.5 to 0.9 and Figure 4.2 and Figure 4.3 show that the mutation rate should be set higher. In the next section, these results are used as a guide to identify the fixed rate and apply it onto all problems to avoid confusion in parameters determination for the hybrid GA.

4.4.1 Crossover Rate

In our cases, due to the different problem sizes and different numbers of parents for recombination, we need to fix these parameters and use them for all numbers of parents. When setting these parameters, the instance FT10 is selected as the reference because it is considered as a difficult problem. Among the multi-parents crossover, three parents crossover are used as reference because we consider them as the starting point of multi-parents recombination (more than two parents).

The dependencies between the crossover and mutation rates are tested by the GA. The crossover rates are set from 1.0 to 0.5 with varied mutation rates from 0.1 to 1.0. Each case (example: crossover rate=1.0, mutation rate =0.1) will be run for 100 times and the average will be figured out. The relative errors are calculated by computing the difference between the average solutions for each crossover rate and the optimal solution of FT10 (930).

Table 4.9: Output for different Crossover Rate and Mutation Rate

	Mut_01	Mut_02	Mut_03	Mut_04	Mut_05	Mut_06	Mut_07	Mut_08	Mut_09	Mut_10	Average	Relative error (%)
Crs_10	971.96	978.20	972.64	968.06	969.28	972.34	971.58	965.46	963.24	968.12	970.09	4.31
Crs_9	967.38	975.94	965.82	968.24	970.30	965.36	963.44	961.14	961.00	960.32	965.89	3.86
Crs_8	971.54	968.00	967.24	965.92	966.10	964.28	964.58	958.92	958.84	959.12	964.45	3.70
Crs_7	973.94	968.32	969.26	963.72	963.48	961.58	958.46	957.84	959.02	958.94	963.46	3.60
Crs_6	972.96	968.32	969.30	965.26	964.02	965.10	964.08	960.16	959.90	959.58	964.87	3.75
Crs_5	971.88	975.96	975.60	969.12	962.58	967.36	960.92	961.60	961.04	957.10	966.32	3.90

Crs_10 represents crossover rate at 1.0, Crs_9 represents crossover rate at 0.9 and so on
Mut_01 represents mutation rate at 0.1, Mut_02 represents mutation rate at 0.2 and so on

In Table 4.9, the relative error for crossover rate at 0.7 appears as the lowest value compared to the other crossover rates. The frequencies of optimal solutions for FT10 at the crossover rate 0.7 are the highest in Figure 4.4. Thus, it is reasonable for us to use the crossover rate at 0.7 for other instances.

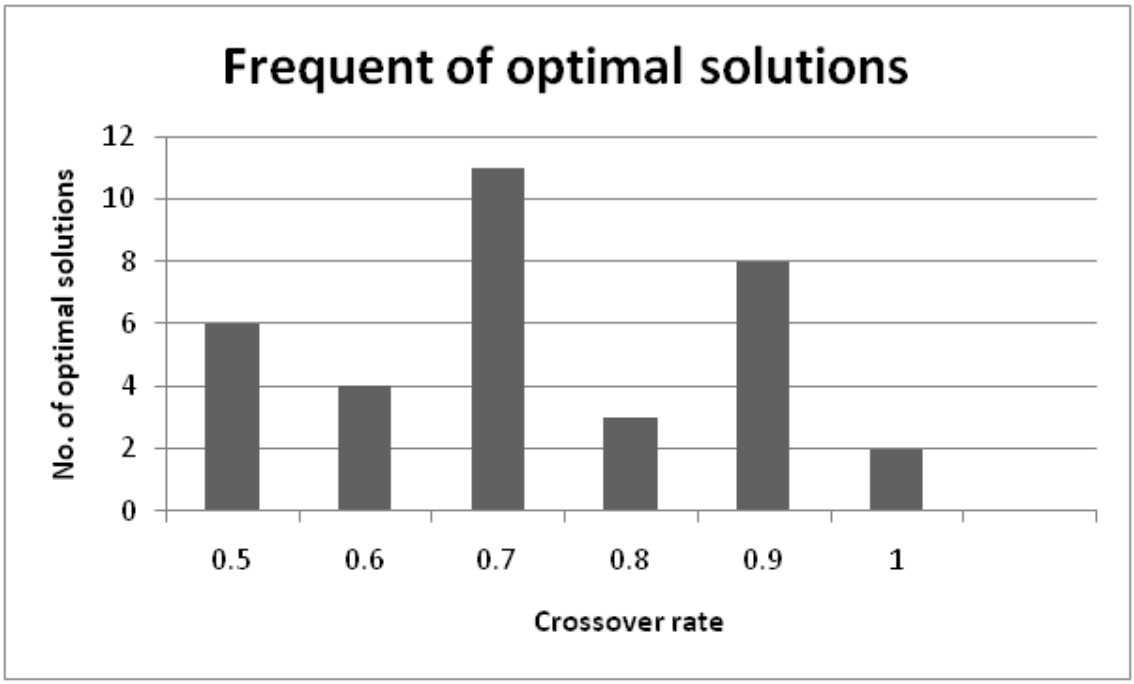


Figure 4.4: Frequent of Optimal Solutions Appear (930) at different Crossover Rate

4.4.2 Mutation Rate

In the literature, there are arguments about the influences of crossover rates and mutation rates in the GA. Some of the researchers preferred for the crossover rate to be set at a high value (higher than 0.7) with low mutation rate (not more than 0.1) as the crossover is the priority operator in the GA. On the other side, some researches show that mutation also plays an important role in the GA to generate better results. In the multi-parents crossover application, especially in the JSSP, there is a lack of information about the mutation rate values. Hence, we try to find the suitable mutation rate for our GA.

Due to the inconsistencies of the results between the crossover rates and mutation rates we obtained from the simulation, Figures 4.5 plotted the best fit line for the problems. All lines for the different crossover rates are decreasing from left to right which means that the average solutions will get better when the mutation rates increase. Thus, we conclude that the last mutation rate (1.0) performs the best when applied in this GA and it will be used as parameter for other instances.

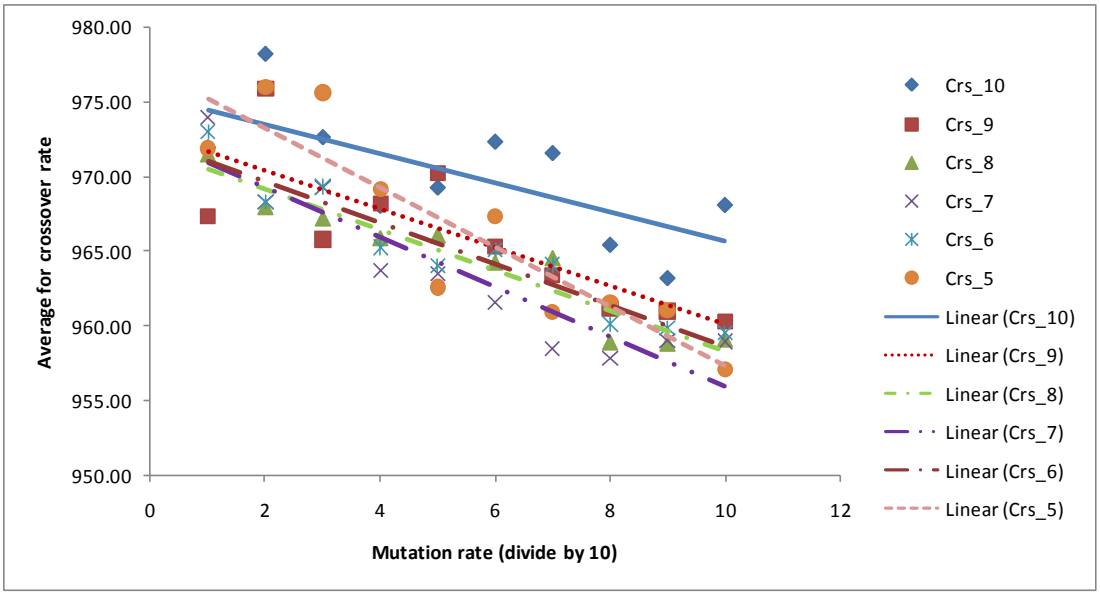


Figure 4.5: Best Fit Line for Crossover with different Mutation Rates

4.5 Results

Table 4.10, Table 4.11, and Table 4.12 summarize the experimental results. Crossover rate and mutation rate are set to 0.7 and 1.0 respectively for all the instances. The first column lists the problem name (instances) and the problem optimal solutions (optimal) or BKS. Second column lists the makespan of the problem which include minimum (best solution), maximum, and average makespan. The last column lists the solutions obtained by different number of parents.

Table 4.10: Results for FT Problem

Instances	Makespan	No. of parents							
		3	4	5	6	7	8	9	10
FT 06	Min. (best)	55	55	55	55	55	55	55	55
Optimal = 55	Max.	55	55	55	55	55	55	55	55
	Average	55	55	55	55	55	55	55	55
FT 10	Min. (best)	930	930	930	930	930	930	930	937
Optimal = 930	Max.	1023	1016	1018	1025	1019	1029	1032	1024
	Average	961.93	965.93	966.61	973.33	969.79	977.18	975.40	978.60
FT 20	Min. (best)	1178	1185	1184	1190	1187	1198	1183	1197
Optimal = 1165	Max.	1272	1252	1266	1259	1262	1283	1294	1286
	Average	1214.59	1216.37	1223.08	1224.9	1224.94	1231.38	1230.88	1232.34

Results from Table 4.10 show that EPPX is considered suitable for solving the JSSP. Instance FT06 can be solved easily with the number of parents used for crossover ranging from 3 to 10 parents. For difficult problems such as FT10, EPPX is able to get the optimum solution with the number of parents of 3 to 9 for the crossover operation. For instance FT20, the smallest deviation from the optimal solution is obtained with the best solution of 1178 (approximately 1%) by using 3 parents for the crossover operation.

Table 4.11: Results for ABZ Problem

Instances	Makespan	Solutions no. of parents							
		3	4	5	6	7	8	9	10
ABZ5	Min. (best)	1238	1234	1238	1238	1238	1238	1238	1238
Optimal = 1234	Max.	1262	1269	1266	1270	1266	1266	1266	1267
	Average	1250.09	1249.27	1251.23	1251.31	1249.5	1249.41	1249.92	1248.88
ABZ6	Min. (best)	947	945	943	945	943	947	946	947
Optimal = 943	Max.	966	967	967	966	967	967	970	970
	Average	948.65	949.24	949.64	949.81	949.14	949.41	950.82	948.83
ABZ7	Min. (best)	680	684	687	684	688	686	693	692
Optimal = 656	Max.	710	711	713	714	713	715	718	712
	Average	696.81	698.39	698.53	700.94	700.68	701.39	702.69	701.93
ABZ8	Min. (best)	701	699	705	699	702	705	705	707
BKS = 665	Max.	727	727	726	728	734	727	733	728
	Average	712.5	713.45	714.83	715.68	716.57	717.28	718.26	718.09
ABZ9	Min. (best)	710	708	713	717	708	721	716	720
BKS= 678	Max.	745	745	745	747	748	749	755	758
	Average	728.75	730.2	730.4	733.34	733.23	733.88	734.4	734.76

Table 4.11 presents the results for ABZ problem which contains five instances with sizes ranging from 10x10 to 20x15. No optimal solutions have been known for ABZ8 and ABZ9, thus the BKS for both instances are obtained from the literatures (Zhang et al., 2008). EPPX is able to get the optimal solution for ABZ5 and ABZ6 but varies between 3 – 5 % from the optimal or the best known results.

Table 4.12: Results for ORB Problem

Instances	Makespan	no. of parents							
		3	4	5	6	7	8	9	10
ORB01	Min. (best)	1077	1077	1087	1077	1086	1070	1086	1089
Optimal = 1059	Max.	1140	1147	1152	1140	1142	1153	1148	1150
	Average	1100.8	1101.03	1098.21	1100.66	1101.37	1102.54	1100.9	1103.32
ORB02	Min. (best)	889	892	889	889	894	892	889	897
Optimal = 888	Max.	934	940	941	942	941	945	941	945
	Average	910.57	912.56	917.16	918.62	920.88	920.90	919.20	922.24
ORB03	Min. (best)	1022	1035	1022	1028	1029	1041	1039	1030
Optimal = 1005	Max.	1114	1121	1138	1134	1156	1156	1146	1174
	Average	1065.21	1071.79	1074.07	1076.22	1081.39	1090.06	1091.36	1092.30
ORB04	Min. (best)	1006	1011	1005	1005	1011	1005	1005	1011
Optimal = 1005	Max.	1052	1062	1054	1060	1060	1062	1056	1062
	Average	1032.32	1034.17	1033.27	1033.36	1030.85	1033.51	1032.43	1031.96
ORB05	Min. (best)	890	890	890	890	891	891	890	890
Optimal = 887	Max.	947	952	943	959	966	959	957	959
	Average	908.93	910.14	910.01	914.56	917.64	918.84	918.31	921.44
ORB06	Min. (best)	1031	1028	1031	1030	1031	1031	1033	1031
Optimal = 1010	Max.	1088	1082	1088	1109	1087	1088	1112	1108
	Average	1055.24	1057.69	1061.23	1064.76	1063.00	1063.44	1065.02	1064.91
ORB07	Min. (best)	397	400	398	397	397	400	399	417
Optimal = 397	Max.	421	422	431	419	422	428	429	421
	Average	408.72	409.54	410.68	406.91	410.19	411.27	410.68	411.24
ORB08	Min. (best)	914	914	899	899	911	899	927	912
Optimal = 899	Max.	983	990	992	1001	1009	1002	1006	1006
	Average	945.73	948.27	951.72	952.84	955.76	958.75	960.66	962.64
ORB09	Min. (best)	934	942	940	943	941	940	939	943
Optimal = 934	Max.	988	996	997	996	997	1007	996	997
	Average	960.29	961.51	963.55	963.48	964.37	963.60	961.48	964.51
ORB10	Min. (best)	944	944	944	944	944	946	944	944
Optimal = 944	Max.	999	993	1004	1005	1005	1004	1004	1012
	Average	959.78	960.28	957.83	960.94	962.10	961.20	962.43	961.96

In the Table 4.12, the results for ORB problem show that EPPX found the optimal solution in 5 instances (ORB04, ORB07-ORB10). The best solutions found are located in the different numbers of parents for different instances thus it is proven that GA is not restricted to two parents crossover in order to find the best solution.

Table 4.13 presents the average computational time in 100 runs for different numbers of parents in each instance.

Table 4.13: Computational Time

		Computational time (in second) no. of parents							
Instances	**Size**	3	4	5	6	7	8	9	10
FT 06	6 x 6	3.24	3.48	2.78	3.72	3.12	3.44	3.52	3.53
FT 10	10x10	192.99	195.63	198.07	194.35	192.10	193.66	203.56	197.60
FT 20	20x 5	290.42	290.61	293.55	316.24	292.71	291.80	276.30	296.74
ORB01	10x10	182.17	186.19	186.76	180.22	185.76	180.31	189.43	186.72
ORB02	10x10	166.47	168.31	167.32	162.40	171.36	165.52	171.48	175.58
ORB03	10x10	191.34	194.01	194.58	187.84	191.01	196.11	204.05	197.58
ORB04	10x10	165.19	173.07	173.17	167.05	171.72	173.59	185.18	183.84
ORB05	10x10	170.83	177.57	176.27	174.45	172.74	171.55	178.28	187.80
ORB06	10x10	189.83	190.79	193.13	184.82	198.95	187.90	192.94	198.92
ORB07	10x10	168.13	171.68	168.20	164.56	169.46	164.22	165.59	171.01
ORB08	10x10	188.48	187.48	188.75	181.74	179.71	179.89	189.04	189.52
ORB09	10x10	165.02	170.29	173.51	165.79	173.23	177.53	174.90	178.03
ORB10	10x10	175.14	178.67	175.92	167.22	173.31	172.76	178.86	179.98
ABZ5	10x10	165.07	165.07	165.99	156.42	161.96	158.92	170.12	171.79
ABZ6	10x10	156.02	156.80	160.49	161.87	162.80	165.04	167.25	173.18
ABZ7	20x15	1507.93	1486.89	1518.80	1542.58	1519.09	1460.59	1468.86	1608.26
ABZ8	20x15	1545.53	1553.63	1536.08	1532.17	1503.69	1472.42	1485.99	1558.98
ABZ9	20x15	1397.51	1368.41	1376.06	1377.28	1353.83	1336.95	1313.06	1393.12

The computational time in all instances varies due to the different structures of the problems. As it uses the heuristic method, computational times for repeated problems also do not have consistent values. The computational time for easy problem FT06 is very short because the optimal solutions can be found quickly and the GA

terminated and stopped running when it reaches the optimal solutions. The average computational time for difficult problem takes a longer time because some of the runs were unable to reach the optimal solution and they are only terminated at the end of the maximum number of generation.

Table 4.14 lists the results of FT problem run with multi-parents crossover and in non-hybrid environment with fixed maximum generation. Table 4.15 lists the results using the complete hybrid GA proposed for the FT problem.

Table 4.14: Before Hybrid

Instances	Optimal	Number of Parents							
		3	4	5	6	7	8	9	10
FT06	55	55	55	55	55	55	55	55	55
FT10	930	953	955	955	950	955	990	976	960
FT20	1165	1204	1208	1211	1206	1228	1236	1254	1250

Table 4.15: After Hybrid

Instances	Optimal	Number of Parents							
		3	4	5	6	7	8	9	10
FT06	55	55	55	55	55	55	55	55	55
FT10	930	930	930	930	930	930	930	930	937
FT20	1165	1178	1185	1184	1190	1187	1198	1183	1197

The proposed hybrid GA shows significant improvement compared to the GA with only multi-parents crossover. The hybrid GA can find the optimal solutions for the problem FT10 and the deviation between optimal solution and best found solution in FT20 is lesser than the GA with multi-parents crossover only. These notable improvements proved that the hybridization of GA with other methods is able to increase the performance of GA.

Table 4.16 presents frequency of the best solutions found in each number of parents. It is observed that the top 4 numbers of parents contributed the most found best solutions are 3, 4, 5 and 6 number of parents. Figure 4.6 depicts the frequency of the number of parents 3, 4, 5 and 6 in achieving the most optimal or near optimal solutions. Thus, they may be considered as the best numbers of parents that may be used for the crossover operations. The number of parents equals 10 appears as the lowest to reach the optimal or near optimal value. The multi-parents crossover that achieved the best solutions has more tendencies to be on the left side of the graph, meaning that if the number of parents increases, the possibilities to find the best solutions will be lower.

Table 4.16: Best Solutions for different No. of Parents

Instances	no. of parents							
	3	4	5	6	7	8	9	10
FT10	1	1	1	1	1	1	1	
FT20	1							
ABZ5		1						
ABZ6			1		1			
ABZ7	1							
ABZ8		1		1				
ABZ9		1			1			
ORB01						1		
ORB02	1		1	1			1	
ORB03	1		1					
ORB04			1	1		1	1	
ORB05	1	1	1	1			1	1
ORB06		1						
ORB07	1			1	1			
ORB08			1	1		1		
ORB09	1							
ORB10	1	1	1	1	1		1	1
Total	9	7	8	8	5	4	5	2

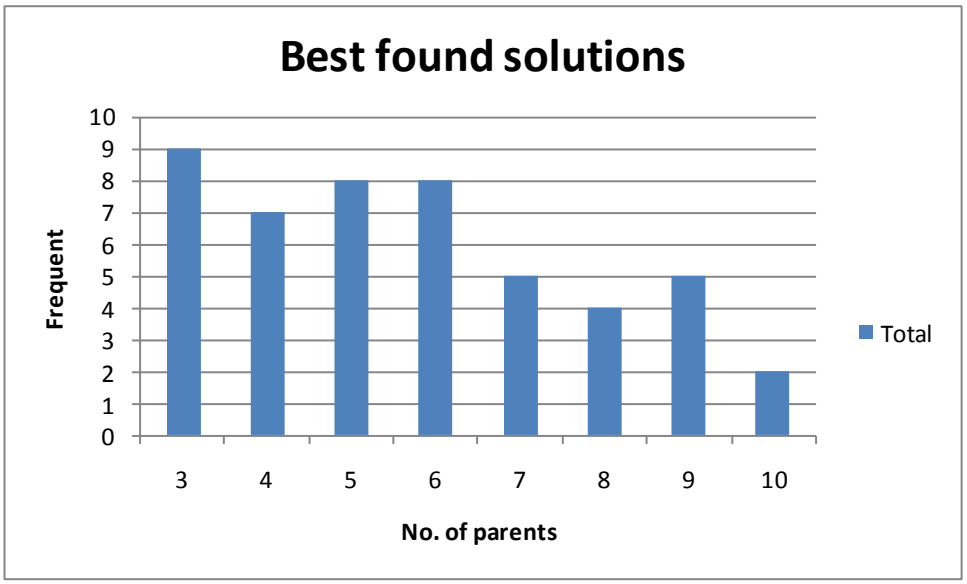


Figure 4.6: Bar Chart for Best Solutions for different No. of Parents

4.6 Comparison with Others that are based on Permutation Crossover Operator

The comparison in Table 4.17 is made with other GA algorithms which adopt the concept of permutation crossover operator. The table contains the authors, year, crossover operator, and best solutions for the instances for comparison. Previously, the authors (Gen et al., 1994; Bierwirth, 1995) used only two parents for the crossover operation and the optimal result achieved for the FT10 and the deviation from the optimal solution for FT20 are less than 2%. The acceptable ranges of comparable GAs are up until 7% (Bierwirth, 1995). Therefore, the deviation ranges more than these values are considered not effective. Tested results reflect that the EPPX is able to obtain the solution within these values and considered applicable for solving the JSSP. The instance FT06 can be solved easily with the number of parents used for crossover ranging from 3 to 10 parents. For difficult problems such as the FT10, the EPPX also performs well as it is able to obtain the optimum solution with the number of parents

ranging from 3 to 9 in crossover operation. Meanwhile, when the variable of JSSP increases (instance FT20), deviation occurs between the optimal solution and EPPX solutions with the best solution 1178 found by using 3 parents in the EPPX's crossover operation.

Table 4.17: Comparison for FT06, FT10, and FT20 with *n* Jobs x *m* Machines

Author(s)	Year	Crossover operator	FT06 (6x6)	FT10 (10x10)	FT20 (20x5)
Optimum			55	930	1165
Gen et al.	1994	Partial schedule exchange crossover	55	962	1175
Bierwirth	1995	Generalized Permutation GP-GA	55	936	1181
Park et al.	2003	Parallel Genetic Algorithm PGA	55	930	1173
Ripon et al.	2010	Improved Precedence Preservation Crossover IPPX	55	930	1180
	2012	Multi-Parents Crossover EPPX	55	930	1178

4.7 Comparison with Results from the Literatures

The ABZ contains five instances with two different sizes of problems: 10x10 and 20x15. In this problem, we compare our tested results with different JSSP strategies such as: hybridization of TS and SA (TSSA) (Zhang et al., 2008), parallel genetic algorithms (PGA) (Park et al., 2003), and greedy randomized adaptive search procedure (GRASP) (Binato et al., 2001). Table 4.18 and Table 4.19 list for each test instance, its name, size (number of jobs x number of machines), the best known solutions (BKS), the best solutions found (Best), multi-parents that obtained the best solutions (MP), and relative error (RE). The RE is calculated from the gap between Best and BKS in percentage. Total RE in the last column shows the total relative error which is used to analyze the effectiveness of the proposed algorithm.

Results in Table 4.18 show the comparison between EPPX with three other methods. The EPPX performs only averagely if compared among the methods listed in the 10x10 problem size. When the sizes of the problems are increased to 20x15, the EPPX performs better compared to the PGA and GRASP with less relative errors with the overall best algorithm is the hybrid tabu search and simulated annealing of Zhang et al. (2008). These significant results indicated that the EPPX is capable to adapt bigger size problems with comparable relative errors ranging from 0.3% to 7.0%. The best solutions found are located in the different numbers of parents for different instances.

Table 4.18: Comparison for ABZ Problem

Instances	Size	BKS	EPPX			TSSA		PGA		GRASP	
			Best	RE	MP	Best	RE	Best	RE	Best	RE
ABZ5	10x10	1234	1234	0.00	4	1234	0.00	1236	0.16	1238	0.32
ABZ6	10x10	943	943	0.00	5, 7	943	0.00	943	0.00	947	0.42
ABZ7	20x15	656	680	3.66	3	658	0.30	685	4.42	723	10.21
ABZ8	20x15	665	699	5.11	4,6	667	0.30	704	5.86	729	9.62
ABZ9	20x15	678	708	4.42	4,7	678	0.00	723	6.64	758	11.80
		Total RE		13.19			0.60		17.08		32.37

Table 4.19: Comparison for ORB Problem

Instances	Size	BKS	EPPX			TSSA		PGA		GRASP	
			Best	RE	MP	Best	RE	Best	RE	Best	RE
ORB01	10x10	1059	1070	1.04	8	1059	0.00	1060	0.09	1070	1.04
ORB02	10x10	888	889	0.11	3,5,6,9	888	0.00	889	0.11	889	0.11
ORB03	10x10	1005	1022	1.69	3,5	1005	0.00	1020	1.49	1021	1.59
ORB04	10x10	1005	1005	0.00	5,6,8,9	1005	0.00	1005	0.00	1031	2.59
ORB05	10x10	887	890	0.34	3-6,9,10	887	0.00	889	0.23	891	0.45
ORB06	10x10	1010	1028	1.78	4	1010	0.00	1013	0.30	1013	0.30
ORB07	10x10	397	397	0.00	3,6,7	397	0.00	397	0.00	397	0.00
ORB08	10x10	899	899	0.00	5,6,8	899	0.00	899	0.00	909	1.11
ORB09	10x10	934	934	0.00	3	934	0.00	934	0.00	945	1.18
ORB10	10x10	944	944	0.00	3-7,9,10	944	0.00	944	0.00	953	0.95
		Total RE		4.96			0.00		2.22		9.32

Table 4.19 lists the comparison for different methods. The hybrid GA (EPPX), which performs better than the GRASP, achieves optimal solutions for five problems (ORB04, ORB07, ORB08, ORB09, and ORB10). The EPPX and PGA have the same unsolved instances (ORB01, ORB02, ORB03, ORB05, and ORB06) but the total relative error (Total RE) of the PGA is slightly better compared to the EPPX. Overall, the performance of EPPX is comparable to all the three methods with the hybrid tabu search and simulated annealing of Zhang et al. (2008) performs the best overall.

Consequently, as can be seen in the problem size 10x10 (ABZ and ORB), both EPPX and PGA which propose a hybrid GA encounter difficulty in solving the problems compared to the TSSA which uses the hybridization of tabu search and simulated annealing. It is evident from Table 4.13 that when the problem sizes become larger and harder, the computational time takes more time to search for the solutions and this is especially true for problem ABZ7, ABZ8 and ABZ9.

4.8 Conclusion

In the experimental results, EPPX using multi-parents is able to get the solutions within the acceptable range of GA values. Results show that the best solutions are obtained from different numbers of parents for crossover thus it is proven that GA able to use more than two parents crossover in order to find the best solution. The number of parents used in EPPX and GA is very much dependent on the problem instances and it may be observed by the best solutions for different instances were produced by different numbers of parents. Although EPPX outperforms some the proposed algorithms in the literatures for relatively larger problems but it still cannot achieve the best solution found especially on the open problems.

CHAPTER 5

CONCLUSION

5.1 Conclusion

In Chapter 2, the job shop scheduling is described and formulated. The background of the job shop scheduling problem is explained in great detail and relevant literatures are presented. The requirements of this combinatorial scheduling problem and their constraints and assumption were converted into mathematical model. Related metaheuristics that are designed for Job Shop Scheduling Problem (JSSP) are reviewed and these algorithms are specially designed to tackle problems that are unable to be solved by the exact method. These heuristic methods are explored in search space with the iterative function to find the potential solutions.

The metaheuristics do not guarantee that the optimal solution can be found. Therefore, additional search methods are embedded or hybridize with metaheuristics to increase the accuracy in solving the problem. In the literature, these hybrid metaheuristics searching methods are classified as intensification and diversification (Zäpfel et al., 2010). The intensification mechanisms tend to find a good solution in a potential solution. Diversification will diversify the solution to escape from the entrapment of the local optima.

One of the hybrid metaheuristics, hybrids GA, also applies both mechanisms in the searching procedure. In most cases, the GA acts as a diversification mechanism that provides diversified potential solutions whereas the local search embedded into the GA

operates as the intensification of the searching and exploiting the potential solutions in search for better solutions.

In Chapter 3, our proposed hybrid GA is built on these approaches. The GA has a limitation in searching solutions because it faces the problems of premature convergence and large search space. Thus, the proposed iterative forward-backward pass and neighborhood search are used to overcome these problems. The proposed methods that build the hybrid GA are divided into three, the multi-parents crossover, neighborhood search on critical path and iterative forward-back pass. The multi-parents crossover proposed requires more than three parents to perform the recombination, the EPPX instead of using two parents for recombination. The neighborhood search in this hybrid GA acts as an intensification mechanism that attempts to search for the best solution by exploiting the provided current solutions. In the problem of the large search space, the search space is reduced by improving the quality of the chromosomes.

In Chapter 4, the algorithms are performed on selected benchmarks problems. Rigorous tests are carried out to determine suitable parameters for the algorithms and the initial parameters are set based on the literatures. The crossover rate and mutation parameter are acquired from the test on the FT 10 problem and they are applied to all instances. The maximum numbers of generations are adjusted to ensure that the comparable results are fair. The results from the simulations are compared with the different methods in the literatures to measure the capabilities of the algorithm. Through observation, the performance of this hybrid GA is comparable to other methods especially in its ability to become prominent when adapted to bigger size problems. This hybrid GA can still be improved to obtain better solutions.

The objectives in this study are met. We proposed a new crossover operator EPPX and it is able to perform well if compared to the other crossover operators that use two parents crossover in the GA. The search spaces are reduced by applying the scheduling method from different areas of scheduling into the JSSP and it shows that there are improvements in searching for solutions. The neighborhood search shows that searching in the critical path reduces the makespan. This is because the critical path is determined by the length of the whole schedule so the search may concentrate on the changes of critical path for better solutions.

Consequently, these three methods combined together in a hybrid GA are able to increase the efficiency of the GA performance. The GA efficiency increases when embedded with the local search and iterative forward-backward schedule. Thus, we may conclude that the local search plays an important role in the GA to achieve the best solutions.

5.2 Future Works

In future works, the hybrid GA needs to explore the combination with other local searches. As shown in the literatures of our research, the local search embedded into the GA perform better compared to the GA that does not combine with other methods.

The proposed future works that need to be done are:

- Adding more efficient local searches into the GA and capitalize the local search by using GA in the searching solution.

- Design a better algorithm to reduce the search space. The quality of the chromosome being increased through reduction of the search space. With better quality of chromosomes, the GA is able to produce better generation that contains the optimal or near optimal solutions.

- The multi-parents crossover needs to be studied more to test its ability in the GA. In the JSSP, the multi-parents crossover variety needs to increase to allow more investigation in exploring the multi-parents crossover effects in the GA.

- Hybridizing with other metaheuristics such as tabu search or simulated annealing and both, may result in a better algorithm. The GA may be used to generate some initial solutions and tabu search or simulated annealing (or both) may be used to intensify the solution.

- The GA takes a longer time to find the best results compared to other methods. This is because it uses the iterative method where time will increase when being hybrid with other local searches. The calculation requires a longer time due to the complexity of the search algorithm. Good calculation methods should comprise of lower computational time and better results. In order to achieve these, the methods need to be improved by creating efficient methods.

REFERENCES

Abdelmaguid, T. F. (2010). Representations in genetic algorithm for the job shop scheduling problem: A computational study. *Journal of Software Engineering and Applications*, *3*(12), 1155-1162.

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management science*, *34*(3), 391-401.

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, *3*(2), 149-156.

Balas, E. (1969). Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations research*, *17*(6), 941-957.

Baker, K.R. (1974). *Introduction to sequencing and scheduling.* Wiley, New York.

Baker, J.E. (1987). Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette, J.J. (Eds.), *Proceedings of the 2nd International Conference on Genetic Algorithms* (pp. 14-21). Hillsdale, New Jersey: L. Erlbaum Associates.

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 1069-1072.

Beligiannis, G. N., Tsirogiannis, G. A., & Pintelas, P. E. (2005). Restartings: A technique to improve classic genetic algorithms' performance. *World Academy of Science, Engineering and Technology*, *1*, 1307-6884.

Bierwirth, C. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spectrum*, *17*(2), 87-92.

Bierwirth, C., & Mattfeld, D. C. (1999). Production scheduling and rescheduling with genetic algorithms. *Evolutionary computation*, *7*(1), 1-17.

Bierwirth, C., Mattfeld, D.C., & Kopfer, H. (1996). On permutation representations for scheduling problems. *Parallel Problem Solving from Nature—PPSN IV*, 310-318.

Binato, S., Hery, W. J., Loewenstern, D. M., & Resende, M. G. C. (2001). A GRASP for job shop scheduling. *Essays and surveys in metaheuristics*, 59-79.

Blickle, T., & Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, *4*(4), 361-394.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, *35*(3), 268-308.

Cai, B., Wang, S., & Hu, H. (2011). An effective hybrid genetic algorithm for job shop scheduling problem. *World Academy of Science, Engineering and Technology*, *58*, 42-48.

Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèmes répartis*, *10*(2), 141-170.

Carlier, J., & Chretienne, P. (1988). *Proble`mes d'ordonnancement*, *col*. ERI. Paris: Masson.

Černý, V. (1985). Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*(1), 41–51.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. *Computers & Industrial Engineering*, *30*(4), 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*(2), 343-364.

Davis, L. (1985). Job shop scheduling with genetic algorithms. *Proceedings of the 1st International Conference on Genetic Algorithms*, 136-140.

Dell'Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, *41*(3), 231-252.

Demirkol, E., Mehta, S., & Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research*, *109*(1), 137-141.

Eiben, A. E. (2003). Multiparent recombination in evolutionary computing. *Advances in evolutionary computing*, 175-192.

Eiben, A. E., & Van Kemenade, C. H. (1997). Diagonal crossover in genetic algorithms for numerical optimization. *Control and Cybernetics*, *26*, 447-466.

Eiben, A., Raué, P., & Ruttkay, Z. (1994). Genetic algorithms with multi-parent recombination. *Parallel Problem Solving from Nature—PPSN III*, 78-87.

Fisher, H., & Thompson, G.L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In J.F. Muth, G.L. Thompson (Eds.), *Industrial Scheduling* (pp. 225-251). Englewood Cliffs, New Jersey: Prentice Hall.

Gen, M., & Cheng, R., (1997). *Genetic algorithms and engineering design*. New York: John Wiley Sons.

Gen, M., Tsujimura, Y., & Kubota, E. (1994, October). Solving job-shop scheduling problems by genetic algorithm. In *Systems, Man, and Cybernetics, 1994.'Humans, Information and Technology'., 1994 IEEE International Conference on* (Vol. 2, pp. 1577-1582). IEEE.

Gen, M., Cheng, R., & Lin, L. (2008). *Network models and optimization: Multiobjective genetic algorithm approach*. Springer.

Giffler, B., & Thompson G.L. (1960). Algorithms for solving production-scheduling problems. *Operations Research, 8*(4), 487-503.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13*(5), 533-549.

Glover, F. (1998). A template for scatter search and path relinking. In *Artificial evolution* (pp. 1-51). Springer Berlin/Heidelberg.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization and machine learning*. Boston: Addison Wesley Longman.

Goldberg, D., & Lingle, R.A. (1985). Logic and the traveling salesman problem. *Proceeding of the 1st Internatinal Conference on GA*, 154–159.

Goldberg, D.E., Deb, K., & Clark J.H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems, 6*, 333–362.

Gonçalves, J.F, Mendes,J.J.D., & Resende, M.G.C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research, 167*(1), 77-95.

Graves, S.C. (1981). A review of production scheduling. *Operations Research, 29*(4), Operations Management, 646-675.

Hasan, S.M.K., Sarker, R., & Cornforth, D. (2007). Hybrid genetic algorithm for solving job-shop scheduling problem. *Computer and Information Science, 2007. ICIS 2007, 6th IEEE/ACIS International Conference* (pp. 519-524).

Jain, A.S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research, 113*, 390-434.

Jones, A., Rabelo, L.C., & Sharawi, A.T. (1999). Survey of job shop scheduling techniques. *Wiley Encyclopedia of Electrical and Electronics Engineering*.

Käschel, J., Teich, T., Köbernik, G., & Meier, B. (1999, June). Algorithms for the job shop scheduling problem: A comparison of different methods. In *European Symposium on Intelligent Techniques. Greece, June* (pp. 3-4).

Kelley Jr, J. E., & Walker, M. R. (1959, December). Critical-path planning and scheduling. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference* (pp. 160-173). ACM.

Kirkpatrick, S., & Vecchi, M. P. (1983). Optimization by simmulated annealing. *Science, 220*(4598), 671-680.

Kumar, S.A., & Suresh, N. (2009). *Operations Management*. New Delhi: New Age International.

Laarhoven, P. J.V., Aarts, E. H., & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations research, 40*(1), 113-125.

Langdon, W.B., McKay, R.I., & Spector, L. (2010). Genetic programming. In Gendreau, M., & Potvin, J. Y. (Eds.), *Handbook of metaheuristics* (pp. 185-225). New York, NY: Springer-Verlag.

Lawrence, S. (1984). *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques*. (Supplement). School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Lourenço, H., Martin, O., & Stützle, T. (2003). Iterated local search. *Handbook of metaheuristics*, 320-353.

Lova, A., Maroto, C., & Tormos, P. (2000). A multicriteria heuristic method to improve resource allocation in multiproject scheduling. *European Journal of Operational Research*, *127*(2), 408-424.

Mellor, P. (1966). A review of job shop scheduling. *OR*, 161-171

Miller, B. L., & Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113-131.

Mühlenbein, H., & Voigt, H. M. (1995). Gene pool recombination in genetic algorithms. In *Proc. of the Metaheuristics Conference*. Kluwer Academic Publishers, Norwell, USA.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, *42*(6), 797-813.

Ochoa, G., Harvey, I., & Buxton, H. (1999, July). On recombination and optimal mutation rates. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Vol. 1, pp. 488-495).

Ólafsson, S. (2006). Metaheuristics. *Handbooks in operations research and management science*, *13*, 633-654.

Osman, I. H., & Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, *63*(5), 511-623.

Park, B. J., Choi, H. R., & Kim, H. S. (2003). A hybrid genetic algorithm for the job shop scheduling problems. *Computers & industrial engineering*, *45*(4), 597-613.

Porter, D. B. (1968). The Gantt chart as applied to production scheduling and control. *Naval Research Logistics Quarterly, 15*,311–317.

Reeves, C. (2003). Genetic algorithms. *Handbook of metaheuristics*, 55-82.

Ripon, K. S. N., Siddique, N. H., & Torresen, J. (2011). Improved precedence preservation crossover for multi-objective job shop scheduling problem. *Evolving Systems*, *2*(2), 119-129.

Sels, V., Craeymeersch, K., & Vanhoucke, M. (2011). A hybrid single and dual population search procedure for the job shop scheduling problem. *European Journal of Operational Research*, *215*(3), 512-523.

Sivanandam, S. N., & Deepa, S. N. (2007). *Introduction to genetic algorithms*. Springer Publishing Company, Incorporated.

Song, Y., Hughes, J. G., Azarmi, N., & Voudouris, C. (2000). A Genetic Algorithm with an incomplete representation for the Job Shop Scheduling Problems. *University of Ulster at Jordanstown*.

Sprecher, A., Kolisch, R., & Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *80*(1), 94-102.

Steinhöfel, K., Albrecht, A., & Wong, C. K. (1999). Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research*, *118*(3), 524-548.

Storer, R. H., Wu, S. D., & Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management science*, *38*(10), 1495-1509.

Taha, H. A. (2011). *Operations research: An introduction*. Upper Saddle River, NJ: Prentice hall.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278-285.

Ting, C. K., Su, C. H., & Lee, C. N. (2010). Multi-parent extension of partially mapped crossover for combinatorial optimization problems. *Expert Systems with Applications*, *37*(3), 1879-1886.

Tsutsui, S., & Jain, L.C. (1998, April). On the effect of multi-parents recombination in binary coded genetic algorithms. *In Knowledge-Based Intelligent Electronic Systems, 1998. Proceedings KES '98. 1998 Second International Conference on* (Vol.3, pp. 155 – 160). IEEE.

Tsutsui, S., & Ghosh, A. (1998, May). A study on the effect of multi-parent recombination in real coded genetic algorithms. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on* (pp. 828-833). IEEE.

Tsutsui, S., Yamamura, M., & Higuchi, T. (1999, July). Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the genetic and evolutionary computation conference* (Vol. 1, pp. 657-664).

Wang, L., & Zheng, D. Z. (2001). An effective hybrid optimization strategy for job-shop scheduling problems. *Computers & Operations Research*, *28*(6), 585-596.

Wang, L., & Zheng, D. Z. (2002). A modified genetic algorithm for job shop scheduling. *The International Journal of Advanced Manufacturing Technology*, *20*(1), 72-76.

Watanabe, M., Ida, K., & Gen, M. (2005). A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem. *Computers & Industrial Engineering*, *48*(4), 743-752.

Wu, A., Tsang, P. W. M., Yuen, T. Y. F., & Yeung, L. F. (2009). Affine invariant object shape matching using genetic algorithm with multi-parent orthogonal recombination and migrant principle. *Applied Soft Computing*, *9*(1), 282-289.

Yamada, T., & Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop problems. *Parallel problem solving from nature*, *2*, 281-290.

Yamada, T., & Nakano, R. (1997, March). Genetic algorithms for job-shop scheduling problems. In *Proceedings of the Modern Heuristics for Decision Support* (pp. 67-81).

Yusof, R., Khalid, M., Hui, G. T., Md Yusof, S., & Othman, M. F. (2011). Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm. *Applied soft computing*, *11*(8), 5782-5792.

Zäpfel, G., Braune, R., & Bögl, M. (2010). *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*. Springer.

Zhang, C., Rao, Y., & Li, P. (2008). An effective hybrid genetic algorithm for the job shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 39(9), 965-974.

Zhang, C. Y., Li, P., Rao, Y., & Guan, Z. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, *35*(1), 282-294.

APPENDIX A

Instances for the Problems

FT Problems

```
instance ft06

 ++++++++++++++++++++++++++++
 Fisher and Thompson 6x6 instance, alternate name (mt06)
 6 6
 2  1  0  3  1  6  3  7  5  3  4  6
 1  8  2  5  4 10  5 10  0 10  3  4
 2  5  3  4  5  8  0  9  1  1  4  7
 1  5  0  5  2  5  3  3  4  8  5  9
 2  9  1  3  4  5  5  4  0  3  3  1
 1  3  3  3  5  9  0 10  4  4  2  1
 ++++++++++++++++++++++++++++

 instance ft10

 ++++++++++++++++++++++++++++
 Fisher and Thompson 10x10 instance, alternate name (mt10)
 10 10
 0 29 1 78 2  9 3 36 4 49 5 11 6 62 7 56 8 44 9 21
 0 43 2 90 4 75 9 11 3 69 1 28 6 46 5 46 7 72 8 30
 1 91 0 85 3 39 2 74 8 90 5 10 7 12 6 89 9 45 4 33
 1 81 2 95 0 71 4 99 6  9 8 52 7 85 3 98 9 22 5 43
 2 14 0  6 1 22 5 61 3 26 4 69 8 21 7 49 9 72 6 53
 2 84 1  2 5 52 3 95 8 48 9 72 0 47 6 65 4  6 7 25
 1 46 0 37 3 61 2 13 6 32 5 21 9 32 8 89 7 30 4 55
 2 31 0 86 1 46 5 74 4 32 6 88 8 19 9 48 7 36 3 79
 0 76 1 69 3 76 5 51 2 85 9 11 6 40 7 89 4 26 8 74
 1 85 0 13 2 61 6  7 8 64 9 76 5 47 3 52 4 90 7 45
 ++++++++++++++++++++++++++++

 instance ft20

 ++++++++++++++++++++++++++++
 Fisher and Thompson 20x5 instance, alternate name (mt20)
 20 5
 0 29 1  9 2 49 3 62 4 44
 0 43 1 75 3 69 2 46 4 72
 1 91 0 39 2 90 4 12 3 45
 1 81 0 71 4  9 2 85 3 22
 2 14 1 22 0 26 3 21 4 72
 2 84 1 52 4 48 0 47 3  6
 1 46 0 61 2 32 3 32 4 30
 2 31 1 46 0 32 3 19 4 36
 0 76 3 76 2 85 1 40 4 26
 1 85 2 61 0 64 3 47 4 90
 1 78 3 36 0 11 4 56 2 21
 2 90 0 11 1 28 3 46 4 30
 0 85 2 74 1 10 3 89 4 33
 2 95 0 99 1 52 3 98 4 43
 0  6 1 61 4 69 2 49 3 53
 1  2 0 95 3 72 4 65 2 25
 0 37 2 13 1 21 3 89 4 55
 0 86 1 74 4 88 2 48 3 79
 1 69 2 51 0 11 3 89 4 74
 0 13 1  7 2 76 3 52 4 45
 ++++++++++++++++++++++++++++
```

ABZ Problems

```
 instance abz5

 ++++++++++++++++++++++++++++
 Adams, Balas, and Zawack 10x10 instance (Table 1, instance 5)
 10 10
 4 88 8 68 6 94 5 99 1 67 2 89 9 77 7 99 0 86 3 92
 5 72 3 50 6 69 4 75 2 94 8 66 0 92 1 82 7 94 9 63
 9 83 8 61 0 83 1 65 6 64 5 85 7 78 4 85 2 55 3 77
```

```
7 94 2 68 1 61 4 99 3 54 6 75 5 66 0 76 9 63 8 67
3 69 4 88 9 82 8 95 0 99 2 67 6 95 5 68 7 67 1 86
1 99 4 81 5 64 6 66 8 80 2 80 7 69 9 62 3 79 0 88
7 50 1 86 4 97 3 96 0 95 8 97 2 66 5 99 6 52 9 71
4 98 6 73 3 82 2 51 1 71 5 94 7 85 0 62 8 95 9 79
0 94 6 71 3 81 7 85 1 66 2 90 4 76 5 58 8 93 9 97
3 50 0 59 1 82 8 67 7 56 9 96 6 58 4 81 5 59 2 96
+++++++++++++++++++++++++++++
```

instance abz6

```
+++++++++++++++++++++++++++++
Adams, and Zawack 10x10 instance (Table 1, instance 6)
10 10
7 62 8 24 5 25 3 84 4 47 6 38 2 82 0 93 9 24 1 66
5 47 2 97 8 92 9 22 1 93 4 29 7 56 3 80 0 78 6 67
1 45 7 46 6 22 2 26 9 38 0 69 4 40 3 33 8 75 5 96
4 85 8 76 5 68 9 88 3 36 6 75 2 56 1 35 0 77 7 85
8 60 9 20 7 25 3 63 4 81 0 52 1 30 5 98 6 54 2 86
3 87 9 73 5 51 2 95 4 65 1 86 6 22 8 58 0 80 7 65
5 81 2 53 7 57 6 71 9 81 0 43 4 26 8 54 3 58 1 69
4 20 6 86 5 21 8 79 9 62 2 34 0 27 1 81 7 30 3 46
9 68 6 66 5 98 8 86 7 66 0 56 3 82 1 95 4 47 2 78
0 30 3 50 7 34 2 58 1 77 5 34 8 84 4 40 9 46 6 44
+++++++++++++++++++++++++++++
```

instance abz7

```
+++++++++++++++++++++++++++++
Adams, Balas, and Zawack 15 x 20 instance (Table 1, instance 7)
20 15
 2 24  3 12  9 17  4 27  0 21  6 25  8 27  7 26  1 30  5 31 11 18 14 16 13 39 10 19 12 26
 6 30  3 15 12 20 11 19  1 24 13 15 10 28  2 36  5 26  7 15  0 11  8 23 14 20  9 26  4 28
 6 35  0 22 13 23  7 32  2 20  3 12 12 19 10 23  9 17  1 14  5 16 11 29  8 16  4 22 14 22
 9 20  6 29  1 19  7 14 12 33  4 30  0 32  5 21 11 29 10 24 14 25  2 29  3 13  8 20 13 18
11 23 13 20  1 28  6 32  7 16  5 18  8 24  9 23  3 24 10 34  2 24  0 24 14 28 12 15  4 18
 8 24 11 19 14 21  1 33  7 34  6 35  5 40 10 36  3 23  2 26  4 15  9 28 13 38 12 13  0 25
13 27  3 30  6 21  8 19 12 12  4 27  2 39  9 13 14 12  5 36 10 21 11 17  1 29  0 17  7 33
 5 27  4 19  6 29  9 20  3 21 10 40  8 14 14 39 13 39  2 27  1 36 12 12 11 37  7 22  0 13
13 32 11 29  8 24  3 27  5 40  4 21  9 26  0 27 14 27  6 16  2 21 10 13  7 28 12 28  1 32
12 35  1 11  5 39 14 18  7 23  0 34  3 24 13 11  8 30 11 31  4 15 10 15  2 28  9 26  6 33
10 28  5 37 12 29  1 31  7 25  8 13 14 14  4 20  3 27  9 25 13 31 11 14  6 25  2 39  0 36
 0 22 11 25  5 28 13 35  4 31  8 21  9 20 14 19  2 29  7 32 10 18  1 18  3 11 12 17  6 15
12 39  5 32  2 36  8 14  3 28 13 37  0 38  6 20  7 19 11 12 14 22  1 36  4 15  9 32 10 16
 8 28  1 29 14 40 12 23  4 34  5 33  6 27 10 17  0 20  7 28 11 21  2 21 13 20  9 33  3 27
 9 21 14 34  3 30 12 38  0 11 11 16  2 14  5 14  1 34  8 33  4 23 13 40 10 12  6 23  7 27
 9 13 14 40  7 36  4 17  0 13  5 33  8 25 13 24 10 23  3 36  2 29  1 18 11 13  6 33 12 13
 3 25  5 15  2 28 12 40  7 39  1 31  8 35  6 31 11 36  4 12 10 33 14 19  9 16 13 27  0 21
12 22 10 14  0 12  2 20  5 12  1 18 11 17  8 39 14 31  3 31  7 32  9 20 13 29  4 13  6 26
 5 18 10 30  7 38 14 22 13 15 11 20  9 16  3 17  1 12  2 13 12 40  6 17  8 30  4 38  0 13
 9 31  8 39 12 27  1 14  5 33  3 31 11 22 13 36  0 16  7 11 14 14  4 29  6 28  2 22 10 17
+++++++++++++++++++++++++++++
```

instance abz8

```
+++++++++++++++++++++++++++++
Adams, Balas, and Zawack 15 x 20 instance (Table 1, instance 8)
20 15
 0 19  9 33  2 32 13 18 10 39  8 34  6 25  4 36 11 40 12 33  1 31 14 30  3 34  5 26  7 13
 9 11 10 22 14 19  5 12  4 25  6 38  0 29  7 39 13 19 11 22  1 23  3 20  2 40 12 19  8 26
 3 25  8 17 11 24 13 40 10 32 14 16  5 39  9 19  0 24  1 39  4 17  2 35  7 38  6 20 12 31
14 22  3 36  2 34 12 17  4 30 13 12  1 13  6 25  9 12  7 18 10 31  0 39  5 40  8 26 11 37
12 32 14 15  1 35  7 13  8 32 11 23  6 22  4 21  0 38  2 38  3 40 10 31  5 11 13 37  9 16
10 23 12 38  8 11 14 27  9 11  6 25  5 14  4 12  2 27 11 26  7 29  3 28 13 21  0 20  1 30
 6 39  8 38  0 15 12 27 10 22  9 27  2 32  4 40  3 12 13 20 14 21 11 22  5 17  7 38  1 27
11 11 13 24 10 38  8 15  9 19 14 13  5 30  0 26  2 29  6 33 12 21  1 15  3 21  4 28  7 33
 8 20  6 17  5 26  3 34  9 23  0 16  2 18  4 35 12 24 10 16 11 26  7 12 14 13 13 27  1 19
 1 18  7 37 14 27  9 40  5 40  6 17  8 22  3 17 10 30  0 38  4 21 12 32 11 24 13 24  2 30
11 19  0 22 13 36  6 18  5 22  3 17 14 35 10 34  7 23  8 19  2 29  1 22 12 17  4 33  9 39
 6 32  3 22 12 24  5 13  4 13  1 11  0 11 13 25  8 13  2 15 10 33 11 17 14 16  9 38  7 24
14 16 13 16  1 37  8 25  2 26  3 11  9 34  4 14  0 20  6 36 12 12  5 29 10 25  7 32 11 12
 8 20 10 24 11 27  9 38  5 34 12 39  7 33  4 37  2 31 13 15 14 34  3 33  6 26  1 36  0 14
 8 31  0 17  9 13  1 21 10 17  7 19 13 14  3 40  5 32 11 25  2 34 14 23  6 13 12 40  4 26
 8 38 12 17  3 14 13 17  4 12  1 35  6 35  0 19 10 36  7 19  9 29  2 31  5 26 11 35 14 37
14 20  3 16  0 33 10 14  5 27  7 31  8 16  6 31 12 28  9 37  4 37  2 29 11 38  1 30 13 36
11 18  3 37 14 16  6 15  8 14 12 11 13 32  5 12  1 11 10 29  7 19  4 12  9 18  2 26  0 39
11 11  2 11 12 22  9 35 14 20  7 31  4 19  3 39  5 28  6 33 10 34  1 38  0 20 13 17  8 28
 2 12 12 25  5 23  8 21  6 27  9 30 14 23 11 39  3 26 13 34  7 17  1 24  4 12  0 19 10 36
+++++++++++++++++++++++++++++
```

instance abz9

```
+++++++++++++++++++++++++++++
Adams, Balas, and Zawack 15 x 20 instance (Table 1, instance 9)
20 15
 6 14  5 21  8 13  4 11  1 11 14 35 13 20 11 17 10 18 12 11  2 23  3 13  0 15  7 11  9 35
```

```
 1 35  5 31  0 13  3 26  6 14  9 17  7 38 12 20 10 19 13 12  8 16  4 34 11 15 14 12  2 14
 0 30  4 35  2 40 10 35  6 30 14 23  8 29 13 37  7 38  3 40  9 26 12 11  1 40 11 36  5 17
 7 40  5 18  4 12  8 23  0 23  9 14 13 16 12 14 10 23  3 12  6 16 14 32  1 40 11 25  2 29
 2 35  3 15 12 31 11 28  6 32  4 30 10 27  7 29  0 38 13 11  1 23 14 17  5 27  9 37  8 29
 5 33  3 33  6 19 12 40 10 19  0 33 13 26  2 31 11 28  7 36  4 38  1 21 14 25  9 40  8 35
13 25  0 32 11 33 12 18  4 32  6 28  5 15  3 35  9 14  2 34  7 23 10 32  1 17 14 26  8 19
 2 16 12 33  9 34 11 30 13 40  8 12 14 26  5 26  6 15  3 21  1 40  4 32  0 14  7 30 10 35
 2 17 10 16 14 20  6 24  8 26  3 36 12 22  0 14 13 11  9 20  7 23  1 29 11 23  4 15  5 40
 4 27  9 37  3 40 11 14 13 25  7 30  0 34  2 11  5 15 12 32  1 36 10 12 14 28  8 31  6 23
13 25  0 22  3 27  8 14  5 25  6 20 14 18  7 14  1 19  2 17  4 27  9 22 12 22 11 27 10 21
14 34 10 15  0 22  3 29 13 34  6 40  7 17  2 32 12 20  5 39  4 31 11 16  1 37  8 33  9 13
 6 12 12 27  4 17  2 24  8 11  5 19 14 11  3 17  9 25  1 11 11 31 13 33  7 31 10 12  0 22
 5 22 14 15  0 16  8 32  7 20  4 22  9 11 13 19  1 30 12 33  6 29 11 18  3 34 10 32  2 18
 5 27  3 26 10 28  6 37  4 18 12 12 11 11 13 26  7 27  9 40 14 19  1 24  2 18  0 12  8 34
 8 15  5 28  9 25  6 32  1 13  7 38 11 11  2 34  4 25  0 20 10 32  3 23 12 14 14 16 13 20
 1 15  4 13  8 37  3 14 10 22  5 24 12 26  7 22  9 34 14 22 11 19 13 32  0 29  2 13  6 35
 7 36  5 33 13 28  9 20 10 30  4 33 14 29  0 34  3 22 11 12  6 30  8 12  1 35  2 13 12 35
14 26 11 31  5 35  2 38 13 19 10 35  4 27  8 29  3 39  9 13  6 14  7 26  0 17  1 22 12 15
 1 36  7 34 11 33  8 17 14 38  6 39  5 16  3 27 13 29  2 16  0 16  4 19  9 40 12 35 10 39
+++++++++++++++++++++++++++++
```

ORB Problems

instance orb01

```
+++++++++++++++++++++++++++++
trivial 10x10 instance from Bill Cook (BIC2)
10 10
0 72 1 64 2 55 3 31 4 53 5 95 6 11 7 52 8  6 9 84
0 61 3 27 4 88 2 78 1 49 5 83 8 91 6 74 7 29 9 87
0 86 3 32 1 35 2 37 5 18 4 48 6 91 7 52 9 60 8 30
0  8 1 82 4 27 3 99 6 74 5  9 2 33 9 20 7 59 8 98
1 50 0 94 5 43 3 62 4 55 7 48 2  5 8 36 9 47 6 36
0 53 6 30 2  7 3 12 1 68 8 87 4 28 9 70 7 45 5  7
2 29 3 96 0 99 1 14 4 34 7 14 5  7 6 76 8 57 9 76
2 90 0 19 3 87 4 51 1 84 5 45 9 84 6 58 7 81 8 96
2 97 1 99 4 93 0 38 7 13 5 96 3 40 9 64 6 32 8 45
2 44 0 60 8 29 3  5 6 74 1 85 4 34 7 95 9 51 5 47
+++++++++++++++++++++++++++++
```

instance orb02

```
+++++++++++++++++++++++++++++
doomed 10x10 instance from Monika (MON2)
10 10
0 72 1 54 2 33 3 86 4 75 5 16 6 96 7  7 8 99 9 76
0 16 3 88 4 48 8 52 9 60 6 29 7 18 5 89 2 80 1 76
0 47 7 11 3 14 2 56 6 16 4 83 1 10 5 61 8 24 9 58
0 49 1 31 3 17 8 50 5 63 2 35 4 65 7 23 6 50 9 29
0 55 6  6 1 28 3 96 5 86 2 99 9 14 7 70 8 64 4 24
4 46 0 23 6 70 8 19 2 54 3 22 9 85 7 87 5 79 1 93
4 76 3 60 0 76 9 98 2 76 1 50 8 86 7 14 6 27 5 57
4 93 6 27 9 57 3 87 8 86 2 54 7 24 5 49 0 20 1 47
2 28 6 11 8 78 7 85 4 63 9 81 3 10 1  9 5 46 0 32
2 22 9 76 5 89 8 13 6 88 3 10 7 75 4 98 1 78 0 17
+++++++++++++++++++++++++++++
```

instance orb03

```
+++++++++++++++++++++++++++++
deadlier 10x10 instance from Bruce Gamble (BRG1)
10 10
0 96 1 69 2 25 3  5 4 55 5 15 6 88 7 11 8 17 9 82
0 11 1 48 2 67 3 38 4 18 7 24 6 62 5 92 9 96 8 81
2 67 1 63 0 93 4 85 3 25 5 72 6 51 7 81 8 58 9 15
2 30 1 35 0 27 4 82 3 44 7 92 6 25 5 49 9 28 8 77
1 53 0 83 4 73 3 26 2 77 6 33 5 92 9 99 8 38 7 38
1 20 0 44 4 81 3 88 2 66 6 70 5 91 9 37 8 55 7 96
1 21 2 93 4 22 0 56 3 34 6 40 7 53 9 46 5 29 8 63
1 32 2 63 4 36 0 26 3 17 5 85 7 15 8 55 9 16 6 82
0 73 2 46 3 89 4 24 1 99 6 92 7  7 9 51 5 19 8 14
0 52 2 20 3 70 4 98 1 23 5 15 7 81 8 71 9 24 6 81
+++++++++++++++++++++++++++++
```

instance orb04

```
+++++++++++++++++++++++++++++
deadly 10x10 instance from Bruce Shepherd (BRS1)
10 10
```

```
0  8 1 10 2 35 3 44 4 15 5 92 6 70 7 89 8 50 9 12
0 63 8 39 3 80 5 22 2 88 1 39 9 85 6 27 7 74 4 69
0 52 6 22 1 33 3 68 8 27 2 68 5 25 4 34 7 24 9 84
0 31 1 85 4 55 8 80 5 58 7 11 6 69 9 56 3 73 2 25
0 97 5 98 9 87 8 47 7 77 4 90 3 98 2 80 1 39 6 40
1 97 5 68 0 44 9 67 2 44 8 85 3 78 6 90 7 33 4 81
0 34 3 76 8 48 7 61 9 11 2 36 4 33 6 98 1  7 5 44
0 44 9  5 4 85 1 51 5 58 7 79 2 95 6 48 3 86 8 73
0 24 1 63 9 48 7 77 8 73 6 74 4 63 5 17 2 93 3 84
0 51 2  5 4 40 9 60 1 46 5 58 8 54 3 72 6 29 7 94
++++++++++++++++++++++++++++++
```

instance orb05

```
++++++++++++++++++++++++++++++
10x10 instance from George Steiner (GES1)
10 10
9 11 8 93 0 48 7 76 6 13 5 71 3 59 2 90 4 10 1 65
8 52 9 76 0 84 7 73 5 56 4 10 6 26 2 43 3 39 1 49
9 28 8 44 7 26 6 66 4 68 5 74 3 27 2 14 1  6 0 21
0 18 1 58 3 62 2 46 6 25 4  6 5 60 7 28 8 80 9 30
0 78 1 47 7 29 5 16 4 29 6 57 3 78 2 87 8 39 9 73
9 66 8 51 3 12 7 64 5 67 4 15 6 66 2 26 1 20 0 98
8 23 9 76 6 45 7 75 5 24 3 18 4 83 2 15 1 88 0 17
9 56 8 83 7 80 6 16 4 31 5 93 3 30 2 29 1 66 0 28
9 79 8 69 2 82 4 16 5 62 3 41 6 91 7 35 0 34 1 75
0  5 1 19 2 20 3 12 4 94 5 60 6 99 7 31 8 96 9 63
++++++++++++++++++++++++++++++
```

instance orb06

```
++++++++++++++++++++++++++++++
trivial 10X10 instance from Bill Cook (BIC1)
10 10
0 99 1 74 2 49 3 67 4 17 5  7 6  9 7 39 8 35 9 49
0 49 3 67 4 82 2 92 1 62 5 84 8 45 6 30 7 42 9 71
0 26 3 33 1 82 2 98 5 83 4 16 6 64 7 65 9 36 8 77
0 41 1 62 4 73 3 94 6 51 5 46 2 55 9 31 7 64 8 46
1 68 0 26 5 50 3 46 4 25 7 88 2  6 8 13 9 98 6 84
0 24 6 80 2 91 3 55 1 48 8 99 4 72 9 91 7 84 5 12
2 16 3 13 0  9 1 58 4 23 7 85 5 36 6 89 8 71 9 41
2 54 0 41 3 38 4 53 1 11 5 74 9 88 6 46 7 41 8 65
2 53 1 50 4 40 0 90 7  7 5 80 3 57 9 60 6 91 8 47
2 45 0 59 8 81 3 99 6 71 1 19 4 75 7 77 9 94 5 95
++++++++++++++++++++++++++++++
```

instance orb07

```
++++++++++++++++++++++++++++++
doomed 10x10 instance from Monika (MON1)
10 10
0 32 1 14 2 15 3 37 4 18 5 43 6 19 7 27 8 28 9 31
0  8 3 12 4 49 8 24 9 52 6 19 7 23 5 19 2 17 1 32
0 25 7 19 3 27 2 45 6 21 4 15 1 13 5 16 8 43 9 19
0 24 1 18 3 41 8 29 5 14 2 17 4 23 7 15 6 18 9 23
0 27 6 29 1 39 3 21 5 15 2 15 9 25 7 26 8 44 4 20
4 17 0 15 6 51 8 17 2 46 3 16 9 33 7 25 5 30 1 25
4 15 3 31 0 25 9 12 2 13 1 51 8 19 7 21 6 12 5 26
4  8 6 29 9 25 3 15 8 17 2 22 7 32 5 20 0 11 1 28
2 41 6 10 8 32 7  5 4 21 9 59 3 26 1 10 5 16 0 29
2 20 9  7 5 44 8 22 6 33 3 25 7 29 4 12 1 14 0  0
++++++++++++++++++++++++++++++
```

instance orb08

```
++++++++++++++++++++++++++++++
deadlier 10x10 instance from Bruce Gamble (BRG2)
10 10
0 55 1 74 2 45 3 23 4 76 5 19 6 18 7 61 8 44 9 11
0 63 1 43 2 51 3 18 4 42 7 11 6 29 5 52 9 29 8 88
2 88 1 31 0 47 4 10 3 62 5 60 6 58 7 29 8 52 9 92
2 16 1 71 0 55 4 55 3  9 7 49 6 83 5 54 9  7 8 57
1  7 0 41 4 92 3 94 2 46 6 79 5 34 9 38 8  8 7 18
1 25 0  5 4 89 3 94 2 14 6 94 5 20 9 23 8 44 7 39
1 24 2 21 4 47 0 40 3 94 6 71 7 89 9 75 5 97 8 15
1  5 2  7 4 74 0 28 3 72 5 61 7  9 8 53 9 32 6 97
0 34 2 52 3 37 4  6 1 94 6  6 7 56 9 41 5  5 8 16
0 77 2 74 3 82 4 10 1 29 5 15 7 51 8 65 9 37 6 21
++++++++++++++++++++++++++++++
```

```
instance orb09

+++++++++++++++++++++++++++++
deadly 10x10 instance from Bruce Shepherd (BRS2)
10 10
0 36 1 96 2 86 3  7 4 20 5  9 6 39 7 79 8 82 9 24
0 16 8 95 3 67 5 63 2 87 1 24 9 62 6 49 7 92 4 16
0 65 6 71 1  9 3 67 8 70 2 48 5 49 4 66 7  5 9 96
0 50 1 31 4  6 8 13 5 98 7 97 6 93 9 30 3 34 2 83
0 99 5  7 9 55 8 78 7 68 4 81 3 90 2 75 1 66 6 40
1 42 5 11 0  5 9 39 2 10 8 30 3 39 6 50 7 20 4 51
0 38 3 68 8 86 7 77 9 32 2 89 4 37 6 53 1 43 5 89
0 19 9 11 4 37 1 41 5 72 7  7 2 52 6 31 3 68 8 10
0 83 1 21 9 23 7 87 8 58 6 89 4 74 5 29 2 74 3 23
0 44 2 57 4 69 9 50 1 65 5 69 8 60 3 58 6 89 7 13
+++++++++++++++++++++++++++++

instance orb10

+++++++++++++++++++++++++++++
10x10 instance from George Steiner (GES2)
10 10
9 66 8 13 0 93 7 91 6 14 5 70 3 99 2 53 4 86 1 16
8 34 9 99 0 62 7 65 5 62 4 64 6 21 2 12 3  9 1 75
9 12 8 26 7 64 6 92 4 67 5 28 3 66 2 83 1 38 0 58
0 77 1 73 3 82 2 75 6 84 4 19 5 18 7 89 8  8 9 73
0 34 1 74 7 48 5 44 4 92 6 40 3 60 2 62 8 22 9 67
9  8 8 85 3 58 7 97 5 92 4 89 6 75 2 77 1 95 0  5
8 52 9 43 6  5 7 78 5 12 3 62 4 21 2 80 1 60 0 31
9 81 8 23 7 23 6 75 4 78 5 56 3 51 2 39 1 53 0 96
9 79 8 55 2 88 4 21 5 83 3 93 6 47 7 10 0 63 1 14
0 43 1 63 2 83 3 29 4 52 5 98 6 54 7 39 8 33 9 23
+++++++++++++++++++++++++++++
```

APPENDIX B

Main Structure of Hybrid GA Programming in MATLAB

```
%JOB SHOP INPUT
%Machine sequence based on the job and operation
M= [0,1,2,3,4,5,6,7,8,9;...
    0,2,4,9,3,1,6,5,7,8;...
    1,0,3,2,8,5,7,6,9,4;...
    1,2,0,4,6,8,7,3,9,5;...
    2,0,1,5,3,4,8,7,9,6;...
    2,1,5,3,8,9,0,6,4,7;...
    1,0,3,2,6,5,9,8,7,4;...
    2,0,1,5,4,6,8,9,7,3;...
    0,1,3,5,2,9,6,7,4,8;...
    1,0,2,6,8,9,5,3,4,7 ];

p= [29,78, 9,36,49,11,62,56,44,21;...
    43,90,75,11,69,28,46,46,72,30;...
    91,85,39,74,90,10,12,89,45,33;...
    81,95,71,99, 9,52,85,98,22,43;...
    14, 6,22,61,26,69,21,49,72,53;...
    84, 2,52,95,48,72,47,65, 6,25;...
    46,37,61,13,32,21,32,89,30,55;...
    31,86,46,74,32,88,19,48,36,79;...
    76,69,76,51,85,11,40,89,26,74;...
    85,13,61, 7,64,76,47,52,90,45 ];


[Mrow,Mcolumn]=size(M);
for i=1:Mrow
   S=M(i,:);

   for j=1:Mcolumn
    S(j)=S(j)+1;
   end
   M(i,:)=S;
end

%INITIALIZATION
NIND = 100;      %numbers of individuals per populations (population size)
MAXGEN=150;      %maximum number of generations (1 generation = population size* cross over rate)
OXrate= 0.7;     %crossover possibilities (rate)
MUTrate=1.0;     %mutation possibilities (rate)
noprt=3;         %number of parents
gen=0;           %initial counter for iteration

tic
%create chromosomes
BaseV= crtbase ([10 10 10 10 10 10 10 10 10 10],[1 2 3 4 5 6 7 8 9 10]);
Chrom = zeros(NIND, length(BaseV));
for C=1:NIND
    randjob=randperm(length(BaseV));
    Chrom(C,:)=randjob;
    for C1=1:length(BaseV)
        ChromJ=Chrom(C,C1);
        Chrom(C,C1)=BaseV(ChromJ);
    end
end

%EVALUATION
for D=1:NIND
    Chro=Chrom(D,:);
    [MStart,MFinish,MJob,MSeq]=scheduling13(M,p,Chro);
    [Mm,Mn]=size(MFinish);
    Makespan(D,1)=max(reshape(MFinish,1,Mm*Mn));
end
ObjV=Makespan;


%find minimum makespan at initial population
MinVal=min(ObjV);

%calculate solutions generated after recombination
Num=NIND/noprt;
Num1=fix(Num);
ObjVSel=zeros(Num1,1);

% Generational loop
   while gen < MAXGEN

    % Assign fitness-value to entire population
        FitnV = ranking(ObjV);

    % Select individuals for breeding
        SelCh = select('sus', Chrom, FitnV);

    % Recombine selected individuals (crossover)
```

```
        SelCh01=crsovr_multi01_3(SelCh,NIND,OXrate,noprt);

    % Perform mutation on offspring
        SelCh01=mtt(SelCh01,Num1,MUTrate);

        for E=1:Num1
            Chro=SelCh01(E,:);

            % Perform iterative forward-backward pass
            [Chro,MakespanA]=CP_FB(M,p,Chro);

            % Perform neighborhood seach
            [Chro,MakespanA]=CP_SIN(M,p,Chro);

            % Evaluation on the offspring for reinsertion
            SelCh01(E,:)=Chro;
            Makespan01(E,1)=MakespanA;
        end
        ObjVSel=Makespan01;
    %default reinsertion in the GA toolbox
        [Chrom ObjV]=reins(Chrom,SelCh01,1,[1 0.3],ObjV,ObjVSel);


    %get the minimum  makespan new population and compare
        MinRsrt=min(ObjV);

        if MinVal>MinRsrt
           MinVal=MinRsrt;

        end

    % Increment generational counter
   gen = gen+1;

   end

%get the best  solution
BestMinVal=MinVal

 toc

% End of GA
```

Initialize chromosome

```
% CRTBASE.m - Create base vector
%
% This function creates a vector containing the base of the loci
% in a chromosome.
%
% Syntax: BaseVec = crtbase(Lind, Base)
%
% Input Parameters:
%
%       Lind    - A scalar or vector containing the lengths
%                 of the alleles.  Sum(Lind) is the length of
%                 the corresponding chromosome.
%
%       Base    - A scalar or vector containing the base of
%                 the loci contained in the Alleles.
%
% Output Parameters:
%
%       BaseVec - A vector whose elements correspond to the base
%                 of the loci of the associated chromosome structure.
%

function BaseVec = crtbase(Lind, Base)

[ml LenL] = size(Lind) ;
if nargin < 2
    Base = 2 * ones(LenL,1) ; % default to base 2
end
[mb LenB] = size(Base) ;

% check parameter consistency
if ml > 1 | mb > 1
    error( 'Lind or Base is not a vector') ;
elseif (LenL > 1 & LenB > 1 & LenL ~= LenB) | (LenL == 1 & LenB > 1 )
    error( 'Vector dimensions must agree' ) ;
elseif LenB == 1 & LenL > 1
    Base = Base * ones(LenL,1) ;
end

BaseVec = [] ;
for i = 1:LenL
    BaseVec = [BaseVec, Base(i)*ones(Lind(i),1)'];
end
```

Evaluation (Generate Active Schedule by Forward Pass)

```
function [MFinishA,MStartA,MJobM,ChroMMM]=CP_Fwd(M,p,Chro)

[i,j]=size(M);

%get the matrix for the machine
MStartA=zeros(j,i);
MFinishA=zeros(j,i);
MJobM=zeros(j,i);

%get the matrix for the job for record purpose
FpcopyM=zeros(i,j);
SpcopyM=zeros(i,j);
MMJob=zeros(i,j);
maxChro=length(Chro);

%chromosome for the machines
MChro=Chro;

for k=1:maxChro
    gene=MChro(1);
    ind=find(MChro==gene);
    getpos=length(ind);
    remain=j-getpos;

    %change the matrix to chromosome for the time and machine
    NChroM(1,k)=M(gene,1+remain);


    MChro(1)=[];
end

Chro2=Chro;

   %find the ealiest completion time for each operation
   for movB=1:maxChro
       %find the machine position
        geneB=NChroM(1);
        indB=find(NChroM==geneB);
        getposB=length(indB);
        remainB=i-getposB;

        %the chro(job) number
        geneC=Chro2(1);
        indC=find(Chro2==geneC);
        getposC=length(indC);
        remainC=j-getposC;

        %if the job operation is 1
        if  remainC==0

            %check the possiblilities for the for 1st job at machine
            if MStartA(geneB,1)>=p(geneC,1+remainC)

                    %insertion for the job
                    FMMFinishA=MFinishA(geneB,:);
                    SMMStartA=MStartA(geneB,:);
                    MMMJobM=MJobM(geneB,:);
                    insertA=p(geneC,1+remainC) ;
                    FMMFinishA(end)=[];
                    SMMStartA(end)=[];
                    MMMJobM(end)=[];
                    FMMFinishA=[insertA  FMMFinishA(1:end)];
                    SMMStartA=[0 SMMStartA(1:end)];
                    MMMJobM=[Chro2(1) MMMJobM(1:end)];

                    %get the machine time
                    MFinishA(geneB,:)= FMMFinishA;
                    MStartA(geneB,:)=SMMStartA;
                    MJobM(geneB,:)=MMMJobM;

                    %get the job time
                    FpcopyM(geneC,1+remainC)=insertA;
                    SpcopyM(geneC,1+remainC)=(insertA-p(geneC,1+remainC));
                    MMJob(geneC,1+remainC)=geneB;
            else
                    %check the possibilities of insertion for the 2nd job or above for machine
                    movC=1;
               while  (MStartA(geneB,movC+1)-MFinishA(geneB,movC))<p(geneC,1+remainC) && movC<(i-1)
                   movC=movC+1;
               end

               if (MStartA(geneB,movC+1)-MFinishA(geneB,movC))>=p(geneC,1+remainC)

                    FMMFinishA=MFinishA(geneB,:);
```

```
                    SMMStartA=MStartA(geneB,:);
                    MMMJobM=MJobM(geneB,:);

                    insertA=MFinishA(geneB,movC)+p(geneC,1+remainC) ;

                    FMMFinishA(end)=[];
                    SMMStartA(end)=[];
                    MMMJobM(end)=[];


                    FMMFinishA=[FMMFinishA(1:movC) insertA  FMMFinishA((movC+1):end)];
                    SMMStartA=[SMMStartA(1:movC) (insertA-p(geneC,1+remainC))
SMMStartA((movC+1):end)];
                    MMMJobM=[MMMJobM(1:movC) Chro2(1) MMMJobM((movC+1):end)];



                    MFinishA(geneB,:)= FMMFinishA;
                    MStartA(geneB,:)=SMMStartA;
                    MJobM(geneB,:)=MMMJobM;

                    FpcopyM(geneC,1+remainC)=insertA;
                    SpcopyM(geneC,1+remainC)=(insertA-p(geneC,1+remainC));
                    MMJob(geneC,1+remainC)=geneB;


               else
                    MFinishA(geneB,1+remainB)=max(MFinishA(geneB,:))+p(geneC,1+remainC);
                    MStartA(geneB,1+remainB)=MFinishA(geneB,1+remainB)-p(geneC,1+remainC);

                    FpcopyM(geneC,1+remainC)=MFinishA(geneB,1+remainB);
                    SpcopyM(geneC,1+remainC)=MStartA(geneB,1+remainB);
                    MMJob(geneC,1+remainC)=geneB;

                    MJobM(geneB,1+remainB)=Chro2(1);

               end
           end
       %if the job operation is 2 or above for insertion
       else
            movC=1;
               while  ((MStartA(geneB,movC+1)-FpcopyM(geneC,remainC))<p(geneC,1+remainC)
||(MStartA(geneB,movC+1)-MFinishA(geneB,movC))<p(geneC,1+remainC) ) && movC<(i-1)
                   movC=movC+1;
               end
          if (MStartA(geneB,movC+1)-FpcopyM(geneC,remainC))>=p(geneC,1+remainC)
&&(MStartA(geneB,movC+1)-MFinishA(geneB,movC))>=p(geneC,1+remainC)
                    if FpcopyM(geneC,remainC)<MFinishA(geneB,movC)
                        FMMFinishA=MFinishA(geneB,:);
                        SMMStartA=MStartA(geneB,:);
                        MMMJobM=MJobM(geneB,:);

                        insertA=MFinishA(geneB,movC)+p(geneC,1+remainC) ;

                        FMMFinishA(end)=[];
                        SMMStartA(end)=[];
                        MMMJobM(end)=[];


                        FMMFinishA=[FMMFinishA(1:movC) insertA  FMMFinishA((movC+1):end)];
                        SMMStartA=[SMMStartA(1:movC) (insertA-p(geneC,1+remainC))
SMMStartA((movC+1):end)];
                        MMMJobM=[MMMJobM(1:movC) Chro2(1) MMMJobM((movC+1):end)];



                        MFinishA(geneB,:)= FMMFinishA;
                        MStartA(geneB,:)=SMMStartA;
                        MJobM(geneB,:)=MMMJobM;

                        FpcopyM(geneC,1+remainC)=insertA;
                        SpcopyM(geneC,1+remainC)=(insertA-p(geneC,1+remainC));
                        MMJob(geneC,1+remainC)=geneB;
                    else
                        FMMFinishA=MFinishA(geneB,:);
                        SMMStartA=MStartA(geneB,:);
                        MMMJobM=MJobM(geneB,:);

                        insertA=FpcopyM(geneC,remainC)+p(geneC,1+remainC) ;

                        FMMFinishA(end)=[];
                        SMMStartA(end)=[];
                        MMMJobM(end)=[];


                        FMMFinishA=[FMMFinishA(1:movC) insertA  FMMFinishA((movC+1):end)];
                        SMMStartA=[SMMStartA(1:movC) (insertA-p(geneC,1+remainC))
SMMStartA((movC+1):end)];
                        MMMJobM=[MMMJobM(1:movC) Chro2(1) MMMJobM((movC+1):end)];
```

```
                        MFinishA(geneB,:)= FMMFinishA;
                        MStartA(geneB,:)=SMMStartA;
                        MJobM(geneB,:)=MMMJobM;

                        FpcopyM(geneC,1+remainC)=insertA;
                        SpcopyM(geneC,1+remainC)=(insertA-p(geneC,1+remainC));
                        MMJob(geneC,1+remainC)=geneB;


                    end
            else
                if max(MFinishA(geneB,:))>=FpcopyM(geneC,remainC)
                    MFinishA(geneB,1+remainB)=max(MFinishA(geneB,:))+p(geneC,1+remainC);
                    MStartA(geneB,1+remainB)= MFinishA(geneB,1+remainB)-p(geneC,1+remainC);

                    FpcopyM(geneC,1+remainC)=MFinishA(geneB,1+remainB);
                    SpcopyM(geneC,1+remainC)=MStartA(geneB,1+remainB);
                    MMJob(geneC,1+remainC)=geneB;

                    MJobM(geneB,1+remainB)=Chro2(1);

                else
                    MFinishA(geneB,1+remainB)=FpcopyM(geneC,remainC)+p(geneC,1+remainC);
                    MStartA(geneB,1+remainB)= MFinishA(geneB,1+remainB)-p(geneC,1+remainC);

                    FpcopyM(geneC,1+remainC)=MFinishA(geneB,1+remainB);
                    SpcopyM(geneC,1+remainC)=MStartA(geneB,1+remainB);
                    MMJob(geneC,1+remainC)=geneB;

                    MJobM(geneB,1+remainB)=Chro2(1);
                end
            end


        end
        NChroM(1)=[];
        Chro2(1)=[];
    end

% New chromosome generated
[ChroMMM]=timearr_01(MStartA,MJobM);
```

Generate New Chromosome from the Active Schedule

```
function [ChroMMM]=timearr_01(MStartA,MJobM)

[Srow,Scolumn]=size(MStartA);
Chro_SM=reshape(MStartA,1,Srow*Scolumn);
Chro_JM=reshape(MJobM,1,Srow*Scolumn);
ArrTS=sort(Chro_SM);
pjgSM=length(Chro_SM);
ChroMMM=zeros(1,pjgSM);

%Generate new chromosome
for loopA=1:pjgSM
    posArrTS=find(Chro_SM==ArrTS(1));
    ChroMMM(loopA)=Chro_JM(posArrTS(1));

    Chro_SM(posArrTS(1))=[];
    Chro_JM(posArrTS(1))=[];
    ArrTS(1)=[];
end
```

Ranking for Chromosome (Check Fitness)

```
% RANKING.M      (RANK-based fitness assignment)
%
% This function performs ranking of individuals.
%
% Syntax:  FitnV = ranking(ObjV, RFun, SUBPOP)
%
% This function ranks individuals represented by their associated
% cost, to be *minimized*, and returns a column vector FitnV
% containing the corresponding individual fitnesses. For multiple
% subpopulations the ranking is performed separately for each
% subpopulation.
%
% Input parameters:
```

```
%    ObjV      - Column vector containing the objective values of the
%                individuals in the current population (cost values).
%    RFun      - (optional) If RFun is a scalar in [1, 2] linear ranking is
%                assumed and the scalar indicates the selective pressure.
%                If RFun is a 2 element vector:
%                RFun(1): SP - scalar indicating the selective pressure
%                RFun(2): RM - ranking method
%                         RM = 0: linear ranking
%                         RM = 1: non-linear ranking
%                If RFun is a vector with length(Rfun) > 2 it contains
%                the fitness to be assigned to each rank. It should have
%                the same length as ObjV. Usually RFun is monotonously
%                increasing.
%                If RFun is omitted or NaN, linear ranking
%                and a selective pressure of 2 are assumed.
%    SUBPOP    - (optional) Number of subpopulations
%                if omitted or NaN, 1 subpopulation is assumed
%
% Output parameters:
%    FitnV     - Column vector containing the fitness values of the
%                individuals in the current population.
%
%
function FitnV = ranking(ObjV, RFun, SUBPOP);

% Identify the vector size (Nind)
   [Nind,ans] = size(ObjV);

   if nargin < 2, RFun = []; end
   if nargin > 1, if isnan(RFun), RFun = []; end, end
   if prod(size(RFun)) == 2,
      if RFun(2) == 1, NonLin = 1;
      elseif RFun(2) == 0, NonLin = 0;
      else error('Parameter for ranking method must be 0 or 1'); end
      RFun = RFun(1);
      if isnan(RFun), RFun = 2; end
   elseif prod(size(RFun)) > 2,
      if prod(size(RFun)) ~= Nind, error('ObjV and RFun disagree'); end
   elseif prod(size(RFun)) < 2, NonLin = 0;
   end

   if nargin < 3, SUBPOP = 1; end
   if nargin > 2,
      if isempty(SUBPOP), SUBPOP = 1;
      elseif isnan(SUBPOP), SUBPOP = 1;
      elseif length(SUBPOP) ~= 1, error('SUBPOP must be a scalar'); end
   end

   if (Nind/SUBPOP) ~= fix(Nind/SUBPOP), error('ObjV and SUBPOP disagree'); end
   Nind = Nind/SUBPOP;  % Compute number of individuals per subpopulation

% Check ranking function and use default values if necessary
   if isempty(RFun),
      % linear ranking with selective pressure 2
         RFun = 2*[0:Nind-1]'/(Nind-1);
   elseif prod(size(RFun)) == 1
      if NonLin == 1,
         % non-linear ranking
         if RFun(1) < 1, error('Selective pressure must be greater than 1');
         elseif RFun(1) > Nind-2, error('Selective pressure too big'); end
         Root1 = roots([RFun(1)-Nind [RFun(1)*ones(1,Nind-1)]]);
         RFun = (abs(Root1(1)) * ones(Nind,1)) .^ [(0:Nind-1)'];
         RFun = RFun / sum(RFun) * Nind;
      else
         % linear ranking with SP between 1 and 2
         if (RFun(1) < 1 | RFun(1) > 2),
            error('Selective pressure for linear ranking must be between 1 and 2');
         end
         RFun = 2-RFun + 2*(RFun-1)*[0:Nind-1]'/(Nind-1);
      end
   end;

   FitnV = [];

% loop over all subpopulations
for irun = 1:SUBPOP,
   % Copy objective values of actual subpopulation
      ObjVSub = ObjV((irun-1)*Nind+1:irun*Nind);
   % Sort does not handle NaN values as required. So, find those...
      NaNix = isnan(ObjVSub);
      Validix = find(~NaNix);
   % ... and sort only numeric values (smaller is better).
      [ans,ix] = sort(-ObjVSub(Validix));

   % Now build indexing vector assuming NaN are worse than numbers,
   % (including Inf!)...
      ix = [find(NaNix) ; Validix(ix)];
   % ... and obtain a sorted version of ObjV
      Sorted = ObjVSub(ix);

   % Assign fitness according to RFun.
```

```
      i = 1;
      FitnVSub = zeros(Nind,1);
      for j = [find(Sorted(1:Nind-1) ~= Sorted(2:Nind)); Nind]',
         FitnVSub(i:j) = sum(RFun(i:j)) * ones(j-i+1,1) / (j-i+1);
         i =j+1;
      end

   % Finally, return unsorted vector.
      [ans,uix] = sort(ix);
      FitnVSub = FitnVSub(uix);

   % Add FitnVSub to FitnV
      FitnV = [FitnV; FitnVSub];
end

% End of function
```

Selection

```
% SELECT.M          (universal SELECTion)
%
% This function performs universal selection. The function handles
% multiple populations and calls the low level selection function
% for the actual selection process.
%
% Syntax:  SelCh = select(SEL_F, Chrom, FitnV, GGAP, SUBPOP)
%
% Input parameters:
%    SEL_F     - Name of the selection function
%    Chrom     - Matrix containing the individuals (parents) of the current
%                population. Each row corresponds to one individual.
%    FitnV     - Column vector containing the fitness values of the
%                individuals in the population.
%    GGAP      - (optional) Rate of individuals to be selected
%                if omitted 1.0 is assumed
%    SUBPOP    - (optional) Number of subpopulations
%                if omitted 1 subpopulation is assumed
%
% Output parameters:
%    SelCh     - Matrix containing the selected individuals.

function SelCh = select(SEL_F, Chrom, FitnV, GGAP, SUBPOP);

% Check parameter consistency
   if nargin < 3, error('Not enough input parameter'); end

   % Identify the population size (Nind)
   [NindCh,Nvar] = size(Chrom);
   [NindF,VarF] = size(FitnV);
   if NindCh ~= NindF, error('Chrom and FitnV disagree'); end
   if VarF ~= 1, error('FitnV must be a column vector'); end

   if nargin < 5, SUBPOP = 1; end
   if nargin > 4,
      if isempty(SUBPOP), SUBPOP = 1;
      elseif isnan(SUBPOP), SUBPOP = 1;
      elseif length(SUBPOP) ~= 1, error('SUBPOP must be a scalar'); end
   end

   if (NindCh/SUBPOP) ~= fix(NindCh/SUBPOP), error('Chrom and SUBPOP disagree'); end
   Nind = NindCh/SUBPOP;  % Compute number of individuals per subpopulation

   if nargin < 4, GGAP = 1; end
   if nargin > 3,
      if isempty(GGAP), GGAP = 1;
      elseif isnan(GGAP), GGAP = 1;
      elseif length(GGAP) ~= 1, error('GGAP must be a scalar');
      elseif (GGAP < 0), error('GGAP must be a scalar bigger than 0'); end
   end

% Compute number of new individuals (to select)
   NSel=max(floor(Nind*GGAP+.5),2);

% Select individuals from population
   SelCh = [];
   for irun = 1:SUBPOP,
      FitnVSub = FitnV((irun-1)*Nind+1:irun*Nind);
      ChrIx=feval(SEL_F, FitnVSub, NSel)+(irun-1)*Nind;
      SelCh=[SelCh; Chrom(ChrIx,:)];
   end


% End of function
```

Stochastic Universal Sampling, SUS

```
% SUS.M          (Stochastic Universal Sampling)
%
% This function performs selection with STOCHASTIC UNIVERSAL SAMPLING.
%
% Syntax:  NewChrIx = sus(FitnV, Nsel)
%
% Input parameters:
%    FitnV     - Column vector containing the fitness values of the
%                individuals in the population.
%    Nsel      - number of individuals to be selected
%
% Output parameters:
%    NewChrIx  - column vector containing the indexes of the selected
%                individuals relative to the original population, shuffled.
%                The new population, ready for mating, can be obtained
%                by calculating OldChrom(NewChrIx,:).

function NewChrIx = sus(FitnV,Nsel);

% Identify the population size (Nind)
   [Nind,ans] = size(FitnV);

% Perform stochastic universal sampling
   cumfit = cumsum(FitnV);
   trials = cumfit(Nind) / Nsel * (rand + (0:Nsel-1)');
   Mf = cumfit(:, ones(1, Nsel));
   Mt = trials(:, ones(1, Nind))';
   [NewChrIx, ans] = find(Mt < Mf & [ zeros(1, Nsel); Mf(1:Nind-1, :) ] <= Mt);

% Shuffle new population
   [ans, shuf] = sort(rand(Nsel, 1));
   NewChrIx = NewChrIx(shuf);


% End of function
```

Mutation

```
function ChromNew=mtt(SelCh,NIND,MUTrate)

 ChromNew=SelCh;
 opr=length(SelCh(1,:));
 opr=randperm(opr);

for i=1:NIND

a=rand;
 if MUTrate>a;

j=2;
    S=SelCh(i,:);
    while S((opr(1)))==S(opr(j))
    j=j+1;
    end

    temp=S((opr(1)));
    S(opr(1))=S(opr(j));
    S(opr(j))=temp;


    ChromNew(i,:)=S;
 end
end
```

Reinsertion

```
% REINS.M        (RE-INSertion of offspring in population replacing parents)
%
% This function reinserts offspring in the population.
%
% Syntax: [Chrom, ObjVCh] = reins(Chrom, SelCh, SUBPOP, InsOpt, ObjVCh, ObjVSel)
%
% Input parameters:
%    Chrom     - Matrix containing the individuals (parents) of the current
%                population. Each row corresponds to one individual.
%    SelCh     - Matrix containing the offspring of the current
%                population. Each row corresponds to one individual.
%    SUBPOP    - (optional) Number of subpopulations
%                if omitted or NaN, 1 subpopulation is assumed
%    InsOpt    - (optional) Vector containing the insertion method parameters
%                ExOpt(1): Select - number indicating kind of insertion
```

```
%                           0 - uniform insertion
%                           1 - fitness-based insertion
%                           if omitted or NaN, 0 is assumed
%                 ExOpt(2): INSR - Rate of offspring to be inserted per
%                           subpopulation (% of subpopulation)
%                           if omitted or NaN, 1.0 (100%) is assumed
%    ObjVCh     - (optional) Column vector containing the objective values
%                 of the individuals (parents - Chrom) in the current
%                 population, needed for fitness-based insertion
%                 saves recalculation of objective values for population
%    ObjVSel    - (optional) Column vector containing the objective values
%                 of the offspring (SelCh) in the current population, needed for
%                 partial insertion of offspring,
%                 saves recalculation of objective values for population
%
% Output parameters:
%    Chrom      - Matrix containing the individuals of the current
%                 population after reinsertion.
%    ObjVCh     - if ObjVCh and ObjVSel are input parameters, then column
%                 vector containing the objective values of the individuals
%                 of the current generation after reinsertion.
%


function [Chrom, ObjVCh] = reins(Chrom, SelCh, SUBPOP, InsOpt, ObjVCh, ObjVSel);

% Check parameter consistency
   if nargin < 2, error('Not enough input parameter'); end
   if (nargout == 2 & nargin < 6), error('Input parameter missing: ObjVCh and/or ObjVSel'); end

   [NindP, NvarP] = size(Chrom);
   [NindO, NvarO] = size(SelCh);

   if nargin == 2, SUBPOP = 1; end
   if nargin > 2,
      if isempty(SUBPOP), SUBPOP = 1;
      elseif isnan(SUBPOP), SUBPOP = 1;
      elseif length(SUBPOP) ~= 1, error('SUBPOP must be a scalar'); end
   end

   if (NindP/SUBPOP) ~= fix(NindP/SUBPOP), error('Chrom and SUBPOP disagree'); end
   if (NindO/SUBPOP) ~= fix(NindO/SUBPOP), error('SelCh and SUBPOP disagree'); end
   NIND = NindP/SUBPOP;  % Compute number of individuals per subpopulation
   NSEL = NindO/SUBPOP;  % Compute number of offspring per subpopulation

   IsObjVCh = 0; IsObjVSel = 0;
   if nargin > 4,
      [mO, nO] = size(ObjVCh);
      if nO ~= 1, error('ObjVCh must be a column vector'); end
      if NindP ~= mO, error('Chrom and ObjVCh disagree'); end
      IsObjVCh = 1;
   end
   if nargin > 5,
      [mO, nO] = size(ObjVSel);
      if nO ~= 1, error('ObjVSel must be a column vector'); end
      if NindO ~= mO, error('SelCh and ObjVSel disagree'); end
      IsObjVSel = 1;
   end

   if nargin < 4, INSR = 1.0; Select = 0; end
   if nargin >= 4,
      if isempty(InsOpt), INSR = 1.0; Select = 0;
      elseif isnan(InsOpt), INSR = 1.0; Select = 0;
      else
         INSR = NaN; Select = NaN;
         if (length(InsOpt) > 2), error('Parameter InsOpt too long'); end
         if (length(InsOpt) >= 1), Select = InsOpt(1); end
         if (length(InsOpt) >= 2), INSR = InsOpt(2); end
         if isnan(Select), Select = 0; end
         if isnan(INSR), INSR =1.0; end
      end
   end

   if (INSR < 0 | INSR > 1), error('Parameter for insertion rate must be a scalar in [0, 1]'); end
   if (INSR < 1 & IsObjVSel ~= 1), error('For selection of offspring ObjVSel is needed'); end
   if (Select ~= 0 & Select ~= 1), error('Parameter for selection method must be 0 or 1'); end
   if (Select == 1 & IsObjVCh == 0), error('ObjVCh for fitness-based exchange needed'); end

   if INSR == 0, return; end
   NIns = min(max(floor(INSR*NSEL+.5),1),NIND);   % Number of offspring to insert

% perform insertion for each subpopulation
   for irun = 1:SUBPOP,
      % Calculate positions in old subpopulation, where offspring are inserted
         if Select == 1,    % fitness-based reinsertion
            [Dummy, ChIx] = sort(-ObjVCh((irun-1)*NIND+1:irun*NIND));
         else               % uniform reinsertion
            [Dummy, ChIx] = sort(rand(NIND,1));
         end
         PopIx = ChIx((1:NIns)')+ (irun-1)*NIND;
      % Calculate position of Nins-% best offspring
```

```
            if (NIns < NSEL),  % select best offspring
               [Dummy,OffIx] = sort(ObjVSel((irun-1)*NSEL+1:irun*NSEL));
            else
               OffIx = (1:NIns)';
            end
            SelIx = OffIx((1:NIns)')+(irun-1)*NSEL;
         % Insert offspring in subpopulation -> new subpopulation
            Chrom(PopIx,:) = SelCh(SelIx,:);
            if (IsObjVCh == 1 & IsObjVSel == 1), ObjVCh(PopIx) = ObjVSel(SelIx); end
      end

   % End of function
```

APPENDIX C

Multi-Parents Crossover

```
function SelCh01 = crsovr_multi01_3(SelCh,NIND,OXrate,noprt)

lgChro=length(SelCh(1,:));
%number of parents
Num=fix(NIND/noprt);


%create space to store
Vc00=zeros(1,lgChro);
SelCh01=zeros(Num,lgChro);



SelNum=randperm(NIND); %randomly select chromosome
S11=zeros(1,lgChro);
g=1;

%Crossover operation
for i=1:noprt:(Num*noprt)
    a=rand;

    if OXrate>a;

       for Vec=1:lgChro
           rand_num=randperm(noprt);
           Vc00(1,Vec)=rand_num(1);
       end

    %already fine Vc00 and Vc01

        S1=SelCh(SelNum(i),:);
        S2=SelCh(SelNum(i+1),:);
        S3=SelCh(SelNum(i+2),:);

        for k=1:lgChro
            x=Vc00(k);

            switch x
                case 1
                    S11(k)=S1(1);
                    pos02=find(S2==S1(1));
                    pos03=find(S3==S1(1));
                    S1(1)=[];
                    S2(pos02(1))=[];
                    S3(pos03(1))=[];

                case 2
                    S11(k)=S2(1);
                    pos01=find(S1==S2(1));
                    pos03=find(S3==S2(1));
                    S2(1)=[];
                    S1(pos01(1))=[];
                    S3(pos03(1))=[];

                case 3
                    S11(k)=S3(1);
                    pos01=find(S1==S3(1));
                    pos02=find(S2==S3(1));
                    S3(1)=[];
                    S1(pos01(1))=[];
                    S2(pos02(1))=[];
            end
        end

    else
        S11=SelCh(SelNum(i),:);
    end
        SelCh01(g,:)=S11;
        g=g+1;
end
```

APPENDIX D

Neighborhood Search

```
function [Chro,Makespan]=CP_SIN(M,p,Chro)

do=1;
while do==1
%Earliest start time
[EF,ES,EJ,Chro]=CP_Fwd(M,p,Chro);
Makespan=max(EF(:,end));

%Latest start,finish time, job
[LF,LS,LJ]=CP_Bwd_NoShift(M,p,Chro,Makespan);
MStartA=ES;
MFinishA=EF;
MJobA=EJ;


%identified the critical operations
[Mrow,Mcolumn]=size(LS);
for i=1:Mrow

   for j=1:Mcolumn

       sLS=find(EJ(i,:)==LJ(i,j));
       if LS(i,j)~=ES(i,sLS)
           LS(i,j)=-1;
           LF(i,j)=-1;
           LJ(i,j)=-1;

           ES(i,sLS)=-1;
           EF(i,sLS)=-1;
           EJ(i,sLS)=-1;
       end
   end
end

[ChES,ChEF,ChEJ,ChEM]=timearr_01_critical_path(ES,EF,EJ);

sTES=ChES(1);
pChES=ChES(1);
pChEF=ChEF(1);
pChEJ=ChEJ(1);
pChEM=ChEM(1);
sg=0;
CChES=ChES;
CChEF=ChEF;
CChEJ=ChEJ;
CChEM=ChEM;
ms=1;

% identified critical path
while sg<length(ChES) && sTES==0
 ms=0;

    while max(pChEF(:,end))<Makespan && ms<length(ChES)+2% the path

        [xp,yp]=size(pChES);

        pc=0;
        for loDP=1:xp
            if loDP==1
            pChES(:,end+1)=0;
            pChEF(:,end+1)=0;
            pChEJ(:,end+1)=0;
            pChEM(:,end+1)=0;
            end
            loDP=loDP-pc;
            mst=0;
            InP=find(CChES==pChEF(loDP,end-1));

            if length(InP)>1
                for Ei=1:length(InP)
                    mst(Ei)=CChEM(InP(Ei));
                end
                mmst=mode(mst);
                indmst=find(mst==mmst);
                if length(indmst)>1
                pChES(loDP,end)=CChES(InP(indmst(1)));
                pChEF(loDP,end)=CChEF(InP(indmst(1)));
                pChEJ(loDP,end)=CChEJ(InP(indmst(1)));
                pChEM(loDP,end)=CChEM(InP(indmst(1)));
                InP=InP(indmst(2));
                pChES(:,end+1)=0;
                pChEF(:,end+1)=0;
                pChEJ(:,end+1)=0;
                pChEM(:,end+1)=0;
```

```
                end

            end



            if pChEF(loDP,end-1)==0
                pChES(loDP,:)=[];
                pChEF(loDP,:)=[];
                pChEJ(loDP,:)=[];
                pChEM(loDP,:)=[];
                pc=pc+1;
            else


            if length(InP)==1
                pChES(loDP,end)=CChES(InP);
                pChEF(loDP,end)=CChEF(InP);
                pChEJ(loDP,end)=CChEJ(InP);
                pChEM(loDP,end)=CChEM(InP);


            else
                if isempty(InP)==1;
                    pc=pc+1;
                end
                for loEP=1:length(InP)

                    pChES(loDP,end)=CChES(InP(1));
                    qChES(loEP,:)=pChES(loDP,:);

                    pChEF(loDP,end)=CChEF(InP(1));
                    qChEF(loEP,:)=pChEF(loDP,:);

                    pChEJ(loDP,end)=CChEJ(InP(1));
                    qChEJ(loEP,:)=pChEJ(loDP,:);

                    pChEM(loDP,end)=CChEM(InP(1));
                    qChEM(loEP,:)=pChEM(loDP,:);

                end

                pChES(loDP,:)=[];
                pChES=[pChES;qChES];
                qChES=[];

                pChEF(loDP,:)=[];
                pChEF=[pChEF;qChEF];
                qChEF=[];

                pChEJ(loDP,:)=[];
                pChEJ=[pChEJ;qChEJ];
                qChEJ=[];

                pChEM(loDP,:)=[];
                pChEM=[pChEM;qChEM];
                qChEM=[];

            end
            end

        end
    ms=ms+1;
    end

sg=sg+1;
sTES=CChES(1);

end

spChEF=find(pChEF(:,end)==Makespan);
if length(spChEF)~=1
    pChES=pChES(spChEF(1),:);
    pChEF=pChEF(spChEF(1),:);
    pChEJ=pChEJ(spChEF(1),:);
    pChEM=pChEM(spChEF(1),:);
end

%find critical blocks and possible swaps
BlkMac=pChEM(1);
NoBlk=1;
BlkPos=1; %block position based on machinein pchro
cr=1;
for loCrB=2:length(pChEM)

    if pChEM(loCrB)==pChEM(loCrB-1)
        NoBlk(cr,1)=NoBlk(cr,1)+1;
        BlkPos(cr,2)=loCrB;
    else
        cr=cr+1;
        BlkMac(cr,1)=pChEM(loCrB);
```

```
        NoBlk(cr,1)=1;
        BlkPos(cr,:)=loCrB;
    end
end

lia=0;
for loCrS=1:cr
    TtlBlk=BlkPos(loCrS,2)-BlkPos(loCrS,1)+1;

    if TtlBlk>=2
        swpA=randperm(TtlBlk);
        BlkA=pChEJ(BlkPos(loCrS,1)+swpA(1)-1);
        BlkB=pChEJ(BlkPos(loCrS,1)+swpA(2)-1);
        lia=lia+1;
        LiaMac(lia,1)=BlkMac(loCrS,1);
        LiaJob(lia,:)=[BlkA BlkB];
    end
end
do=0;

% Evaluate the swap and maintain the best swap
for losw=1:length(LiaMac)
    MJobM=MJobA;
    swp=ismember(MJobM(LiaMac(losw),:),LiaJob(losw,:));
    swpos=find(swp==1);
    MJobM(LiaMac(losw),swpos(1))=LiaJob(losw,2);
    MJobM(LiaMac(losw),swpos(2))=LiaJob(losw,1);

    [ChroMMM]=timearr_01(MStartA,MJobM);
    [BMFinish,BMStart,BMJob,ChroMMM]=CP_Fwd(M,p,ChroMMM);
    [BMm,BMn]=size(BMFinish);
    BMakespan=max(reshape(BMFinish,1,BMm*BMn));
    if BMakespan<Makespan
        Chro=ChroMMM;
        Makespan=BMakespan;
        do=1;
    end

end
end
```

Late Start time of the Operations

```
function [MFinishA,MStartA,MJobM]=CP_Bwd_NoShift(M,p,Chro,Makespan)
[i,j]=size(M);

%get the matrix for the machine

MStartA=zeros(j,i);
MFinishA=zeros(j,i);
MJobM=zeros(j,i);


%get the matrix for the job for record purpose
FpcopyM=zeros(i,j);
SpcopyM=zeros(i,j);
MMJob=zeros(i,j);


maxChro=length(Chro);

%matrix for the machines
MChro=Chro;

for k=1:maxChro
    gene=MChro(1);
    ind=find(MChro==gene);
    getpos=length(ind);
    remain=j-getpos;

    %change the matrix to chromosome for the time and machine
    NChroM(1,k)=M(gene,1+remain);

    MChro(1)=[];
end

Chro2=Chro;

   %find the ealiest completion time for each operation
   for movB=1:maxChro
       %find the machine position
        geneB=NChroM(end);
        indB=find(NChroM==geneB);
        getposB=length(indB);
        remainB=getposB;

        %the chro(job) number
```

```
        geneC=Chro2(end);
        indC=find(Chro2==geneC);
        %indCC=find(Chro3==geneC);
        getposC=length(indC);
        %getposCC=length(indCC);
        remainC=getposC;

        %if the job operation is 1
        if  remainC==j


                    if remainB==i
                     MFinishA(geneB,remainB)=Makespan;
                     MStartA(geneB,remainB)=Makespan-p(geneC,remainC);

                     FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                     SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                     MMJob(geneC,remainC)=geneB;

                     MJobM(geneB,remainB)=Chro2(end);

                    else
                     MFinishA(geneB,remainB)=MStartA(geneB,remainB+1);
                     MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                     FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                     SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                     MMJob(geneC,remainC)=geneB;

                     MJobM(geneB,remainB)=Chro2(end);

                    end


        %if the job operation is 2 or above
        else


                    if remainB==i
                     MFinishA(geneB,remainB)=SpcopyM(geneC,remainC+1);
                     MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                     FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                     SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                     MMJob(geneC,remainC)=geneB;

                     MJobM(geneB,remainB)=Chro2(end);

                    else
                        if MStartA(geneB,remainB+1)<=SpcopyM(geneC,remainC+1)

                     MFinishA(geneB,remainB)=MStartA(geneB,remainB+1);
                     MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                     FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                     SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                     MMJob(geneC,remainC)=geneB;

                     MJobM(geneB,remainB)=Chro2(end);


                        else
                     MFinishA(geneB,remainB)=SpcopyM(geneC,remainC+1);
                     MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                     FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                     SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                     MMJob(geneC,remainC)=geneB;

                     MJobM(geneB,remainB)=Chro2(end);


                        end
                    end


        end
        NChroM(end)=[];
        Chro2(end)=[];
    end
```

Sequencing Job in Critical Path (Based on Time Priority)

```
function [ChES,ChEF,ChEJ,ChEM]=timearr_01_critical_path(ES,EF,EJ)

[Srow,Scolumn]=size(ES);
EM=zeros(Srow,Scolumn);

%Machine
for SM=1:Srow
    EM(SM,:)=crtbase(Scolumn,SM);
end

Chro_ES=reshape(ES,1,Srow*Scolumn);
Chro_EF=reshape(EF,1,Srow*Scolumn);
Chro_EJ=reshape(EJ,1,Srow*Scolumn);
Chro_EM=reshape(EM,1,Srow*Scolumn);


ArrTS=sort(Chro_ES);

pjgES=length(Chro_ES);

ChES=zeros(1,pjgES);
ChEF=zeros(1,pjgES);
ChEJ=zeros(1,pjgES);
ChEM=zeros(1,pjgES);

%Generated the sequnces base on time
for loopA=1:pjgES
    posArrTS=find(Chro_ES==ArrTS(1));

    ChES(loopA)=Chro_ES(posArrTS(1));
    ChEF(loopA)=Chro_EF(posArrTS(1));
    ChEJ(loopA)=Chro_EJ(posArrTS(1));
    ChEM(loopA)=Chro_EM(posArrTS(1));

    Chro_ES(posArrTS(1))=[];
    Chro_EF(posArrTS(1))=[];
    Chro_EJ(posArrTS(1))=[];
    Chro_EM(posArrTS(1))=[];

    ArrTS(1)=[];

end

delCh=max(find(ChES==-1));
ChES(1:delCh)=[];
ChEF(1:delCh)=[];
ChEJ(1:delCh)=[];
ChEM(1:delCh)=[];
```

APPENDIX E

Iterative Forward-Backward Pass

```
function [Chro, AMakespan]=CP_FB(M,p,Chro)

doA=1;
ASMP=0;
AMakespan=0;
BSMP=-1;
BMakespan=-1;

% Perform iterative forward-backward pass.

while doA==1

    % Compare with backward pass
    if ASMP~=BSMP && doA==1
    [EF,ES,EJ,Chro]=CP_Fwd(M,p,Chro);
    AMakespan=max(EF(:,end));
    else
        doA=0;
    end

    % Compare with forward pass
    if BMakespan~=AMakespan && doA==1
        [LF,LS,LJ,Chro]=CP_Bwd_Shift(M,p,Chro,AMakespan);
        BSMP=min(LS(:,1));
        BMakespan=AMakespan-BSMP;

    else
        doA=0;
    end

end
```

Backward Pass

```
function [MFinishA,MStartA,MJobM,ChroMMM]=CP_Bwd_Shift(M,p,Chro,Makespan)

[i,j]=size(M);

%get the matrix for the machine

MStartA=zeros(j,i);
MFinishA=zeros(j,i);
MJobM=zeros(j,i);

%get the matrix for the job for record purpose
FpcopyM=zeros(i,j);
SpcopyM=zeros(i,j);
MMJob=zeros(i,j);
maxChro=length(Chro);

%matrix for the machines
MChro=Chro;

for k=1:maxChro
    gene=MChro(1);
    ind=find(MChro==gene);
    getpos=length(ind);
    remain=j-getpos;

    %change the matrix to chromosome for the time and machine
    NChroM(1,k)=M(gene,1+remain);
    MChro(1)=[];
end

Chro2=Chro;

   %find the ealiest completion time for each operation
   for movB=1:maxChro
       %find the machine position
        geneB=NChroM(end);
        indB=find(NChroM==geneB);
        getposB=length(indB);
        remainB=getposB;

        %the chro(job) number
        geneC=Chro2(end);
        indC=find(Chro2==geneC);
        getposC=length(indC);
        remainC=getposC;
```

```
        %if the job operation is 1
        if  remainC==j

            %check the possiblilities for the for 1st job at machine
            if (Makespan-MFinishA(geneB,i))>=p(geneC,remainC)

                    FMMFinishA=MFinishA(geneB,:);
                    SMMStartA=MStartA(geneB,:);
                    MMMJobM=MJobM(geneB,:);

                    insertA=p(geneC,remainC) ;

                    FMMFinishA(1)=[];
                    SMMStartA(1)=[];
                    MMMJobM(1)=[];


                    FMMFinishA=[FMMFinishA(1:end) Makespan];
                    SMMStartA=[SMMStartA(1:end) Makespan-insertA];
                    MMMJobM=[MMMJobM(1:end) Chro2(end)];
                    %get the machine time
                    MFinishA(geneB,:)= FMMFinishA;
                    MStartA(geneB,:)=SMMStartA;
                    MJobM(geneB,:)=MMMJobM;
                    %get the job time
                    FpcopyM(geneC,remainC)=Makespan;
                    SpcopyM(geneC,remainC)=(Makespan-p(geneC,remainC));
                    MMJob(geneC,remainC)=geneB;

            %check the possibilities fot the 2nd job or above for machine
            else
                movC=0;
               while  (MStartA(geneB,i-movC)-MFinishA(geneB,i-movC-1))<p(geneC,remainC) && movC<(i-2)
                   movC=movC+1;
               end

               if (MStartA(geneB,i-movC)-MFinishA(geneB,i-movC-1))>=p(geneC,remainC)

                    FMMFinishA=MFinishA(geneB,:);
                    SMMStartA=MStartA(geneB,:);
                    MMMJobM=MJobM(geneB,:);

                    insertA=MStartA(geneB,i-movC)-p(geneC,remainC) ;

                    FMMFinishA(1)=[];
                    SMMStartA(1)=[];
                    MMMJobM(1)=[];


                    FMMFinishA=[FMMFinishA(1:i-movC-2) insertA+p(geneC,remainC)  FMMFinishA((i-movC-
1):end)];
                    SMMStartA=[SMMStartA(1:i-movC-2) insertA SMMStartA((i-movC-1):end)];
                    MMMJobM=[MMMJobM(1:i-movC-2) Chro2(end) MMMJobM((i-movC-1):end)];



                    MFinishA(geneB,:)= FMMFinishA;
                    MStartA(geneB,:)=SMMStartA;
                    MJobM(geneB,:)=MMMJobM;

                    FpcopyM(geneC,remainC)=insertA+p(geneC,remainC);
                    SpcopyM(geneC,remainC)=insertA;
                    MMJob(geneC,remainC)=geneB;

               else
                   if remainB==i
                    MFinishA(geneB,remainB)=Makespan;
                    MStartA(geneB,remainB)=Makespan-p(geneC,remainC);

                    FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                    SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                    MMJob(geneC,remainC)=geneB;

                    MJobM(geneB,remainB)=Chro2(end);

                   else
                    MFinishA(geneB,remainB)=MStartA(geneB,remainB+1);
                    MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                    FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                    SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                    MMJob(geneC,remainC)=geneB;

                    MJobM(geneB,remainB)=Chro2(end);

                   end

               end
            end
        %if the job operation is 2 or above
        else
```

```
            movC=0;
               while  (SpcopyM(geneC,remainC+1)-(MFinishA(geneB,i-movC-1))<p(geneC,remainC)
||(MStartA(geneB,i-movC)-MFinishA(geneB,i-movC-1))<p(geneC,remainC)) && movC<(i-2)
                   movC=movC+1;
               end
           if (SpcopyM(geneC,remainC+1)-MFinishA(geneB,i-movC-1))>=p(geneC,remainC) &&
(MStartA(geneB,i-movC)-MFinishA(geneB,i-movC-1))>=p(geneC,remainC)
                    if SpcopyM(geneC,remainC+1)>=MStartA(geneB,i-movC)
                        FMMFinishA=MFinishA(geneB,:);
                        SMMStartA=MStartA(geneB,:);
                        MMMJobM=MJobM(geneB,:);

                        insertA=MStartA(geneB,i-movC)-p(geneC,remainC) ;

                        FMMFinishA(1)=[];
                        SMMStartA(1)=[];
                        MMMJobM(1)=[];

                        FMMFinishA=[FMMFinishA(1:i-movC-2) insertA+p(geneC,remainC)  FMMFinishA((i-
movC-1):end)];
                        SMMStartA=[SMMStartA(1:i-movC-2) insertA SMMStartA((i-movC-1):end)];
                        MMMJobM=[MMMJobM(1:i-movC-2) Chro2(end) MMMJobM((i-movC-1):end)];



                        MFinishA(geneB,:)= FMMFinishA;
                        MStartA(geneB,:)=SMMStartA;
                        MJobM(geneB,:)=MMMJobM;

                        FpcopyM(geneC,remainC)=insertA+p(geneC,remainC);
                        SpcopyM(geneC,remainC)=insertA;
                        MMJob(geneC,remainC)=geneB;


                    else
                        FMMFinishA=MFinishA(geneB,:);
                        SMMStartA=MStartA(geneB,:);
                        MMMJobM=MJobM(geneB,:);

                        insertA=SpcopyM(geneC,remainC+1)-p(geneC,remainC) ;

                        FMMFinishA(1)=[];
                        SMMStartA(1)=[];
                        MMMJobM(1)=[];

                        FMMFinishA=[FMMFinishA(1:i-movC-2) insertA+p(geneC,remainC)  FMMFinishA((i-
movC-1):end)];
                        SMMStartA=[SMMStartA(1:i-movC-2) insertA SMMStartA((i-movC-1):end)];
                        MMMJobM=[MMMJobM(1:i-movC-2) Chro2(end) MMMJobM((i-movC-1):end)];



                        MFinishA(geneB,:)= FMMFinishA;
                        MStartA(geneB,:)=SMMStartA;
                        MJobM(geneB,:)=MMMJobM;

                        FpcopyM(geneC,remainC)=insertA+p(geneC,remainC);
                        SpcopyM(geneC,remainC)=insertA;
                        MMJob(geneC,remainC)=geneB;
                    end
           else
                   if remainB==i
                    MFinishA(geneB,remainB)=SpcopyM(geneC,remainC+1);
                    MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                    FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                    SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                    MMJob(geneC,remainC)=geneB;

                    MJobM(geneB,remainB)=Chro2(end);

                   else
                       if MStartA(geneB,remainB+1)<=SpcopyM(geneC,remainC+1)

                    MFinishA(geneB,remainB)=MStartA(geneB,remainB+1);
                    MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                    FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                    SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                    MMJob(geneC,remainC)=geneB;

                    MJobM(geneB,remainB)=Chro2(end);


                       else
                    MFinishA(geneB,remainB)=SpcopyM(geneC,remainC+1);
                    MStartA(geneB,remainB)=MFinishA(geneB,remainB)-p(geneC,remainC);

                    FpcopyM(geneC,remainC)=MFinishA(geneB,remainB);
                    SpcopyM(geneC,remainC)=MStartA(geneB,remainB);
                    MMJob(geneC,remainC)=geneB;
```

```
                    MJobM(geneB,remainB)=Chro2(end);


                        end
                    end
                end
            end
        end
        NChroM(end)=[];
        Chro2(end)=[];
    end
% Generated new chromosome
[ChroMMM]=timearr_01(MStartA,MJobM);
```