Genetic Algorithm Based Parts Scheduling in a Three-Machine Robotic Cell

BY

NOOTNAPANG RAKTAMAKIT

A thesis submitted to the Faculty of Graduate studies
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Department of Mechanical and Manufacturing Engineering
University of Manitoba
Winnipeg, Manitoba

Genetic Algorithm Based Parts Scheduling in a Three-Machine Robotic Cell

BY

Nootnapang Raktamakit

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

MASTER OF SCIENCE

Acknowledgments

I would like to express my deep and sincere gratitude to my advisor, Professor S. Balakrishnan, Ph.D. for his encouragement, advice, and guidance. Throughout my thesis-writing period, he painstakingly reviewed this work and gave assistance and instruction. His vast knowledge and logical way of thinking has been of immense value to me. Without his knowledge, ideas, and patience, this work could not have been completed.

I would like to thank Dr. Q. Peng, who provided inspiring advices and encouragement. His comments and guidance helped me perfect this work.

I wish to thank my entire family for their support; my mother for her unconditional love and being there whenever I needed, my father for his encouragement, my brother and my sister for their understanding and help getting me through the difficult times, and my son and my daughter for their love and laughter.

Lastly, and most importantly, I owe my loving thanks to my husband, Songraj Raktamakit who's support me in every possible way, helped me to accomplish this work. Without his love, support, encouragement and understanding this endeavor would have been impossible. To him I dedicate this thesis.

Abstract

This thesis proposes a genetic algorithm for finding part schedule that produces the best makespan in a three machine flexible manufacturing cell. The type of production operation considered is 'different part type' problem. Key constraints considered in the analysis are: (i) each part has a specific order of sequencing through machines which cannot be changed, and each part may not pass through every machine, and (ii) processing times of parts on machines they visit need not be the same. A pair grouping and priority procedure is developed to generate the makespan of part families, without violating any of the constraints imposed. Influence of different genetic parameters on the resulting makespan are studied and compared using three different test data types. Results from the studies indicate that good makespan values are obtained for all types of crossover when (a) the crossover probability take the values ranging from 0.3 to 0.5, and (b) an 'Arbitrary two-job exchange mutation type' is selected with probability values ranging from 0.4 to 1.0. Good results are obtained when a population size of 20 to 35 was selected. The number of iterations required for convergence of solution was found to be the least for the above chosen values. Other choices were found to require greater iterations for convergence.

Table of Contents

List of Figures

Figure **Page**

List of Tables

Table **Page**

Nomenclature

C	Chromosome
Cp	Cumulative probability
FV	Fitness value of chromosome
G	Group of pairing part in chromosome
m	Number of machine that part will be processed
n	Number of parts
p	Selection probability
P	Part name
P_c	Crossover probability
P_m	Mutation probability
r	Random number
R	Random number for mutation operation
S	Random number for crossover operation
S_1	The first machine that part will be processed
S_2	The next machine that part will be processed after machine S_1
S_3	The next machine that part will be processed after machine S_2
T	Makespan of individual part
TC	Makespan of chromosome
$TC_{\min}$	Minimum makespan of the chromosome
TG	Makespan of group in chromosome
$W_{(S_1)}$	Processing time of part on S_1

$W_{(S_2)}$	Processing time of part on S_2
$W_{(S_3)}$	Processing time of part on S_3

Chapter 1 Introduction

1.1 Background

The way operations are carried out in manufacturing system has changed with time. In early days the tasks in manufacturing depended mainly on operator to manually load, unload and operate machines. Later on, with the intense market competition as well as progress in intelligent machines and computer control, the need for human interactions has reduced. Since operator's salaries tend to increase while the cost of intelligent machines and automatic control system has decreased, more and more tasks have been automated [1]. Nowadays, with the considerably shortened product life cycles and more demand of customized products, manufacturers have discovered that they can no longer capture market share and gain higher profit by producing large volumes of standard product for a mass market. Success in manufacturing in next decade requires the processes with ability to flexibly and rapidly respond to the change of market conditions [2]. In order to meet the increased demand for customized products and to reduce production lot sizes, the industry has adapted new techniques and production concepts by introducing flexibility into the production system so that a variety of products can be manufactured on the same equipment. Advancements in the area of flexible manufacturing systems (FMS) have paved the way for gaining competitive edge to many manufacturing enterprises.

A general FMS is defined as an automated system composed of computer-controlled machining centers and automated material-handling devices. Parts are produced by computer-controlled machining centers and are transferred between machines/processes by automated material-handlings. A traditional FMS is large and very complex,

consisting of dozens of multi-purpose machines with sophisticated material handling and controlled by incredibly complex software systems. Since they are very complex and very expensive, only a limited number of industries can afford the investment. Currently, the trend in FMS is toward small versions of the traditional FMS, called flexible manufacturing cell (FMC). They are less expensive, low risk and also satisfy many of the benefits gained through FMS. It is the most feasible approach for automating the job shop manufacturing [4]. A FMC typically consists of internal and external material handling systems, and one or more flexible machines [4].

In a FMC, the industrial robots are widely used for internal material handling. The machines in a robotic cell are generally computer numerical controlled machines (CNCs) and usually equipped with necessary cutting tools capable of performing different processes including milling, drilling, grinding, tapping and cutting with minimum set-up time between successive operations. As a result, the robotic cell is capable of performing different operations on a variety of parts. The layout of machines in a robotic cell can structured in many different ways and depends on the type of robot. A commonly used cell layout is called robot-centered cell. It consists of m machines which are placed in a circular or semicircular layout with a handling robot at the center. The machines have to be laid out to be within the reach of a robot so that the robot can perform all the material handling functions. The material handling functions include loading, unloading and transferring parts between machines.

The performance of a robotic cell depends on several operational and system characteristics which include part scheduling, robot and machine characteristics. Since operations in a robotic cell are automated, effective part scheduling and efficient

sequences of robot moves is crucial. The part characteristics such as their sequence through machines and processing times on individual machines are the key factors to be considered in arriving at the part schedule. If the parts in a part family have identical characteristics, they are referred to as an 'identical part type'. Parts in a part family having different sequences and machine processing times, will be referred to as a 'different part type' in this thesis. To determine an efficient part scheduling methodology in a robotic cell especially in a different part type problem is an important and challenging task. This research focuses on efficient part scheduling in a three-machine robotic cell for the different part type problem.

1.2 Objective

The research focuses on the development of a novel genetic algorithm for the 'different part type' problem in a three-machine robotic cell. The objective of the thesis is to find the best part schedule that will produce the minimum makespan for the problem considered. The solution will consider the two key constraints, namely: (i) each part has a specific order of sequencing through machines which cannot be changed, and each part may not pass through every machine, and (ii) processing times of parts on machines they visit need not be the same. A genetic algorithm will be proposed. A pair grouping and priority procedure will be developed to generate the makespans of part families, without violating any of the constraints imposed. Different genetic parameters will be studied and compared using three different test data types. Results from the studies will be discussed.

1.3 Thesis Structure

This thesis is organized as follows. A literature review is presented in Chapter 2. The problem formulation is introduced in Chapter 3. The methodology of part scheduling using genetic algorithm in a three-machine robotic cell is developed in Chapter 4. The results and evaluation of genetic algorithm are presented in Chapter 5. Lastly, conclusions and recommendations are described in Chapter 6.

Chapter 2 Literature Review

2.1 Robotic Cells

Decreases in economic growth rate and the increases in dependence upon international markets are two major factors that have created incentives to improve manufacturing productivities [5]. Increasing the application of automation in manufacturing is one way towards improvement. As a result, use of industrial robots for automation is also on the increase. Robots have been installed in order to increase output, to reduce cost, to provide more flexibility in manufacturing operations, and to replace people working in dangerous and hazardous conditions [6]. Their task performances are versatile from processing operation, assembly, testing to inspection. One of the important applications is material handling especially in robotic cell. Robotic cells consist of one or more machines, input/output device, and a robot for loading, unloading and material handling between machines in the cell.

There have been several studies pertaining to the scheduling of robotic cells [3-9]. Since the number of machines in a robotic cell is usually no more than four [7], most of the researches concentrated on two and three machine robotic cell. Scheduling problems in a two-machine robotic cell for identical parts have been investigation to provide greater operational flexibility [6]. It has been proved that that there is exactly $m!$ potential optimal one-part cycles in an m machine cell [8]. Algorithms for scheduling a robotic cell that produces a set of part types on several machines served by a single robot have also been reported [7]. Studies related to the scheduling of operations in a manufacturing cell that repetitively produces a family of similar part on two and three machines robotic cell have also been investigated [5]. A similar study focused on

finding the optimal one-unit cycle in three-machine no-wait robotic cell when parts are classified as "different part type" [9].

In some manufacturing operations, the order of sequencing through machines is very crucial. For example, if we want a cylindrical tube of a specific size with two tapped holes at both ends, the raw material has to be processed in the following order. First, the raw material has to be turned into the specific size, then, it has to be drilled at both ends, and lastly the two holes at the ends have to be tapped. The above example illustrates the importance of sequencing. The short comings of various studies cited are summarized at the concluding section of this chapter.

2.2 Genetic Algorithms

Genetic algorithms are global search algorithms and follow the principles of evolution in nature. It is based on the Darwinian principle of 'survival of the fittest'. Genetic algorithms use probabilistic transition approach for searching from a given population. A basic genetic algorithm is composed of two genetic operators; crossover, and mutation with the goal of finding the best solution to a problem. Before a genetic algorithm can be put to work on any problem, it is necessary to encode potential solutions to that problem in a form that a computer can process. The different coding strategies or genetic description lead to different results [10]. In fact, genetic algorithms are a searching process. The coding strategies play an important rule in contributing to the searching efficiency. If coding strategies are not selected appropriately, the searching space will be too large to find the optimum solution. In contrast, if the codes are created suitably, the searching range can be narrowed and the optimum solution can

be found rapidly. In [10-13] different coding techniques have been proposed in order to satisfy and improve problem solving. After encoding, the chromosomes are evaluated and selected by using what is known as the 'fitness value'. The promising chromosomes which have the better fitness value are chosen to perform genetic operators which are crossover and mutation. Genetic operators especially crossover are regarded as the main contributing factor in the performance of the genetic algorithms. Therefore, many new genetic operators have been studied and adapted to accommodate the coding strategies. A number of genetic operators have been proposed for flow shop scheduling problem [14]. New offsprings are created from an initial population. To keep the population size at a certain level, only the chromosomes that generate better fitness values are selected to be the new generation. The selection operator and genetic operations are repeated until a satisfactory solution is obtained or a specified termination criterion is met.

Genetic algorithms (GA) are proving to be popular and have been found to be effective in solving problems. Even a simple GA appears to be robust and the complexity of algorithms and the result are irrelevant to the length of genetic string and the original state of population [10]. Genetic algorithm is also simple that it only involves some genetic operations. It is so efficient that it can find a near optimum solution even for a large scale problem. Nowadays, genetic algorithms are one of the most widely studied searching techniques in manufacturing scheduling problem. Generally, job shop scheduling problem (JSSP) and flow shop scheduling problem are known to be a typical NP-complete or NP-hard problem meaning that no method is known to find a guaranteed-optimal solution in a reasonable amount of time. The two main issues of using genetic algorithms in manufacturing scheduling problem that have received a lot

of attention are: (i) how to encode a solution of the problem into a chromosome in order to guarantee that a chromosome will correspond to a feasible solution [15]; and (ii) how to enhance the performance of genetic search by incorporating traditional heuristic methods [15].

Three methods based on genetic algorithms have been proposed to design a nearly optimum solution for job-shop scheduling problem, a typical NP-complete problem [10]. The three methods are decimal idle time coding genetic algorithms (DITCGA), binary idle time coding genetic algorithms (BITCGA) and adaptive idle time coding genetic algorithm (AITCGA). The differences between the first two methods were in the coding strategies. For DITCGA method, chromosomes were coded by several sub strings that contained data pertaining to parts and idle time that was processed by a machine. For BITCGA, the structure of the chromosome and the coding strategy were almost the same as DITCGA except the idle time coding used a binary number of certain bits instead of decimal. The purpose of binary coding is to create the fitter gene for crossover and mutation operation. For AITCGA method, the concept of self-adjusting the upper limit of idle time is used. Because genetic algorithms are search approaches, the proper upper limit idle time can adjust searching space and improve searching efficiency. As a result, DITCGA and BITCGA provided almost the same searching efficiency. AITCGA was more complicated. However, as the upper limit of idle time plays an important rule for the searching space, AITCGA which can adjust upper limit of idle time (enlarge searching space) had a greater possibility to find the optimum solution at the expense of needing a larger searching time. In conclusion, these

three methods can satisfy the rapid response requirement in scheduling if the upper limit of idle time is decided properly.

An adaptive genetic algorithm has been developed to solve the flow shop scheduling that deals with processing a set of jobs through a set of machines where all jobs have the same order of sequencing through machines [16]. One of the well-known drawbacks that can occur with a genetic algorithm is known as premature convergence. It can occur if the chromosomes with a higher fit than the others emerges early in the search. Under fitness-proportionate selection, they and their descendents will multiply quickly and it may lead the algorithm to converge on the local optimum that these chromosomes represent rather than searching the fitness landscape thoroughly enough to find the global optimum [17], [18]. To improve the performance, the adaptive probabilities of crossover and mutation were introduced. The probabilities of the genetic operations are dynamically adjusted according to the actual situation of the evolution process. If the fitness of the individual is higher, the probability of the crossover and mutation operation should be lower, and vise verse. As a result, the adaptive genetic algorithm increases the velocity of the convergence of the algorithm and provides better effectiveness and efficiency than the basic genetic algorithms.

The genetic algorithm for solving flow shop scheduling problem in a type of production line that had the following attributes has been proposed: (i) parallel machines (a set of machines that perform operations, simultaneously) and (ii) special procedure constraints such as some jobs can only be processed on special machines [19]. In this flow shop scheduling problems, there are n jobs scheduled to be processed, m sequenced operations in the production line, and all the jobs have the same flow through machines.

In order to handle special constraints, some techniques were introduced to the traditional genetic algorithms. For coding scheme, the chromosome was presented in term of a vector with $(m+1)n$ ordered elements. The vector represents two parts, a permutation of the serial number of n jobs and a matrix $n \times m$ which denotes a serial number of machine chosen to perform operation of job. Because the two parts of the chromosome have different format and meaning, two different crossover operators, mutation operators and their probability are applied for different parts. The comparisons between genetic algorithms and other algorithms such as shortest processing time (SPT), largest processing time (LPT), shortest remaining processing time (SRPT) and largest remaining processing time (LRPT) were also studied. As a result, genetic algorithms seemed to provide better results than others when measured for efficiency and computing time.

A genetic algorithm application to solve the flexible job shop scheduling problem has been studied [11]. In a classical $n \times m$ job shop scheduling problem (JSSP), there are n jobs and m machines. Each job must be processed on every machine by following the assigned operation sequences. In this study, the typical JSSP are extended. Each job may not have to be processed on every machine and the uncompleted jobs may be able to pass though the same machine. For chromosome coding, each chromosome is made up of a chain of operation template. The operation template can be defined as the mapping between a set of natural numbers and operations which are job number, operation number, processing machine and processing time. Because the operation in each job has precedence constraints, not all the coded chromosome can represent the feasible schedules directly. A decoding procedure called virtual job shop is developed

to decode each chromosome to a feasible schedule. The operating time for each operation is calculated by Gantt chart. In summary, the utilization of the operation template and virtual job shop was shown to solve the flexible job shop scheduling problems easily.

A genetic algorithm to solve the multiobjective optimization of the flexile job shop scheduling problem has been proposed [12]. The flexible job shop scheduling problem (FSJP) is a problem of planning and organizing a set of tasks that have to be processed on a set of resources with variable performances. Two main difficulties associated with this problem are: (i) the assignment of each operation to the suitable machine, and (ii) the calculation of the starting time of each operation. In this study, an Approach by Localization (AL) and a scheduling algorithm have been proposed to assign a set of feasible solution before applying genetic algorithm. For chromosome coding, Tasks Sequencing List (T.S.L) is presented. Each chromosome is presented in the form of cell list which represent the sequencing of tasks on the machines. For the genetic operators, the Pox operator (Precedence Preserving order based crossover) and the PPS (precedence Preserving Shift mutation) are introduced. Moreover, the assignment operators which are the operators that deal with the only assignment properties of the individuals were also applied. In summary, the solution from the genetic algorithm was found to be generally satisfactory and promising.

The various genetic operators for flow shop scheduling problem have been studied [14]. In crossover operators, ten crossover operations (which are one-point crossover, two-point crossover version I, two-point crossover version II, two-point crossover version III, position based crossover version I, position based crossover version II, edge

recombination crossover, enhanced edge recombination crossover, and partially matched crossover and cycle crossover) were compared by using computer simulation using test problems and user selectable termination conditions. The two-point crossover version I provided the best result. Similarly, four mutation operations (which are adjacent two-job change, arbitrary two-job change, arbitrary three-job change and shift change) were compared. In this case, the shift change mutation operation provided the best result. However, to obtain the best performance for the problem, the type of genetic operators as well as the genetic operation probabilities had to be considered [16]. In addition, the genetic algorithm has been compared with other search algorithms such as local search, taboo search, simulated annealing and random sampling technique. The results show that genetic algorithm provided a much superior performance when compared to a random sampling technique. However, the performance was a bit inferior when compared to other search algorithms.

A genetic-based algorithm to solve the cell formation problem and batch scheduling problem in Cellular Manufacturing (CM) has been developed [13]. The cell formation problem is the problem of grouping machines into different cells on certain standard. The batch scheduling problem is the problem of scheduling a number of n-jobs through a single machine which are capable of handling only a number of job at one-time. In batch scheduling problem, coding strategy is to divide the chromosome into two parts. Each chromosome consisted of Part A and Part B. The values of genes in Part A represent the number of job to be scheduled and the number of genes represents the number of batches. Part B represents the actual jobs to be processed. The number of jobs to be allocated in each batch depends on Part A. For Part A, two types of

operations were introduced which are: (i) a unary swapping operator, and (ii) a string swapping operator. Similarly, two type of operation are used for Part B which are the order-based operator and the position-based operator. This work concluded that genetic algorithms can solve batch scheduling problem. It can also provide sequence of jobs to be processed as well as determine the number of jobs to be processed in each batch. For genetic operators, the position-based operators provided the best result.

A genetic algorithm-based parts scheduling in a two-machine and three-machine robotic cell have been studied [3], [20]. The parts to be scheduled belong to the different part types. It was assumed that there are no intermediate buffers between machines. For encoding scheme, the chromosome consisted of a string of number which represents part scheduling. Since the problem chosen consisted of different part types, there are two alternative sequences of robot moving in a two-machine robotic cell and six alternative sequences of robot moving in a three-machine robotic cell. For this reason, these two and six alternative cycle times are used as a fitness function for two-machine and three-machine robotic cell respectively. For genetic operators, one-point cut crossover and arbitrary two-part exchange mutation are applied. In conclusion, the genetic algorithms provided a practical and effective solution. However, to prevent local optimum, converging speed of the algorithm had to be controlled. Different scheme of genetic operator affects converging speed of the algorithm. This aspect of GA's application to sequencing and scheduling problem will be fully explored in this thesis.

2.3 Summary

During the past decade, genetic algorithms have been employed to solve manufacturing schedule problem. A number of researchers dealing with manufacturing schedule problem using genetic algorithms have been reported in literatures. Manufacturing schedule problems including flow shop scheduling problem and job shop scheduling problem are NP-hard combinatorial optimization problems. Normally, these problems have to deal with multi-objective optimization problems with complex constraints. Genetic algorithms seem to be suitable for solving such problems because the first and most important point of genetic algorithms is its ability to explore the solution space in multiple directions. Since genetic algorithms have multiple offsprings, many different routes can be explored at the same time and if one path turns out to be a dead end, they can easily eliminate it and continue work on the more promising path, giving them a better change each run to find the optimal solution. Most of the other algorithms are serial, and can explore the solution space in one direction at a time, and if the solution turns out to be suboptimal, all the previous works may have to be abandoned and started all over. Moreover genetic algorithms are particularly well-suited to solving problems where the space of all potential solutions is truly huge - too vast to search exhaustively in any reasonable amount of time [21]. However, no matter how much strength genetic algorithms possessed the key to obtaining an efficient solution depends on the efficient implementation of genetic algorithms. Two main issues namely: how to encode a solution in the chromosome, and how to enhance the performance of genetics search always have to be considered. From a review of various researchers cited above, with appropriate modification to the standard genetic algorithms, it can be concluded that

they have the potential to produce efficient results for many complex problems in manufacturing schedule problems.

From a review of literature it can be concluded that genetic algorithms to solve batch scheduling problem with a number of jobs though a single machine, and with a single part type have been developed. Genetic algorithms to solve 'different part type' have been the focus only in a very limited number of works. The order of sequencing through machines of each part in 'different part type' has not been developed to an acceptable level due to the complexity of encoding strategies and fitness calculation. As mentioned in section 2.1, there are cases when the orders of the operations are very crucial. Only two researchers [3], [20] have considered material handling between machines and developed genetic algorithms for a two-machine and three-machine robotic cell.

This research will develop genetic algorithms for a three-machine circular robotic cell to deal with scheduling 'different part type'. It is a NP-hard problem and will consider part scheduling as well as robot scheduling for part handling. The problem has two main issues: (i) each part has a specific order of sequencing through machines which cannot be changed, and each part may not pass through every machine, and (ii) processing times of parts on machines they visit need not be the same. The objective is to find the best part schedule which provides the minimum makespan. A pair grouping and priority procedure will be employed to formulate the makespan in order to maintain the order of sequencing through machines of a part. Further detail will be provided in the following chapters.

Chapter 3 Problem Formulation

This chapter will present the details of problem formulation for part scheduling in a three-machine robotic cell. It includes a description of the three-machine robotic cell considered, data structure, part scheduling, part grouping and makespan calculation by using time chart and the priority method.

3.1 Three-Machine Robotic Cell

The three-machine robotic cell consists of three machines: machine 1 (M1), machine 2 (M2) and machine 3 (M3), an input station *I* (M0), an output station *O* (M4), and a central robot as shown in Figure 3.1. The machines and the stations are assumed to be located in a semi-circle with the robot located at the centre. The exact location of the machines is not that critical and variations in spacing can be easily incorporated. The robot transfers part between machines as well as from input and output stations. At the completion of all processing requirements, the parts are deposited in the output station.

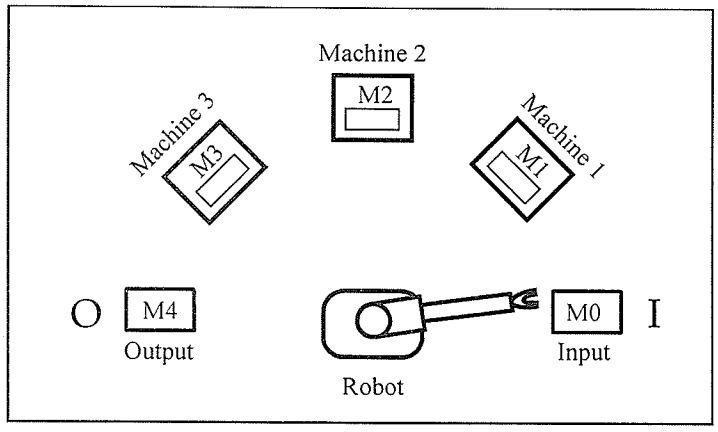


Figure 3.1: The layout of three-machine robotic cell

The robot is assumed to be equipped with a one hand gripper and can handle one part at a time. There are no intermediate storage buffers in front of machines. Therefore, any part in the cell is always either in one of the machines or being handled by the robot. The machines can process one part at a time and the parts in the machine have to be unloaded before a new part is loaded. The pick-up, loading, unloading, drop-off and transportation of part between machines or stations are carried out by the robot. The robot travel time between any of pair of adjacent stations is equal as a result of distances between machines being assumed equal. However, it can be changed, if need to be. The notations that represent robot movements of a robotic cell in this work are shown below.

pick = Time used by robot to pick up or unload part.

drop = Time used by robot to drop off or load part.

move = Movement time between any pair of adjacent location.

It is assumed that the setup time is included in the processing time of each part as it moves through different machines.

3.2 Data Structure

There are n parts $P_1, P_2 \ldots, P_n$ to be processed through the machines. Each part has a specific order of sequencing through machines and this cannot be changed. Hence routing of a part through the cell is critical. The parts may not need to pass though every machine and the processing time of parts on each machine may be different. Every part starts at the input station I (M0) and finishes at the output station O (M4). At the beginning of a production, all the machines are assumed to be idle with no parts in them and the robot is at the input station I (M0) ready to pick up a part.

A part schedule is represented using the notation $P_{\sigma(1)}, P_{\sigma(2)} \ldots, P_{\sigma(n)}$. In other words, $P_{\sigma(1)}, P_{\sigma(2)} \ldots, P_{\sigma(n)}$ is a schedule for a given permutation σ of n parts. The information of each part can be represented to contain the order of sequence through machines and the corresponding processing time. Any part information $(P_{\sigma(i)})$ will be presented in a standard form shown below.

$P_{\sigma(i)} = S_{1(\sigma_i)}(W_{(S_{1(\sigma_i)})}) \rightarrow S_{2(\sigma_i)}(W_{(S_{2(\sigma_i)})}) \rightarrow S_{3(\sigma_i)}(W_{(S_{3(\sigma_i)})})$, where

$S_{1(\sigma_i)}$ = The first machine that part $P_{\sigma(i)}$ will be processed.

$S_{2(\sigma_i)}$ = The next machine that part $P_{\sigma(i)}$ will be processed after machine $S_{1(\sigma_i)}$.

$S_{3(\sigma_i)}$ = The next machine that part $P_{\sigma(i)}$ will be processed after machine $S_{2(\sigma_i)}$.

$W_{(S_{1(\sigma_i)})}$ = Processing time of part $P_{\sigma(i)}$ on $S_{1(\sigma_i)}$.

$W_{(S_{2(\sigma_i)})}$ = Processing time of part $P_{\sigma(i)}$ on $S_{2(\sigma_i)}$.

$W_{(S_{3(\sigma_i)})}$ = Processing time of part $P_{\sigma(i)}$ on $S_{3(\sigma_i)}$.

$m_{(\sigma_i)}$ = Number of machine that part $P_{\sigma(i)}$ will be processed. $m_{(\sigma_i)} = \{1,2,3\}$

$T_{(\sigma_i)}$ = Makespan of individual part $P_{\sigma(i)}$.

$T_{(\sigma_1)+(\sigma_2)\ldots+(\sigma_n)}$ = Makespan of part schedule $P_{\sigma(1)}, P_{\sigma(2)} \ldots, P_{\sigma(n)}$.

In the event when $m_{(\sigma_i)}$ is equal to 2, there will be no $S_{3(\sigma_i)}$ and $W_{(S_{3(\sigma_i)})}$ in the equation representing part $P_{\sigma(i)}$. When $m_{(\sigma_i)}$ is equivalent to 1, there will be no $S_{3(\sigma_i)}$, $S_{2(\sigma_i)}$, $W_{(S_{3(\sigma_i)})}$ and $W_{(S_{2(\sigma_i)})}$ in the equation representing part $P_{\sigma(i)}$.

For example if part P_A has the following data, $M2(5) \rightarrow M3(5) \rightarrow M1(6)$, it means that the part starts at M2, upon finishing at M2 moves to M3 and then to M1. The processing times on each of those stations are as shown within brackets. The sequence of robot moves for this part is shown in Figure 3.2.

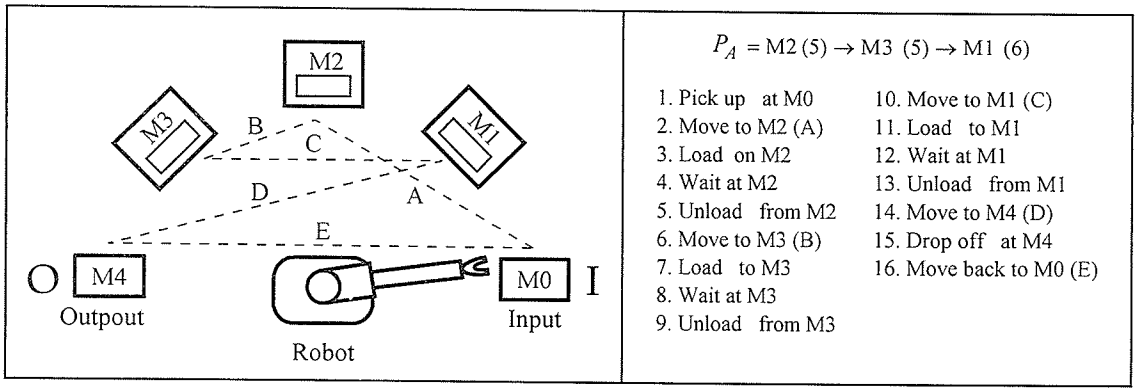


Figure 3.2: The sequence of robot moves for part P_A

To illustrate, consider an example where a part family F1 consists of seven parts: P_1, P_2, P_3, P_4, P_5, P_6, and P_7 to be processed through the three machines. For the example considered, the part family F1 has the following part information.

P1 = M1 (11) → M2 (13) → M3 (9)

P2 = M2 (8) → M1 (10)

P3 = M3 (14)→ M1 (12)

P4 = M2 (11)→ M3 (7)

P5 = M2 (10)

P6 = M3 (15)→ M1 (13) → M2 (11)

P7 = M1 (7) → M2 (9)

The information provided above is also shown in a tabular form in Table 3.1.

Table 3.1: Part family F1

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	11	M2	13	M3	9
P2	M2	8	M1	10	None	None
P3	M3	14	M1	12	None	None
P4	M2	11	M3	7	None	None
P5	M2	10	None	None	None	None
P6	M3	15	M1	13	M2	11
P7	M1	7	M2	9	None	None

To find the part schedule that provides the minimum makespan for the part family consisting of many parts in F1, is very complicated. One part family can generate many part schedules and there are many different options to calculate the makespan. Moreover the important constraint that the order of sequence through machines of every part cannot be changed should also be considered. In this work, a pair grouping and priority method is applied to assure that makespan calculation will meet the constraint considered. Section 3.3 will provide more details about the part scheduling and the grouping and section 3.4 will present the priority method and time chart for makespan calculation.

3.3 Part Scheduling / Grouping

As mentioned in Section 3.2, any part schedule will be represented as $P_{\sigma(1)}, P_{\sigma(2)} \ldots, P_{\sigma(n)}$. If there are n parts P_1, $P_2 \ldots, P_n$ to be processed through the three-machine cell, the parts can be arranged into many part schedule groups. Specifically, the number of part schedules that can be generated is equivalent to $n!$, where n is the number of parts to be scheduled. For example, if there are three parts: P_1, P_2 and P_3, to be scheduled, parts can be arranged into six (3!) part schedules which are $\{P_1, P_2, P_3\}$, $\{P_1, P_3, P_2\}$, $\{P_2, P_1, P_3\}$, $\{P_2, P_3, P_1\}$, $\{P_3, P_1, P_2\}$, and $\{P_3, P_2, P_1\}$. However, in each part schedule, there are many different options to calculate the makespan. For example, to calculate the makespan of part schedule P_1, P_2, P_3, there are four options to calculate makespan. For the first option, it can be formulated by calculating the individual makespan of P_1, P_2 and P_3, and then summing all three individual makespans together (makespan of option 1 $= T_{P_1} + T_{P_2} + T_{P_3}$). For the second option, it can be formulated by calculating makespan of P_1 and P_2 together and then summing this makespan with the individual makespan of P_3, (makespan of option 2 $= T_{P_1+P_2} + T_{P_3}$). For the third option, it can be formulated by calculating makespan of P_2 and P_3 together and then summing this makespan with the individual makespan of P_1, (makespan of option 3$= T_{P_2+P_3} + T_{P_1}$). For the last option, it can be formulated by calculating makespan of P_1, P_2 and P_3 all together at once, (makespan of option 4 $= T_{P_1+P_2+P_3}$). Each option may generate different makespans. This means that there can be many makespans generated from one part schedule. The difference in each option is

the way groupings are formed for the part. In other words, the different grouping of the parts in the calculation will provide different results for the makespan. As the number of parts in the part schedule increase, more grouping option will be generated. The fundamental issue that needs to be looked at is to determine which grouping option is the best to use in calculating the makespan. In fact, the makespan that will represent the makespan of the part schedule should be the minimum makespan that is selected from every grouping option generated from the part schedule. However, it is difficult to calculate all the makespan from every group option, especially in the case when part schedule contains a number of parts.

One of the grouping options is a pair grouping. In a pair grouping, every part in the part schedule will be paired into groups. For example, if part schedule A is $P_1, P_2, P_3, P_4, P_5, P_6$, by using a pair grouping, the parts can be grouped into three groups: $\{P_1, P_2\}$, $\{P_3, P_4\}$, and $\{P_5, P_6\}$. The pair grouping method is explored further in this work. A pair is defined as the smallest group that any part schedule can be grouped into (excluding one). In other words, every part schedule can be divided into groups of pair, no matter how many parts are contained in the part schedule. A computer based methodology was developed to calculate the makespan of any pair grouping. The program was written using "Java" as the programming language. The choice of programming language was based on the familiarity of the author with "Java". Section 4.2.3 will provide further details of the methodology for a pair grouping.

3.4 Priority Method / Time Chart

To calculate the makespan of an individual part is quite simple. A time chart is used as a tool to determine the makespan. The java language is used to create the graphical display and the code for the time chart. The time chart consists of the time table of machine 1 (M1), machine 2 (M2), machine 3 (M3) and robot (R). Since the order of sequencing of a part through machines cannot be changed, the makespan can be determined by plotting the processing data of parts as a time chart by following the given order of sequence through machines. The processing data include the processing time of part in the machines and the time of robot activities (loading time, unloading time, pick up time, drop off time, and moving time between machines) that robot uses in order to complete the process through a set of machines for a part. The total time utilized to complete the part in the time chart is the makespan. For example, the data of part P_A from section 3.2 has the sequence, $\mathrm{M2}(5) \rightarrow \mathrm{M3}(5) \rightarrow \mathrm{M1}(6)$. Figure 3.3 shows the time chart of part P_A. From the time chart, the makespan of part P_A is equivalent to 26 units.

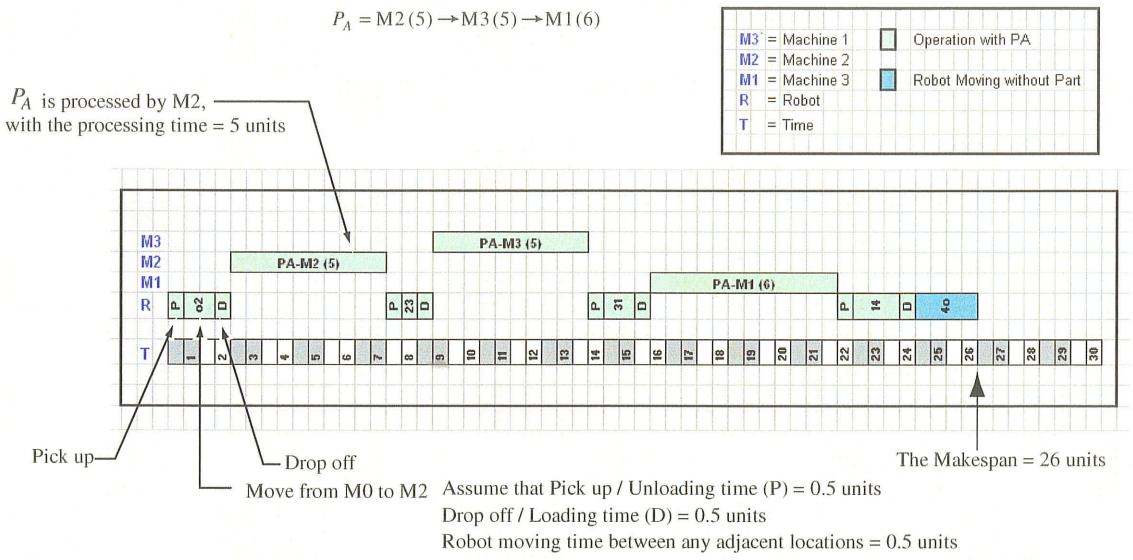


Figure 3.3: Time chart of part P_A

To calculate the makespan of a group of parts that contains more than one part is complicated. The makespan of the group can be obtained by utilizing the same method of calculating a makespan for an individual part, by plotting all of the processing data of every part in the group in the time chart. However, this does not mean that the makespan of the part schedule is obtained by accumulating the individual makespan of each part in the group. In that case, there will be a lot of wasted production time as a result of greater idle time from machines and robot. When a pair grouping method is used, the member of the group will be two. There are many ways to plot two parts into one time chart. For example, the least robot idle time method and the least machine idle time method are two options. Even though these two methods will tabulate the results considering the least idle time of machines or the least idle time of the robot, they do not consider the order of sequencing of the part through machines, an important constraint considered in this work.

A priority based method is introduced. The concept of the priority method is based on "first come, first served" principle. This is one of many other scheduling rules normally followed in scheduling studies. The parts will be attended to in the order that they arrive. It means that the first part of the pair has to be processed before the second part of the pair. The processing of the first part will have the first priority and any activities (such as robot movement or machine processing) arising from processing the second part cannot stop the processing of the first part. In other words, the part in the first order of the group will have the priority over the part that comes later. This assumption does not however imply that the first part has to be finished before the second part. Since the order of sequence of the part through machines cannot be changed, this method will maintain the order of the sequence through machines and the position of a part in the group.

The process starts by plotting the processing data of the first part in the time chart (this process is identical to the process of determining the makespan of an individual part). Then the processing data of the second part will be incorporated on the same time chart. Two processes are employed: (i) insertion and (ii) connection. The process starts with insertion. The processing data of the second part will be analyzed in order to determine the insertion position. The computer based algorithm will compute the time interval (idle times) in the first part's schedule. If the space is big enough to insert any processing data of the second part, that processing data will be inserted into the time chart. In the event when the processing data cannot be inserted into those idle spaces, the connection process will be used. For the connection process, the processing data will be input after the processing data of the first part. In other words, the processing

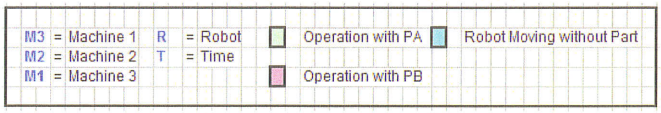
data of the second part will be inserted into the time chart in the ways that provide the shortest makespan of the pair without causing any interference to the first part. That means the first part will be the control part. The first part does not have to finish before the second part. The processing of the second part can finish ahead of the first part if the result will provide a shorter makespan. However the inserting data has to meet the condition that the order of machine processing sequence of any part cannot be changed. The total time to complete all parts in the time chart is the makespan. For example a pair grouping G1 consists of P_A (the first part of the pair) and P_B, and is composed of the following part information:

$P_A = M2(5) \rightarrow M3(5) \rightarrow M1(6)$; and

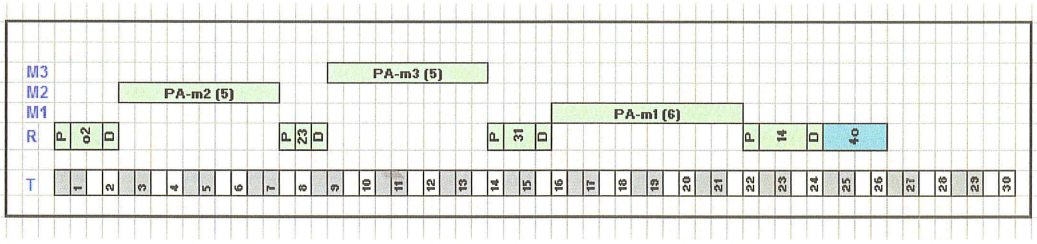
$P_B = M1(6) \rightarrow M2(4)$.

The process of makespan calculation from time chart of the pair grouping G1 is shown in Figure 3.4. The sequence of robot moves for the pair grouping G1 is shown in Figure 3.5.

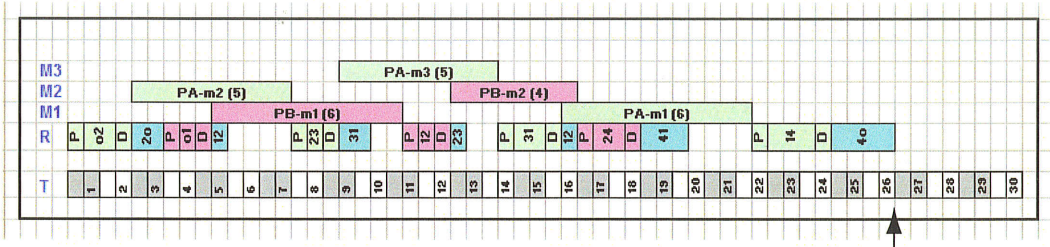
P_A = M2 (5) → M3 (5) → M1 (6)
P_B = M1 (6) → M2 (4)



1. Set the first part (P_A) as the main part.



2. Add the second part (P_B) to the chart without interference to the process of the first part (P_A)

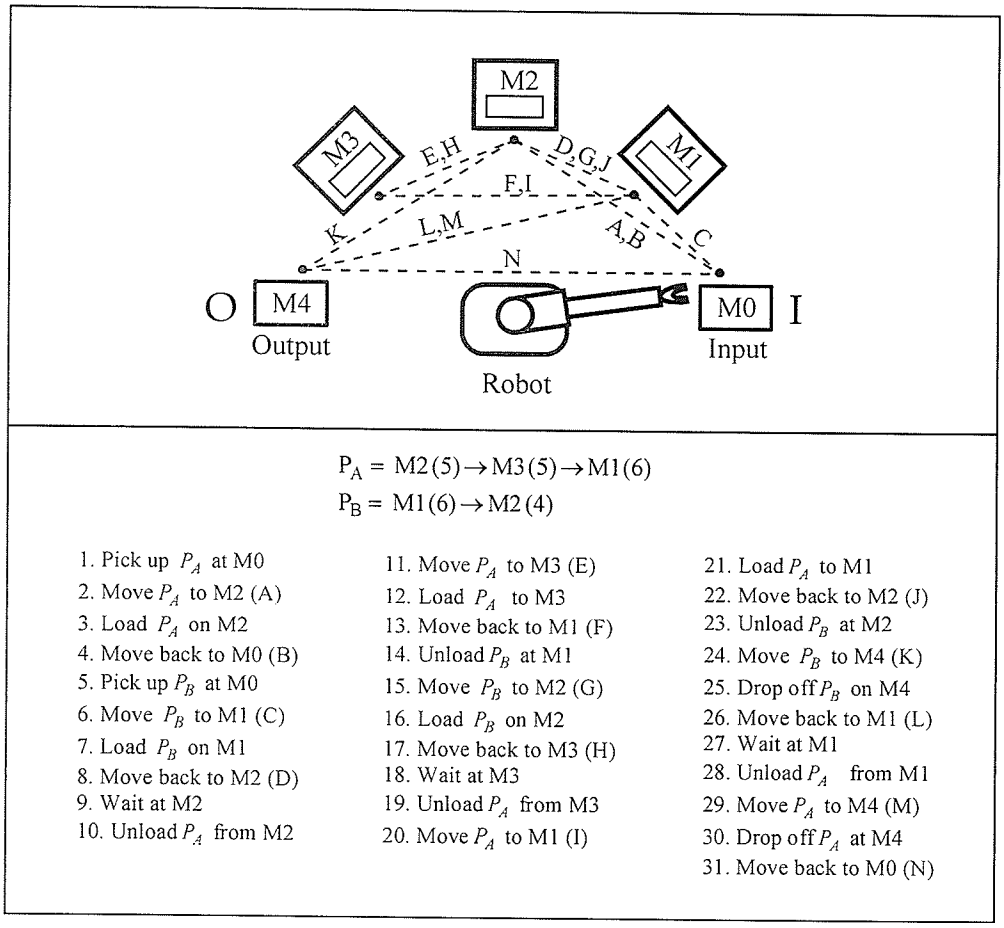


Assume that pick up/loading time (P) and drop off /unloading time (D) = 0.5 units
Robot movement time between any adjacent locations = 0.5 units.

Makespan of P_A - P_B (T_{A+B}) = 26 Units

Figure 3.4: Time chart of pair grouping G1

From Figure 3.4, the makespan of the group G1 is equivalent to 26 units which is the same as the individual makespan of part P_A. This means that processing data of part P_B can be completely inserted in the idle times of part P_A.



P_A = M2(5)→M3(5)→M1(6)
P_B = M1(6)→M2(4)

1. Pick up P_A at M0
2. Move P_A to M2 (A)
3. Load P_A on M2
4. Move back to M0 (B)
5. Pick up P_B at M0
6. Move P_B to M1 (C)
7. Load P_B on M1
8. Move back to M2 (D)
9. Wait at M2
10. Unload P_A from M2
11. Move P_A to M3 (E)
12. Load P_A to M3
13. Move back to M1 (F)
14. Unload P_B at M1
15. Move P_B to M2 (G)
16. Load P_B on M2
17. Move back to M3 (H)
18. Wait at M3
19. Unload P_A from M3
20. Move P_A to M1 (I)
21. Load P_A to M1
22. Move back to M2 (J)
23. Unload P_B at M2
24. Move P_B to M4 (K)
25. Drop off P_B on M4
26. Move back to M1 (L)
27. Wait at M1
28. Unload P_A from M1
29. Move P_A to M4 (M)
30. Drop off P_A at M4
31. Move back to M0 (N)

Figure 3.5: Sequence of robot moves for pair grouping G1

As stated above, there are many part schedules that can be generated from a part family and each part schedule may result in different makespan values. The makespan of these part schedules have to be calculated by using time chart methodology. A single equation for calculating the minimum makespan could not be found. In case that the part family consists of many parts, the solution space of the problem will be very big and complex. To find out which part schedule provides the best (minimum) makespan is a challenge. A methodology based on genetic algorithm is proposed next. Genetic algorithm utilizes simple and efficient searching techniques and represents a popular approach to stochastic optimization. Genetic algorithm is most appropriate for complex non-linear

models such as the problem considered here. Genetic algorithm has the ability to explore the solution in multiple directions and hence a powerful tool to find a solution even in a vast solution space. Chapter 4 will explain the methodology of genetic algorithms developed for solving this problem.

Chapter 4 Methodology

4.1 Outline of Genetic Algorithms

Genetic algorithms were formally introduced in the United States in the 1970s by John Holland and utilize probabilistic search approaches. They mimic the evolutionary process. The basic concept of genetic algorithm is designed to simulate the Darwinian principle of survival of the fittest. As per Darwinian evolution, all of the creatures are constantly evolving. They are continually adapting themselves to all changing environments to survive. The weaker creatures tend to die. Only the stronger and the fitter creatures live to mate and to create offspring and ensure the continuing survival of the species.

Genetic algorithms are techniques that mimic biological evolution as a problem-solving strategy. Genetic algorithms begin with a set of solutions called initial population. The population consists of chromosomes, with each chromosome representing a solution to the problem. After initial populations are created, each candidate chromosome is evaluated by using some measure of fitness. To form a better population, only the fitter chromosomes are selected as the parents and allowed to reproduce the new chromosomes as the offspring. The more suitable they are, the more chances they have to reproduce. The offsprings are created by two basic genetic operators; crossover and mutation. After the operation, the new population consists of two groups: (i) the preceding chromosomes and (ii) the new offspring. To maintain population size, only the fitter chromosomes of the new population are selected to take part on evolving the life cycle. Every evolutionary step is known as a generation. The new generation is produced from the surviving chromosome and the better offsprings from the previous

generation. By this way, the new generation will become stronger than the last generation. Then, the new generation is used to reproduce its next generation. The population evolves again and again, containing more and more highly fit chromosomes. When a certain convergence criterion is reached, wherein no significant further increase in the average fitness of the population results, the best chromosome produced will be selected to determine optimal solution to the problem.

4.2 The Implementation of Genetic Algorithm

A typical genetic algorithm consists of six processes: (i) encoding, (ii) initialization, (iii) fitness evaluation, (iv) parent selection, (v) genetic operations and (vi) evolution. In each process, there are different methods for implementation. For example, in the encoding process, solution can be encoded in several ways such as permutation encoding, binary encoding, or tree encoding, [22]. It depends on the nature of the problem to decide which method is suitable for the problem. For instance, permutation encoding is suitable for the ordering problem such as traveling salesman problem (TSP), [22] as well as the problem focused in this work. Therefore, in order to obtain an efficient solution of any problem, every process of genetic algorithms should be implemented to suit the problem considered. In this section, the implementation of genetic algorithm for part scheduling for the three-machine robotic cell considered is presented.

Java language is used to code the proposed genetic algorithm. At the start, the essential data about parts to be produced in the robotic cell, namely : number of parts, the processing sequence, and the processing time at machines for each part, robot loading

and unloading time, and robot movement time between machines will be input. Similarly the necessary information about genetic operations which are population size, number of generations, the type and the probability of crossover and mutation will also be input. The program provides several functions to create initial population, to evaluate fitness, to select parents, to perform genetic operations and evolution with the goal of finding the optimal part schedule which provides minimum makespan. Figure 4.1 shows the flow chart of genetic algorithm implementation.

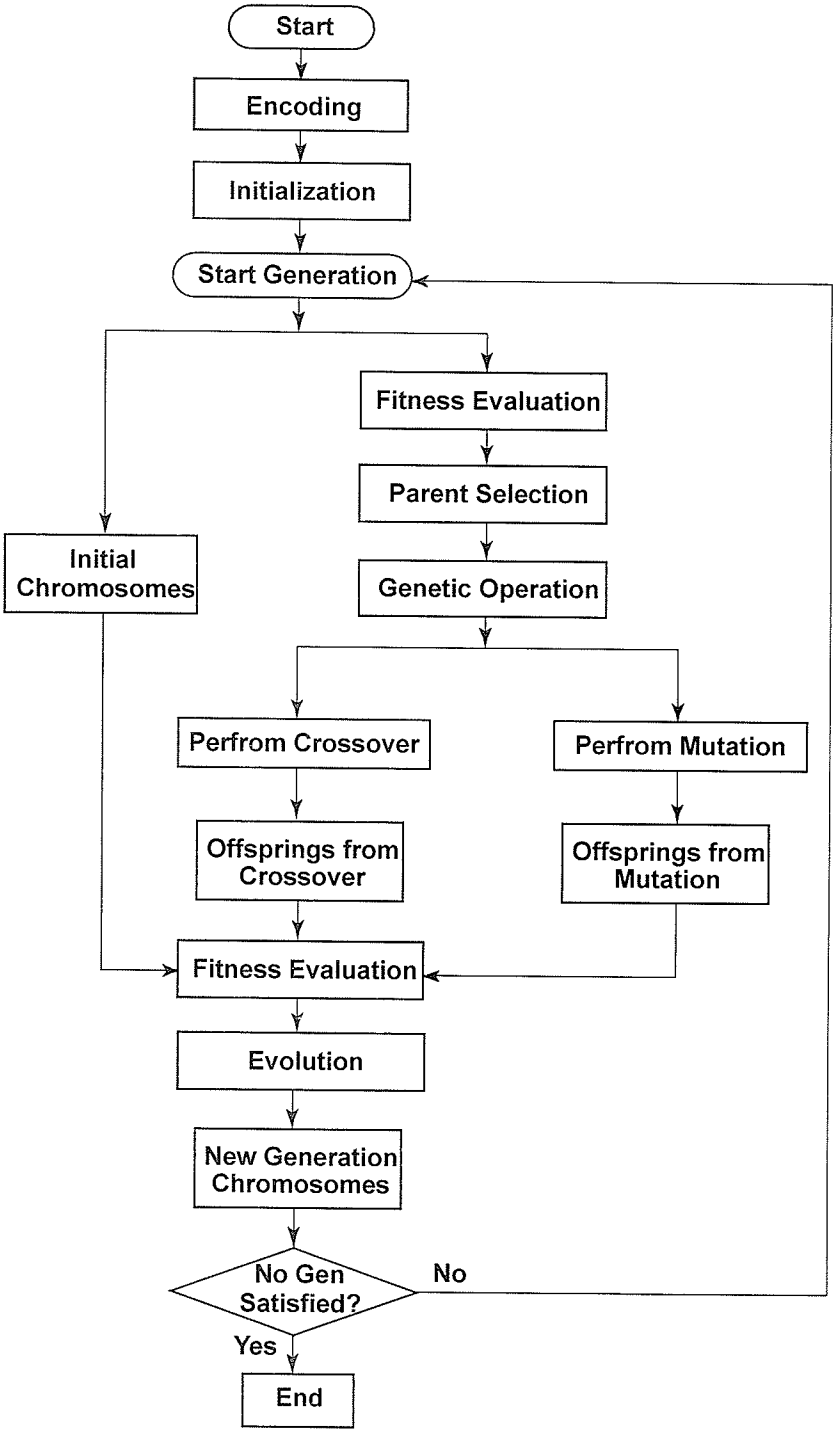


Figure 4.1: Flow chart of genetic algorithm

4.2.1 Encoding

Before a genetic algorithm can be applied to solve a problem, a method is needed to encode potential solutions to the problem in a computer implementable format. The process is called "Encoding". It represents the potential solution to the problem in the form of a chromosome. Specifically, part schedules are represented by chromosomes. In this work, permutation encoding is adopted, [22]. Permutation encoding is particularly suitable for sequencing problems. In permutation encoding, every chromosome consists of a string of genes. Each gene is an integer that represents a part and the position of each gene represents the position of the part in the part schedule. Therefore, if there are *n* parts to be scheduled, each chromosome will consist of an integer from 1 to *n* with the sequence of numbers being different from chromosome to chromosome. In short, the length of the chromosome is equivalent to the number of parts. Figure 4.2 shows chromosome "C1" which is represented as 1 3 6 2 8 4 5 7. It means chromosome "C1" has part schedule that starts with processing of part 1 first, followed by part 3, 6, 2, 8, 4, 5 and 7 in that order. Each part goes through a variety of machine in a certain sequence, as contained in the data for each part.

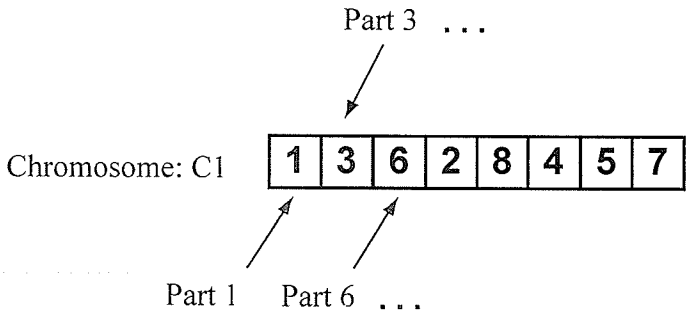


Figure 4.2: Representation of chromosome C1

4.2.2 Initialization

To start the genetic algorithm, a group of chromosomes are required. The group of chromosome that is used to start genetic algorithm is called "initial population". Initialization is the process to generate the initial population. The first step is to define the population size. The initial population will be generated, represented as an integer in a two-dimensional array. Each row of the array represents a chromosome which is a part schedule. The number of rows which is the number of the initial chromosomes is equivalent to the population size. As stated in Section 4.2.1, each chromosome contains an integer 1 to n where n is the number of parts. The process to generate initial population contains the following steps.

(1) Generate chromosome C_i

C_i = 1, 2, 3, 4... n , where n = number of parts and i = 1, 2 ... population size.

(2) For each C_i, generate random number $(r_k)_i$ from a range [0, 1],

where k = 1, 2 ... number of part and i = 1, 2 ... population size.

(3) Group the random number $(r_k)_i$ with the integer in C_i.

$(r_1)_i$ = 1, $(r_2)_i$ = 2.... $(r_n)_i = n$

(4) Use bubble sort method to sort $(r_k)_i$ in descending order.

(5) Sort the integer in C_i according to the sorted $(r_k)_i$ from (4).

Figure 4.3 shows the example of generating initial chromosome C1.

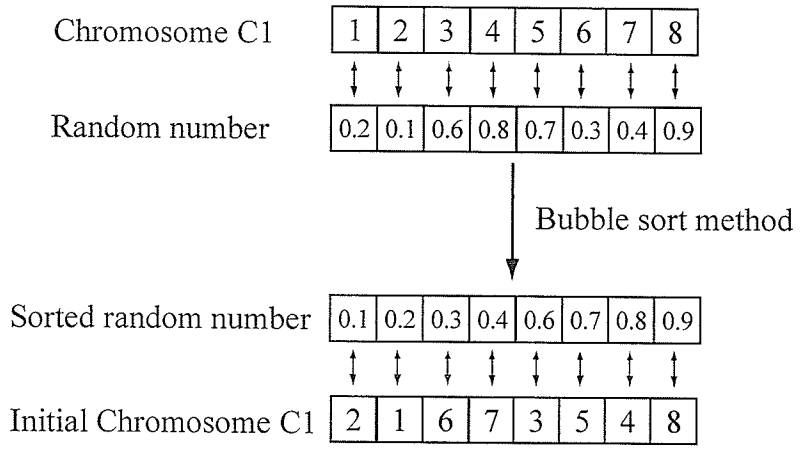


Figure 4.3: Example of generating initial chromosome C1

To generate initial population, first, the chromosome is created as an array of integers from 1 to *n*, where *n* is number of parts. Next, the number from 0 to 1 is generated randomly. Each random number represents each gene (integer) in the chromosome. Then, by using the value of the random numbers, the genes in the chromosome are sorted in descending order by using bubble sort method. The next chromosome is generated using the same procedure until the number of chromosome is equivalent to population size. In this way the chromosomes for the initial population are created.

4.2.3 Fitness Evaluation

Fitness evaluation is a method to measure the fitness value of a particular chromosome. In order to select the fitter chromosome to progress to subsequent process of genetic algorithms, each candidate chromosome has to be evaluated by using some measure of fitness. For part scheduling in a three-machine robotic cell problem, the makespan is used as a fitness measurement. Specifically, the reciprocal value of the makespan is a

fitness value. However, the makespan has to be calculated within the condition that achieves the most crucial issue of the problem, namely: each part in chromosome has a specific order of sequencing through machines and the order cannot be changed. Hence, in this work, the pair grouping method is designed to calculate the makespan. The concept of the pair grouping method is that the makespan of each chromosome can be calculated by the summing all the makespan values of the groups of parts that are paired in chromosome. The process begins with every part in the chromosome being paired with its adjacent part with the intention of maintaining the order of the part in the chromosome. Next, the makespan of each pair will be calculated by using the priority method as mentioned in Section 3.4. Then, the makespan of the chromosome will be formulated from the summation of the makespans from every pair. Figure 4.4 shows the example of the pair grouping method for chromosome C1.

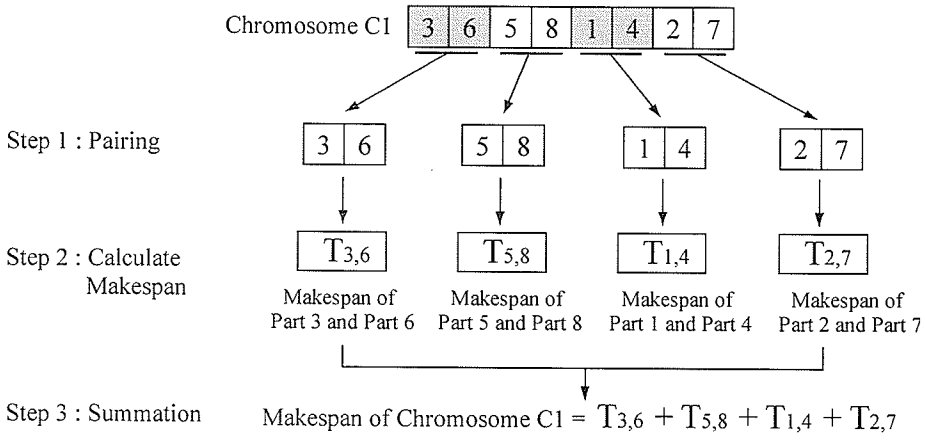


Figure 4.4: Example of pair grouping method in chromosome C1

The process of pair grouping method contains the following steps.

Assume that "*C*" is a particular chromosome considered to measure fitness value where chromosome "*C*" consists of *n* parts which are $P_1, P_2, P_3 \ldots, P_n$.

In other words, $C = P_1,\ P_2,\ P_3 \ldots, P_n$, where n = number of parts.

(1) Create G_i , a group i of chromosomes by pairing part with its next part.

$$G_1 = \{P_1, P_2\},\ G_2 = \{P_3, P_4\} \ldots, G_k = \{P_{(n-1)}, P_n\},$$

where i = 1, 2... , k and $k = \frac{n}{2}$.

(2) Calculate TG_i , the makespan of group G_i .

The detail of makespan calculation is shown in Section 3.4

(3) Calculate TC , the makespan of chromosome "C" from

$$TC = \sum_{i=1}^{k} TG_i$$

(4) Calculate FV , the fitness value of chromosome "C" from

$$FV = \frac{1}{TC}$$

However, the above process can be used more efficiently when the number of parts in the chromosome is an even number. When the number of part is an odd number, there will be an excess part that can not be paired. The excess part will be treated as a group with just that part as a member. The position of the excess part can take many options. Each different position of the excess part provides a different model of grouping. Therefore, there is more than one option to pair the parts in chromosome when the number of part is an odd number. The number of options of a pair grouping is equivalent to $(n+1)/2$, where n is the number of parts. Figure 4.5 shows the four alternatives for pairing the group of chromosome C1, where C1 = 3, 6, 5, 1, 4, 2, 7.

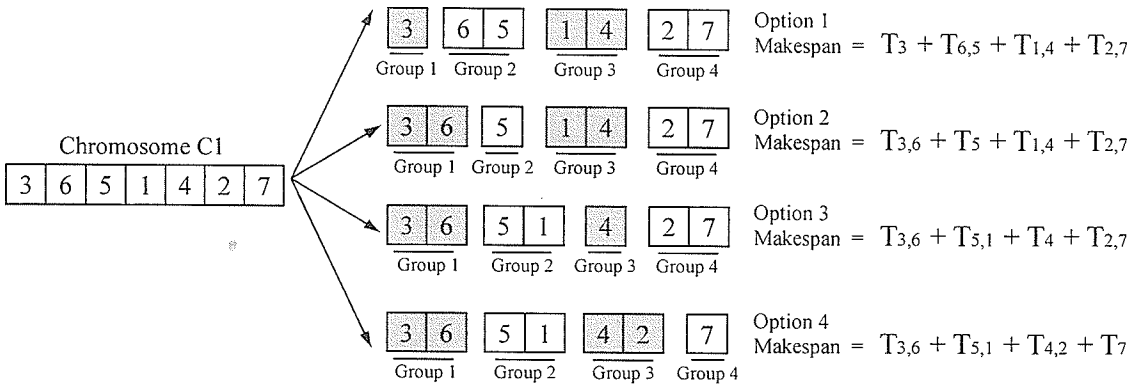


Figure 4.5: Four alternatives of pair grouping of chromosome C1

As a result, when the number of parts is an odd number, all options of a pair grouping in the chromosome have to be considered. The option that provides the best (minimum) makespan will represent the fitness value of the chromosome. The process of the pair grouping method when the number of parts is an odd number contains the following steps.

Assume that "*C*" is a particular chromosome whose fitness value needs to be calculated. $C = P_1,\ P_2,\ P_3 \ldots, P_n$, with n = number of parts.

(1) Create G_{ij}, a group j of option i by pairing part with its next part and at any position where $i = j$, G_{ij} will contain only one member which is $P_{(2j-1)}$. In other word, $G_{jj} = \{P_{(2j-1)}\}$.

$$G_{11} = \{P_1\},\ G_{12} = \{P_2, P_3\},\ G_{13} = \{P_4, P_5\} \ldots, G_{1k} = \{P_{(n-1)}, P_n\},$$

$$G_{21} = \{P_1, P_2\},\ G_{22} = \{P_3\},\ G_{23} = \{P_4, P_5\} \ldots, G_{2k} = \{P_{(n-1)}, P_n\},$$

$$\vdots$$

$$G_{j1} = \{P_1, P_2\},\ G_{j2} = \{P_3, P_4\} \ldots,\ G_{jj} = \{P_{(2j-1)}\} \ldots, G_{jk} = \{P_{(n-1)}, P_n\},$$

$$\vdots$$

$$G_{k1} = \{P_1, P_2\},\ G_{k2} = \{P_3, P_4\},\ G_{k3} = \{P_5, P_6\} \ldots, G_{kk} = \{P_n\}$$

Where i , j $= 1, 2 \ldots, k$ and $k = \frac{n+1}{2}$.

(2) Calculate TG_{ij} , the makespan of group G_{ij}.

The detail of makespan calculation is shown in Section 3.4

(3) Calculate TC_i , the makespan of option *i* of chromosome "*C*" from

$$TC_i = \sum_{j=1}^{k} TG_{ij}$$

(4) The TC_i that provides the minimum makespan, will be used to calculate the fitness value FV of chromosome "*C*" from

$$FV = \frac{1}{TC_{\min}}$$

Figure 4.6 shows the flow chart of fitness evaluation procedure.

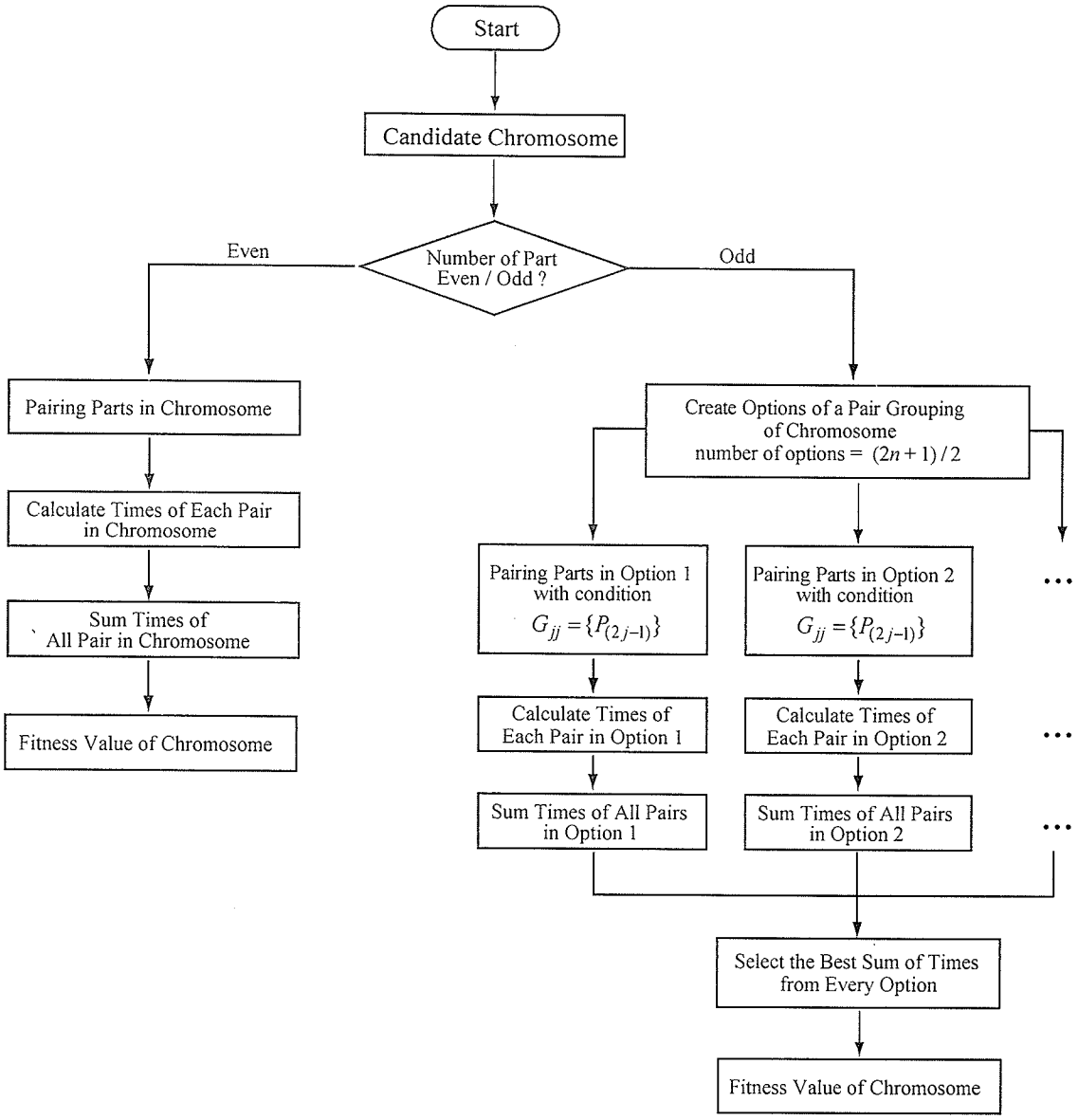


Figure 4.6: Flow chart of fitness evaluation procedure

4.2.4 Parent Selection

"Parent selection" is used to select the populations from the previous generation to serve as the parents for the next generation. The principle followed is to select only the fitter chromosomes as the parents to reproduce the new offspring by performing the genetic operation. In this work, a roulette wheel selection approach is applied for parent

selection. The roulette wheel selection is a form of fitness selection procedure. In this the chance of an individual being selected is determined. The amount of fitness is measured by quantifying the fitness as a proportion. The concept can be represented as a game of roulette with each individual getting a slice of the wheel. The more fit ones get larger slices than less fit ones. When the wheel is spun, the section on which it lands and the individual who owns the location is chosen. The places on the wheel can be representing different chromosomes. There is a good chance for some chromosomes to be selected more than once. The better the fitness, the greater is the chance it will be selected. The selected chromosome will go forward to form the mating pool for creating the next generation. Multiple copies of the same chromosome can exist in the mating pool. The roulette wheel selection approach contains the following steps.

(1) Calculate fitness value, FV_i from makespan, TC_i as described in 4.2.3

$$FV_i = \frac{1}{TC_i}$$, where i = 1, 2... population size.

(2) Calculate selection probability, p_i for each individual using,

$$p_i = \frac{FV_i}{\sum_{i=1}^{pop\_size} FV_i}$$, where i = 1, 2... population size.

(3) Calculate cumulative probability, Cp_i for each individual from

$$Cp_i = \sum_{n=1}^{i} P_n$$, where i = 1, 2... population size.

(4) Generate random number (r_k) from a range [0, 1].

(5) If $r_k \leq Cp_i$ then select individual i.

If $Cp_i < r_k \leq Cp_{i+1}$, then select individual $i+1$.

The roulette wheel approach is stochastic. The approach begins with determining the slice area of each individual in the wheel. By calculating the selection probability and cumulative probability from fitness value, the slice area of each individual are defined. Then, the number from 0 to 1 is generated randomly. If the random number lands in a certain area, for example the generated random number is between cumulative probability Cp_i and Cp_{i+1}, the individual i + 1 is selected. Using this procedure, the chromosomes are chosen to form mating pool. In other words, the mating pool contains the selected chromosomes that will be chosen to be the parent chromosomes when genetic operations occur. The number of selected chromosomes in mating pool is equivalent to population size. Hence, there are chances that some chromosomes are selected more than once and some may not be chosen to be included in the mating pool.

4.2.5 Genetic Operators

Once parent selection has chosen the fitter parents, there must be ways of improving their fitness in the next generation. There are two basic genetic operators to accomplish this, which are crossover and mutation. The crossover and mutation are the most important parts of the genetic algorithms to generate the better chromosome.

4.2.5.1 Crossover

Crossover is a genetic operator that combines the two selected chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both the parents if it takes the best characteristics from each of the parents. There are several types of crossover operators. In this work, three crossover operators which are (i) one-point crossover, (ii) two-point crossover (version I) and (iii) two-point crossover (version II) are adopted. The following sections provide further detail.

(1) One-Point Crossover

The one-point crossover begins with a random selection of a crossover point (a cutting point) in the chromosome. Then, the two new offsprings are generated by interchanging the genes from the two parent chromosomes at this point.

Figure 4.7 shows the example of a one-point crossover wherein parent 1 and parent 2 are divided into two sections by the cutting point.

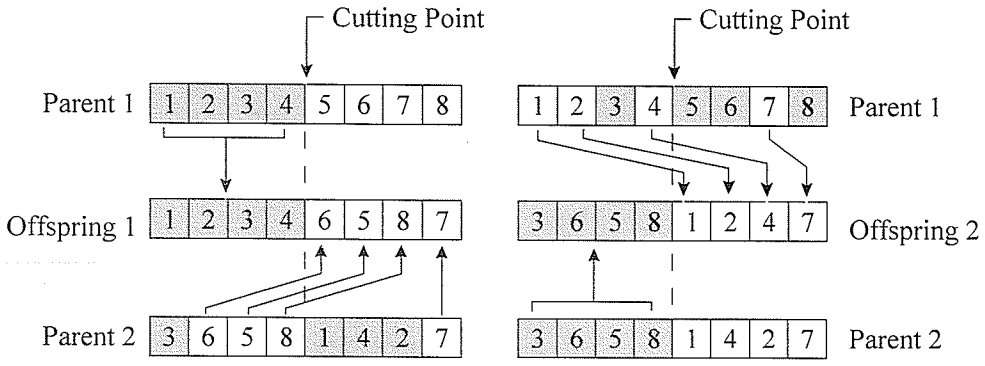


Figure 4.7: Example of one-point crossover

The two offsprings, offspring 1 and offspring 2, are generated from the two parents. The left section of offspring 1 imitates the genes from the left section of parent 1 while offspring 2 inherits from parent 2. The right section of offspring 1 is the copy of the genes that are not contained in the first section of offspring 1 from parent 2. The creation of offspring 2 is similar, except in this case, the right section of offspring 2 is created as shown in Figure 4.7.

(2) Two-Point Crossover (Version I)

The two-point crossover (version I) begins with a random selection of two crossover points in the chromosome. Then, the two new offsprings are created by interchanging the genes from the two parent chromosomes at these two points. Figure 4.8 illustrates the example of two-point crossover (version I) where the two offsprings: offspring 1 and offspring 2 are generated from two parents.

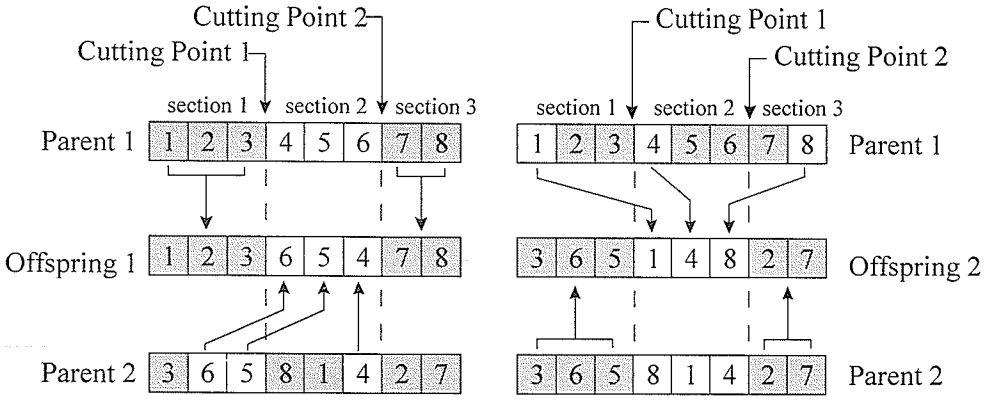


Figure 4.8: Example of two-point crossover (version I)

Parent 1 and parent 2 are divided into three sections: section 1, section 2 and section 3 by cutting point 1 and cutting point 2. The genes in section 1 and section 3 of offspring 1 inherit from section 1 and section 3 of parent 1 respectively while offspring 2 inherits those genes from parent 2. The genes in section 2 of offsprings 1 is the copy of the genes from parent 2 that are not contained in the section 1 and the section 3 of offspring 1. The section 2 of offspring 2 is created in similar way, except that the genes are a copy from parent 1.

(3) Two-Point Crossover (Version II)

The two-point crossover (version II) is similar to the two-point crossover (version I). It starts with randomly choosing two crossover points in the chromosome of the two parents. As a result, the chromosomes of the two parents are divided into three sections: section 1, section 2 and section 3. Then, the two new offsprings are created by interchanging the genes from the two parent chromosomes at these two points. Figure 4.9 shows the example of a two-point crossover (version II).

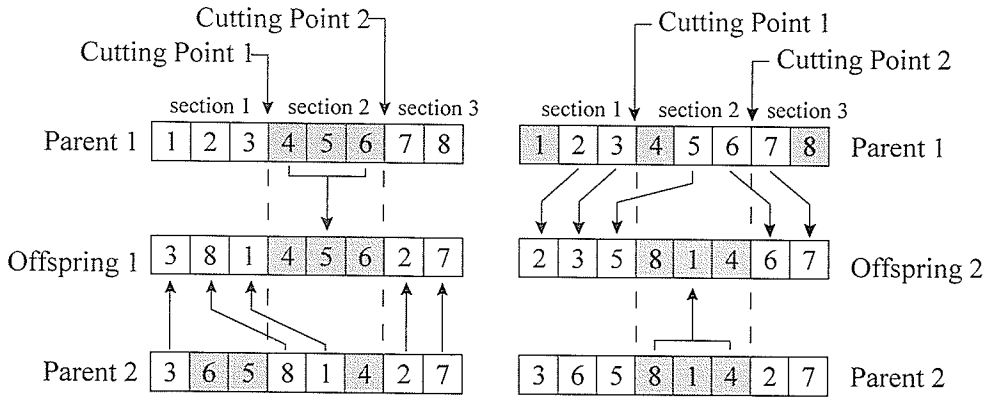


Figure 4.9: Example of two-point crossover (version II)

In the case of two-point crossover (version II), the genes in the section 2 of offspring 1 inherits from the genes in the section 2 of parent 1 whereas offspring 2 inherits those genes from parent 2. The genes in the section 1 and section 3 of offspring 1 are the copy of the genes from parent 2 that are not contained in the section 2 of offspring 1. Similarly to generate the section 1 and the section 3 of offspring 2, the genes of the corresponding section are created from parent 1 as shown in Figure 4.9.

The number of children from crossover operation depends on the crossover probability (*Pc*). For instance, if *Pc* equals 0.6, it means that 60% of selected chromosomes in the mating pool should undergo crossover operation. In other words, the number of offsprings from crossover should be equivalent to 60% of population size. The pair of parents in which crossovers are performed is randomly picked from the mating pool. The process of choosing parents is as described below.

(1) Generate a random number S_i from the range [0, 1] of each chromosome i (C_i) in mating pool, where i = 1, 2... , population size.

(2) If $S_i \leq Pc$, then chromosome i (C_i) is a parent chromosome that will undergo crossover operation, where Pc = crossover probability.

4.2.5.2 Mutation

Mutation is a genetic operator that randomly alters one or more gene positions within chromosomes (parents) from its initial state. This can result in entirely new chromosome values being added to the populations. With these new chromosome values, the genetic algorithm may be able to arrive at a better solution than was previously possible. Mutation is an important part of the genetic search as it helps to prevent the population from stagnating at any local optima. There are several types of mutation operators. In this work, three mutation operators which are (i) arbitrary two-job exchange mutation, (ii) arbitrary three-job exchange mutation and (iii) shift change mutation are adopted, for sequencing a 'different part type'. The detail of these mutation operators are provided below.

(1) Arbitrary Two-Job Exchange Mutation

The arbitrary two-job exchange mutation starts with a random selection of two positions of genes in a parent chromosome. Then, the offspring is generated by exchanging the position of the two genes. Figure 4.10 illustrates the example of the arbitrary two-job exchange mutation.

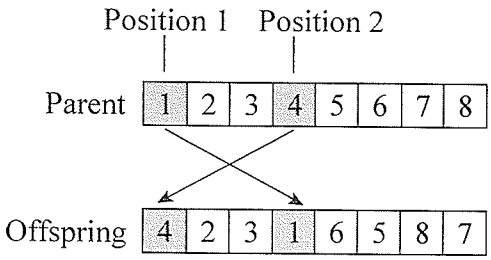


Figure 4.10: Example of arbitrary two-job exchange mutation

(2) Arbitrary Three-Job Exchange Mutation

The arbitrary three-job exchange mutation begins with a random selection of three positions of genes in a parent chromosome. An example is shown in Figure 4.11. The offspring is created by exchanging the position of those genes as: (a) position 1 exchanges with position 2, (b) position 2 exchanges with position 3 and (c) position 3 exchanges with position 1. Figure 4.11 illustrates the arbitrary three-job exchange mutation.

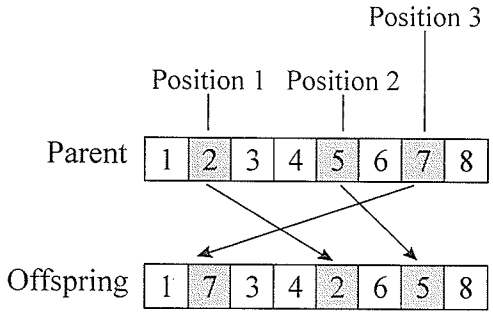


Figure 4.11: Example of arbitrary three-job exchange mutation

(3) Shift Change Mutation

The shift change mutation starts with randomly choosing two positions of genes in a parent chromosome. Then, the offspring is created as follows: (a) move the

gene at position 2 to position 1, and (b) shift the genes between position 1 and the position before position 2 by one location. Figure 4.12 demonstrates the example of the shift change mutation.

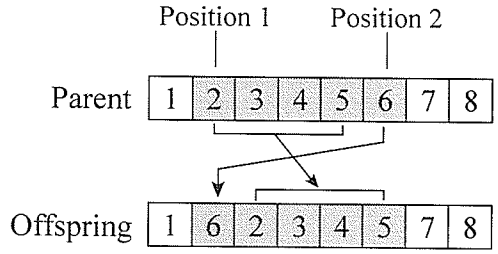


Figure 4.12: Example of shift change mutation

The number of offspring from mutation depends on Mutation probability (*Pm*). The parents considered to perform mutation operation are randomly chosen from the mating pool generated from parent selection process. The process of choosing parent is described below.

(1) Generate random number R_i from the range [0,1] of each chromosome i (C_i) in mating pool, where i = 1, 2 .. , population size.

(2) If $R_i \leq Pm$ then chromosome i (C_i) is a parent chromosome that will undergoes mutation operation, where *Pm* = crossover probability.

4.2.6 Evolution

Evolution strategy is a method that uses to select the stronger (fitter) chromosomes and to eliminate the weaker chromosomes. After performing genetic operations, total population is increased due to the offsprings from genetic operations. The new population consists of three groups of chromosome: (i) the initial chromosomes, (ii) the chromosomes from crossover operation and (iii) the chromosomes from mutation operation. In order to maintain the population size, only the chromosomes that have the best or the better fitness values are chosen. The fitness evaluation presented in section 4.2.3 is used to measure fitness value. The selected chromosomes which are called the surviving chromosomes will become a new population and will be used as the starting chromosomes for the next generation. In this way, the next generation will be created from the stronger chromosome than the last generation. As a result, the new population will become stronger than the last generation. Then, the population evolves again and again, resulting in more and more highly fit chromosomes. When the number of generation reaches a user selected value, the best chromosome produced the minimum makespan will be determined as the best solution or the best part schedule of the problem. The next chapter presents the results of tests conducted using the methodology presented.

Chapter 5 Results and Discussion

5.1 Testing and Evaluation of Genetic Algorithm (GA)

In this section, the GA developed is tested to obtain the best part scheduling that will produce the minimum makespan for the three-machine robotic cell. The algorithm was coded using Java as the programming language. To test the program, a part family 'F2' containing 16 different parts is used. The data for part family F2 is shown in a tabulated form in Table 5.1. Each part in the part family has a different order of sequencing through machines with the exception of part P02 and P16 which have the same order of sequencing through machines but have different processing times on each machine. The processing times of parts in the part family were generated randomly to produce a range from 1 to 60 seconds.

Table 5.1: Part family F2

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	15	M2	16	M3	11
P2	M1	9	M3	43	M2	20
P3	M2	45	M1	28	M3	29
P4	M2	27	M3	58	M1	46
P5	M3	55	M1	11	M2	60
P6	M3	58	M2	38	M1	37
P7	M1	29	M2	9	None	-
P8	M1	16	M3	28	None	-
P9	M2	16	M1	38	None	-
P10	M2	51	M3	49	None	-
P11	M3	40	M1	19	None	-
P12	M3	51	M2	20	None	-
P13	M1	10	None	-	None	-
P14	M2	24	None	-	None	-
P15	M3	48	None	-	None	-
P16	M1	22	M3	59	M2	49

The robot movement times between two adjacent stations is assumed to be 3 secs. The pick and drop times was also assumed to be 3 secs at all station.

The part family F2 is tested by using four different sets of genetic probabilities, namely [*Pc*, *Pm*] = [0.2, 0.2], [0.6, 0.4], [0.5, 0.5], [0.8, 0.8]. This results in four tests for part family F2. The tests are designed to terminate when the number of generation reach 100. The other relevant parameters for the test are as shown below.

Population size:	20
Crossover Type:	One-Point Crossover
Mutation Type:	Arbitrary Two-Job Exchange Mutation
Number of Evolutions:	100

Figures 5.1(a) through (d) show makespan values obtained versus the number of generations required for the chosen *Pc* and *Pm*. The chromosome that produces the best makespan and the corresponding pair grouping are also shown.

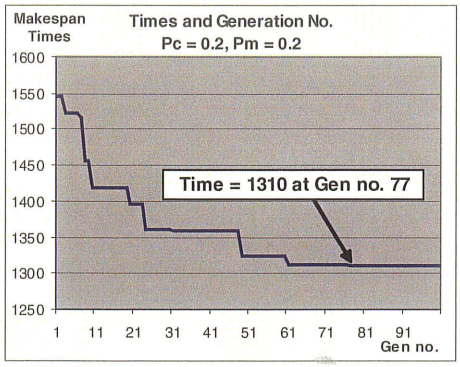


Figure 5-1(a)

Pc = 0.2, Pm = 0.2
Minimum makespan: 1310 sec.
at generation # 77
Chromosome with the minimum makespan:
15 13 16 03 06 09 04 01 10 07 12 08 05 02 14 11
Paired groups:
P15-P13 = 84.0
P16-P03 = 196.0
P06-P09 = 193.0
P04-P01 = 191.0
P10-P07 = 142.0
P12-P08 = 139.0
P05-P02 = 234.0
P14-P11 = 131.0

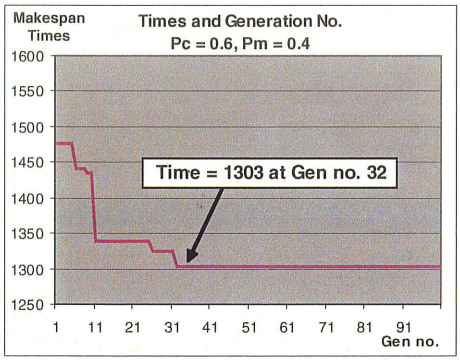


Figure 5-1(b)

Pc = 0.6, Pm = 0.4
Minimum makespan: 1303 sec.
at generation # 32
Chromosome with the minimum makespan:
06 02 16 03 15 01 14 13 12 08 07 05 04 10 09 11
Paired groups:
P06-P02 = 211.0
P16-P03 = 196.0
P15-P01 = 119.0
P14-P13 = 84.0
P12-P08 = 139.0
P07-P05 = 198.0
P04-P10 = 203.0
P09-P11 = 153.0

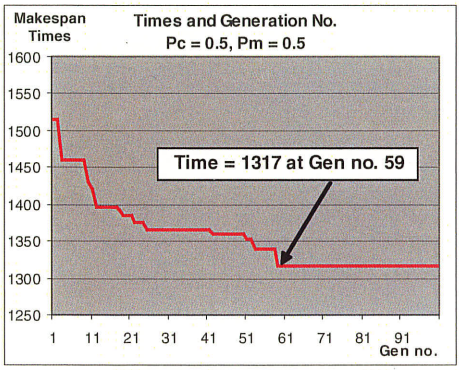


Figure 5-1(c)

Pc = 0.5, Pm = 0.5
Minimum makespan: 1317 sec.
at generation # 59
Chromosome with the minimum makespan:
07 12 05 02 16 03 06 09 08 13 11 14 15 01 04 10
Paired groups:
P07-P12 = 131.0
P05-P02 = 234.0
P16-P03 = 196.0
P06-P09 = 193.0
P08-P13 = 110.0
P11-P14 = 131.0
P15-P01 = 119.0
P04-P10 = 203.0

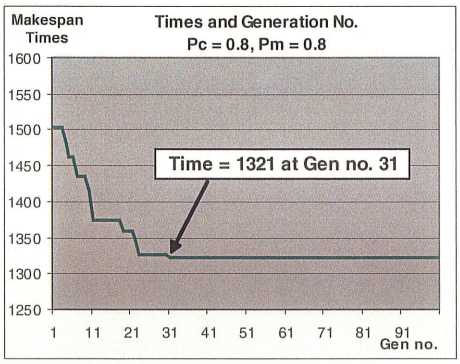


Figure 5-1(d)

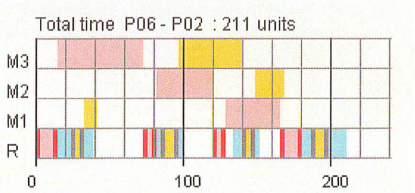
Pc = 0.8, Pm = 0.8
Minimum makespan: 1321 sec.
at generation # 31
Chromosome with the minimum makespan:
09 11 16 03 06 02 05 12 14 07 15 01 08 13 04 10
Paired groups:
P09-P11 = 153.0
P16-P03 = 196.0
P06-P02 = 211.0
P05-P12 = 230.0
P14-P07 = 99.0
P15-P01 = 119.0
P08-P13 = 110.0
P04-P10 = 203.0

Figure 5.1(a) through (d): Makespan times versus generation number of part family F2 with four genetic probabilities

From the four graphs shown in Figure 5.1, it can be observed that the makespan values reach a minimal (best value) at different generation times. These values can be identified as the best makespan for that test. The chromosome providing the minimum makespan for the test will represent the best part schedule for that test. However, the minimum makespan values and the generation number at which it is reached, is different for each case. For example Plot 5.1(b) provides a minimum makespan of 1303 sec., while Plot 5.1(d) provides a fast convergent rate, at 31 generations. From this result it can be concluded that values of *Pc* and *Pm* have an impact on the result. Results of further studies are discussed in Section 5.2.

In addition to the results shown above, the GA program is also capable of providing details pertaining to how each pair of chromosome is scheduled through the machines. This is illustrated in Figure 5.2. With this detail, the part scheduling of a part family can be easily visualized. The best makespan is obtained for the conditions shown in graph 5.1(b). The chromosome with minimum makespan from graph 5.1(b) is used as an example for illustrating details of pair grouping in term of time chart as shown in Figure 5.2. Figure 5.2 illustrates the time charts generated from GA program of the first three pairs; P06-P02, P16-P03 and P15-P01. The detail sequence of the robot moves of each pair grouping are provided below each time chart. A time chart of all chromosome pairs providing the minimal makespan in graph 5.1(b) are presented in Appendix A.

Time Chart of pair: P06 and P02

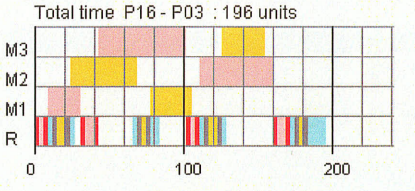


(r) = red
(p) = pink
(g) = gray
(y) = yellow
(b) = blue

The sequence of the robot moves of pair grouping between part P06 and P02

1. (r) Pick up P06 at M0
2. (p) Move P06 to M3
3. (r) Load P06 on M3
4. (b) Move back to M0
5. (g) Pick up P02 from M0
6. (y) Move P02 to M1
7. (g) Load P02 to M1
8. (b) Move back to M3
9. Wait at M3 until P06 finished
10. (r) Unload P06 from M3
11. (p) Move P06 to M2
12. (r) Load P06 to M2
13. (b) Move back to M1
14. (g) Unload P02 from M1
15. (y) Move P02 to M3
16. (g) Load P02 on M3
17. (b) Move back to M2
18. Wait at M2 until P06 finished
19. (r) Unload P06 from M2
20. (p) Move P06 to M1
21. (r) Load P06 on M1
22. (b) Move back to M3
23. (g) Unload P02 from M3
24. (y) Move P02 to M2
25. (g) Load P02 at M2
26. (b) Move back to M1
27. Wait at M1 until P06 finished
28. (r) Unload P06 from M1
29. (p) Move P06 to M4
30. (r) Drop off P06 at M4
31. (b) Move back to M2
32. (g) Unload P02 from M2
33. (y) Move P02 to M4
34. (g) Drop off P02 at M4
35. (b) Move back to M0

Time Chart of pair: P16 and P03

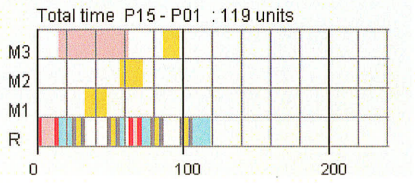


(r) = red
(p) = pink
(g) = gray
(y) = yellow
(b) = blue

The sequence of the robot moves of pair grouping between part P16 and P03

1. (r) Pick up P16 at M0
2. (p) Move P16 to M1
3. (r) Load P16 on M1
4. (b) Move back to M0
5. (g) Pick up P03 at M0
6. (y) Move P03 to M2
7. (g) Load P03 on M2
8. (b) Move back to M1
9. Wait at M1 until P16 finished
10. (r) Unload P16 from M1
11. (p) Move P16 to M3
12. (r) Load P16 on M3
13. (b) Move back to M2
14. Wait at M2 until P03 finished
15. (g) Unload P03 from M2
16. (y) Move P03 to M1
17. (g) Load P03 at M1
18. (b) Move back to M3
19. Wait at M3 until P16 finished
20. (r) Unload P16 from M3
21. (p) Move P16 to M2
22. (r) Load P16 at M2
23. (b) Move back to M1
24. (g) Unload P03 from M
25. (y) Move P03 to M3
26. (g) Load P03 at M3
27. (b) Move back to M2
28. Wait at M2 until P16 finished
29. (r) Unload P16 from M2
30. (p) Move P16 to M4
31. (r) Drop off P16 at M4
32. (b) Move back to M3
33. (g) Unload P03 from M3
34. (y) Move P03 to M4
35. (g) Drop off P03 at M4
36. Move back to M0

Time Chart of pair: P15 and P01



(r) = red
(p) = pink
(g) = gray
(y) = yellow
(b) = blue

The sequence of the robot moves of pair grouping between part P15 and P01

1. (r) Pick up P15 at M0
2. (p) Move P15 to M3
3. (r) Load P15 on M3
4. (b) Move back to M0
5. (g) Pick up P01 at M0
6. (y) Move P01 to M1
7. (g) Load P01 on M1
8. Wait at M1 until P01 finished
9. (g) Unload P01 from M1
10. (y) Move P01 to M2
11. (g) Load P01 at M2
12. (b) Move back to M3
13. Wait at M3 until P15 finished
14. (r) Unload P15 from M3
15. (p) Move P15 to M4
16. (r) Drop off P15 at M4
17. (b) Move back to M2
18. (g) Unload P01 from M2
19. (y) Move P01 to M3
20. (g) Load P01 to M3
21. Wait at M3 until P01 finished
22. (g) Unload P01 from M3
23. (y) Move P01 to M4
24. (g) Drop off P01 to M4
25. (b) Move back to M0

Figure 5.2: Time charts for the first three group pairs: P06-P02, P16-P03 and P15-P01 in the best chromosome in graph 5.1(b)

5.2 Comparison between Various Genetic Operators

Before starting a GA program, there are five necessary genetic operators that need to be input in order to obtain the minimal makespan in part scheduling. These genetic operators are (1) population size, (2) crossover type, (3) crossover probability, (4) mutation type, and (5) mutation probability. However, the various combinations of genetic operators may affect the result differently. In other words, the different values of genetic operators may provide different results for the minimum makespans in part scheduling. For example, as shown in the graphs in Figure 5.1 of Section 5.1, variations in the two genetic operators (crossover and mutation probabilities) have produced different minimum makespan times.

In order to evaluate the effects of genetic operators, the influence of all the five genetic operators is studied. The studies are categorized into three major groups which are (i) crossover, (ii) mutation and (iii) population size. The studies are performed by running the program for three types of data which are named: (1) short, (2) medium and (3) mix180. The test data "short", "medium", and "mix180" are test data that contain parts that have machine processing time ranging from 1 through 60, 60 through 120 and 1 through 180 seconds, respectively. These ranges of machine processing times were selected to study whether each data type is affected differently by genetic operators. Each data type consists of three part families as shown in column 1 of Table 5.2. A total of nine different part families from the three data types were tested for various genetic parameters. Each part family contains 40 different parts, and all the relevant test data are randomly generated. The details of the test data used to perform the studies on

genetic operators are presented in Table 5.2. Further details specific to each of the nine part families are shown in Appendix B.

Table 5.2: Test data used to perform study on genetic operators

Data Type Name	Specification	Part Family's Name	Number of Parts in Part Family
1. Short	Contains parts that have machine processing time between 1-60 sec.	short1	40
		short2	40
		short3	40
2. Medium	Contains parts that have machine processing time between 60-120 sec.	med1	40
		med2	40
		med3	40
3. Mix180	Contains parts that have machine processing time between 1-180 sec.	mix1	40
		mix2	40
		mix3	40

As before the robot times for all studies are assumed to be:

pick : 3 sec.

drop : 3 sec.

move : 3 sec.

For every study, the tests are designed to terminate when the number of generation reaches a value of 1000.

5.2.1 Study of Crossover Operator

In crossover operation, the two crossover parameters: crossover type and crossover probabilities are considered. For crossover type, there are three types of crossover that can be studied (as mentioned in Section 4.2.5.1). The three types of crossover are (1) one-point crossover, (2) two-point crossover (version I) and (3) two-point crossover (version II). They will be referred to as crossover type 1, crossover type 2 and crossover type 3, respectively in further discussions. For crossover probabilities, the values of crossover probabilities can vary from a value of 0 to 1. A value of *Pc* equals 0 means no chromosome in mating pool will be utilized in crossover, while *Pc* equals 1 means all of the chromosomes in mating pool will considered. All the nine part families from the three data types will be tested using each crossover type for the ten values of crossover probabilities: 0.1, 0.2. 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. In order to evaluate the effect of the crossover parameters, other genetic operators (mutation parameters and population size) are set as constants as shown below for every tests.

Population size:	20
Mutation probability:	0.5
Mutation type:	Arbitrary Two-Job Exchange Mutation

The results of the test are divided into three groups: short, medium and mix180. For each data type, the results are presented in a graphical form categorized by the types of crossover. The results can be divided into two types. The first type is the plot showing the minimum makespans generated and the *Pc*. The intent is to study the effect of *Pc* on the minimum makespan in each crossover type. An additional plot between the

generation numbers that produce the minimum makespans and *Pc* will also be shown to evaluate the effect of *Pc* on the convergent rate in each crossover type. The results of all part families for the same crossover parameters will be presented in the same graph. Hence, there will be three results obtained from the three part families in each graph. The results of the crossover study are presented below.

5.2.1.1 Crossover Study with Data Type: Short

The results from the tests are shown in Figure 5.3. Figure 5.3(a, b and c) show the results for average minimum makespans obtained (average from ten data) and *Pc* of crossover type 1, crossover type 2 and crossover type 3, respectively for the part families short1, short2 and short3. Figure 5.3(d, e and f) show the plots between average number of generation (average from ten data) that produce the minimum makespan and *Pc* of crossover type 1, crossover type 2 and crossover type 3, respectively for the short1, short2 and short3 part family.

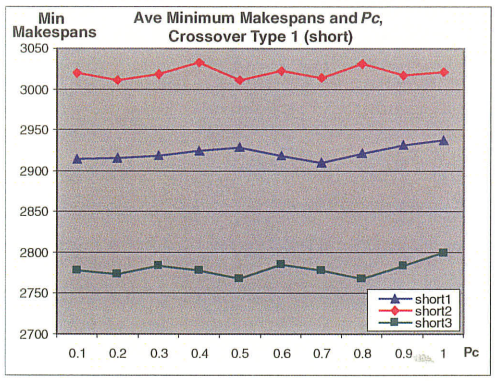


Figure 5.3(a): Graph showing average min makespans and *Pc* for crossover type 1

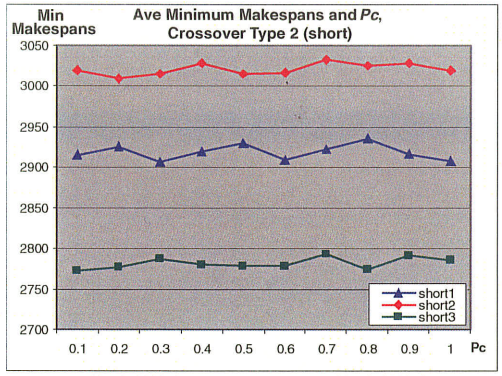


Figure 5.3(b): Graph showing average min makespans and *Pc* for crossover type 2

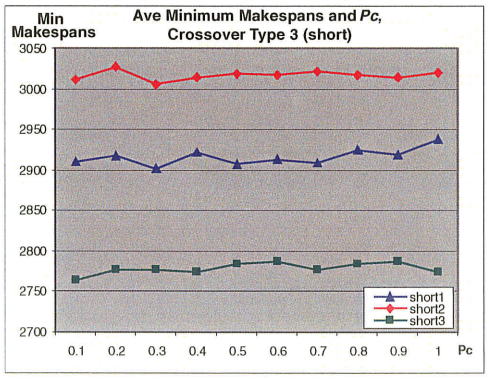


Figure 5.3(c): Graph showing average min makespans and *Pc* for crossover type 3

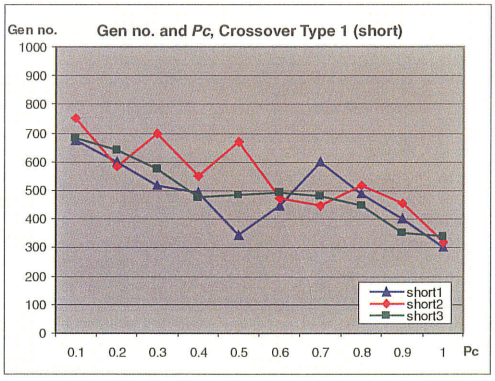


Figure 5.3(d): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 1

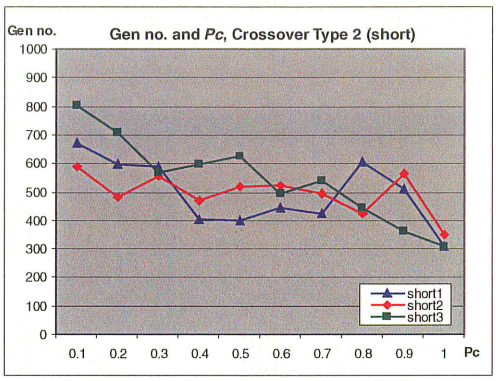


Figure 5.3(e): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 2

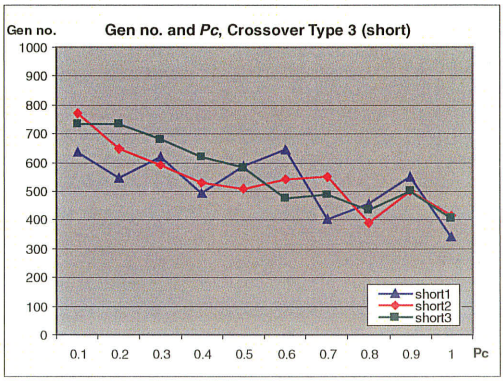


Figure 5.3(f): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 3

In order to evaluate the effect of the types of crossover on the minimum makespans, the average minimum makespans of short1, short2 and short3 part family at the same *Pc* value of each crossover type are averaged and plotted in a single graph as shown in

Figure 5.4. Figure 5.4(a) shows the average minimum makespans for all three part families in "Short" data type and *Pc* of crossover types 1, crossover type 2 and crossover type 3.

Similarly, to study the effect of crossover types on the convergent rate, the generation numbers that produce the minimum makespan at the same *Pc* point of each crossover type are averaged and are plotted as a single graph as shown in Figure 5.4. Figure 5.4(b) shows the average generation number that produce the minimum makespans from all three part families in "Short" data type and *Pc* of crossover types 1, crossover type 2 and crossover type 3.

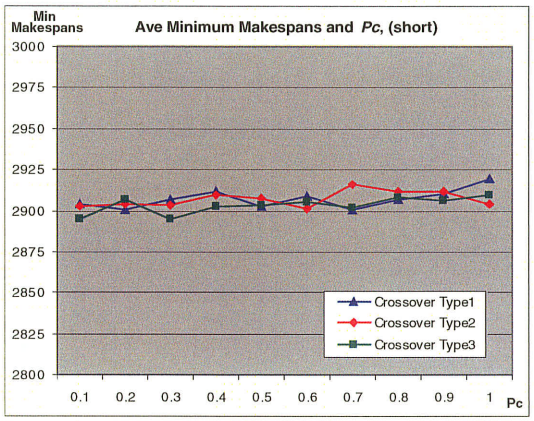


Figure 5.4(a): Graph showing the average min makespans from all three part families and *Pc* for crossover types 1, 2 and 3

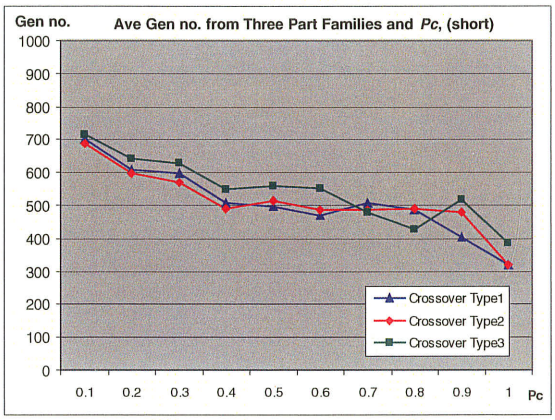


Figure 5.4(b): Graph showing the average number of generations that produce min makespans from all three part families and *Pc* for crossover types 1, 2 and 3

The plots shown in Figure 5.3(a, b and c), clearly indicate that the effect of *Pc* on the minimum makespan is not significant in any crossover types for all three part families. However, Figure 5.4(a) shows that a low value of *Pc* may provide a slightly better result. In addition, Figure 5.4(a) also illustrates that the type of crossover does not seem to have considerable effect on the minimum makespan. On the other hand, value of *Pc*

does seen to have an influence on the number of generations required to obtain the minimum makespans as can be seen in Figure 5.3(d, e and f) as well as Figure 5.4(b). The higher the values of *Pc*, the lower are the number of generations required to produce the minimum makespans. A high *Pc* results in a faster converging process. However, from Figure 5.4(b), one can conclude that that every crossover type appears to provide a similar pattern for convergent rate which means that the type of crossover does not have a significant effect on converging speed.

5.2.1.2 Crossover Study with Data Type: Medium

The results for the average minimum makespan (average from ten data) and *Pc* of crossover type 1, crossover type 2 and crossover type 3, respectively of med1, med2 and med3 part family are shown in Figure 5.5(a, b and c). Figure 5.5(d, e and f) show average number of generations (average from ten data) that produce the minimum makespan and *Pc* of crossover type 1, crossover type 2 and crossover type 3 respectively of med1, med2 and med3 part family.

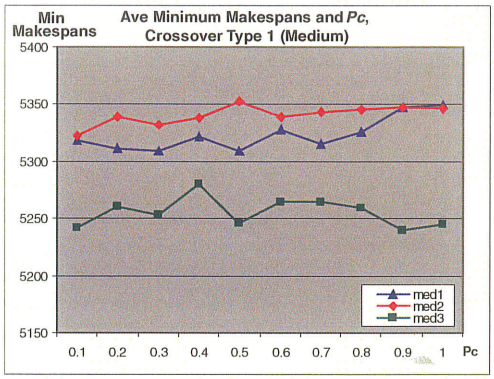


Figure 5.5(a): Graph showing average min makespans and *Pc* for crossover type 1

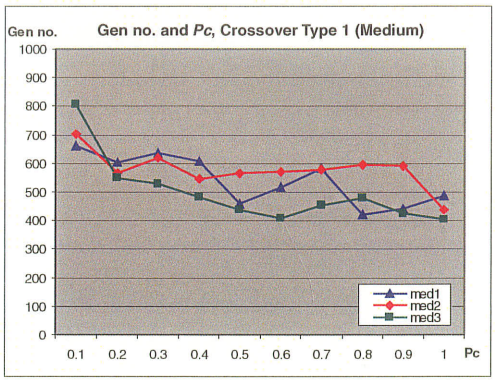


Figure 5.5(d): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 1

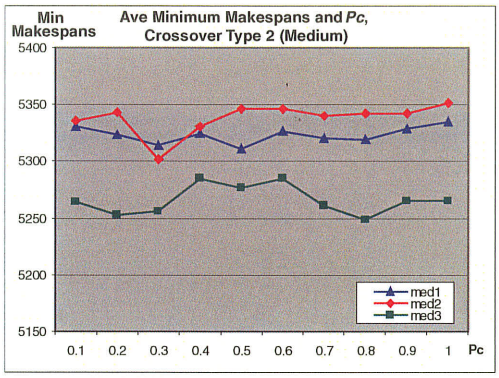


Figure 5.5(b): Graph showing average min makespans and *Pc* for crossover type 2

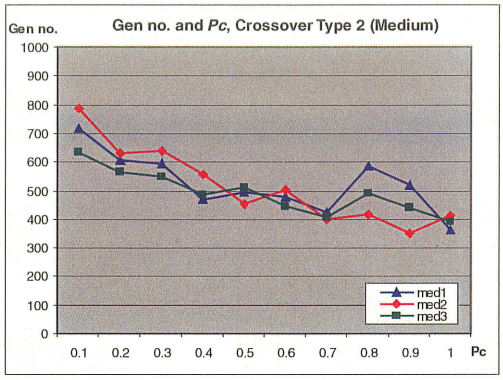


Figure 5.5(e): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 2

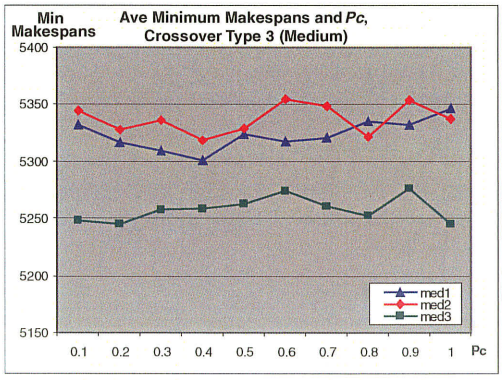


Figure 5.5(c): Graph showing average min makespans and *Pc* for crossover type 3

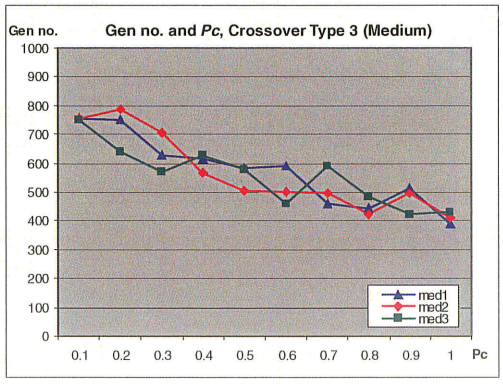


Figure 5.5(f): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 3

Similar to the procedure followed for "Short" data type, the averaged minimum makespans of med1, med2 and med3 part family at the same *Pc* point of each crossover type are plotted as a single graph, as shown in Figure 5.6(a). The purpose is to evaluate

the effect of the types of crossover on the minimum makespans. Figure 5.6(a) is the graph between the averaged minimum makespans from all three part families in "Medium" and *Pc* of crossover types 1, crossover type 2 and crossover type 3.

The generation numbers that produce the minimum makespan of all three part families at the same *Pc* point of each crossover type are also averaged and plotted into a single graph to study the consequence of crossover types on the convergent rate. Figure 5.6(b) shows the average generation number that produces the minimum makespans from all three part families of "Medium" and *Pc* of crossover types 1, crossover type 2 and crossover type 3.

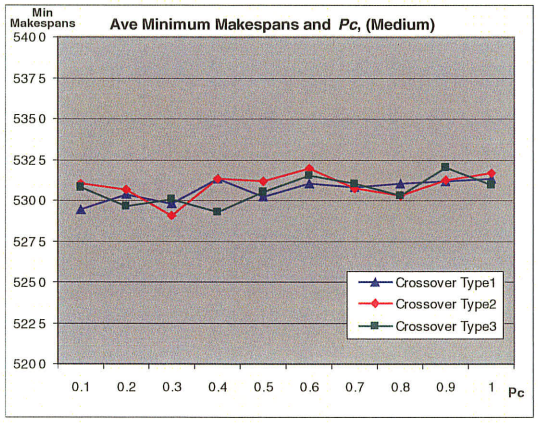


Figure 5.6(a): Graph showing the average min makespans from all three part families and *Pc* for crossover types 1, 2 and 3

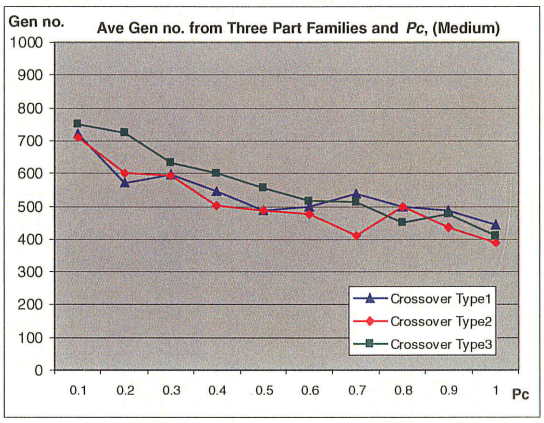


Figure 5.6(b): Graph showing the average number of that produce the min makespans from all three part families and *Pc* for crossover types 1, 2 and 3

The result shown for "Medium" data type show very similar trend to the result obtained from "Short" data type. In Figure 5.5(a, b and c), the graphs show that a change in *Pc* value does not significantly affect the minimum makespans in any crossover types. Figure 5.6(a) also illustrates that a low value of *Pc* may provide somewhat better result. Furthermore, Figure 5.6(a) also shows that the type of crossover chosen does not have a

considerable effect on the minimum makespans. In contrast, Figure 5.5(d, e and f) and Figure 5.6(b) indicate that a change in *Pc* value does have a substantial effect on the number of generations required to produce the minimum makespans. The higher the values of *Pc*, the lower are the numbers of generation required to produce the minimum makespans. In other words, a higher *Pc* provides a quicker converging process. However, Figure 5.6(b) shows that every crossover type appears to provide a similar convergent rate which means that the type of the crossover does not impact considerably the converging speed.

5.2.1.3 Crossover Study with Data Type: Mix180

The results for the tests with "Mix180" data type are shown in Figure 5.7. Figure 5.7(a, b and c) show the average minimum makespans (average from ten data) and *Pc* of crossover type 1, crossover type 2 and crossover type 3, respectively of mix1, mix2 and mix3 part family. Figure 5.7(d, e and f) show the average number of generation (average from ten data) required to produce the minimum makespans and *Pc* of crossover type 1, crossover type 2 and crossover type 3 respectively of mix1, mix2 and mix3 part family.

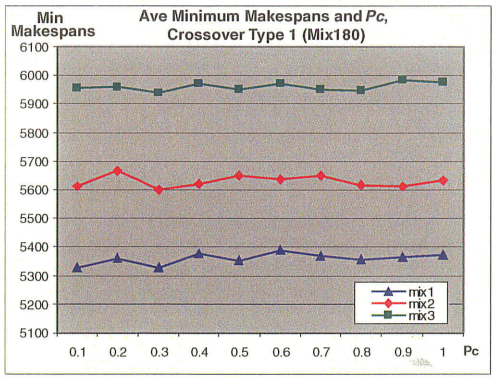


Figure 5.7(a): Graph showing average min makespans and *Pc* for crossover type 1

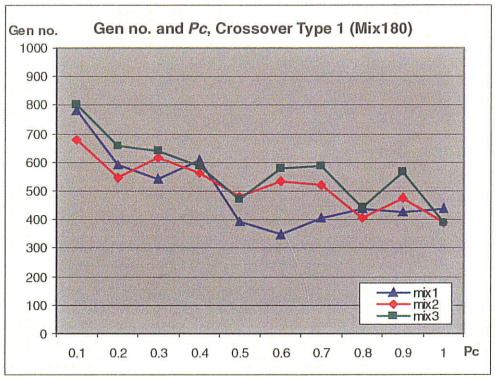


Figure 5.7(d): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 1

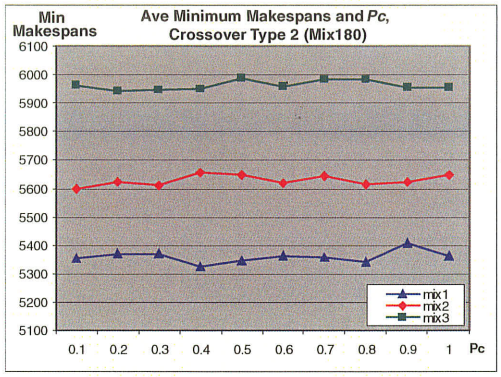


Figure 5.7(b): Graph showing average min makespans and *Pc* for crossover type 2

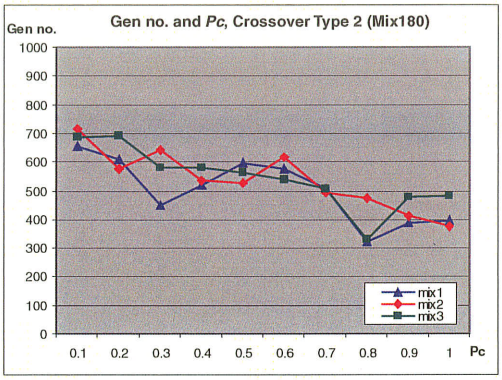


Figure 5.7(e): Graph showing number of generations that produce the min makespans and *Pc* for crossover type 2

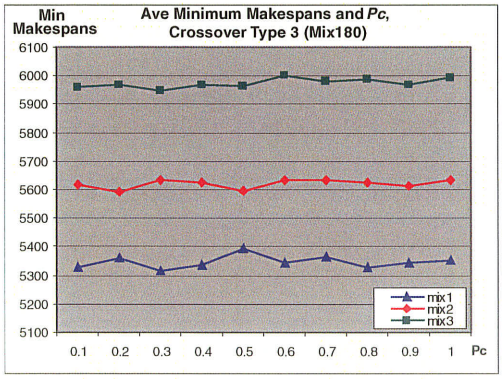


Figure 5.7(c): Graph showing average min makespans and *Pc* for crossover type 3

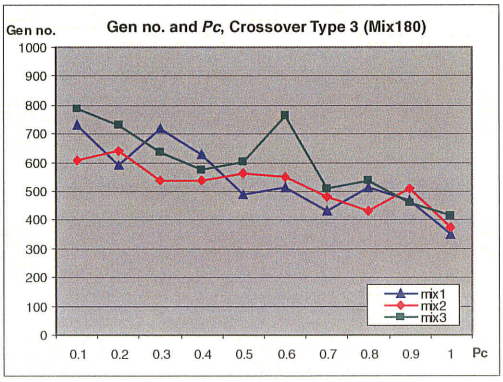


Figure 5.7(f) Graph showing number of generations that produce the min makespans and *Pc* for crossover type 3

Similar to the study of "Short" and "Medium" data type, the average minimum makespans of mix1, mix2 and mix3 at the same *Pc* value of each crossover type are averaged and plotted as a single graph to study the effect of the types of crossover on

the minimum makespans. Figure 5.8(a) show the average minimum makespans from mix1, mix2 and mix3 and *Pc* of crossover types 1, crossover type 2 and crossover type 3. The generation numbers required to produce the minimum makespans of mix1, mix2 and mix3 at the same *Pc* point of each crossover type are averaged and plotted as a single graph to study the consequence of crossover types on the convergent speed. Figure 5.8(b) shows the average generation number that produces the minimum makespans times for mix1, mix2 and mix3 and *Pc* of crossover types 1, crossover type 2 and crossover type 3.

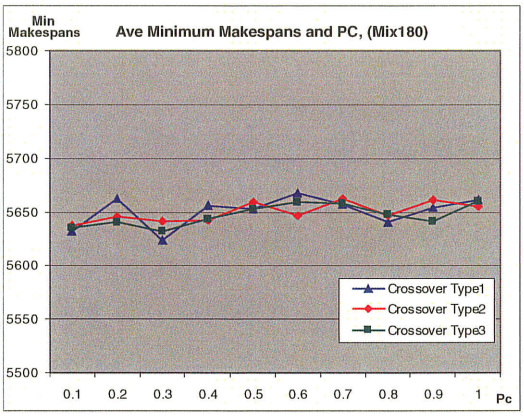


Figure 5.8(a): Graph showing the average min makespans from all three part families and *Pc* for crossover types 1, 2 and 3

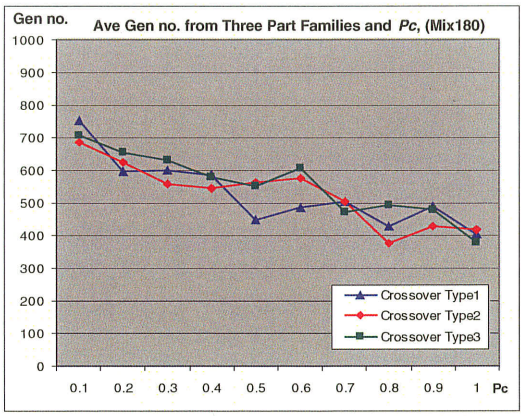


Figure 5.8(b): Graph showing the average number of generations that produce the min makespans from all three part families and *Pc* for crossover types 1, 2 and 3

The results from the test conducted for "Mix180" data type show similar trends as those from "Short" and "Medium" data type. Figure 5.7(a, b and c) illustrate that a change in the value of *Pc* does not have a significant effect on the minimum makespans in any crossover types. However, Figure 5.8(a) suggests that a low value of *Pc* may provide a slightly better result. Additionally, Figure 5.8(a) also shows that the change in crossover type does not have a considerable effect on the minimum makespans. On the contrary,

Figure 5.7(d, e and f) and Figure 5.8(b) show that a change in *Pc* has a considerable effect on the number of generation producing the minimum makespans. The higher the value of *Pc*, the lower is the number of generation required to produce the minimum makespans. In other words, a higher *Pc* provides a faster convergence towards the solution. Nonetheless, Figure 5.8(b) also suggests that the convergent rate from every crossover type appear to be the same, indicating that the type of the crossover does not have a significant effect on the converging speed.

5.2.2 Study of Mutation Operator

In mutation operation, the two mutation parameters: mutation type and mutation probabilities are considered. For mutation type, three types of mutation are analyzed using the GA program (as mentioned in Section 4.2.5.2). These three types of mutation are (1) arbitrary two-job exchange mutation, (2) arbitrary three-job exchange mutation and (3) shift change mutation. They will be referred to as mutation type 1, mutation type 2 and mutation type 3, respectively when discussing the results. For mutation probabilities, the values of crossover probabilities can be chosen from 0 – 1. Similar to the study conducted for both crossover parameters, all nine part families from three data types will be included in GA with each mutation type. The ten values of mutation probabilities considered are: 0.1, 0.2. 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. With an aim to evaluate solely the effect of the mutation parameters, other genetic operators (crossover parameters and population size) are set to a constant value for every test as shown below.

Population size:	20
Crossover probability:	0.5
Crossover type:	One-Point Crossover

The results from the test are divided into three groups: short, medium and mix180 (Section 5.2.2.1, 5.2.2.2 and 5.2.2.3 respectively). The results from each data type are presented in term of graphs categorized by the types of crossover. Two types of results are presented. The first one is a plot between the minimum makespans generated and *Pm*, in order to study the effect of the *Pm* on the minimum makespans for each mutation type. A second plot shows the relationship between the generation numbers that produce the minimum makespans and *Pm*. The purpose of the second graph is to evaluate the effect of the *Pm* on the convergence speed for each mutation type. All the part families with the same mutation parameters will be plotted as a single graph. Thus, there will be three plots obtained from the three part families in every graph. The results of the mutation study are discussed next.

5.2.2.1 Mutation Study with Data Type: Short

The results for these tests are shown in Figure 5.9. Figure 5.9(a, b and c) show the relationship between the average minimum makespans (average from ten data) and *Pm* of mutation type 1, mutation type 2 and mutation type 3, respectively of the part families short1, short2 and short3. Figure 5.9(d, e and f) are the graphs between average number of generation (average from ten data) that produce the minimum makespans and

Pm of mutation type 1, mutation type 2 and mutation type 3 respectively of the short1, short2 and short3 part family.

In order to evaluate the effect of the type of mutation on the minimum makespans, the average minimum makespans of short1, short2 and short3 part family at the same *Pm* point of each mutation type are averaged and are plotted into one graph as shown in Figure 5.10. Figure 5.10(a) illustrates the graph between the average minimum makespans from all three part families in "Short" data type and *Pm* of mutation types 1, mutation type 2 and mutation type 3.

To study the consequence of mutation types one the convergent rate, the generation numbers that produce the minimum makespans at the same *Pm* of each mutation type are averaged and are plotted into a single graph as presented in Figure 5.10. Figure 5.10(b) shows the average generation number that produce the minimum makespans from all three part families in "Short" data type and *Pm* of mutation types 1, mutation type 2 and mutation type 3.

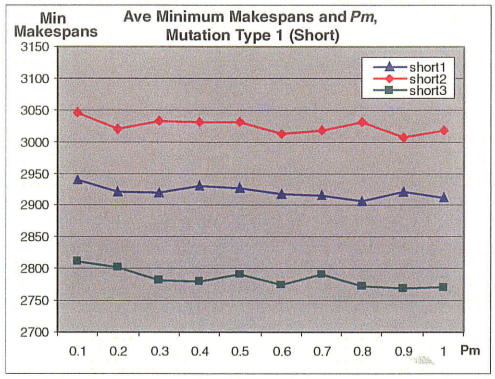


Figure 5.9(a): Graph showing average min makespans and *Pm* for mutation type 1

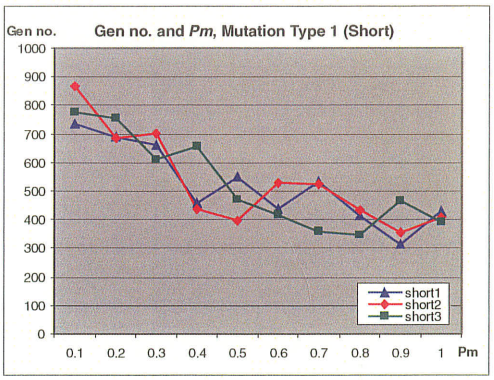


Figure 5.9(d): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 1

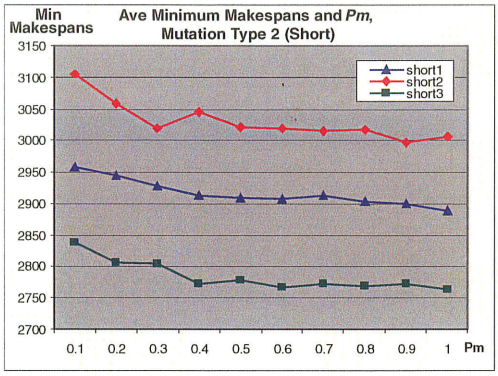


Figure 5.9(b): Graph showing average min makespans and *Pm* for mutaion type 2

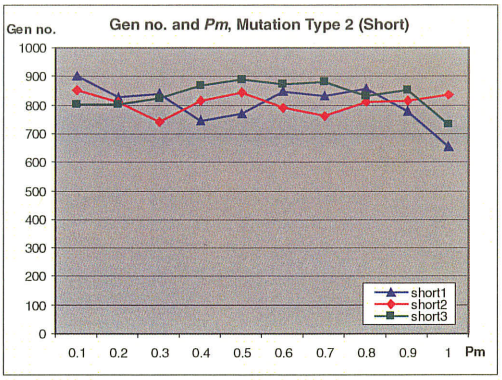


Figure 5.9(e): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 2

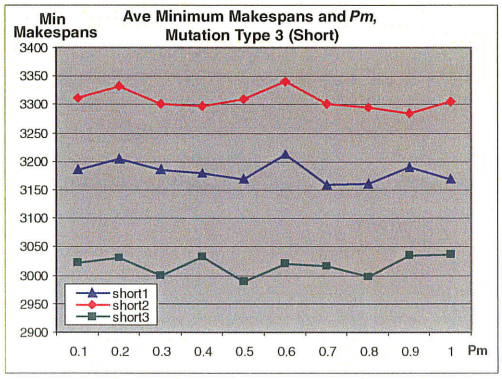


Figure 5.9(c): Graph showing average min makespans and *Pm* for mutation type 3

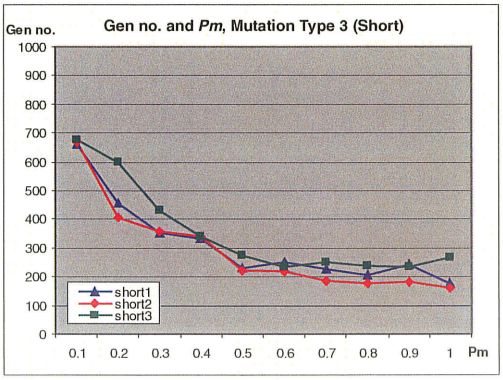


Figure 5.9(f): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 3

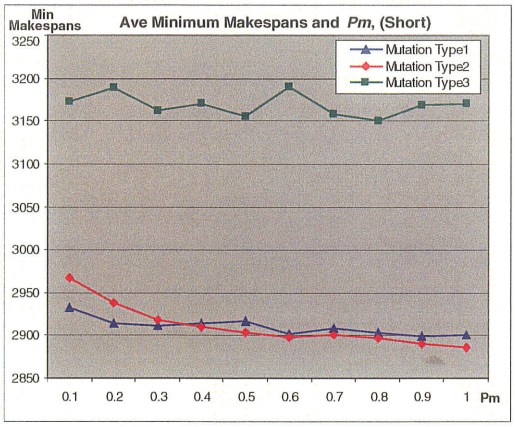


Figure 5.10(a): Graph showing the average min makespans from all three part families and *Pm* for mutation types 1, 2 and 3

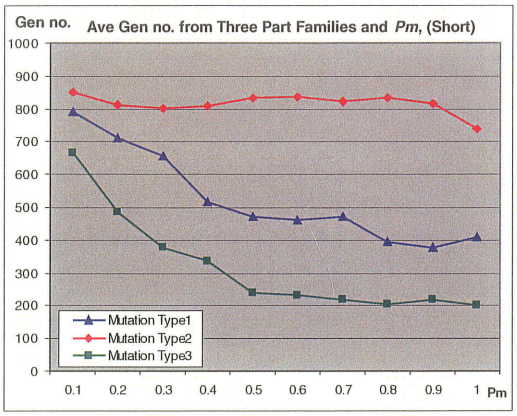


Figure 5.10(b): Graph showing the average number of generationsthat produce the min makespans from all three part families and *Pm* for mutation types 1, 2 and 3

From Figure 5.9(a, b and c) and Figure 5.10(a), one can observe that in mutation type 1 and mutation type 2, a high *Pm* value provide a slightly better result for the minimum makespans especially at a *Pm* value ranging from 0.3 to 1.0. However, in mutation type 3, a change in *Pm* does not alter the minimum makespan significantly. Furthermore, Figure 5.10(a) illustrates that mutation type 3 (the shift change mutation) provides poor results.

From Figure 5.9(d, e and f) and Figure 5.10(b), one can notice that for mutation type 1 and mutation type 3 a high *Pm* value tends to provide a more rapidly converging solution. However, this effect cannot be seen in mutation type 2. A change of *Pm* in mutation type 2 does not have a considerable effect on convergence speed. Specifically, mutation type 3 provides the best convergent rate while mutation type 2 provides the worst.

5.2.2.2 Mutation Study with Data Type: Medium

The results of the tests are shown in Figure 5.11. Figure 5.11(a, b and c) show the results between the average minimum makespans (average from ten data) and *Pm* for mutation type 1, mutation type 2 and mutation type 3, respectively of the part families med1, med2 and med3. Figure 5.11(d, e and f) show the plot between average number of generation (average from ten data) that produce the minimum makespans and *Pm* for mutation type 1, mutation type 2 and mutation type 3, respectively of the med1, med2 and med3 part family.

Similar to the study of "Short" data type, the average minimum makespans of med1, med2 and med3 part family at the same *Pm* point of each mutation type are averaged and are plotted as a single graph as shown in Figure 5.12 to study the effect of the types of mutation on the minimum makespans. Figure 5.12(a) shows the average minimum makespans from all three part families in "Medium" and *Pm* of mutation type 1, mutation type 2 and mutation type 3.

The generation numbers that produce the minimum makespans of all three part families at the same *Pm* point of each crossover type are averaged and are plotted into a single graph to study the consequence of mutation types to the convergent speed. Figure 5.12(b) shows the average generation number that produces the minimum makespan from all three part families of "Medium" and *Pm* of mutation types 1, mutation type 2 and mutation type 3.

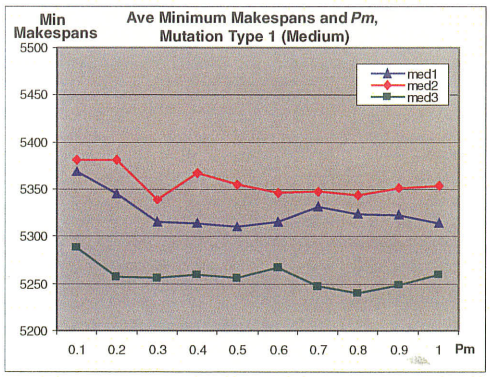


Figure 5.11(a): Graph showing average min makespans and *Pm* for mutation type 1

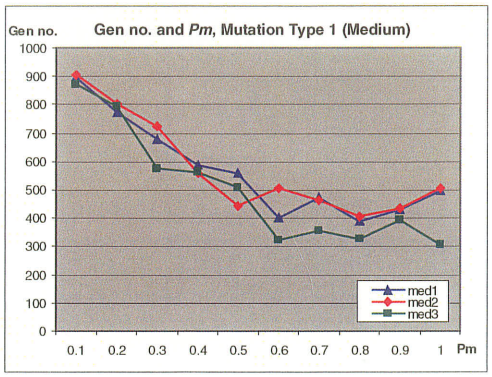


Figure 5.11(d): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 1

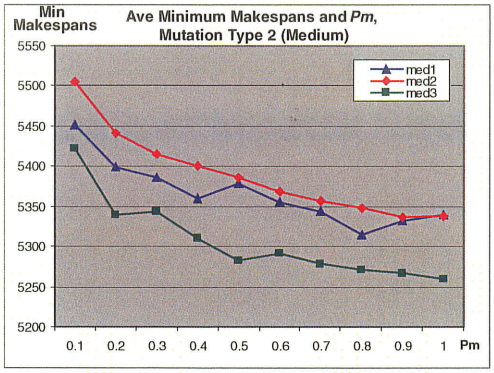


Figure 5.11(b): Graph showing average min makespans and *Pm* for mutaion type 2

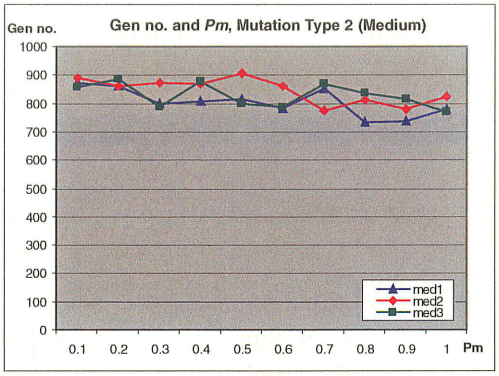


Figure 5.11(e): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 2

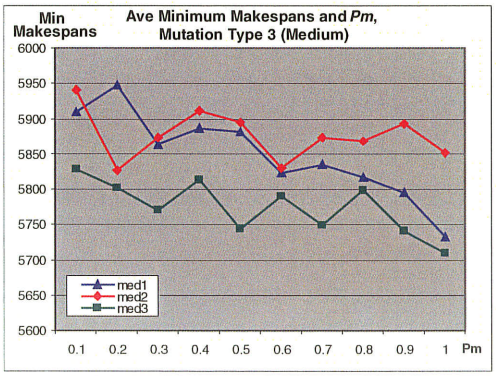


Figure 5.11(c): Graph showing average min makespans and *Pm* for mutation type 3

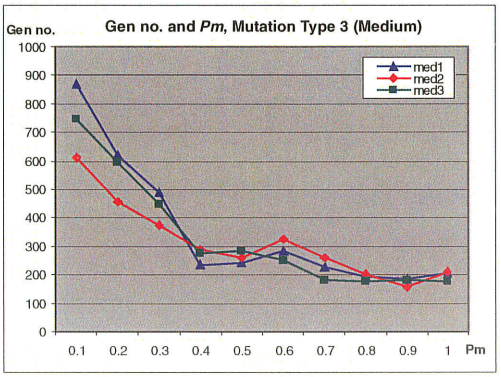


Figure 5.11(f): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 3

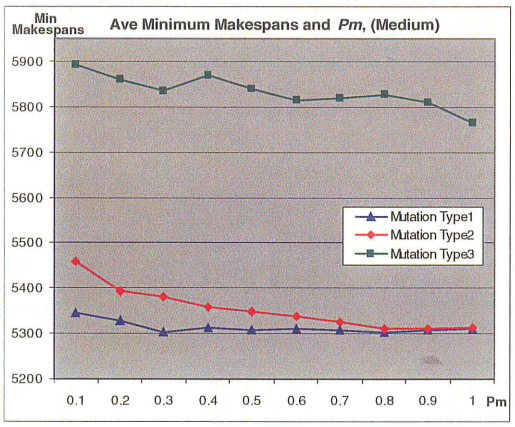


Figure 5.12(a): Graph showing the average min makespans from all three part families and *Pm* for mutation types 1, 2 and 3

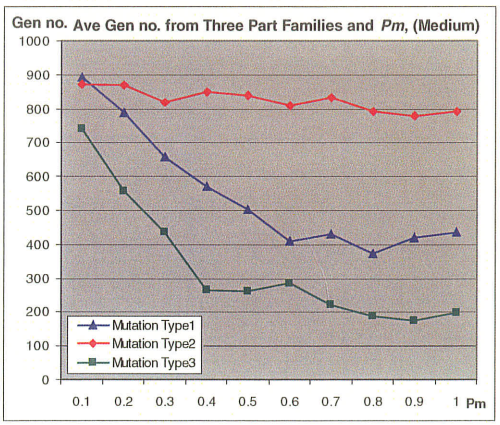


Figure 5.12(b): Graph showing the average number of generationsthat produce the min makespans from all three part families and *Pm* for mutation types 1, 2 and 3

In Figure 5.11(a, b and c) and Figure 5.12(a), one can observe that a high *Pm* value provides a better result for minimum makespans. A better result of the minimum makespans can be obtained at *Pm* value of 0.3 to 1.0 for mutation type 1, at *Pm* value of 0.8 to 1.0 for mutation type 2 and at *Pm* value of 0.6 to 1.0 for mutation type 3. Moreover, Figure 5.12(a) shows that mutation type 3 provides poor results.

Figure 5.11(d, e and f) and Figure 5.12(b) shows that in mutation type 1 and mutation type 3, a high *Pm* tends to provide the fast converging process while in mutation type 2 the change of *Pm* does not have a major effect. One can say that, mutation type 3 provides the best convergence rate while mutation type 2 provides the worst.

5.2.2.3 Mutation Study with Data Type: Mix180

The results of the tests for this study are shown in Figure 5.13. Figure 5.13(a, b and c) show the plots for the average minimum makespans (average from ten data) and *Pm* of mutation type 1, mutation type 2 and mutation type 3, respectively of the part families mix1, mix2 and mix3. Figure 5.13(d, e and f) are similar plots between average number

of generation (average from ten data) that produce the minimum makespans and *Pm* of mutation type 1, mutation type 2 and mutation type 3 respectively of the mix1, mix2 and mix3 part family.

Similar to the study done for "Short" and "Medium" data type, the average minimum makespans of mix1, mix2 and mix3 at the same *Pm* value of each mutation type are averaged and are plotted as a single graph to study the effect of the types of mutation on the minimum makespans. Figure 5.14(a) is the graph between the average minimum makespans from mix1, mix2 and mix3 and *Pm* of mutation types 1, mutation type 2 and mutation type 3.

Also, the generation numbers that produce the minimum makespans of mix1, mix2 and mix3 at the same *Pm* point of each mutation type are averaged and are plotted as a single graph to study the relationship between mutation type and the convergence speed. Figure 5.14(b) shows the average generation number that produces the minimum makespans times for mix1, mix2 and mix3 and *Pc* of mutation types 1, mutation type 2 and mutation type 3.

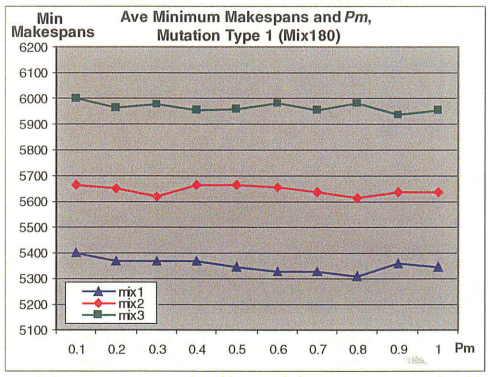


Figure 5.13(a): Graph showing average min makespans and *Pm* for mutation type 1

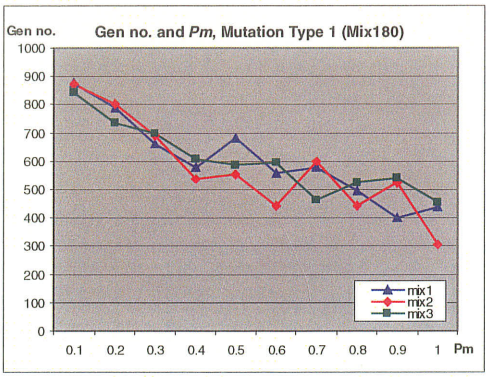


Figure 5.13(d): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 1

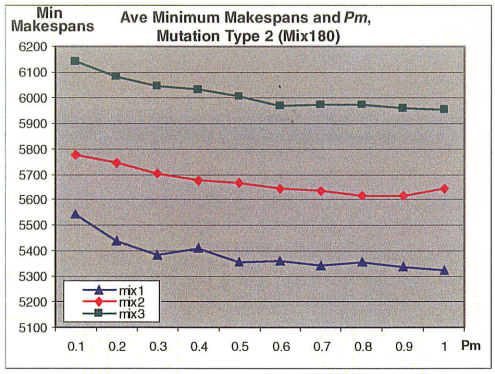


Figure 5.13(b): Graph showing average min makespans and *Pm* for mutaion type 2

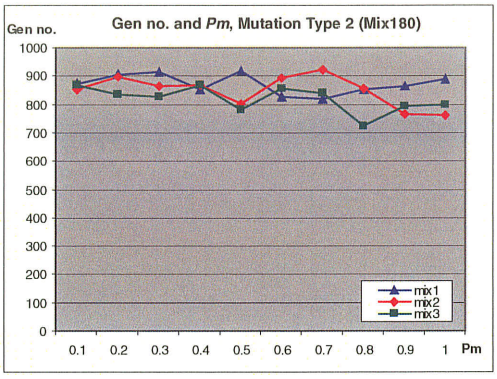


Figure 5.13(e): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 2

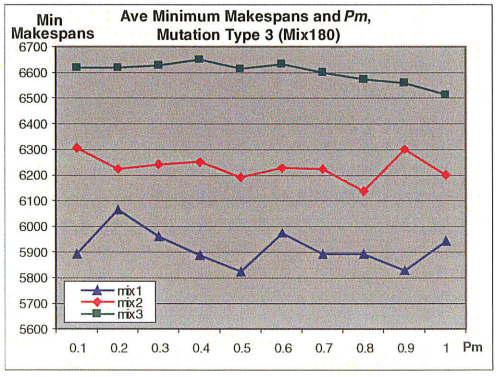


Figure 5.13(c): Graph showing average min makespans and *Pm* for mutation type 3

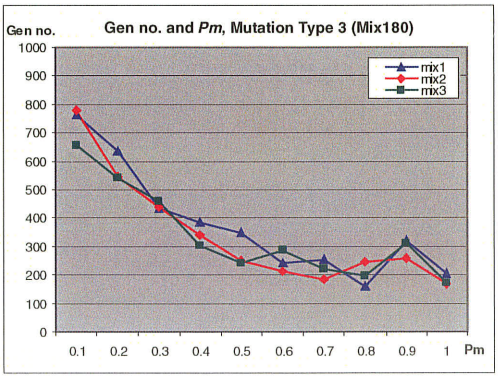


Figure 5.13(f): Graph showing number of generations that produce the min makespans and *Pm* for mutation type 3

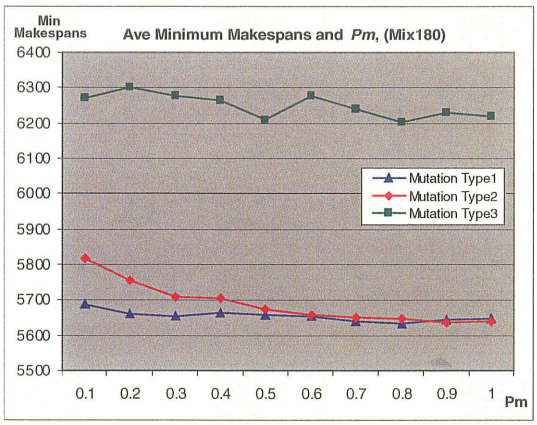


Figure 5.14(a): Graph showing the average min makespans from all three part families and *Pm* for mutation types 1, 2 and 3

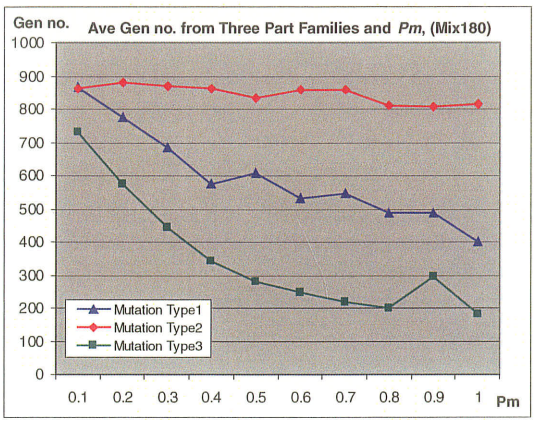


Figure 5.14(b): Graph showing the average number of generationsthat produce the min makespans from all three part families and *Pm* for mutation types 1, 2 and 3

The results for "Mix180" are very similar to the result obtained for "Medium". From Figure 5.13(a, b and c) and Figure 5.14(a), it can be observed that a high *Pm* value provides better makespan times. Specifically, the good results for the minimum makespans can be obtained at *Pm* 0.3 to 1.0 for mutation type 1, at *Pm* 0.6 to 1.0 for mutation type 2 and at *Pm* 0.8 to 1.0 for mutation type 3. In addition, graph (w) in Figure 5.11 illustrates that mutation type 3 provides poor results.

Figure 5.13(d, e and f) and Figure 5.14(b) show that in mutation type 1 and mutation type 3 a high *Pm* tends to provide a more rapidly converging process while in mutation type 2 a change in *Pm* does not produce significant change. Mutation type 3 provides the best convergent rate while mutation type 2 provides the worst.

5.2.3 Study of Population Size

To study the influence of population size, a group of chromosomes called initial chromosomes are needed. As mentioned in Section 4.2.2, initial chromosomes are created randomly. The number of chromosomes created is called population size. Population size is also the number that limits the amount of new chromosomes that will carry forward to the next generation of the evolution process. Therefore, the population size is a very important specification. As stated in Section 5.2, population size is one of the necessary parameters that have to be input before performing the analysis.

To study the influence of population size on the best makespans and the convergent rate, various population sizes with values 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 were selected. Similar to the crossover and mutation studies, all nine part families from the three data types (Short, Medium, and Mix180) were analyzed using ten population sizes for the genetic parameter shown below.

Crossover probability:	0.5
Crossover type:	One-Point Crossover
Mutation probability:	0.5
Mutation type:	Arbitrary Two-Job Exchange Mutation

The results of the test are divided into three groups: short, medium and mix180 based on the data types. The results are presented in a graphical form as before. Similar to the results shown for crossover and mutation, two types of results can be analyzed. The first type is a plot between the minimum makespan and population sizes, to study the effect of the population size on the minimum makespan. The second type is the graph between

the generation numbers that produce the minimum makespan and population size to evaluate the effect of the population size on the convergence rate.

The results from the tests are shown in Figure 5.15. Figure 5.15(a, b and c) show the results for the average minimum makespans (average from ten data) and population size (which is referred as "pop size" in graphs) of part families from data type "Short", "Medium" and "Mix180" respectively. Figure 5.15(d, e and f) show the average number of generation (average from ten data) that produce the minimum makespans and population size of part families from data type "Short", "Medium" and "Mix180" respectively. The results of the population size study are presented below.

In order to evaluate the effect of the population size and its influence on the minimum makespans, the average minimum makespans of all part families at the same population size of each data type are averaged and plotted into one graph as shown in Figure 5.16. Figure 5.16(a) shows the plot between the average minimum makespans from all three part families and population size in "Short", "Medium" and "Mix180" data type.

In order to study the effect of population size on the convergent rate, the generation numbers that produce the minimum makespan at the same population size of each data type are averaged and are plotted into a single graph as presented in Figure 5.16. Figure 5.16(b) shows the average generation number that produces the minimum makespan from all three part families in "Short" data type families and population size in "Short", "Medium" and "Mix180" data type. Figure 5.17 shows the relation between the average times that are used to obtain the result from GA for various population sizes (chosen as 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50). The purpose of the graph is to compare the makespan obtained for each population size.

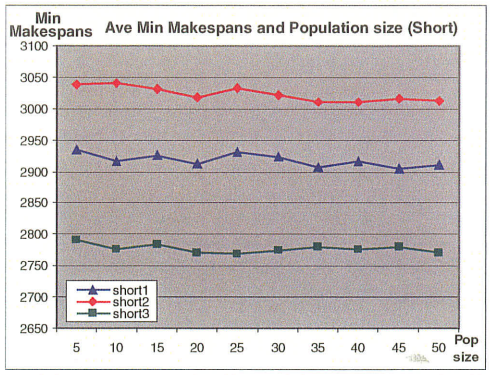


Figure 5.15(a): Graph showing average min makespans and popopulaiton size for data type "Short"

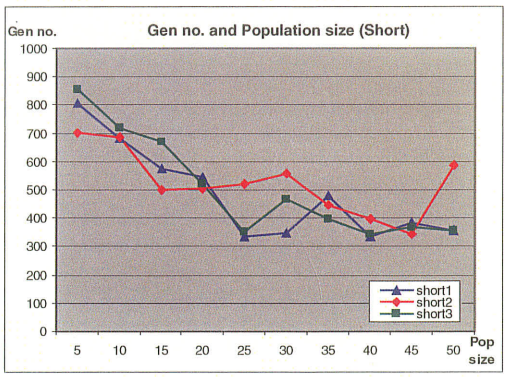


Figure 5.15(d): Graph showing number of generations that produce the minimum makespans and population size for data type "Short"

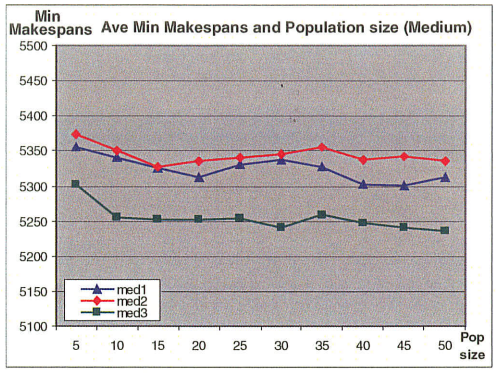


Figure 5.15(a): Graph showing average min makespans and popopulaiton size for data type "Medium"

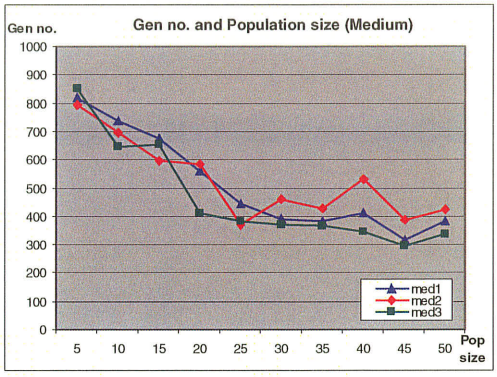


Figure 5.15(d): Graph showing number of generations that produce the minimum makespans and population size for data type "Medium"

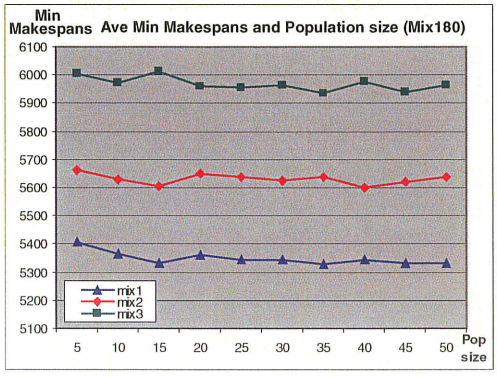


Figure 5.15(a): Graph showing average min makespans and popopulaiton size for data type "Mix180"

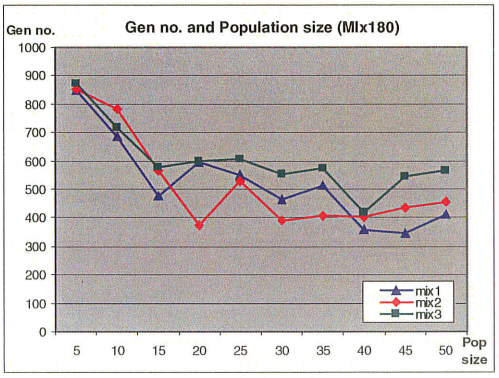


Figure 5.15(d): Graph showing number of generations that produce the minimum makespans and population size for data type "Mix180"

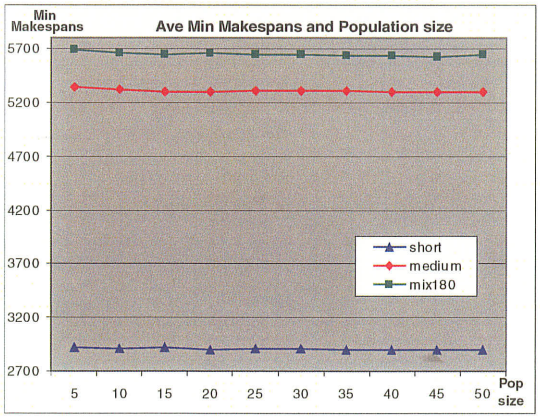


Figure 5.16(a): Graph showing the average min makespans from all three part families in each data type and Population size

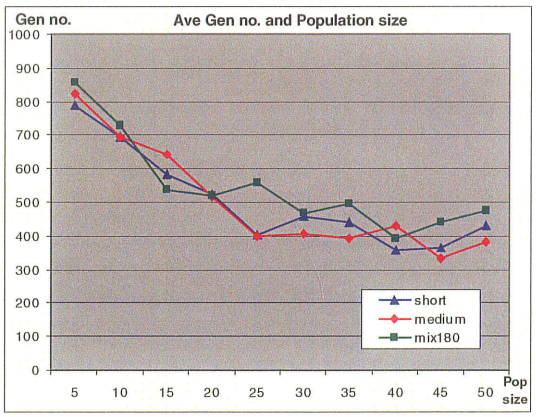


Figure 5.16(b): Graph between the average number of generationsthat produce the min makespans from all three part families in each data type and Population size

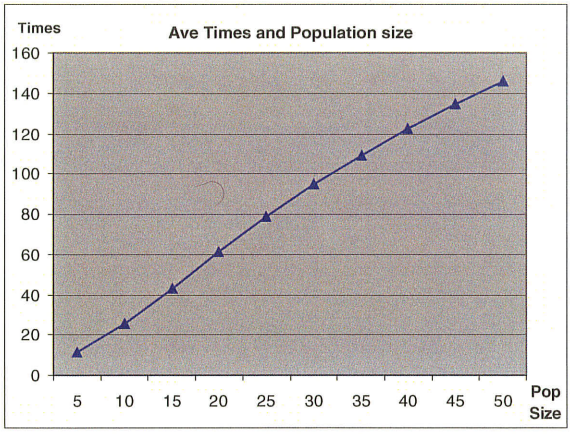


Figure 5.17: Graph showing the average times that are used to obtain the resulf from GA program and Population size

From graphs 5.15(a, b and c) and Figure 5.16(a), one can observe a population size of 5 provides a slightly poorer result for the minimum makespans in every data types. Figure 5.15(d, e and f) and Figure 5.16(b) indicate that at population size from 5 to 25, the convergent rate increases as the population size increases. However, at population size from 30 to 50, the graphs show that the change in population size produce very minimal change in converging speed.

The GA program was tested using an AMD Athlon 64 Processor 3200+, 2.01 GHz, and 2.5 GB of Ram. The tested data contained 40 parts in part family with a population size of 20. The computational time was 0.0607 second per generation.

Chapter 6 Conclusions and Recommendations

This thesis has developed a GA based methodology for scheduling parts through a flexible manufacturing cell with a robot performing all the material handling related tasks. The methodology investigated feasible solutions for obtaining minimum makespan in a 'different part type' production operation wherein parts are required to go through a set of machines with each having its own sequence and different processing times. The problem is complex and a single analytical solution for this type of problem was not found in existing literature. Hence, no comparisons could be made with any previously published work. The proposed GA approach also investigated the influence of different genetic parameters while obtaining the minimum makespan for producing the family of parts. The results from the study have lead to the following conclusions.

- Crossover/ mutation operators and population size for the three data types show that all three test data types provide similar results for every genetic operator considered.
- The effect of the genetic parameters on the minimum makespan and the generation number producing the minimum makespan is not strongly influenced by the different types of data considered.
- All three crossover types, namely: (i) one-point crossover, (ii) two-point crossover (version I) and (iii) two-point crossover (version II) provided similar results for the minimum makespan and convergent rate at any *Pc*. A low value of *Pc* provides a slightly better makespan values while a high *Pc* gave a better convergent speed in every crossover type.

- In regards to mutation parameters and their influence, a high *Pm* provided a better makespan in every mutation type. In addition, a high *Pm* resulted in a better convergence speed in mutation type 1 and mutation type 3. The mutation type 3 (shift change mutation) provided a poor result for the minimum makespan while the mutation type 2 (arbitrary three-job exchange mutation) provided a poor convergent rate.
- In regards to the population size, a high value of population provided a better convergent speed. However it also results increased computational times. An appropriate choice of the different parameters is listed on next page.
- From the results one can conclude that, irrespective of the type of crossover considered, *Pc* values ranging from 0.3 to 0.5 should be selected in order to obtain the minimum makespan.
- The *Pm* values ranging from 0.4 to 1.0 with mutation type 1 (arbitrary two-job exchange mutation) provided a better minimum makespan than mutation type 3 and presents the shorter convergence time than mutation type 2.
- In regards to the population size, ranges from 20 – 35 produced good results within a reasonable generation time.
- In conclusion, the specification of the genetic operators that has a tendency to provide a high performance in part scheduling problem can be summarized as:

Crossover probability:	0.3 – 0.5
Crossover type:	Any type
Mutation probability:	0.4 – 1.0
Mutation type:	Arbitrary Two-Job Exchange Mutation
Population size:	20 - 35
Number of Evaluations:	600 – 700

A step by step procedure for obtaining the sequences for a part data is provided in Appendix C.

It is recommended that extension to this study be undertaken to include multiple objectives such as minimizing idle time of machines, maximization of robot usage, customer imposed constraints such as meeting early due dates for some of the parts, minimization of penalty etc. In its present form the methodology does not detect any dead-lock states in machines. This can also be included in future studies. It should be noted that the problem size will become even more complex when some or all of these are included in the model. However genetic algorithms have the power to expand the horizon of search space and may possibly lead to more powerful solutions.

References

[1] A. Adlemo, "Balanced automation in flexible manufacturing systems", Studies in Informatics and Control, Vol.5, pp.179-187, 1996.

[2] G. Qiao, R. F. Lu and C. Mclean, "Flexible manufacturing system for mass customization manufacturing", International Journal of Mass Customization, Vol.1, pp.374-393, 2006.

[3] T. Luan, "Scheduling in robotic cells with two and three machines", M.Sc.Thesis - University of Manitoba, pp.1-81, 2005.

[4] M. Savsar, "Reliability analysis of a flexible manufacturing cell", Reliability Engineering and System Safety 67, pp.147-152, 2000.

[5] N. G. Hall, H. Kamoun, and C. Sriskandarajah, "Scheduling in robotic cells: classification, two and three machine cells", Operations Research, Vol.45, pp.421-439, 1997.

[6] M. S. Akturk, H. Gultekin, and O. E. Karasan, "Robotic cell scheduling with operational flexibility", Discrete Applied Mathematics, Vol.145, pp.334-348, 2005.

[7] H. Chen, C. Chu and J. Proth, "Sequencing of Parts in Robotic Cells", International Journal of Flexible Manufacturing Systems, Vol.9, pp.81-104, 1997.

[8] S. P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz and W. Kubiak, "Sequencing of parts and robot moves in a robotic cell", International Journal of Flexible Manufacturing Systems, Vol.4, pp.331-358, 1992.

[9] A. Agnetis and D. Pacciarelli, "Part sequencing in three-machine no-wait robotic cells", Operations Research Letters, Vol.27, pp.185-192, 2000.

[10] W. Ying and L. Bin, "Job-shop scheduling using genetic algorithm", Signal Processing 3rd International Conference, pp.1441- 1444, 1996.

[11] C. G. Wu, X. L. Xing, H. P. Lee, C. G. Zhou and Y. C. Liang, "Genetic algorithm application on the job shop scheduling problem", International Conference on Machine Learning and Cybernetics, pp.2102-2106, 2004.

[12] I. Kacem, "Genetic algorithm for the flexible job-shop scheduling problem", IEEE International Conference on Systems, Man and Cybernetics, Vol.4, pp.3464-3469, 2003.

[13] N. Morad and A. Zalzala, "A genetic-based approach to the formation of manufacturing cell and batch scheduling" , Proceedings of the IEEE Conference on Evolutionary Computation, pp.485-490, 1996.

[14] T. Murata and H. Ishibuchi, "Performance evaluation of genetic algorithms for flowshop scheduling problems", Proceedings of the IEEE Conference on Evolutionary Computation, pp.812-817, 1994.

[15] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies", Computers & Industrial Engineering, pp.343-364, 1999.

[16] Y. Yin, J. Yu and Z. Cheng, "A genetic algorithm based approach to flowshop scheduling", World Congress on Intelligent Control and Automation, pp.3019-3021, 2004.

[17] S. Forrest, "Genetic algorithms: principles of natural selection applied to computation", Science, Vol.261, pp.872-878, 1993.

[18] M. Mitchell, "An introduction to genetic algorithms", MIT Press, pp.167-168, 1996.

[19] Y. Wu, M. Liu and C. Wu, "Genetic algorithm for solving flow shop scheduling problem with parallel machine and special procedure constraints", International Conference on Machine Learning and Cybernetics, pp.1774-1779, 2003.

[20] T. Luan and Q. Peng, "Genetic algorithm-based parts scheduling in a two-machine robotic Cell", Flexible Automation and Intelligent Manufacturing, pp.974-980, 2004.

[21] S. Forrest, "Genetic algorithms: principles of natural selection applied to computation" Science, Vol.261, pp.872-878,1993.

[22] R. Simon, "Robust encodings in genetic algorithms: a survey of encoding issues", Proceedings of IEEE International Conference on Evolutionary Computation, pp.43-48, 1997.

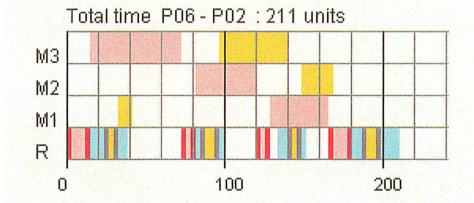
Appendix A

Time charts of all chromosome pairs of part family F2 (tested by using genetic probabilities, namely [*Pc*, *Pm*] = [0.6, 0.4]) in graph 5-1(b) in Chapter 5, Section 5.1 are shown below.

Time Chart of pair: P06 and P02

P06 = M3 (58) → M2 (38) → M1 (37)

P02 = M1 (9) → M3 (43) → M2 (20)



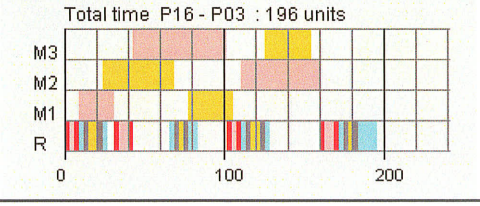
The sequence of the robot moves of pair grouping between part P06 and P02

1. (r) Pick up P06 at M0
2. (p) Move P06 to M3
3. (r) Load P06 on M3
4. (b) Move back to M0
5. (g) Pick up P02 from M0
6. (y) Move P02 to M1
7. (g) Load P02 to M1
8. (b) Move back to M3
9. Wait at M3 until P06 finished
10. (r) Unload P06 from M3
11. (p) Move P06 to M2
12. (r) Load P06 to M2
13. (b) Move back to M1
14. (g) Unload P02 from M1
15. (y) Move P02 to M3
16. (g) Load P02 on M3
17. (b) Move back to M2
18. Wait at M2 until P06 finished
19. (r) Unload P06 from M2
20. (p) Move P06 to M1
21. (r) Load P06 on M1
22. (b) Move back to M3
23. (g) Unload P02 from M3
24. (y) Move P02 to M2
25. (g) Load P02 at M2
26. (b) Move back to M1
27. Wait at M1 until P06 finished
28. (r) Unload P06 from M1
29. (p) Move P06 to M4
30. (r) Drop off P06 at M4
31. (b) Move back to M2
32. (g) Unload P02 from M2
33. (y) Move P02 to M4
34. (g) Drop off P02 at M4
35. (b) Move back to M0

Time Chart of pair: P16 and P03

P16 = M1 (22) → M3 (59) → M2 (49)

P03 = M2 (45) → M1 (28) → M3 (29)



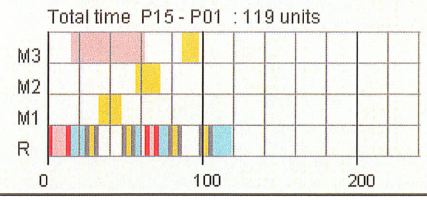
The sequence of the robot moves of pair grouping between part P16 and P03

1. (r) Pick up P16 at M0
2. (p) Move P16 to M1
3. (r) Load P16 on M1
4. (b) Move back to M0
5. (g) Pick up P03 at M0
6. (y) Move P03 to M2
7. (g) Load P03 on M2
8. (b) Move back to M1
9. Wait at M1 until P16 finished
10. (r) Unload P16 from M1
11. (p) Move P16 to M3
12. (r) Load P16 on M3
13. (b) Move back to M2
14. Wait at M2 until P03 finished
15. (g) Unload P03 from M2
16. (y) Move P03 to M1
17. (g) Load P03 at M1
18. (b) Move back to M3
19. Wait at M3 until P16 finished
20. (r) Unload P16 from M3
21. (p) Move P16 to M2
22. (r) Load P16 at M2
23. (b) Move back to M1
24. (g) Unload P03 from M
25. (y) Move P03 to M3
26. (g) Load P03 at M3
27. (b) Move back to M2
28. Wait at M2 until P16 finished
29. (r) Unload P16 from M2
30. (p) Move P16 to M4
31. (r) Drop off P16 at M4
32. (b) Move back to M3
33. (g) Unload P03 from M3
34. (y) Move P03 to M4
35. (g) Drop off P03 at M4
36. Move back to M0

Time Chart of pair: P15 and P01

P15 = M3 (48)

P01 = M1 (15) → M2 (16) → M3 (11)



(r) = red
(p) = pink
(g) = gray
(y) = yellow
(b) = blue

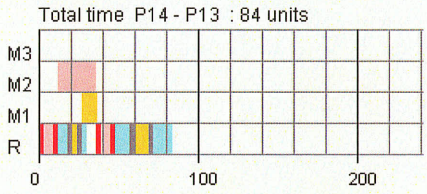
The sequence of the robot moves of pair grouping between part P15 and P01

1. (r) Pick up P15 at M0
2. (p) Move P15 to M3
3. (r) Load P15 on M3
4. (b) Move back to M0
5. (g) Pick up P01 at M0
6. (y) Move P01 to M1
7. (g) Load P01 on M1
8. Wait at M1 until P01 finished
9. (g) Unload P01 from M1
10. (y) Move P01 to M2
11. (g) Load P01 at M2
12. (b) Move back to M3
13. Wait at M3 until P15 finished
14. (r) Unload P15 from M3
15. (p) Move P15 to M4
16. (r) Drop off P15 at M4
17. (b) Move back to M2
18. (g) Unload P01 from M2
19. (y) Move P01 to M3
20. (g) Load P01 to M3
21. Wait at M3 until P01 finished
22. (g) Unload P01 from M3
23. (y) Move P01 to M4
24. (g) Drop off P01 to M4
25. (b) Move back to M0

Time Chart of pair: P14 and P13

P14 = M2 (24)

P13 = M1 (10)



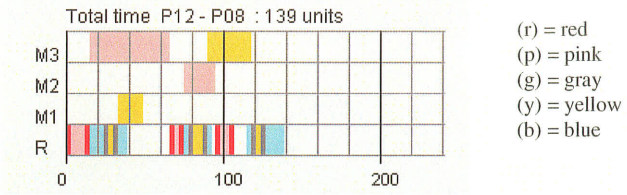
The sequence of the robot moves of pair grouping between part P14 and P13

1.(r) Pick up P14 at M0
2.(p) Move P14 to M2
3.(r) Load P14 on M2
4.(b) Move back to M0
5.(g) Pick up P13 at M0
6.(y) Move P13 to M1
7.(g) Load P13 on M1
8.(b) Move back to M2
9. Wait at M2 until P14 finished
10.(r) Unload P14 from M2
11.(p) Move P14 to M4
12.(r) Drop off P14 at M4
13.(b) Move back to M1
14.(g) Unload P13 from M1
15.(y) Move P13 to M4
16.(g) Drop off P13 at M4
17.(b) Move back to M0

Time Chart of pair: P12 and P08

P12 = M3 (51) → M2 (20)

P08 = M1 (16) → M3 (28)



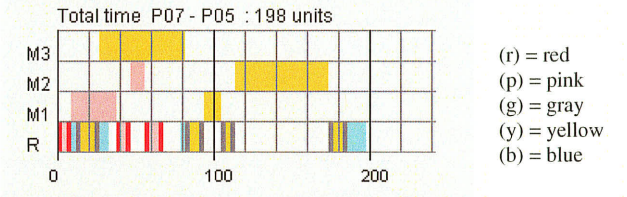
The sequence of the robot moves of pair grouping between part P12 and P08

1.(r)	Pick up P12 at M0	13.(b)	Move back to M1	25.(y)	Move P08 to M4
2.(p)	Move P12 to M3	14.(g)	Unload P08 from M1	26.(g)	Drop off P08 at M4
3.(r)	Load P12 on M3	15.(y)	Move P08 to M3	27.(b)	Move back to M0
4.(b)	Move back to M0	16.(g)	Unload P08 to M3		
5.(g)	Pick up P08 at M0	17.(b)	Move back to M2		
6.(y)	Move P08 to M1	18.	Wait at M2 until P12 finished		
7.(g)	Load P08 on M1	19.(r)	Unload P12 from M2		
8.(b)	Move back to M3	20.(p)	Move P12 to M4		
9.	Wait at M3 until P12 finished	21.(r)	Drop off P12 at M4		
10.(r)	Unload P12 from M3	22.(b)	Move back to M2		
11.(p)	Move P12 to M2	23.	Wait at M2 until P08 finished		
12.(r)	Load P12 on M2	24.(g)	Unload P08 from M2		

Time Chart of pair: P07 and P05

P07 = M1 (29) → M2 (9)

P05 = M3 (55) → M1 (11) → M2 (60)



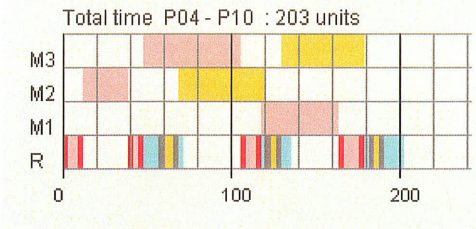
The sequence of the robot moves of pair grouping between part P07 and P05

1.(r)	Pick up P07 at M0	13.	Wait at M2 until P07 finished	25.(g)	Load P05 on M2
2.(p)	Move P07 to M1	14.(r)	Unload P07 from M2	26.	Wait at M2 until P05 finished
3.(r)	Load P07 on M1	15.(p)	Move P07 to M4	27.(g)	Unload P05 from M2
4.(b)	Move back to M0	16.(r)	Drop off P07 on M4	28.(y)	Move P05 to M4
5.(g)	Pick up P05 at M0	17.(b)	Move back to M3	29.(g)	Drop off P05 at M4
6.(y)	Move P05 to M3	18.	Wait at M3 until P05 finished	30.(b)	Move back to M0
7.(g)	Load P05 on M3	19.(g)	Unload P05 from M3		
8.(b)	Move back to M1	20.(y)	Move P05 to M1		
9.	Wait at M3 until P07 finished	21.(g)	Load P05 on M1		
10.(r)	Unload P07 from M1	22.	Wait at M1 until P05 finished		
11.(p)	Move P07 to M2	23.(g)	Unload P05 from M1		
12.(r)	Load P07 on M2	24.(y)	Move P05 to M2		

Time Chart of pair: P04 and P10

P04 = M2 (27) → M3 (58) →M1 (46)

P10 = M2 (51) → M3 (49)



(r) = red
(p) = pink
(g) = gray
(y) = yellow
(b) = blue

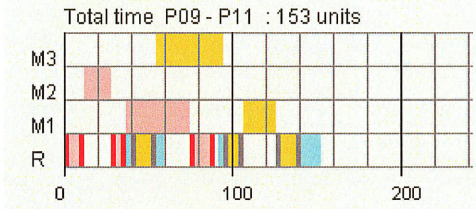
The sequence of the robot moves of pair grouping between part P04 and P10

1.(r)	Pick up P04 at M0	13.	Wait at M3 until P04 finished	25.(r)	Drop off P04 at M4
2.(p)	Move P04 to M2	14.(r)	Unload P04 from M3	26.(b)	Move back to M3
3.(r)	Load P04 on M2	15.(p)	Move P04 to M1	27.(g)	Unload P10 from M3
4.	Wait at M2 until P04 finished	16.(r)	Load P04 at M1	28.(y)	Move P10 to M4
5.(r)	Unload P04 from M2	17.(b)	Move back to M2	29.(g)	Drop off P10 at M4
6.(p)	Move P04 to M3	18.(g)	Unload P10 from M2	30.(b)	Move back to M0
7.(r)	Load P04 at M3	19.(y)	Move P10 to M3		
8.(b)	Move back to M0	20.(g)	Load P10 on M3		
9.(g)	Pick up P10 at M0	21.(b)	Move back to M1		
10.(y)	Move P10 to M2	22.	Wait at M1 until P04 finished		
11.(g)	Load P10 at M2	23.(r)	Unload P04 from M1		
12.(b)	Move back to M3	24.(p)	Move P04 to M4		

Time Chart of pair: P09 and P11

P09 = M2 (16) → M1 (38)

P11 = M3 (40) → M1 (19)



(r) = red
(p) = pink
(g) = gray
(y) = yellow
(b) = blue

The sequence of the robot moves of pair grouping between part P09 and P

1.(r)	Pick up P09 at M0	13.	Wait at M1 until P09 finished	25.(g)	Drop off P11 at M1
2.(p)	Move P09 to M2	14.(r)	Unload P09 from M1	26.(b)	Move back to M0
3.(r)	Load P09 on M2	15.(p)	Move P09 to M4		
4.	Wait at M2 until P09 finished	16.(r)	Drop off P09 at M4		
5.(r)	Unload P09 from M2	17.(b)	Move back to M3		
6.(p)	Move P09 to M1	18.	Wait at M3 until P11 finished		
7.(r)	Load P09 at M1	19.(g)	Pick up P11 from M3		
8.(b)	Move back to M0	20.(y)	Move P11 to M1		
9.(g)	Pick up P11 at M0	21.(g)	Load P11 at M1		
10.(y)	Move P11 to M3	22.	Wait at M1 until P11 finished		
11.(g)	Load P11 at M3	23.(g)	Pick up P11 from M1		
12.(b)	Move back to M1	24.(y)	Move P11 to M4		

Appendix B

The information for part family: short1

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	5	M2	30	M3	10
P2	M1	30	M3	40	M2	50
P3	M2	10	M1	25	M3	60
P4	M2	20	M3	12	M1	20
P5	M3	30	M1	40	M2	50
P6	M3	15	M2	5	M1	45
P7	M1	20	M2	50	None	-
P8	M1	60	M3	15	None	-
P9	M2	45	M1	30	None	-
P10	M2	20	M3	5	None	-
P11	M3	55	M1	10	None	-
P12	M3	20	M2	60	None	-
P13	M1	25	None	-	None	-
P14	M2	35	None	-	None	-
P15	M3	3	None	-	None	-
P16	M1	60	M2	60	M3	60
P17	M1	20	M3	45	M2	10
P18	M2	55	M1	40	M3	12
P19	M2	15	M3	15	M1	15
P20	M3	25	M1	35	M2	30
P21	M3	10	M2	8	M1	6
P22	M1	35	M2	40	None	-
P23	M1	10	M2	20	None	-
P24	M2	30	M1	30	None	-
P25	M2	45	M3	9	None	-
P26	M3	22	M1	10	None	-
P27	M3	50	M2	18	None	-
P28	M1	60	None	-	None	-
P29	M2	55	None	-	None	-
P30	M3	45	None	-	None	-
P31	M1	5	None	-	None	-
P32	M2	10	None	-	None	-
P33	M3	8	None	-	None	-
P34	M1	40	M2	25	M3	40
P35	M1	45	M3	50	None	-
P36	M2	60	M3	35	M1	10
P37	M3	55	M1	15	None	-
P38	M2	45	M1	40	None	-
P39	M3	25	M1	45	None	-
P40	M2	25	M1	12	M3	30

The information for part family: short2

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	5	M2	30	M3	10
P2	M1	30	M3	40	M2	50
P3	M2	10	M1	25	M3	60
P4	M2	20	M3	12	M1	20
P5	M3	30	M1	40	M2	50
P6	M3	15	M2	5	M1	45
P7	M1	20	M2	50	None	-
P8	M1	60	M3	15	None	-
P9	M2	45	M1	30	None	-
P10	M2	20	M3	5	None	-
P11	M3	55	M1	10	None	-
P12	M3	20	M2	60	None	-
P13	M1	25	None	-	None	-
P14	M2	35	None	-	None	-
P15	M3	3	None	-	None	-
P16	M1	60	M2	60	M3	60
P17	M1	20	M3	45	M2	10
P18	M2	55	M1	40	M3	12
P19	M2	15	M3	15	M1	15
P20	M3	25	M1	35	M2	30
P21	M3	10	M2	8	M1	6
P22	M1	35	M2	40	None	-
P23	M1	10	M2	20	None	-
P24	M2	30	M1	30	None	-
P25	M2	45	M3	9	None	-
P26	M3	22	M1	10	None	-
P27	M3	50	M2	18	None	-
P28	M1	60	None	-	None	-
P29	M2	55	None	-	None	-
P30	M3	45	None	-	None	-
P31	M1	5	None	-	None	-
P32	M2	10	None	-	None	-
P33	M3	8	None	-	None	-
P34	M1	40	M2	25	M3	40
P35	M1	45	M3	50	None	-
P36	M2	60	M3	35	M1	10
P37	M3	55	M1	15	None	-
P38	M2	45	M1	40	None	-
P39	M3	25	M1	45	None	-
P40	M2	25	M1	12	M3	30

The information for part family: short3

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	19	M2	57	M3	37
P2	M1	4	M3	40	M2	14
P3	M2	10	M1	60	M3	48
P4	M2	6	M3	53	M1	42
P5	M3	23	M1	24	M2	36
P6	M3	5	M2	39	M1	21
P7	M1	8	M2	25	None	-
P8	M1	28	M3	49	None	-
P9	M2	28	M1	1	None	-
P10	M2	37	M3	28	None	-
P11	M3	28	M1	8	None	-
P12	M3	44	M2	5	None	-
P13	M1	43	None	-	None	-
P14	M2	2	None	-	None	-
P15	M3	48	None	-	None	-
P16	M1	17	M2	45	M3	31
P17	M1	30	M3	36	M2	44
P18	M2	34	M1	20	M3	2
P19	M2	8	M3	12	M1	30
P20	M3	42	M1	41	M2	13
P21	M3	15	M2	24	M1	27
P22	M1	11	M2	49	None	-
P23	M1	15	M2	48	None	-
P24	M2	3	M1	23	None	-
P25	M2	10	M3	14	None	-
P26	M3	11	M1	41	None	-
P27	M3	8	M2	16	None	-
P28	M1	38	None	-	None	-
P29	M2	30	None	-	None	-
P30	M3	33	None	-	None	-
P31	M1	22	None	-	None	-
P32	M2	41	None	-	None	-
P33	M3	12	None	-	None	-
P34	M1	17	M2	14	M3	10
P35	M1	25	M3	44	None	-
P36	M2	24	M3	13	M1	30
P37	M3	57	M1	37	None	-
P38	M2	17	M1	27	None	-
P39	M3	52	M1	34	None	-
P40	M2	40	M1	54	M3	26

The information for part family: med1

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	65	M2	75	M3	100
P2	M1	90	M3	80	M2	120
P3	M2	60	M1	120	M3	60
P4	M2	110	M3	95	M1	85
P5	M3	100	M1	100	M2	100
P6	M3	65	M2	65	M1	75
P7	M1	110	M2	90	None	-
P8	M1	120	M3	95	None	-
P9	M2	60	M1	85	None	-
P10	M2	85	M3	120	None	-
P11	M3	60	M1	60	None	-
P12	M3	105	M2	115	None	-
P13	M1	75	None	-	None	-
P14	M2	75	None	-	None	-
P15	M3	75	None	-	None	-
P16	M1	120	M2	120	M3	120
P17	M1	60	M3	60	M2	60
P18	M2	105	M1	75	M3	75
P19	M2	110	M3	85	M1	90
P20	M3	100	M1	70	M2	80
P21	M3	100	M2	65	M1	115
P22	M1	100	M2	75	None	-
P23	M1	95	M2	80	None	-
P24	M2	115	M1	120	None	-
P25	M2	75	M3	75	None	-
P26	M3	120	M1	80	None	-
P27	M3	90	M2	65	None	-
P28	M1	120	None	-	None	-
P29	M2	120	None	-	None	-
P30	M3	120	None	-	None	-
P31	M1	90	None	-	None	-
P32	M2	80	None	-	None	-
P33	M3	115	None	-	None	-
P34	M1	90	M2	90	M3	90
P35	M1	85	M3	70	None	-
P36	M2	75	M3	60	M1	60
P37	M3	120	M1	60	None	-
P38	M2	115	M1	80	None	-
P39	M3	70	M2	85	None	-
P40	M2	115	M1	65	M3	80

The information for part family: med2

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	92	M2	93	M3	92
P2	M1	79	M3	86	M2	83
P3	M2	101	M1	70	M3	65
P4	M2	102	M3	101	M1	82
P5	M3	102	M1	66	M2	103
P6	M3	61	M2	85	M1	106
P7	M1	108	M2	86	None	-
P8	M1	118	M3	110	None	-
P9	M2	93	M1	118	None	-
P10	M2	76	M3	103	None	-
P11	M3	62	M1	88	None	-
P12	M3	86	M2	110	None	-
P13	M1	75	None	-	None	-
P14	M2	89	None	-	None	-
P15	M3	103	None	-	None	-
P16	M1	86	M2	85	M3	62
P17	M1	98	M3	66	M2	72
P18	M2	116	M1	89	M3	60
P19	M2	110	M3	92	M1	83
P20	M3	96	M1	63	M2	107
P21	M3	78	M2	111	M1	108
P22	M1	90	M2	61	None	-
P23	M1	119	M2	89	None	-
P24	M2	109	M1	67	None	-
P25	M2	69	M3	116	None	-
P26	M3	112	M1	87	None	-
P27	M3	98	M2	111	None	-
P28	M1	86	None	-	None	-
P29	M2	74	None	-	None	-
P30	M3	78	None	-	None	-
P31	M1	80	None	-	None	-
P32	M2	116	None	-	None	-
P33	M3	64	None	-	None	-
P34	M1	64	M2	61	M3	79
P35	M1	113	M3	103	None	-
P36	M2	112	M3	84	M1	73
P37	M3	70	M1	113	None	-
P38	M2	90	M1	89	None	-
P39	M3	104	M1	118	None	-
P40	M2	93	M1	61	M3	76

The information for part family: med3

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	78	M2	85	M3	83
P2	M1	106	M3	94	M2	85
P3	M2	119	M1	72	M3	66
P4	M2	105	M3	98	M1	69
P5	M3	82	M1	87	M2	83
P6	M3	65	M2	113	M1	89
P7	M1	73	M2	62	None	-
P8	M1	77	M3	84	None	-
P9	M2	113	M1	65	None	-
P10	M2	81	M3	118	None	-
P11	M3	65	M1	76	None	-
P12	M3	118	M2	78	None	-
P13	M1	93	None	-	None	-
P14	M2	97	None	-	None	-
P15	M3	71	None	-	None	-
P16	M1	96	M2	66	M3	114
P17	M1	85	M3	114	M2	97
P18	M2	98	M1	78	M3	77
P19	M2	70	M3	61	M1	63
P20	M3	101	M1	107	M2	64
P21	M3	90	M2	77	M1	111
P22	M1	120	M2	62	None	-
P23	M1	118	M2	73	None	-
P24	M2	74	M1	108	None	-
P25	M2	63	M3	104	None	-
P26	M3	88	M1	69	None	-
P27	M3	85	M2	98	None	-
P28	M1	84	None	-	None	-
P29	M2	93	None	-	None	-
P30	M3	84	None	-	None	-
P31	M1	106	None	-	None	-
P32	M2	105	None	-	None	-
P33	M3	85	None	-	None	-
P34	M1	117	M2	86	M3	60
P35	M1	65	M3	111	None	-
P36	M2	88	M3	61	M1	104
P37	M3	89	M1	64	None	-
P38	M2	99	M1	100	None	-
P39	M3	73	M1	85	None	-
P40	M2	115	M1	95	M3	104

The information for part family: mix1

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	5	M2	105	M3	20
P2	M1	65	M3	150	M2	30
P3	M2	100	M1	55	M3	150
P4	M2	15	M3	90	M1	20
P5	M3	180	M1	150	M2	165
P6	M3	20	M2	15	M1	90
P7	M1	10	M2	100	None	-
P8	M1	90	M3	35	None	-
P9	M2	155	M1	145	None	-
P10	M2	65	M3	78	None	-
P11	M3	20	M1	95	None	-
P12	M3	15	M2	145	None	-
P13	M1	15	None	-	None	-
P14	M2	20	None	-	None	-
P15	M3	35	None	-	None	-
P16	M1	170	M2	35	M3	150
P17	M1	45	M3	95	M2	155
P18	M2	30	M1	155	M3	40
P19	M2	160	M3	15	M1	100
P20	M3	75	M1	75	M2	60
P21	M3	85	M2	70	M1	35
P22	M1	160	M2	90	None	-
P23	M1	165	M2	55	None	-
P24	M2	25	M1	65	None	-
P25	M2	15	M3	20	None	-
P26	M3	50	M1	15	None	-
P27	M3	90	M2	180	None	-
P28	M1	175	None	-	None	-
P29	M2	155	None	-	None	-
P30	M3	145	None	-	None	-
P31	M1	75	None	-	None	-
P32	M2	90	None	-	None	-
P33	M3	80	None	-	None	-
P34	M1	65	M2	45	M3	165
P35	M1	180	M3	145	None	-
P36	M2	110	M3	35	M1	85
P37	M3	15	M1	105	None	-
P38	M2	115	M1	25	None	-
P39	M3	95	M2	175	None	-
P40	M2	45	M1	145	M3	65

The information for part family: mix2

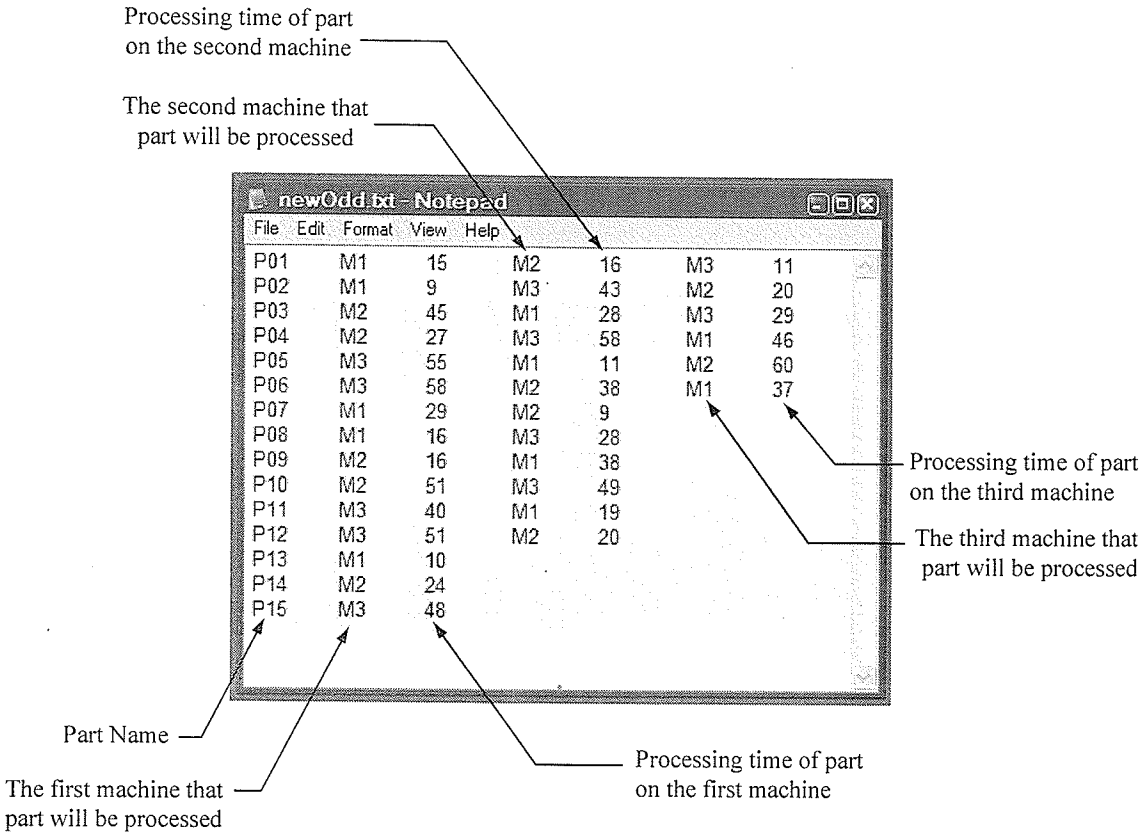
Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	1	M2	63	M3	24
P2	M1	168	M3	59	M2	145
P3	M2	105	M1	6	M3	56
P4	M2	38	M3	89	M1	37
P5	M3	170	M1	55	M2	57
P6	M3	27	M2	110	M1	96
P7	M1	89	M2	125	None	-
P8	M1	153	M3	83	None	-
P9	M2	24	M1	161	None	-
P10	M2	38	M3	144	None	-
P11	M3	121	M1	88	None	-
P12	M3	43	M2	6	None	-
P13	M1	134	None	-	None	-
P14	M2	161	None	-	None	-
P15	M3	1	None	-	None	-
P16	M1	69	M2	116	M3	163
P17	M1	152	M3	83	M2	39
P18	M2	48	M1	44	M3	175
P19	M2	55	M3	108	M1	4
P20	M3	90	M1	65	M2	96
P21	M3	94	M2	151	M1	166
P22	M1	23	M2	47	None	-
P23	M1	31	M2	136	None	-
P24	M2	152	M1	51	None	-
P25	M2	153	M3	123	None	-
P26	M3	109	M1	55	None	-
P27	M3	104	M2	45	None	-
P28	M1	147	None	-	None	-
P29	M2	3	None	-	None	-
P30	M3	156	None	-	None	-
P31	M1	36	None	-	None	-
P32	M2	19	None	-	None	-
P33	M3	79	None	-	None	-
P34	M1	79	M2	167	M3	60
P35	M1	157	M3	168	None	-
P36	M2	133	M3	72	M1	108
P37	M3	23	M1	103	None	-
P38	M2	54	M1	92	None	-
P39	M3	160	M1	52	None	-
P40	M2	180	M1	175	M3	105

The information for part family: mix3

Part Name	First Process		Second Process		Third Process	
	Machine	Time	Machine	Time	Machine	Time
P1	M1	142	M2	49	M3	91
P2	M1	157	M3	148	M2	29
P3	M2	154	M1	179	M3	171
P4	M2	42	M3	115	M1	171
P5	M3	59	M1	164	M2	155
P6	M3	5	M2	23	M1	74
P7	M1	37	M2	151	None	-
P8	M1	32	M3	123	None	-
P9	M2	137	M1	3	None	-
P10	M2	33	M3	113	None	-
P11	M3	26	M1	74	None	-
P12	M3	147	M2	163	None	-
P13	M1	178	None	-	None	-
P14	M2	5	None	-	None	-
P15	M3	114	None	-	None	-
P16	M1	165	M2	163	M3	67
P17	M1	117	M3	132	M2	56
P18	M2	64	M1	109	M3	93
P19	M2	83	M3	157	M1	161
P20	M3	93	M1	72	M2	51
P21	M3	71	M2	163	M1	126
P22	M1	31	M2	95	None	-
P23	M1	36	M2	5	None	-
P24	M2	101	M1	99	None	-
P25	M2	164	M3	54	None	-
P26	M3	29	M1	146	None	-
P27	M3	178	M2	111	None	-
P28	M1	61	None	-	None	-
P29	M2	174	None	-	None	-
P30	M3	152	None	-	None	-
P31	M1	179	None	-	None	-
P32	M2	117	None	-	None	-
P33	M3	76	None	-	None	-
P34	M1	120	M2	73	M3	28
P35	M1	169	M3	92	None	-
P36	M2	6	M3	138	M1	113
P37	M3	60	M1	167	None	-
P38	M2	49	M1	158	None	-
P39	M3	161	M1	82	None	-
P40	M2	21	M1	77	M3	11

Appendix C

1. Prepare part family data as shown below in .txt format



2. Input robot information into the GA program

Robot pick up / unloading time (second)

Robot drop off/ loading time (second)

Robot movement time between two adjacent stations (second)

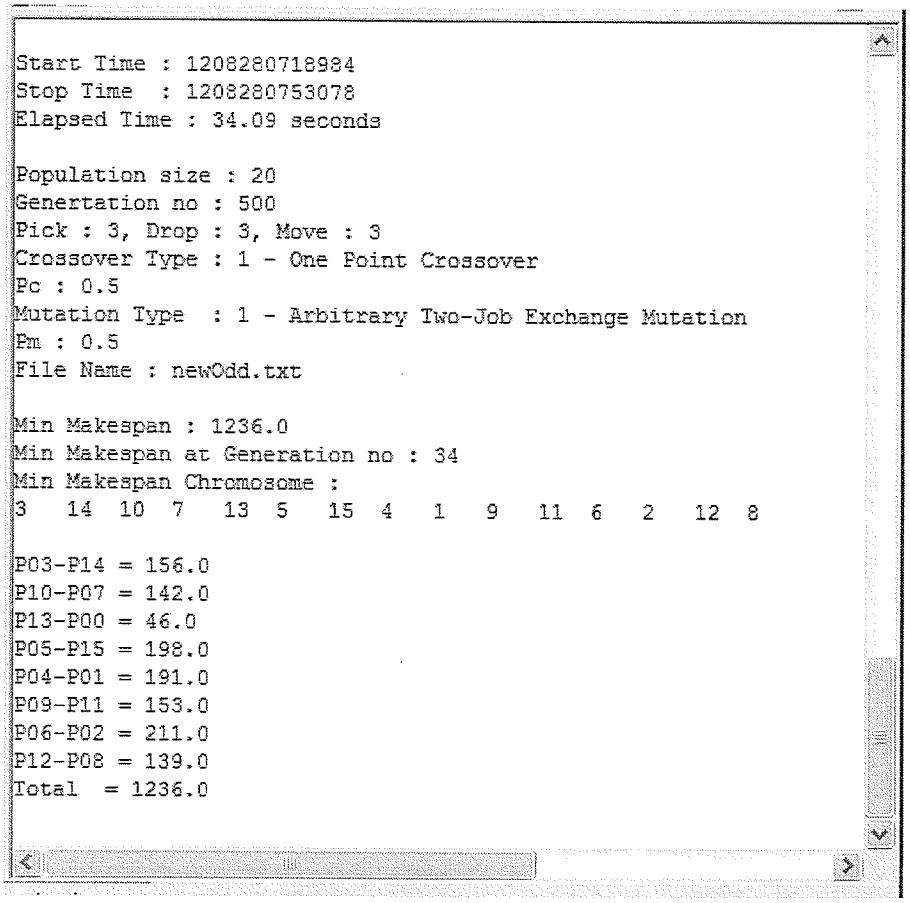
3. Input the necessary parameters into the GA program

File name:	filename.txt (prepared form step 1)
Crossover probability:	0.3 – 0.5
Crossover type:	Any type
Mutation probability:	0.4 – 1.0
Mutation type:	Arbitrary Two-Job Exchange Mutation
Population size:	20 - 35
Number of Evaluations:	600 – 700

4. Run GA program

5. Get the result from GA program

The result will provide the minimum makespan, number of generation that produces min makespan, chromosome that produces the minimum makespan, its pair grouping detail and the time charts. The result examples are as shown below.

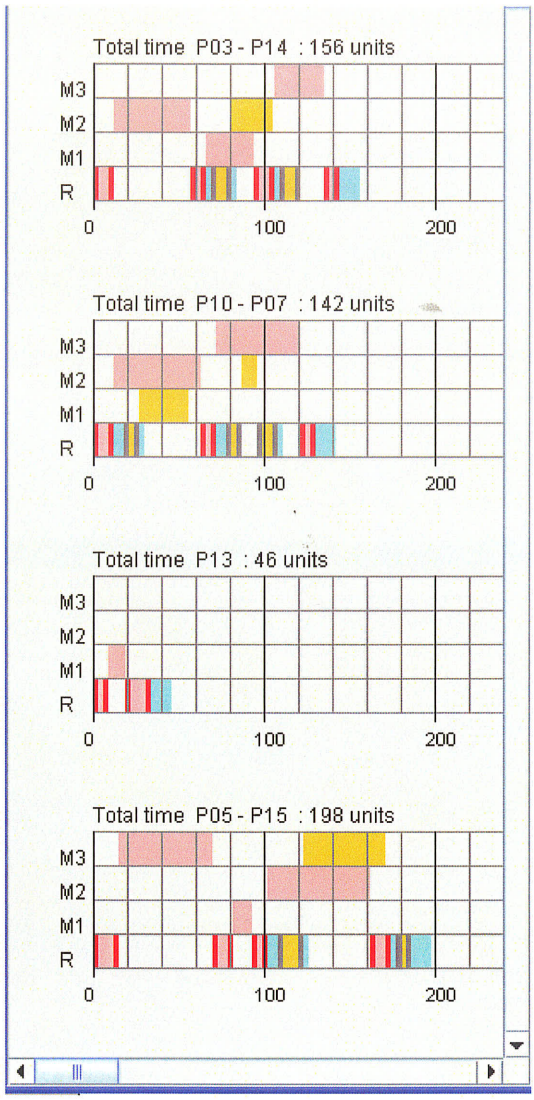


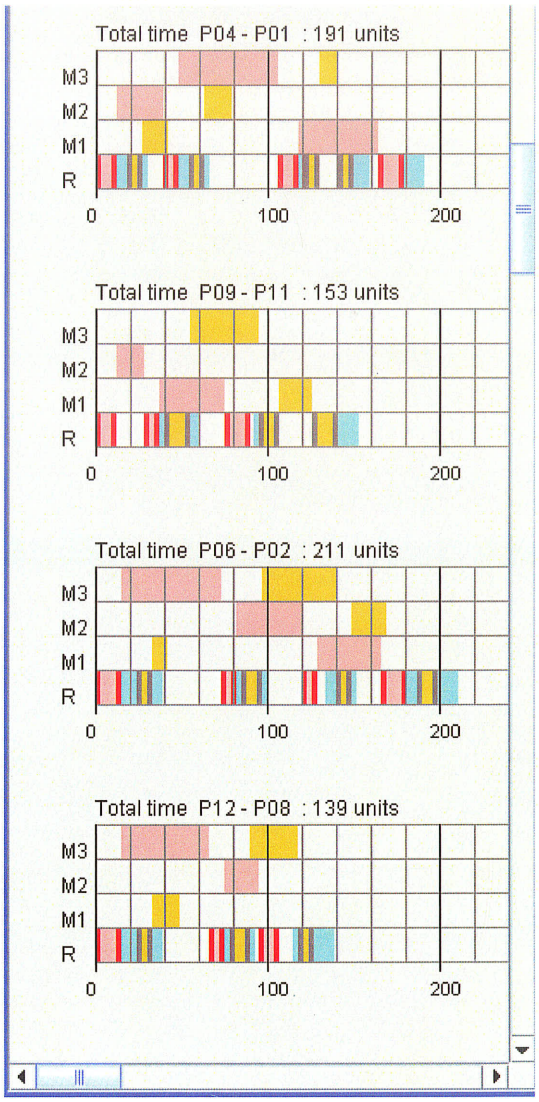
```
Start Time : 1208280718984
Stop Time  : 1208280753078
Elapsed Time : 34.09 seconds

Population size : 20
Genertation no : 500
Pick : 3, Drop : 3, Move : 3
Crossover Type : 1 - One Point Crossover
Pc : 0.5
Mutation Type  : 1 - Arbitrary Two-Job Exchange Mutation
Pm : 0.5
File Name : newOdd.txt

Min Makespan : 1236.0
Min Makespan at Generation no : 34
Min Makespan Chromosome :
3   14  10  7   13  5   15  4   1   9   11  6   2   12  8

P03-P14 = 156.0
P10-P07 = 142.0
P13-P00 = 46.0
P05-P15 = 198.0
P04-P01 = 191.0
P09-P11 = 153.0
P06-P02 = 211.0
P12-P08 = 139.0
Total  = 1236.0
```





(The above result use move, drop, pick times = 3 seconds.)