

南 華 大 學

資訊管理學系

碩士論文

以知識為基礎之基因演算法於 JSP 問題上的應用

A knowledge-based genetic algorithm for the job  
shop schedule problem



研 究 生：簡琬蓉

指 導 教 授：邱宏彬

中 華 民 國 95 年 6 月 3 日

南 華 大 學  
資訊管理研究所  
碩 士 學 位 論 文

A Knowledge-Based Genetic Algorithm for the Job Shop  
Scheduling Problem

研究生：簡 琬 蓉

經考試合格特此證明

口試委員：


謝 品 霖

李 翔 詣

\_\_\_\_\_

\_\_\_\_\_

指導教授：邱 宏 村

系主任(所長)：

口試日期：中華民國 95 年 6 月 3 日

# 以知識為基礎之基因演算法於 JSP 問題上的應用

學生：簡琬蓉

指導教授：邱宏彬

南 華 大 學 資 訊 管 理 學 系 碩 士 班

## 摘 要

本論文提出一新的改良式基因演算法 (KGA)，此演算法透過由傳統基因演算法的結果搭配屬性的辨識去收集知識，並利用知識引導 KGA 的過程與交配時基因優良度的評估。此外，為了避免因為知識的應用而使演算過程容易落入區域最佳解，本研究利用突變的方式做區域搜尋，並在結果確定落入區域最佳解時，重新置換母體及替換舊有知識，藉由這些改變來使得本演算法可以同時兼顧集中性和多樣性。最後透過實驗的結果證實，本演算法確實是穩定的且可以找出不錯的排程，而知識的應用也可有效的提供引導的資訊。

關鍵字：零工式排程問題、改良式基因演算法、知識

# A knowledge-based Genetic Algorithm for the job shop scheduling problem

Student : Wan-Jung Chien

Advisors : Dr. Hung-Pin Chiu.

Department of Information Management  
The M.B.A. Program  
Nan-Hua University

## ABSTRACT

This study presents a novel use of attribution for the extraction of knowledge from job shop scheduling problem. Our algorithm improves the traditional GA and using knowledge to keep the quality of solution. Based on the knowledge, the search space will be led to a better search space. In addition, this study uses mutation to do local search and refresh the knowledge and population when the solution fall into local minimum. Based on those methods, our algorithm will have the intensification and diversification. Those can make the algorithm have good convergence and leap for the search space to find the better solution. The experiment results show that algorithm steadily and can find the approximate optimal solution. And the knowledge is useful in provide the gene selection information.

Key words: JSP problem, hybrid GA, knowledge

# LIST OF CONTENT

<b>CHAPTER 1 INTRODUCTION.....</b>	<b>1</b>
<b>CHAPTER 2 RELATED WORKS .....</b>	<b>4</b>
2.1 JOB SHOP SCHEDULING PROBLEM.....	4
2.2 GENETIC ALGORITHM.....	5
2.2.1 Crossover operator .....	5
2.2.2 Mutation operator .....	6
2.3 GA-BASED JSP ALGORITHM.....	6
2.3.1 Encoding of chromosome .....	7
2.3.2 Decoding of chromosome.....	7
<b>CHAPTER 3 RESEARCH APPROACH .....</b>	<b>9</b>
3.1 DATA PREPARATION.....	9
3.1.1 Attribution.....	9
3.1.2 Operations Table.....	11
3.2 COLLECT KNOWLEDGE.....	12
3.3 DESIGNING CROSSOVER OPERATOR .....	14
3.3.1 Eugenic crossover .....	15
3.3.2 Adjust child .....	16
<b>CHAPTER 4 THE FLOWCHART OF ALGORITHM .....</b>	<b>18</b>
4.1 COLLECT KNOWLEDGE BY SOLUTION FOR GA .....	18
4.1.1 Collect solutions by GA.....	18
4.1.2 Analyze solution to collect knowledge.....	20
4.2 KNOWLEDGE-BASED GA .....	22
4.2.1 Blended crossover .....	23
4.2.2 Forced mutation .....	24
4.2.3 Collect or replace new knowledge .....	25
4.2.4 The flowchart of KGA .....	25
<b>CHAPTER 5 EXPERIENTIAL RESULTS.....</b>	<b>29</b>
5.1 ESTABLISH PARAMETER .....	30
5.2 EVALUATE OFFSPRING.....	31
5.3 COMPARE WITH GA .....	33
5.4 10×10 BENCHMARK PROBLEM.....	34
<b>CHAPTER 6 CONCLUSIONS AND DISCUSSIONS .....</b>	<b>38</b>

**REFERENCE ..... 41**

## LIST OF TABLES

Table 2.1	JSP problem.....	5
Table 3.1	Classes substitutions.....	11
Table 3.2	Example of operations table.....	12
Table 3.3	Example of attribute association in the gene position.....	14
Table 5.1	Offspring evaluation.....	32
Table 5.2	Progress of the 10×10 benchmark instance.....	36

## LIST OF FIGURES

Fig. 2.1	Job shop scheduling gene.....	7
Fig. 2.2	Decoded active schedule.....	8
Fig. 3.1	Example of eugenic crossover.....	15
Fig. 3.2	Example of adjusted child.....	17
Fig. 4.1	The flowchart of collecting knowledge.....	22
Fig. 4.2	The flowchart of KGA.....	28
Fig. 5.1	Crossover probability = 0.6 (GA) .....	30
Fig. 5.2	Crossover probability =0.7(GA) .....	31
Fig. 5.3	Crossover probability =0.8(GA) .....	31
Fig. 5.4	The result of KGA*, KGA** and GA (run 200 times) .....	33
Fig. 5.5	Makespan of case 1.....	34
Fig. 5.6	Makespan of case 2.....	34
Fig. 5.7	The makespan of the10×10 benchmark problem.....	35
Fig. 5.8	The makespan for KGA (run 100 times) .....	37



# Chapter 1 Introduction

Scheduling problems exist everywhere in real-world circumstance, especially in the flexible manufacturing world. Many people pay close attention to it because poor scheduling can lead to higher cost for manufacturers and consequently higher prices for customer. Therefore, if we want to have the better efficiency, we must have a good schedule to promote the efficiency and reduce the time in the manufacturing process. Nevertheless, scheduling problems are categorized into different groups in the different machine environment (e.g., single machine problems, parallel machine problems, job shop problems, etc.). In those groups, job shop problem (JSP) emphasizes the order of job in the every machine. In the other word, it considers the order of every operator of job but not prescribes which machine is the first machine for the job. As a result, JSP is more complicated than other scheduling problem. Therefore, this study has focused on the JSP problem.

JSP is among the hardest combinational optimization problem [21]. Most of the researches used different approaches to solved JSP such as: Tabu search [8, 16], simulated annealing [22, 26], ant colony system [1], neural network algorithm [3], genetic algorithm (GA) [7, 9, 28, 18], and others. GA-based approach was used to solve JSP problem considerably in recent years among these studying. Cheng, Gen, and Tsujimura [24] have given a detailed sort survey on papers using GA to solve classical JSP in Part I survey. Nevertheless, using traditional GA can't give consideration convergence rate, quality of solution and stability of search process. On the other hand, this

algorithm can't balance intensification and diversification. Ignore the intensification will spend much time to search. And disregard the diversification will fall into local minimum easily. So many researches tried modify GA with other algorithm. Cheng [25] discuss various hybrid GA to solve JSP.

In recent years, many researches wanted to improve intensification or diversification. Mattfeld and Bierwirth [5] used a heuristic reduction of search space which can help GA to find better solution in a shorter computation time. Goncalves, Mendes, and Resende [15] constructed the scheduling to generate parameterized active schedules and used a local search heuristic to improve the solution in evolutionary process of GA. To sum up, there studies focused on improving the search space. Therefore, better solutions could be expected but the quality of solutions could not be guarded. Watanabe, Ida and Gen [20] use GA with modified crossover operator for JSP problem. It made use of random number to decide what gene must be reserved for children chromosome. If the offspring do not conform to constrain the JSP problem, it will be regulate by some rules. This paper changed the traditional crossover operator and considered influence of each gene. Nevertheless, using the random number to decide which gene can be retained to offspring did not exclude random effect.

In order to keep the quality of solutions, some studies used the better chromosome to replace the bad chromosome. This method is accomplished by first coping some of the best individuals from each generation to the next, in what is called an elitist strategy [9]. Chang, Hsieh and Hsiao [23] reserved some better chromosomes and replaced some bad chromosomes in each

generation. Those methods supposed that if there is a better population, it will be easy to produce the better offspring in crossover operator. However, it was not exactly so and it may be easy to fall into local minimum. For this reason, we propose the idea that if we can evaluate the fitness for genes and choose the better gene to generate the offspring which may lead to a better solution. And if reserving the better chromosomes can help the quality of solution, those better chromosomes may have useful information for finding the better solution.

Based on those ideas, we will collect some best solutions by GA to sort some knowledge and use it to evaluate the fitness for gene. And then make use of concluded result to design the suitable crossover operator for JSP problem. Hope to use this idea to speed up the convergence and improve the solution for JSP problem. Besides, we use mutation to do the local search, hope this can keep the diversification and avoid intensification overly. We will describe the design and the logic behind this method. And use the experiment to demonstrate the feasibility. This research is a new attempt and which can apply to other optimization problem. Therefore, it is a very important problem and merit discussion about it.

In this paper, we present a new knowledge-based genetic algorithm (KGA) for solving the job shop scheduling problem. The paper will be divided into the following sections. Section 2 describes the related work of the problem. Section 3 will describe the flow of the knowledge-based GA. Section 4 shows the several experiment results in the comparison to those of existing GA. Section 5 makes conclude this paper.

## Chapter 2 Related works

### 2.1 Job shop scheduling problem

JSP problem has been described as follows [6]: there are  $m$  different jobs and  $n$  different machines to be scheduled. Each job is composed of a set of operation and the operation order on each machine is prespecified. The required machine and the fixed processing time characterize each operation. There are several constraints on jobs and machines:

- | A job does not visit the same machine twice.
- | There are no precedence constraints among the operations of different jobs.
- | Operations cannot be interrupted.
- | Each machine can process only one job at a time.
- | Neither release times nor due dates are specified.

A schedule is an allocation of the operations to time intervals on the machines. According to the allocated operation sequences in a schedule, the time required to complete all jobs is called makespan of the schedule. Table 2.1 shows a 3×3 JSP problem and concluded operations, job number, machine number, process time. For example, when we observe J1 and O1, it means that operation 1 of job 1 be arranged for machine 2 (M2) and spend 2 time units.

Table 2.1 JSP problem

	<b>Operations</b>		
<b>Job</b>	<b>O1</b>	<b>O2</b>	<b>O3</b>
<b>J1</b>	M2(2)	M3(3)	M1(7)
<b>J2</b>	M3(1)	M1(9)	M2(3)
<b>J3</b>	M1(3)	M2(8)	M3(5)

## 2.2 Genetic algorithm

Genetic algorithm is search algorithm developed to explain and simulate the mechanisms of natural systems. In GA applications, variables of the solution are encoded into a structures string that presents a list of genes. A fitness function is also required, which assigns a figure of merit to each encoded solution. During the run, parents must be selected for reproduction, and recombined or mutated to generate offspring. The GA uses a measure of fitness of individual chromosome to carry out reproduction. As reproduction takes place, the crossover operator exchanges parts of two single chromosomes and the mutation operator changes the gene value in some randomly chosen location of the chromosome. As a result, after a number of successive reproductions, the less fit chromosomes become extinct, while those best able to survive gradually come to dominate the population.

### 2.2.1 Crossover operator

Crossover operator was used to recombine parents and generate offspring. There are many studies on the design of crossover operator. Partially Mapped Crossover (PMX) copies a section of genes from one parent and the rest by

position-wise exchanging [10]. Order crossover (OX) can be viewed as a kind of variation of PMX that used a different repairing procedure [17]. And, Order-based Crossover (OC2) is a slight variation of position-based crossover in that the order of symbols in the selected position in one parent is imposed on the corresponding ones in the other parent [12].

### **2.2.2 Mutation operator**

Some mutation operators have been proposed for permutation representation. Inversion mutation selects two positions, within a chromosome at random and then inverts the substring between selects two positions. Shift mutation first chooses a gene randomly and shifts it to a random position of right or left from the gene's position [19].

## **2.3 GA-based JSP algorithm**

The JSP problem has captured the interest of a significant number of researches and a lot of literature has been published, but no efficient solution algorithm has found yet for solving it to optimality in polynomial time. For this reason, there are many researches use the GA or hybrid GA to solve the JSP problem. For example, Kennedy provided the offspring will pass on its traits acquired during this local optimization to future offspring. Giffler and Thompson used to deduce a schedule from the encoding of priority dispatching rules. Roughly is to combine the GA with beam search technique in JSP problem [24]. Those algorithms must be used in executing GA. So encoding the chromosome, decoding chromosome and representing the

chromosome must be understood.

### 2.3.1 Encoding of chromosome

Using genetic algorithm to solve problem, solutions must be encoded in a format that allows for the operations of crossover and mutation. This research uses Syswerda's [13] list of order operations representation as the gene model for JSP problem. The  $3 \times 3$  JSP is encoded into a 9 integer array. For example, a solution might be the sequence:  $\{1,3,3,2,1,2,1,2,3\}$ . This sequence is called a chromosome. Each element in the array corresponds to a job. Successive references to a job in the array imply the next available operation for that job. Fig. 2.1 shows the schedule ordering from the previous solution sequence.

Job	1	3	3	2	1	2	1	2	3
Operation	1	1	2	1	2	2	3	3	3

Fig. 2.1 Job shop scheduling gene

### 2.3.2 Decoding of chromosome

Every chromosome represents a schedule. We must decode a chromosome to its corresponding schedule to evaluate the makespan. The schedule with the shortest makespan is regarded as the final solution for the given JSP problem. The makespan was calculated by order of operations for chromosome. For example,  $o_{jim}$  denote the  $i$ th operation of job  $j$  on machine  $m$  [24]. The chromosome shown in Fig 2.1 can be translated into a unique list of ordered operations of  $\{o_{112}, o_{123}, o_{131}, o_{213}, o_{221}, o_{232}, o_{331}, o_{322}, o_{333}\}$ . Operation  $o_{112}$  has the highest priority and is scheduled first, then  $o_{123}$ , and so on. The

resulting active schedule is shown in Fig 2.2.

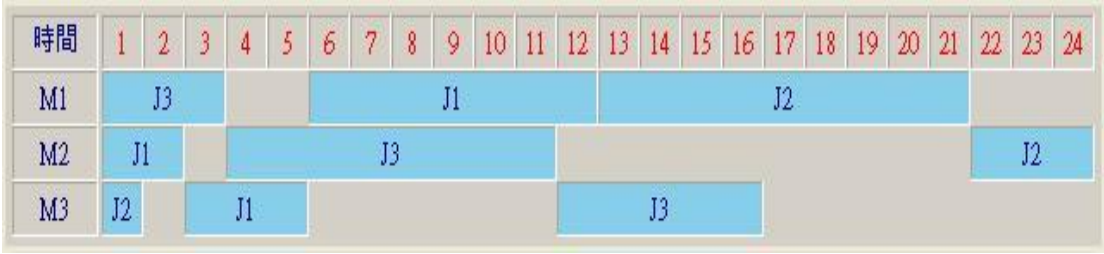


Fig. 2.2 Decoded active schedule



## **Chapter 3 Research approach**

Our algorithm was modified GA's deficiency. We use some better solutions (chromosomes) to collect knowledge and designing a eugenic crossover. However, those better solutions just bring the limited information. We can understand the machine number and processing time for this operation, but we can not collect the integrate information. Therefore, we design the operation table to classify those operations before collecting knowledge. Use knowledge to decide the fitness of gene in crossover operator and adjust the chromosome.

### **3.1 Data preparation**

Hsieh and Hsiao [23] reserved some better chromosomes and replace some bad chromosome in each generation and improve the solution of GA. According to this result, we can assume that those better chromosomes may be included some useful information for improve solution. But GA can not demonstrate repeat-ability or provide an explanation of how a solution is developed. For this reason, we can't induce information from the solutions of GA. Therefore, we will use the method which was brought up by Koonce and Tsai [6]. This method used attributions to induce information from the solution of GA.

#### **3.1.1 Attribution**

Koonce and Tsai [6] proposed that the JSP problem could be stored in another relation with the following structure: {Job, Operation, Machine, and

Process}. Therefore, they used five attributions to do hierarchies which were related to the JSP problem. Every attribution had classes itself. In this study, we use the attribution to determine the gene in each gene position. This idea could not decide the priority of gene. So we will use four of those attributions to make the attribution table be used in the next research phase. The following classification hierarchies are based on the  $3 \times 3$  JSP problem and four attributions is listed as following:

- (1) Operation: The Operation attribute is an ordinal variable representing the sequence of the operation in the job. It was divided into three classes if operation would be adequate for induction. Operation 1 was classified as “first”, operation 2 was classified as “middle”, and operation 3 was classified as “later”.
- (2) Process time: Process represent the time for processing for that particular operation. It was classified three classes: the first  $1/3$  as “short”, the second  $1/3$  as “middle”, and the third as “long”. For the  $3 \times 3$  cases studied, processing times ranged from 1 to 9 time units. A simple division of time is when time less than 3 is “short”; long than 3 and short than 6 is “middle”; and long than 6 is “long”.
- (3) Remaining process time: Remaining process represents the cumulative processing time for all subsequent operations for that job. Because it’s range is domain, so we will find the maximum remaining process time and use maximum remaining process time to divide into three classes: the first  $1/3$  as “short”, the second  $1/3$  as “middle”, and the third as “long”. Giving an example, if the maximum remaining process time is 24, the “short”, “middle” and “long” with bounds of 7, 14, and 24.

(4) Machine loading: Machine loading is a property of the machine on which an operation is scheduled. It was divided into two classes: the first as “light”, and the rest as “heavy”. If the value below average, it is classified as “light”. If the value below average, it is over average as “heavy”. Table 3.1 shows the attribution table related to the classes.

Table 3.1 Classes substitutions

Attributes	Value		
	Classes (Code)		
Operation	first 1/3	second 1/3	third 1/3
	first ( 1 )	middle ( 2 )	later ( 3 )
Process time	first 1/3	second 1/3	third 1/3
	short ( 1 )	middle ( 2 )	long ( 3 )
Remaining process time	first 1/3	second 1/3	third 1/3
	short ( 1 )	middle ( 2 )	long ( 3 )
Machine loading	less than average	others	
	light ( 1 )	heavy ( 2 )	

### 3.1.2 Operations Table

After the attribution table is set, the operation of a job will be coded using the combination of attribute and recorded in operation table. Table 3.2 shows the test case in the operations table. For example, Table 2.1 shows that operation 1 of job 1 is the first operation, so operation attribute belong to first 1/3 and be coded 1. And this operation takes 2 time units. While the processing times ranges from 1 to 9 time units, so it belongs to first 1/3 and be coded 1. For operation 1, remain processing time was the sum of the time units from operation 2 and operation 3. In this test case, the longest remain

processing time is 13, so it belongs to third 1/3 and be coded 3 (see table 3.1). The average of machine loading is 13 and all operations be assigned to machine 2 takes 13 time units. So machine loading attribution belongs to the heavy class and be coded 2 (see table 3.1).

Table 3.2 Example of operations table

Operation (Job, operation)	Process time	Remain processing time	Machine loading	Operation (Job, operation)	Process time	Remain processing time	Machine loading
(1,1)	1	3	2	(2,3)	1	1	2
(1,2)	2	2	1	(3,1)	1	3	2
(1,3)	3	1	2	(3,2)	3	2	2
(2,1)	1	3	1	(3,3)	2	1	1
(2,2)	3	1	2				

### 3.2 Collect knowledge

Better solutions (chromosome) may be have some information and can help us to find optimal solution. Therefore, we could take advantage of those better solutions to collect knowledge. However, those solutions just tell me which operation could be sort in this position. We could not understand the influence factor on gene position. So we used the operations table to analyze those operations in each position. Every operation was defined by four attributions and has its attribute value. Attribute combination combined those

attribute value to determine the difference of operation. We use that to decide which operation fits better in each position. If this attribution combination in this position has higher occurrence times and called fitness value or count. Those fitness values were collected by some better solution. If it had higher value, it meant this gene in this position were discovered in better solution sever times. So we will use this fitness value to evaluate fittest-gene in each gene position and those fitness values in each position were called the knowledge. Next, we will use an example to explain it.

Table 3.3 shows the attribute combination in the gene position. We suppose there are three better solutions such as {2, 1, 3, 1, 1, 3, 3, 2, 2}, {1, 3, 2, 3, 1, 1, 2, 2, 3}, and {1, 3, 2, 1, 2, 2, 1, 3, 3}. The gene 2, 1, and 1 are the first gene of three solutions. Those mean the operation 1 of job 2 and operation 1 of job 1. According to the Table 3.2, we can know the attribute combinations are (1, 1, 3, 1), (1, 1, 3, 2), and (1, 1, 3, 2). Therefore, there are two kinds of attribute combinations in gene position 1 (see Table 3.3). The occurrence time (count) of (1, 1, 3, 1) is 1 and of (1, 1, 3, 2) is 2. We used this method to collect knowledge and those will be used in the crossover operator. Based on the position and attribute combination of gene, we could find the count. The count was used to evaluate fitness of gene. If the count is higher, the gene had higher fitness value and it was reserved in child.

Table 3.3 Example of attribute combination in the gene position

Gene position	Attribute combination	Count	Gene position	Attribute combination	Count
1	1131	1	6	2322	1
	1132	2		3312	1
2	1132	3		3112	1
3	1132	1	7	3211	1
	1131	2		2312	1
4	2121	2		3312	1
	2322	1	8	2312	1
5	3312	1		3112	1
	2121	1		2322	1
	2312	1	9	3112	1
		3211		2	

### 3.3 Designing crossover operator

“Keep the better gene in the chromosome” is the important notion in the crossover operator. Therefore, we use the knowledge which collected from some better solutions to decide which gene is better and can be retained. In other words, use operations table to decide the attribute combination of operation from parent chromosome. And then using those attribute combination and gene position to find the count. This count is the fitness value of the gene. And we must choose one gene of parent chromosomes which has higher fitness value in the same position. Based on this method to retain the better gene, and hope it to improve the quality of offspring. If the solution does not conform to JSP problem, we use the fitness value (count) to adjust it. The adjusting method is the job which does not have enough

operations. The targeted replaced operation is the operation with lowest fitness value.

### 3.3.1 Eugenic crossover

This crossover was based on the knowledge to decide which gene can be retained to the child chromosome and was called eugenic crossover. Eugenic crossover is accomplished through the following steps, and Fig 3.1 demonstrates an example:

1. Choose two chromosomes, named parent  $A$  and parent  $B$ .
2. Use the operations table to determine the attribute combination for parent chromosome. If the attribute combination in this position was never existed before, the count will be coded 0.
3. Using the attribute combination and gene position to retrieve the count.
4. Choose one gene from parent which has higher count in the same position.

If the count is the same, randomly choose one gene from parent.

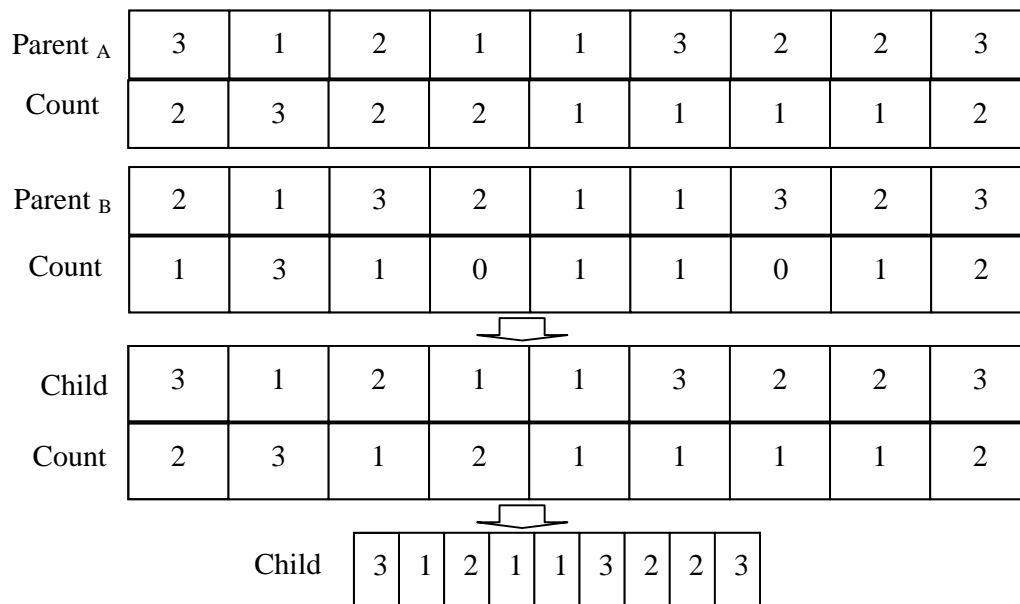


Fig. 3.1 Example of eugenic crossover

### 3.3.2 Adjust child

If the child does not conform to JSP problem, we will use the count to adjust it. The adjust phase is accomplished through the following steps, and Fig. 3.2 demonstrates an example:

1. The child was produced by crossover and not conformed to JSP problem.
2. Sort the child gene by count and record the original gene position.
3. Select the job which has more than  $n$  operations and have the lower count
4. Replace the job which has less operation.
5. According the position to sort the gene and produce child.

Fig. 3.2 shows the example of adjusted child. In the crossover processing, it produces a child which has four operations for job 3 and two operations for job 2. This result does not conformed to JSP problem, so we must adjust it. We used the count to sort the gene and recoded the original gene position. According the sort result to change from gene position 7 which was job 3 and have the lower count to job 2. Making every job three operations and sort the gene based on original gene position.



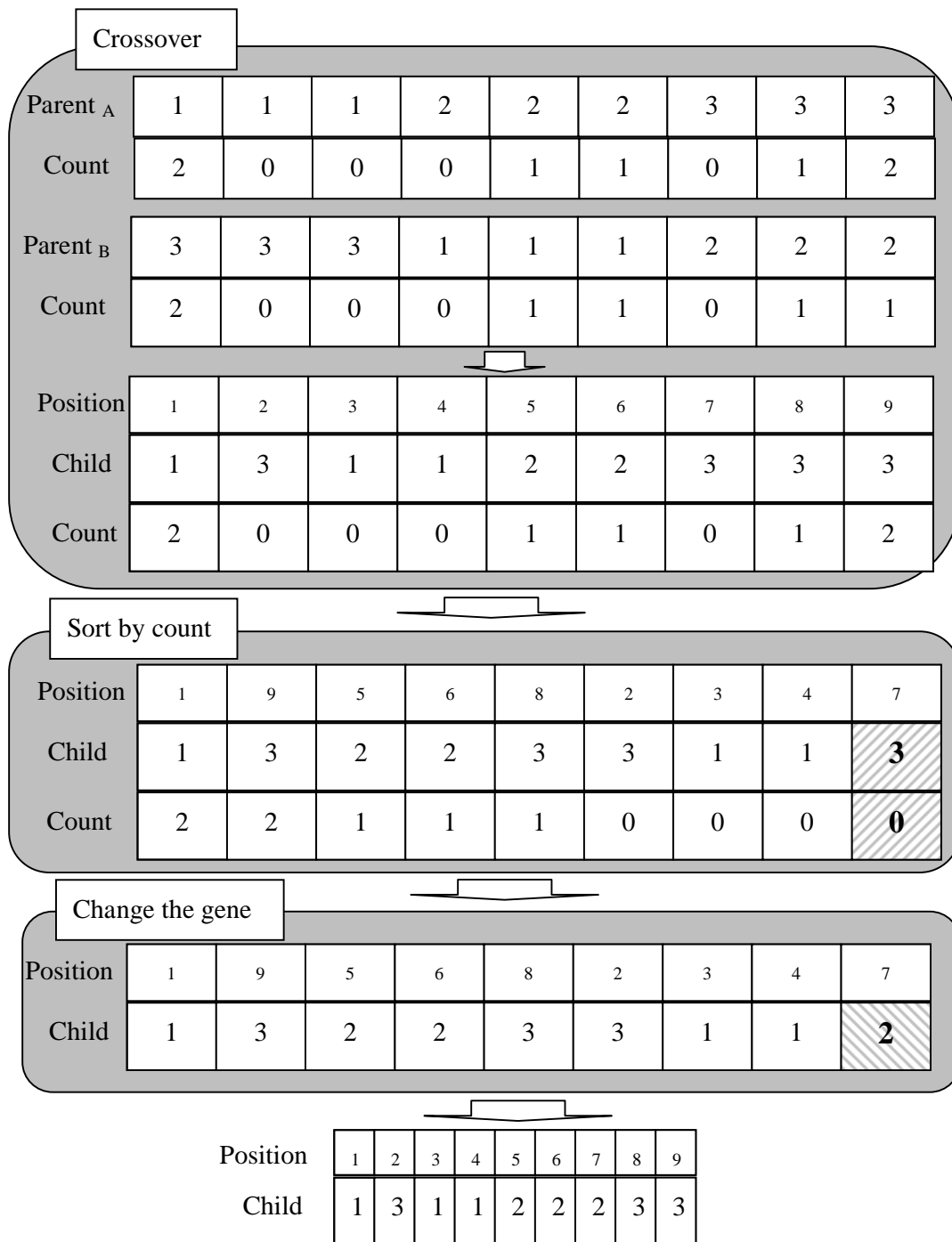


Fig. 3.2 Example of adjusted child

## **Chapter 4 The flowchart of Algorithm**

Our algorithm divides into two parts. The first part is using the GA's solutions to collect knowledge. In this part, there are two phases which are finds better solution by GA and using the operations table to analysis the better solution. The second part is using knowledge to improve the GA. This part was called the knowledge-based GA (KGA). There are three phases which was different from GA. Those phases are called blended crossover, forced mutation, and research knowledge.

### **4.1 Collect knowledge by solution for GA**

If we retain better solution, it will improve the quality of solution by GA. Therefore, we try to collect some useful information from GA's solution. But GA does not demonstrate repeat ability or provide an explanation of how a solution is developed. On the other hand, those solutions are not stable and the information of gene different is not available, so we must use operation table to decide them. This part has two phases and was introduced as following:

#### **4.1.1 Collect solutions by GA**

In this phase, we used the GA to collect the better solutions. There are two methods to generate the solutions. The first method was to executed the GA several times and retain the best solution from each times. But this method will spend much time to collect solutions. The second method was executes the GA one times and retain some better solutions. We will set size

of better solutions and collect those in each generation. In case that new solution better than original solution, we will replace it. The process will be stopped when the pre-set generation number is reached. The flowchart of collect solution was shown in Fig. 4.1 and described as following:

***Step1: Create initial population.***

We used random number to produce some chromosomes. Those chromosomes were called initial population and must conform to constrain of the JSP problem.

***Step2: Compute fitness value.***

Each chromosome represents one solution. So we transform the makespan into the fitness value. If the chromosome has lesser makespan, it will have the higher fitness value.

***Step3: Generate new generation.***

When we have initial population and fitness value, we will use those data to do the next step.

***Step4: Select population.***

We use the roulette wheel method to select two chromosomes. In the roulette wheel, if this chromosome has higher fitness value, it will have higher probability to be selected.

***Step5: Crossover.***

The crossover operator is performed if a randomly generated float number is less than the crossover rate. In this step, we use the PMX crossover to do it.

***Step6: Mutate.***

The mutation operator is performed if a randomly generated float number

is less than the mutation rate. In this step, we use the inversion mutation to do it.

***Step7: Those chromosomes in this generation is better than retained solution or not.***

Many chromosomes were produced in each generation. In the generation 1 and generation 2, we will retain all chromosomes as the better solutions. After generation 3, we use the fitness value of chromosome compare with those retrained better solutions. If the chromosome has higher fitness value than that of any retained solutions, the chromosome will replace the better solution.

***Step8: reach the generation number or not.***

If the GA process has haven enough generation number, the algorithm will be stopped.

#### **4.1.2 Analyze solution to collect knowledge**

We use the GA to collect some better solutions, but we can't get the information. So we use the operations table to find attribute combination of the operation and to evaluate the difference of operation. Besides, analysis genes which have the certain of attribute combination of each gene position. The process of analyze the solutions was showed in Fig. 4.1 and described as following:

***Step1: Make the operations table.***

In this step, we must use the attribute combination to decide the difference of gene for each chromosome. Therefore, four attributions be used and each attribution has several classes (see section 3.1.1). According those

classes of attribution, we will obtain the attribute combination of operation of job. Those attribute combination were be recorded in operations table (see section 3.1.2).

***Step2: Transform those better solutions by operations table.***

We use the operations table to code those better solutions.

***Step3: Record the attribute combination and occurrence times for the same position.***

For each gene position, we can collect which attribute combination had been occurred and it's occurrence times (see 3.2). Those data which include the gene position, attribution combination and occurrence times (count) were called the knowledge. The knowledge will be used in the second part (KGA).

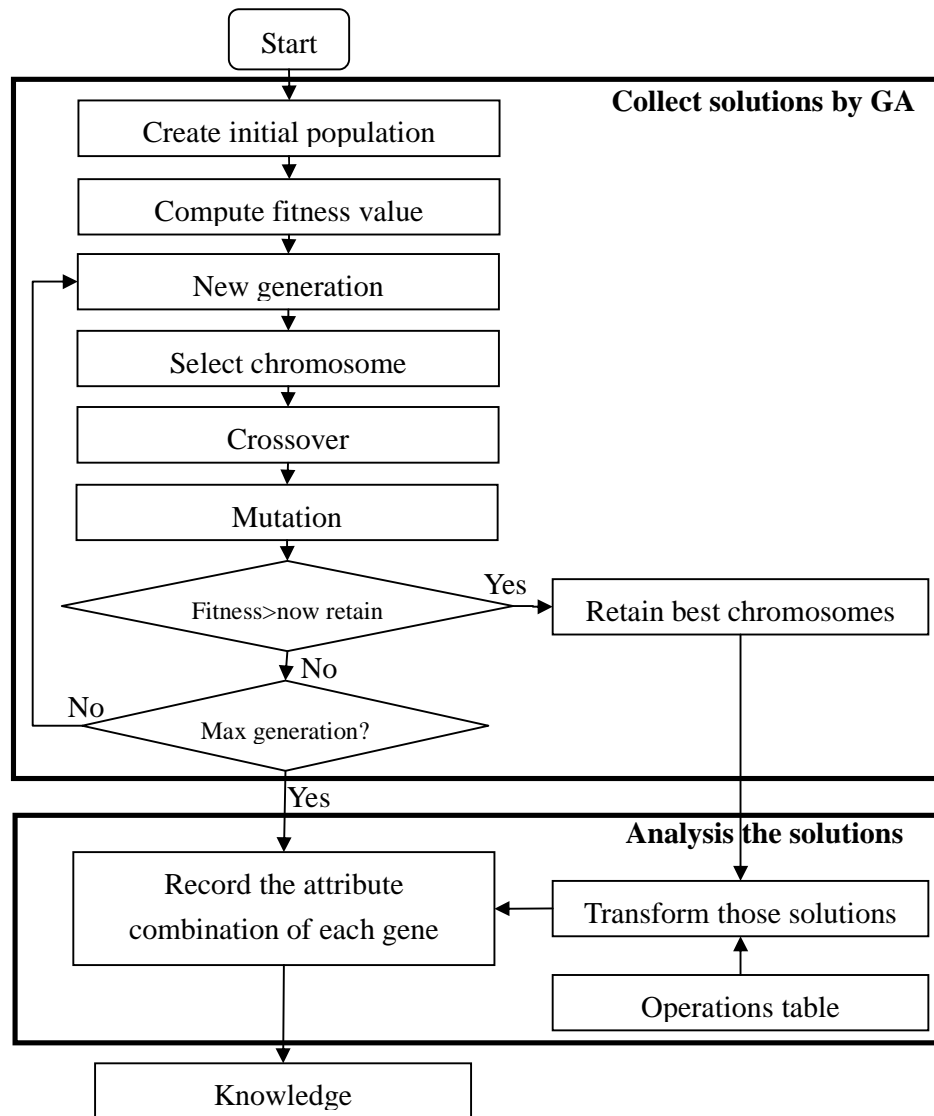


Fig. 4.1 The flowchart of collecting knowledge

## 4.2 Knowledge-based GA

Knowledge-based GA (KGA) has three phases that is different from traditional GA. The first phase is blended crossover. We will reduce the chance that the knowledge made the solution which fell into the local minimum easily. We use the two crossover methods to produce child

chromosome. The second phase is forced mutation. We remove the mutation rate and mutate the child chromosomes which were produced by crossover if it's not better than parents. The third phase is to recollect the better solution and replace the knowledge.

#### **4.2.1 Blended crossover**

Traditional crossover methods randomly select two points to recombine and produce the child chromosome. Therefore, using random can't assure the quality of child. So we use the knowledge to decide the fitness-gene from parent chromosome and execute the eugenic crossover (see 3.3.1). However, this crossover can find some optima solution but fall into the local minimum easily. Based on this reason, we use the PMX crossover to increase the diversification and try to balance intensification and diversification. It showed in Fig. 4.2 and described as following:

***Step1: Select the crossover operator.***

We use the threshold to choose the crossover operator. If the generation number is lower than threshold, using eugenic crossover and making this crossover operator to speed up the convergence rate. And if the generation number is higher than threshold, using PMX crossover and making this crossover operator to raise the variation of chromosome. Hope the algorithm leap from the local minimum.

***Step2: Determine if the child is better than parent or not.***

The child that was produced by crossover must be better than one of parents. If the child better than parents, this chromosome was retained to the new population. And if the child was not better than one of parents, this

chromosome must be mutated.

#### **4.2.2 Forced mutation**

Traditional mutation operator was executed when the random number is lower than the mutation rate. In our algorithm, mutation operator was executed when the child is not better than parents. This process increases the search space and uses the two-point mutation to do the local search. The forced mutation was showed in Fig. 4.2 and described as following:

##### ***Step1: Mutate the child.***

If the child produced by crossover is not better than one of parents, this chromosome must be mutated. The mutation operator was to randomly select two gene positions and exchange these genes.

##### ***Step2: Determine if the child is better than parent or not.***

If the child produced by mutation is better than one of parents, this child was retained to the new population. And if the child was not better than parents, this child must be re-mutated. This process will be ended when the child was better than one of parents or the mutation times was higher than the threshold.

##### ***Step3: Determine if the mutation times is higher than threshold or not.***

The mutation times was counted when the child not better than one of parents by crossover. If the mutation time was not higher than the threshold, the child must be mutated until the child better than parents. Otherwise, we must give up this child and select two parents to produce the new child by crossover again.



***Step4: Determine if the crossover times is higher than threshold or not.***

The crossover times was counted when the child had been mutated several times and had been done crossover. If the crossover times is equal the threshold, the child was retained to new population. Otherwise, we select two parents to produce the new child by crossover again.

#### **4.2.3 Collect or replace new knowledge**

In the part 1, we had collected the knowledge. But the knowledge can't be used until the KGA is completed because the knowledge help the solution become better than GA's. If we still use the past knowledge, the solution will be limited and can't find the other better solution in other place. Therefore, we must retain new better solution in each generation and analyze those to collect new knowledge. If the algorithm was fall into local minimum, we will refresh knowledge.

#### **4.2.4 The flowchart of KGA**

Fig. 4.2 shows the flowchart of KGA. We will decide if the knowledge collected can be used to other JSP problem or not. In the KGA, we can use the same JSP problem with GA or select different JSP problem to do. The KGA process was described as following:

***Step1: Create initial population.***

We used random number to produce some chromosomes.

***Step2: Compute fitness value.***

Transform the makespan into the fitness value. If the chromosome has lesser makespan, it will have the higher fitness value.

***Step3: Generate new generation.***

When we have initial population and fitness value, we will use those data to do the next step.

***Step4: Select population.***

We use the roulette wheel method to select two chromosomes. This step is the same with GA.

***Step5: Crossover.***

In the blended crossover, we must select which crossover operator by generation number (see 4.2.1). If the child was not better than one of parents, we must do the mutation.

***Step6: Mutate.***

In the forced mutation, we will mutate the child which is not better than parents in crossover (see 4.2.2).

***Step7: Meet the population size or not.***

If this generation has enough population, this generation will be over.

***Step8: Compute fitness value.***

In this process, we will compute fitness value of new population and using new population in the next generation.

***Step9: Reach the generation number or not.***

If the KGA process has enough generation number, the algorithm will be over. Otherwise, continuing the KGA and determine the solution fall into local minimum or not.

***Step10: If the solution fall into local minimum or not.***

If the best solution in each generation had not change several times, we will determine the solution has fall into local minimum. When the solution

does not fall into local minimum, we will collect the better solution from this generation. The collecting of the better solution is the same the part 1. If the solution is better than one of the better solution which was retained, we will retain this solution until the solution fall into local minimum. When the solution was fells into local minimum, we will collect new knowledge and refresh knowledge by new knowledge.

***Step11: Refresh population***

When the solution fell into local minimum, we must refresh knowledge. And this situation was represented this search space that can't find the better solution. So we must refresh the population and search other space again. In the refresh population process, we will sort the original population by fitness value and selecting the first 20% population to new population. The other new population was produced by random number.

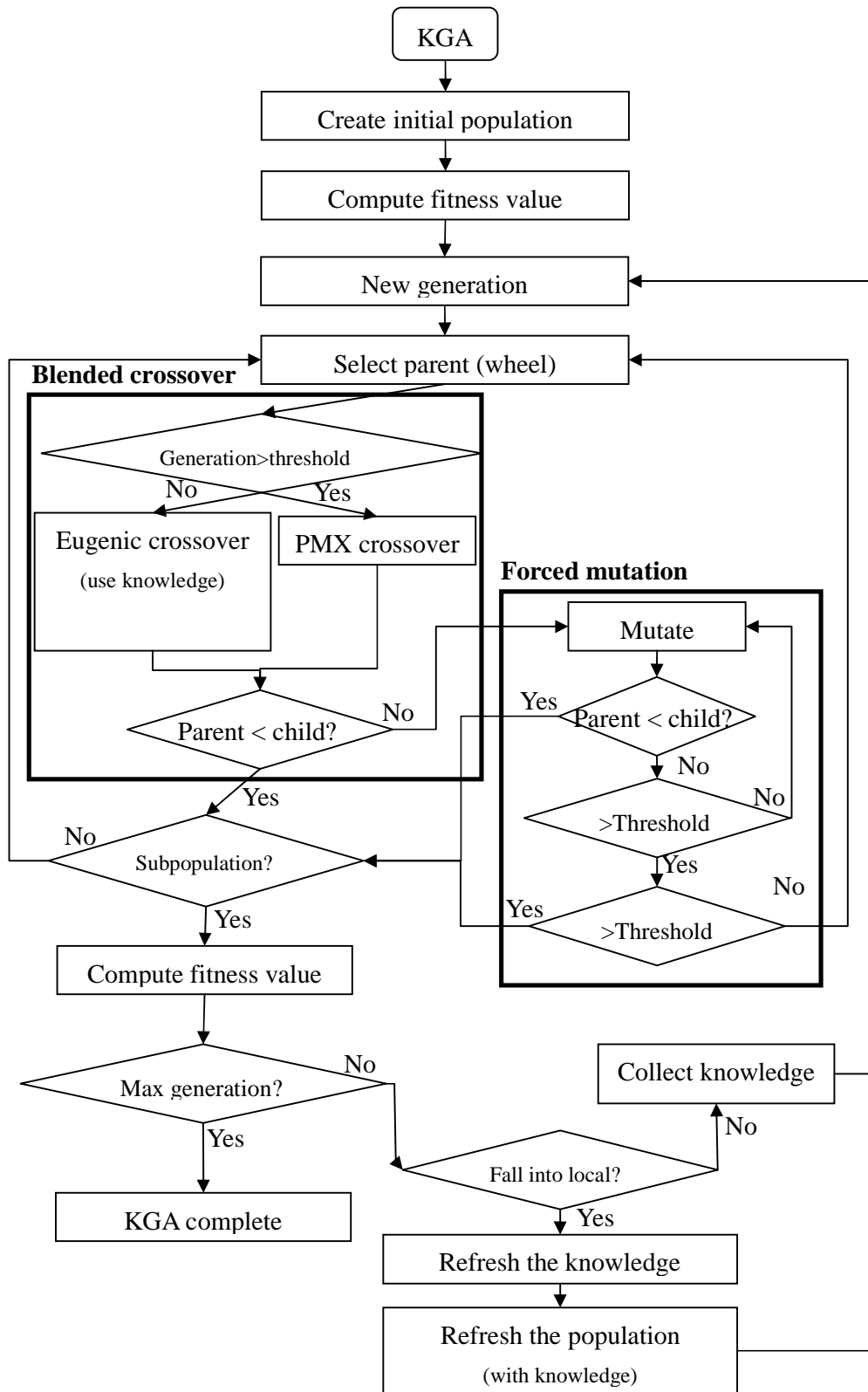


Fig. 4.2 The flowchart of KGA

## Chapter 5 Experiential results

In our experiment, we compare with GA. Therefore, we use three different algorithm processes to do experiment. Those are called KGA\*, KGA\*\* and KGA. KGA\* excluded the random effect during the crossover and mutation steps. Therefore, we use the random number to decide if crossover or mutation showed be down. If the random number lower than crossover probability or mutation probability, we will crossover or mutate the chromosome. GA was run several times to collect knowledge and retains the every best solution in every run times. KGA\*\* still use random number to collect crossover or mutation. But the collect knowledge method was to run GA one time and retains the better solution in each generation. KGA removed the influence of random number and increased search space by change population and forced mutation.

The following experiments showed the 6×6 JSP problems solving only for the purpose of illustrating the computational procedure discussed above. In the first experiment, we try to establish parameter of GA. Then we used the best parameter to do the second experiment. In the second experiment, we try to evaluate the offspring which was produced by eugenic crossover. This result proved that knowledge is useful in selecting fitness-gene and can improve the quality of child. However, KGA\* must spent much time to collect knowledge. So we use KGA\*\* to improve the knowledge collecting speed and found that this method did not influence the result. In the third experiment, in order to prove the knowledge is useful in finding better solution, we run different 6×6 JSP cases to compare with GA. Finally,

combining all the result and using KGA to solve the 10×10 benchmark problem.

### 5.1 Establish parameter

In our algorithm, we used the GA to select some better solutions in each generation. And use those better solutions to collect information for finding the best solution. Therefore, we used three kinds of crossover probability ( $P_c$ ) and four kinds of mutation probability ( $P_m$ ) to do run experiments. Fig. 5.1 to Fig. 5.3 displays the results by using crossover probability of 0.6, 0.7 and 0.8. We find that the result in Fig. 5.3 is better than others, because the astringency is better in each mutation probability. When mutation probability is 0.2, the astringency is better and can find the best solution first. According to this, we decided to use 0.8 and 0.2 for crossover probability and mutation probability respectively.

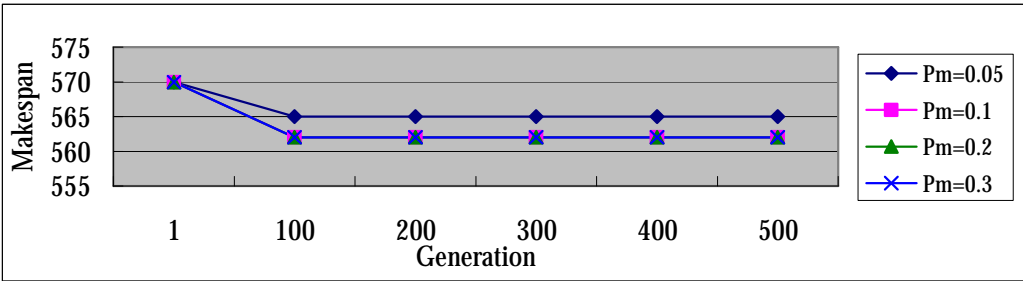


Fig. 5.1 Crossover probability = 0.6 (GA)

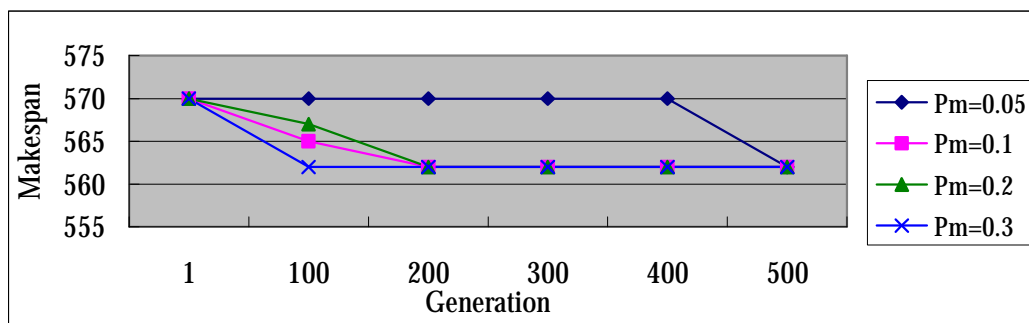


Fig. 5.2 Crossover probability =0.7(GA)

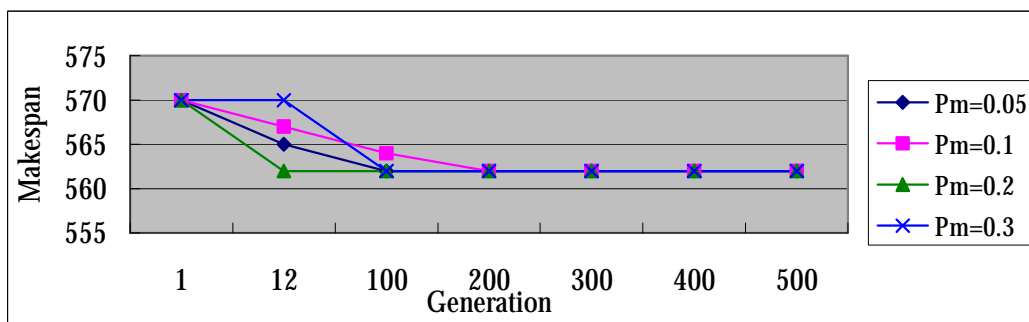


Fig. 5.3 Crossover probability =0.8(GA)

## 5.2 Evaluate offspring

KGA\* used the information that we obtained from GA to evaluate which gene is better and can be reserve as offspring in the same position by crossover operator. Therefore, we try to evaluate whether those information be used by KGA\* can bring the better offspring or not. In this experimentation, we joined the KGA\*\* to assure that using information can make our offspring better than that of GA. The process of KGA\*\* is the same as KGA\*, but it collected information from each generation by run GA one time and reduces more processing time than KGA\*. Table 5.1 shows the probability that offspring better than parents in crossover step. Superior rate is the percentage that offspring better than both parents and improving rate is

offspring equal or better than one of parents. We can find that using KGA\* or KGA\*\* are all better than GA in Superior rate. But, the Superior rate of KGA\* is very close to KGA\*\*. This result shows that using our algorithm can bring the better offspring in crossover step. But the KGA\*\* can reduce the knowledge collecting time as it run GA one time only.

Table 5.1 Offspring evaluation

	GA	KGA*	KGA**
<b>Number of total offspring</b>	17628	17620	20043
<b>Number of superior offspring</b>	9197	10472	11819
<b>Number of improving offspring</b>	4361	1645	2392
<b>Superior rate</b>	52.17 %	59.43 %	58.97 %
<b>Improving rate</b>	24.74 %	9.33 %	11.93 %

After evaluating offspring, we can assure that our knowledge is useful. Therefore, we must test whether the knowledge used in the JSP problem is useful or not. Fig. 5.4 shows the solutions of the JSP problem be run 200 times for KGA\* and KGA\*\*. Based on the result, we know the knowledge is useful in providing gene selection information which leads to better solution. According to the result, we find that KGA\* and KGA\*\* are very close for the 200 times trials. Therefore, we use the KGA\*\* to as the collecting knowledge method because it save collecting time.



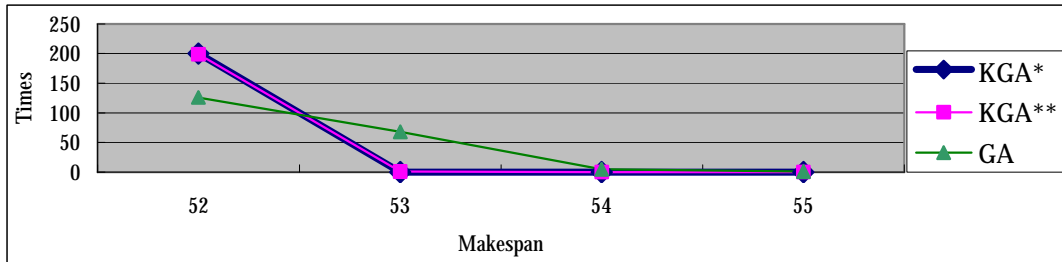


Fig. 5.4 The result of KGA\*, KGA\*\* and GA (run 200 times)

### 5.3 Compare with GA

This algorithm used some better solutions from GA to collect the knowledge and used the knowledge to generate better offspring in crossover step. Based on the idea, we suppose that using our algorithm can bring better solution than GA or its convergence may be faster than GA. In the next experiment, we will compare the performance of KGA\*\* with GA on the two 6×6 randomly produced JSP problems.

This experiment solved those problems by KGA\*\* and GA. According to the results, we have two conclusions. Firstly, the algorithm (KGA\*\*) have the faster convergence than GA whether both method could reach the same minimum makespan. Secondly, the different of convergence time is larger when the minimum makespan is larger. The makespan of case 1 (see Fig. 5.5) and case 2 (see Fig. 5.6) are 464 and 619. We assure that our algorithm can find the same makespan with GA, but we can find the makespan in the 5 and 15 generation. And the generation number is lesser than GA's more. Based on the result, we can prove that our algorithm have faster convergence. Besides, when the problems have larger makespan, the algorithm must spend much

time to find the minimum makespan. But the search speed of KGA\*\* is faster than GA.

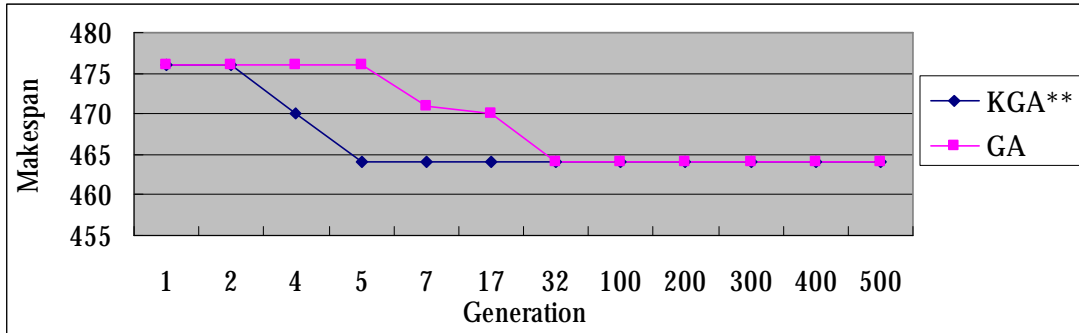


Fig. 5.5 Makespan of case 1

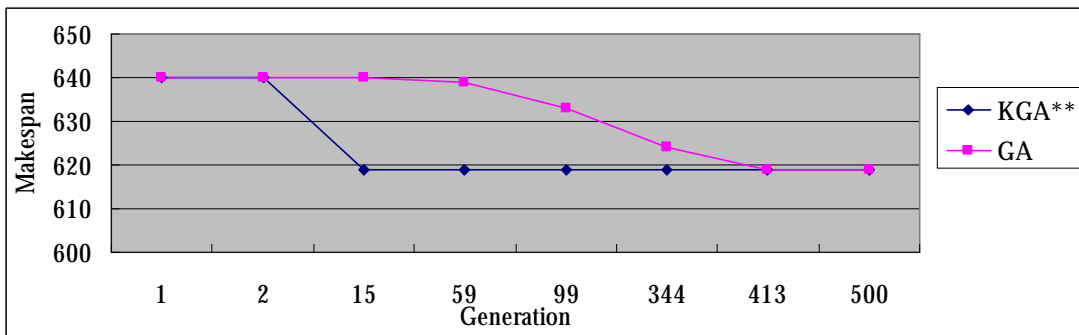


Fig. 5.6 Makespan of case 2

## 5.4 10×10 Benchmark problem

Based on the prevent results, we know that our algorithm can make the convergence faster. But it may fall into the local minimum easily. The study of Michalewicz [29], Jain and ELMaraghy [2] showed that the setting of parameters may performance of algorithm in different problem. Therefore, we establish the KGA algorithm by excluded the parameters (mutation probability and crossover probability) from the KGA\*\*. In this method, the offspring is generated by crossover and this method uses the mutation to do local search. In the blended crossover, we blended the PMX crossover and

eugenic crossover into process. We used the eugenic crossover to do the first several generations and used the PMX crossover in the rest generation to skip over the local minimum. To sum up, this method may skip over the local minimum and help it convergence fast. Besides, if the child chromosome is not better than one of parents, it will proceed to forced mutation process until it become better than parents or the times is more than 30. This forced mutation process will help the algorithm to do the local search. Beside, if the algorithm fell into local minimum, we will refresh the population and refresh the knowledge by recollect in KGA process.

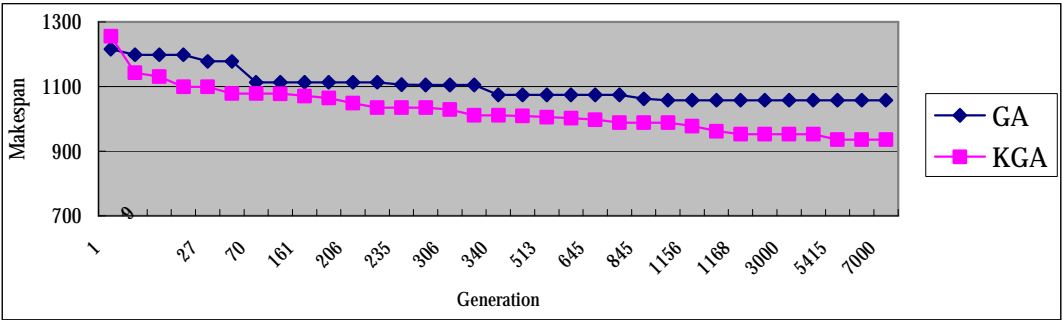


Fig. 5.7 The makespan of the 10×10 benchmark problem

In this experiment, we use KGA for the 10×10 benchmark problem. This problem was generated by Fisher and Thompson [27]. Lawler et al. [11] report that within 6000s when applying a deterministic local search to this problem and find more than 9000 local optima. It is perceived that this problem has the difficult to find the optimal solution. Besides, Carlier and Pinson [14] proved that the optimal makespan is 930. We can use this result to determine whether the solution by KGA is good or not. Fig. 5.7 shows the result by GA and KGA. In this figure, the best solution by KGA is 936 and by

GA is 1053. And it just spent 440 generations to find the makespan 964. This result proved that KGA had faster convergence than GA and its result better than GA. Table 5.2 shows the progress of the 10×10 benchmark instance. According to this table, we can know that we did not find the optimal makespan, but the solution by our algorithm is very close the optimal makespan.

Table 5.2 Progress of the 10×10 benchmark instance

Researchers who achieved solution	Makespan	Researchers who achieved solution	Makespan
Fisher and Thompson(1963)	1101	Lageweg(1982)	935
Balas(1969)	1177	Fisher et al.(1983)	1084
Schrage(1970)	1156	Lageweg(1984)	930
Florian et al.(1971)	1041	Barker and McMahan(1985)	960
Bratley et al.(1973)	980	Adams et al.(1988)(sbII)	930
McMahon and Florian(1975)	972	Carlier and Pinson(1989)	930
Lageweg et al.(1977)	1082		

These results are achieved from experiments performed by Jain and Meeran [4]

Fig. 5.8 shows the makespan for KGA for 100 time trial. We can find that most of the solutions fall into the range between 960 and 969. This result can represent our algorithm is steady. And the best solution by KGA is 936. This solution is not the optimum solution, but it is close the optimum solution. For those result, we can prove KGA is useful and using the knowledge can improve the quality of solution.

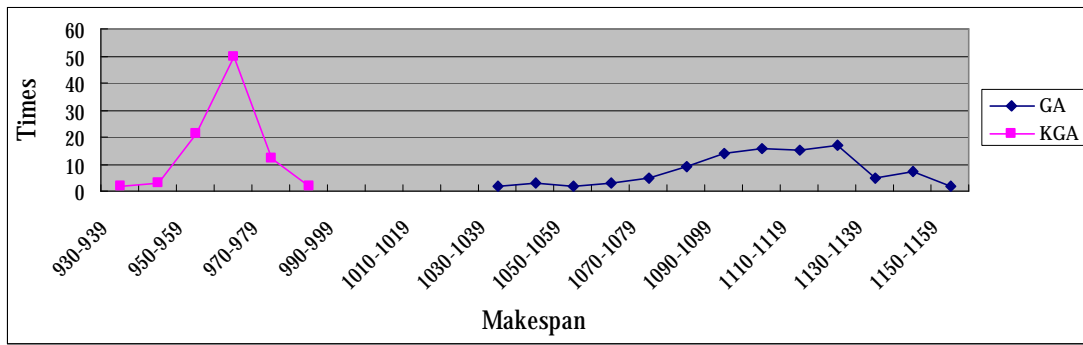


Fig. 5.8 The makespan for KGA (run 100 times)

## **Chapter 6 Conclusions and discussions**

This paper has presented a theoretical and experimental study of the KGA process and concept. KGA is an algorithm which was used the knowledge to improve the deficiency of GA. Therefore, retaining some better solutions from GA and evaluate those to collect knowledge. The knowledge will be used in crossover operator and the KGA process. However, if we still use the random number to decide if crossover showed be downed or not, it will make our algorithm unsteadily. And the parameters could not be used in each case. So we use the child to decide whether crossover or mutation should take place and used knowledge to speed up the convergence. Further, we want to avoid the chance that knowledge will make the solution fall into local minimum easily. In the KGA, we used two crossover operators to do it. Knowledge was used in eugenic crossover to increase the intensification. The PMX crossover was used to increase the diversification. And forced mutation process uses the simple change to do local search. If the solution fell into local minimum, we will refresh the knowledge and population.

According to those experiments, we can obtain some conclusions. The first, the knowledge is useful. In the eugenic crossover, the knowledge was used to evaluate the fitness-gene and retained the higher fitness-gene in offspring. This method can raise the quality of offspring and produce better offspring. Based on this result, the knowledge is useful in provide the gene selection information. But this method will make the algorithm fall into local minimum easily. This is because when we can find the better solution than GA, the knowledge becomes useless. Therefore, the knowledge must be

renewed in KGA process. The second, this algorithm can raise the convergence. Because the algorithm used the knowledge to improve the crossover, it will be led to search the specific space. For this reason, the method could search out the better offspring in short time and raised the convergence.

The third, the algorithm can balance the intensification and diversification. This algorithm used the knowledge to search special space and improve the convergence. Therefore, this method achieves the intensification which makes the algorithm to search the space where better solution exists. But if the algorithm did not have enough diversification, it could not find better solution and fall into local minimum completely. So the mutated process and refreshed population were used in KGA. Using the forced mutation to do the local search and to find other new solution neighbored on the better solution. And when no better solution could be found better in several generations, the population must be renew so that search other space. Based on the KGA result, we can find the better solution better than GA and achieve to balance between intensification and diversification. The fourth, the KGA is steady. In the  $10 \times 10$  JSP problem, although we can't find the optimum solution, we can find the solution which is close to optimum solution. And those solutions which were found by KGA are steadily. Because the solutions are center on certain space and very close the optimism solution. For those reasons, we can prove our algorithm (KGA) is useful and may be used in other optimum problem.

In the future, we can modify the two points. The first, we can use other the algorithm to collect knowledge. As GA can't find the optimal solution, if we can use other algorithm to collect knowledge, the knowledge may be

better than now. The second, we use the simple statistic to count the occurrence times. Some statistic method could be used in inference more sophisticated the result. Those points may be help the KGA become more solid.





## Reference

1. A. Colomi, M. Dorigo, V. Maniezzo and M. Trubian, "Ant system for Job-shop Scheduling", *Statistics and Computer Science*, vol. 34, pp.39-53, 1994.
2. A. K. Jain and H. A. Elmaraghy, "Production Scheduling / Rescheduling in flexible manufacturing", *International Journal of Production Research*, vol. 35, pp. 281-309, 1997.
3. A. S. Jain and S. Meeran, "Job-Shop Scheduling Using Neural Networks", *International Journal of Production Research*, vol.36, No.5, pp.1249-1272, 1998.
4. A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future", *European Journal of operational research*, vol.13, pp. 390-434, 1999.
5. C. Dirk Mattfeld and Christian Bierwirth, "An efficient genetic algorithm for job shop scheduling with tardiness objectives", *European journal of operational research*, vol. 155, pp.616-630, 2004.
6. D.A. Koonce and S.-C. Tsai, "Using data mining to find patterns in genetic algorithm solutions to a job chop schedule", *Computers & Industrial engineering*, vol.38, pp.361-374, 2000.
7. E. Falkenauer, S. Bouffoix, "A genetic algorithm for job shop", *Proc. of the 1991 IEEE international Conference on Robotics and Automution*, 1991.
8. E. Nowicki, C. Smutnicki, "A Fast Taboo Search Algorithm: for the Job Shop Problem", *Management Science*, vol. 42, pp. 797-813, 1996.
9. D.E. Goldberg, "Genetic Algorithms in Search", *Optimization, and Machine Learning*, Addison-Wesley, USA, 1989.
10. D. Goldberg, R. Lingle Alleles, "Loci and the traveling Salesman Problem", *Proceedings of the First International Conference on Genetic Algorithms*, pp. 154-163, 1985.
11. E.L. Lawler, J.K. Lenstra and Rinnooy Kan etc., "Sequencing and scheduling: Algorithms and complexity", *Handbook in operations research and management*, 1993.
12. G. Syswerda, "Uniform crossover in genetic algorithms", *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 2-11, 1989.
13. G. Syswerda, "A study of reproduction in generational and steady-state

- genetic algorithms”, foundations of Genetic Algorithms, pp. 94-101, 1991.
14. J. Carlier and E. Pinson, “An algorithm for solving the job shop problem”, *Management science*, vol. 35, pp.164-176, 1989
  15. J. F. Goncalves, M. Mendes, and Maurício G.C. resende, “A hybrid genetic algorithm for the job shop scheduling problem”, *European journal of operational research*, vol. 167, pp.77-95, 2005.
  16. K. Morikawa, T. Furuhashi, Y. Uchikawa., “Single Populated Genetic Algorithm and its Application to Job-shop Scheduling”, *Proc. of Industrial Electronics, Control, Instrumentation, and Automation on Power Electronics and Motion Control*, pp. 1014-1019, 1992.
  17. L. Davis, “Applying adaptive algorithms to epistatic domains”, *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 162-166, 1985.
  18. L. Davis, “Applying adaptive algorithms to epistatic domains”, *In Proc. of the Inter. Joint Con5 on Artificial Intelligence*, pp. 162-164, 1985.
  19. M. Gen and R. Cheng, “Genetic algorithms and engineering design”, New York: John Wiley & Sons, 1997.
  20. M. Watanabe, K. Ida, and M. Gen, “A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem”, *Computers & Industrial Engineering*, vol. 48, pp.743-752, 2005.
  21. M. Garey, D. Johnson and R. Sethi, “The complexity of flow shop and job shop scheduling”, *Maths Ops Res*, vol.1, pp.117-129, 1976.
  22. M. Kolonko. Some new results on simulated, “annealing applied to the job shop scheduling problem”, *European Journal of Operational Research*, pp. 123-.136, 1999.
  23. P.C. Chang, J.C. Hsieh and C.H. Hsiao, “Application of genetic algorithm to the unrelated parallel machine scheduling problem”, *Chinese industrial of Industrial Engineers*, vol. 19, pp.79-95, 2002.
  24. R. Cheng, M. Gen, and Y. Tsujimura. “A tutorial survey of job-shop scheduling problems using genetic algorithms---I:Representation”. *Computers ind. Engng* vol. 30, No. 4, pp.983-997, 1996
  25. R. Cheng, M. Gen, and Y. Tsujimura. “A tutorial survey of job-shop scheduling problems using genetic algorithms--- II: Hybrid genetic search strategies”. *Computers & Engineering*, vol. 36, pp.343-364, 1999
  26. V.L. PJM, A. EHL, and L. JK, “Job shop scheduling by simulated

- annealing”, *Operations Research*, 40, pp.113-125, 1992.
27. W. K. Fisher, E.O. Thompson, “Amino acid sequence studies on sheep liver fructose-bisphosphatase. II. The complete sequence”, *Aust J Biol Sci.*, vol. 36, pp. 235-250, 1983.
  28. X. Li, W. Liu, S. Ren, and X. Wang, “A solution of job-shop scheduling problems based on genetic algorithms”, *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, pp. 1823 -1828, 2001.
  29. Z. Michalewicz and M. Schoenauer, “Evolutionary algorithms for constrained parameter optimization problems”, *Evolutionary Computation*, vol. 4, pp. 1-32, 1996.