國立交通大學
工業工程與管理學系

碩士論文

零工式排程之巨集啟發式演算法的比較

A Comparison of Meta-heuristics Algorithms for Job Shop Scheduling Problem

研究生：呂光棓
指導教授：巫木誠　博士

中華民國九十六年六月

零工式排程之巨集啟發式演算法的比較

A Comparison of Meta-heuristics Algorithms for Job Shop Scheduling Problem

研究生：呂光棓　　Student：Kuang-Pei Lu

指導教授：巫木誠 博士　　Advisor：Dr. Muh-Cherng Wu

國立交通大學

工業工程與管理學系

碩士論文

A Thesis

Submitted to Department of Industrial Engineering and Management

College of Management

National Chiao Tung University

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In

Industrial Engineering

June 2007

Hsin-Chu, Taiwan, Republic of China

中華民國九十六年六月

零工式排程之巨集啟發式演算法的比較

研究生：呂光棓　　指導教授：巫木誠 博士

國立交通大學工業工程與管理研究所

中文摘要

零工式排程問題(Classical Job Shop Scheduling Problem；JSSP)是一個已經研究很久的問題，也已有許多文獻發表，目前以 Huang & Liao (2006)的演算法表現最佳。本研究結合共識因子和田口方法提出多種巨集演算法(meta-heuristics)，希望找出一種演算法，能在績效上改進 Huang & Liao (2006)的演算法。本研究使用 19 個案例實驗測試，贏了 5 個(26%)，平手 7 個(37%)，輸了 7 個(37%)。

關鍵字：共識因子、田口方法、禁忌搜尋法、瓶頸飄移法、零工式排程

A Comparison of Meta-heuristics Algorithms for Job Shop Scheduling Problem

Student：Kuang-Pei Lu　　Advisor：Dr. Muh-Cherng Wu

Institute of Industrial Engineering

National Chiao Tung University

ABSTRACT

The classical job shop scheduling problem (JSSP) has been studied for decades. Much literature has been published, and the algorithm proposed by Huang & Liao (2006) is the most leading one. We applied the notions of consensus and Taguchi genetic operators and proposed various meta-heuristics algorithms. Numerical tests that include 19 problem instances have been carried out. Compared with the algorithm proposed by Huang & Liao (2006), our algorithm excel in 5 problem instances, and has a tie in 7 problem instances, and lose in 7 problem instances.

Keywords: consensus operator; Taguchi methods; Taboo search; shift bottleneck procedure; Job shop scheduling

誌謝

本論文得以順利完成，主要感謝恩師 巫木誠教授的指導。巫老師除了研究領域以外，還認真地教導生活上的各種層面，並孜孜不倦的教導分析事物的原則與方法，讓學生在研究所兩年中同時學到了學術和生活上的知識，在此獻上最崇高的敬意。同時也感謝許錫美老師、彭德保老師和陳文智老師在口試時候所提供的寶貴建議，讓本論文更加完善。

同時感謝學長吳政翰的經驗傳承，博士班學長施昌甫的引導，此外學術方面有富騰、忠霖和柏儒的細心幫助，生活上有振富、進銘、廷勳、艾苓、秉琦、東森、維琦和幼容的互相幫忙協助，在這些夥伴的協助之下，讓我在這兩年得以成長許多。此外也感謝所有曾經幫助過我的朋友。

最後，要特別感謝我的父母、兄長的支持，在我最需要幫助的時候給予鼓勵與幫助，得以專心完成學業，願將此刻的喜悅與榮耀與你們一同分享，謝謝你們無上的協助與幫忙。

光棓 于新竹交通大學

中華民國九十六年六月

目錄

中文摘要……I

英文摘要…… II

誌謝……III

目錄……IV

表目錄……VI

圖目錄……VII

第一章 序論……1

1.1 研究背景……1

1.2 研究目的與動機……2

1.3 研究問題的定義與限制條件……2

1.4 論文組織……4

第二章 文獻探討……5

2.1 排程問題……5

2.2 關於JSSP的相關排程技術……6

2.3 本研究的概念來源與標竿演算法……8

第三章 本論文演算法……9

3.1 解的表達與計算……9

3.1.1 分離網路圖……9

3.1.2 延遲作業程序……11

3.2 本論文演算法架構……14

3.2.1 初始解……15

3.3 共識因子……17

3.3.1 共識因子架構……17

3.3.2 導引機制……19

3.3.3 產生新解……22
3.4 田口方法……25
3.4.1 田口方法架構……26
3.4.2 直交表……26
3.4.3 田口實驗……27
3.4.3 田口實驗應用在 JSSP 問題……28
3.5 禁忌演算法……29
3.5.1 禁忌演算法架構……29
3.5.2 禁忌演算法演算流程……30
第四章 實驗結果……32
4.1 前置實驗……33
4.2 本論文演算法最佳配方實驗……40
4.3 本論文最佳配方與過去著名方法比較……43
4.4 實驗分析結論……45
4.4.1 母體的差異性……45
4.4.2 母體的結構性……46
4.4.3 更新解的品質……48
4.4.4 田口方法的影響……49
4.4.5 效益的比較……50
第五章 結論……51
參考文獻……52

表目錄

表 2.1　演算法分類……..……..…7

表 3.1　基本組合表……..……...14

表 3.2　m 機器的工件優先關係矩陣 Am……..……21

表 3.3　累積矩陣 M_m……..……21

表 3.4　標準化矩陣 C_M……..……22

表 3.5　L16 標準直交表……..……26

表 3.6　L8 標準直交表……..……..……27

表 3.7　完工時間……..……...……27

表 4.1　基本組合表.....34

表 4.2　基本演算法組合表.....34

表 4.3　實驗一的結果.....37

表 4.4　實驗二的結果.....38

表 4.5　實驗三的結果.....39

表 4.6　指標性題目.....40

表 4.7　指標性題目內容.....41

表 4.8　最佳配方演算法實驗結果比較.....42

表 4.9　本論文演算法和過去著名方法比較……..……..44

表 4.10 母體的差異性.....45

表 4.11 母體的結構性.....46

表 4.12 田口方法的影響.....49

表 4.13 效益的比較.....50

圖目錄

圖 3.1　計算解的概念……..…....10

圖 3.2　起始已知的分離網路圖……..…..…10

圖 3.3　已知的工件途程和演算法計算出來的作業限制……10

圖 3.4　CPM 的計算法(框左上的為作業編號,框中間的為作業時間)…….……11

圖 3.5　不合理的解……..……...……..…..13

圖 3.6　機器二的工件已經排序完畢……..…..…13

圖 3.7　增加 DCP 限制……13

圖 3.8　合理的一組解……....13

圖 3.9　整體架構圖……..……..14

圖 3.10 混合螞蟻塔布演算法邏輯……16

圖 3.11 共識矩陣產生流程……19

圖 3.12 產生 m 機器的工件優先關係矩陣……...…20

圖 3.13 參考指標……..……..……24

圖 4.1　第一種實驗流程概念.....……..……...35

圖 4.2　第二種實驗流程概念.....……..……..35

圖 4.3　第三種實驗流程概念.....……..……...36

圖 4.4　V13 和 ACOFT-MWR 的方法比較:更新解部分……..47

圖 4.4　V13 和 ACOFT-MWR 的方法比較:修正解部分…..……..…..47

第一章　緒論

1.1 研究背景

排程(scheduling)就是決定一群工件(jobs)的加工順序，好的排程代表可以提升生產的效率，以最小化完工時間(minimization makespan)為例，有效率的排程不但完工時間較短，也可減少機台的停機時間，相對也就可以產出較多的產品，而達到提升產品競爭力，擁有一組好的製程排序，一直是許多製造企業渴望追求的目標之一。

由於排程幾乎都屬於 NP-Hard (Garey & Johnson, 1979)的題目，也就是會隨著工件(job)和機器(machine)的數量而大量的增加其複雜度，如：2 個工件 2 個機器就有$(2!)^2$種組合，而 10 個工件 10 個機器則有 $(10!)^{10}$種組合，不過是增加了 8 個工件和 8 種機器，其組合就幾乎已經達到天文數字，更何況現場大型工廠裡面的工件和機器更是成千上萬數量在計算的，如此情況之下，已經不是簡單的傳統排列組合就可以計算出來，這時候我們需要更有效率的演算法(algorithm)來解決這個問題。

在中小型問題的時候，尚可用數學模式的求解方法來取得最佳解，如整數規劃法(Integral Programming；IP)或分支界線法(Branch and Bound；B&B)這兩種方法來求得最佳解，可是一旦問題變大，上述兩種方法缺乏效益性，於是近年來許多學者開始使用搜尋式演算法(Meta-Heuristic)來解排程的問題，如基因演算法(genetic algorithm；GA)、螞蟻演算法(ant colony optimization；ACO)、模擬退火法(simulated annealing；SA)、雖然此類方法大多只能夠找尋到近似最佳解(near-optimal solution)，但在求解效益上優於數學模式的求解方法，可惜的是這些演算法都有各自的優缺點，為了互補彼此的缺陷，現今學者大多結合兩種以上的演算法來求解排程問題，而這樣的演算法我們稱之為混合式演算法(hybrid algorithm) 。

1.2 研究目的與動機

JSSP 比起其他類型的排程，如：旅行者問題(TSP)、流程式生產(Flow Shop)和開放式生產(Open shop)，JSSP 問題看似容易，可是計算的時候卻是十分麻煩，不但要考慮到各種機器加工的先後關係，還要考慮工件的加工順序所造成的死結問題，如果單純的用普通的演算法邏輯來計算，很容易產生不合理的解(Infeasible solution)，或是找出的解品質令人不滿意。

我們希望應用共識因子(consensus operator)和田口方法(Taguchi methods)這兩種演算法來求解 JSSP 的問題，除了希望能夠超越過去的方法以外，也希望能夠發現這兩種方法使用在 JSSP 問題上的特性。

1.3 研究問題的定義與限制條件

零工式排程問題(Job Shop Scheduling Problem；JSSP)在排程問題裡面算是基本的問題，屬於 NP-hard 的題目，題目看起來很容易，可是求解起來卻很困難，著名的例子如 Muth & Thompsom (1963)，所提出的 10 台機器 10 個工件問題，經過了 25 年才被找到最佳解。

本論文所求解的題目 JSSP 可以簡單的敘述為：給定一群工件集合(jobs)和機器集合(machines)，每個工件集合都已知其作業加工順序，且每一項作業的處理時間和所需的機器已知，而工件和機器有以下的限制：

1. 不同的工件之前彼此獨立(independent)。
2. 機器一次只能加工一個工件，且不能中斷(non-preemption)。
3. 工件不可分割，單一工件只能由一台機器作業，且有已知固定的流程和作業時間。

一組可行解(feasible solution)就是在每台機器上面指定加工的工件，且不會有死結(deadlock)的情況；最終目標為最小化完工時間(minimize makespan)。

首先給定一群工件集合 J(其大小以｜J｜表示)，再給定一群機器集合 M (其大小以｜M｜表示)，其中 σ_m^j 代表工件 j 在機器 m 上的作業；PT(σ_m^j)代表工件 i 在機器 m 上的作業時間；$\Pi(m)$ 代表機器 m 上的作業順序，如以下表示：

$$\Pi(m) = \{\prod(m,1), \prod(m,2), \ldots \prod(m,|J|)\} \qquad (1)$$

其中 $\prod(m,i)$ 代表機器 m 上第 i 個位置的作業；一組完整的解使用 $\prod$ 來表示，如以下表示方法：

$$\Pi = \{\prod(1), \prod(2), \ldots \Pi(|M|)\} \qquad (2)$$

通常使用 Roy & Sussmann (1964)所定義的分離網路圖(disjunctive graph)來表示一個 JSSP 問題：

$$\begin{aligned}
&G = \{V, A, E\} \\
&V = \{O \cup \{source, \sin k\}\} \\
&A = \{(\sigma_m^j, \sigma_k^j) \mid \sigma_m^j, \sigma_k^j \in O, \sigma_m^j \prec \sigma_k^j\} \cup \\
&\qquad (source, \sigma_k^j \mid \sigma_k^j \in O, \sigma_m^j \in O \wedge \sigma_m^j \prec \sigma_k^j\} \cup \\
&\qquad (\sigma_m^j, \text{sink} \mid \sigma_m^j \in O, \sigma_k^j \in O \wedge \sigma_m^j \prec \sigma_k^j\} \\
&E = \{(\sigma_m^i, \sigma_m^j) \mid \sigma_m^i, \sigma_m^j \in O\}
\end{aligned}$$

在上面的表示法裡面，*source* 和 *sink* 代表起始和結束的兩個虛擬作業，符號 $\prec$ 則是代表兩個作業有先後順序的關係，也就是 $\sigma_m^i \prec \sigma_k^j$ 代表作業 σ_k^j 一定要在作業 σ_m^j 完成之後才有辦法進行作業；A 代表作業間的有向弧，也就是作業在工作上面的加工順序，E 則是代表無向弧，也就是尚未決定順序的作業。

當所有的作業都有自己的排序以後，且為合理解的情況之下，我們就可以得到一組完整的解$\prod$，也就是把上面所提到的分離網路圖中的 E(無向弧)轉變成A(有向弧)，我們把一組完整的解用下面的方法來表示：

$$
\begin{aligned}
&G = \{V, A, \mathrm{E}(\prod)\} \\
&A = \{(\sigma_m^j, \sigma_k^j) \,\big|\, \sigma_m^j, \sigma_k^j \in O, \sigma_m^j \prec \sigma_k^j\} \cup \\
&\qquad (source, \sigma_k^j \,\big|\, \sigma_k^j \in O, \sigma_m^j \in O \wedge \sigma_m^j \prec \sigma_k^j\} \cup \\
&\qquad (\sigma_m^j, \mathrm{sink} \,\big|\, \sigma_m^j \in O, \sigma_k^j \in O \wedge \sigma_m^j \prec \sigma_k^j\} \\
&E(\prod) = \bigcup_{m=1}^{|M|} \bigcup_{j=2}^{|J|} (\prod(m, j-1), \prod(m, j))
\end{aligned}
$$

特別要注意的是，所謂的合理解$\prod$，代表著這個有向的網路圖，不會有死結(death lock)的狀況發生，也就是不會有迴路(cycle)的問題發生。

1.4 論文組織

本論文後續章節安排如下，第二章為文獻回顧，說明過去發展的演算法，和本論文使用的演算法文獻；第三章為本論文演算法，說明本論文如何使用在 JSSP 問題上面，第四章為本論文演算法的實驗結果及分析，第五章為結論。

第二章　文獻回顧

2.1 排程問題

排程就是決定一群工件的加工順序，簡單說就是在正確的時間使用正確的資源在正確的地點上面；在排程的領域，幾乎所有的排程問題都屬於 NP-Hard 的問題。

Graham (1979)與 Pinedo (1995)，提出可以使用三個符號{α , β , γ}={機器環境 , 加工特性 , 排程目標}來表示一個排程的環境設定：

對 α 而言，通常可分為下列幾種主要的機器環境：

1. 單機問題(Single machine)：單一個機器，此機器用來加工所有的工件。
2. 平行機台(Parallel Machine)：多部功能相同的機器，但根據效率的不同，分為等效機台(Identical)，和不等效機台(Non-identical)。
3. 流程式生產(Flow Shop)：每個工件的加工順序固定，且每個工件的加工流程相同。
4. 零工式生產(Job Shop)：每個工件的加工順序固定，但每個工件的加工流程不一定相同。
5. 開放式生產(Open Shop)：每個工件的加工順序不固定，且每個工件的加工流程也不一定相同。

對 β 而言，通常可分為下列幾種主要加工特性：

1. 能否中斷(preemptive or non- preemptive) 。
2. 先後順序限制(precedence constraint) 。
3. 當機情況(breakdown) 。
4. 無等待時間(no-wait) 。
5. 循環(cycle) 。
6. 阻塞和飢渴(blocking and starvation) 。

對γ而言，通常可分為下列幾種主要排程目標

1. 總完工時間(Total Completion time)
2. 平均流程時間(Mean Flow time)
3. 平均延遲時間(Mean Tardiness)
4. 平均差異時間(Mean Lateness)
5. 平均早交時間(Mean Earliness)

2.2 關於 JSSP 的相關排程技術

排程技術已經有數十年的基礎，可以使用的技術很多且廣泛，不過整體上來說可以分成兩種主要技術：

第一種是最佳化方法(optimization method)，最佳化方法就如同字面上的意義，主要為找到最佳解(optimal solution)為主，如：Carlier & Pinson (1989)年使用的分支界線法(branch and bound ; B&B)、或是 Manne (1960)所使用的整數規劃法(integral programming；IP)，這些方法在計算的時候都需要大量的計算時間(time consuming)，好處是往往都可以找到問題的最佳解，可惜的是求解的速度和問題的大小特性有很直接的關係，於是此種方法的研究重點幾乎都放在發展有效率的求解方式。

第二種是近似求解法(approximation method)，近似求解法也如同字面上的意義，主要以找尋近似解(near-optimal solution)為主，有時也會找到最佳解，最大的優點就在於計算效率相當的高，且不易隨著問題的大小而大量的增加運算時間(相較最佳化方法)，為目前許多學者所喜愛。此外根據求解的方式又可以分成建構式求解法(constructive algorithm)和迭代演算法(iterative algorithm)。

建構式求解法：Adams et al.(1988)使用的瓶頸飄移法(Shifting Bottleneck Procedure)為一標準建構演算法，主要構想為把大型問題(problem)切割成數個小型的子問題(sub-problem)，找出瓶頸子問題(bottleneck sub-problem)並最佳化各個子問題，最後再把這些子問題合併做再次最佳化的(re-optimal)動作

迭代演算法如：基因演算法(genetic algorithm ; GA)、模擬退火法(simulated annealing ; SA)、螞蟻演算法(ant colony optimization ; ACO)、禁忌演算法(taboo search ; TS)，由於求解方法是一代一代的更新，所以稱之為迭代演算法。本研究將這些著名的基本的演算方法整理如表 2.1，其他關於 JSSP 的詳細歷史回顧資訊在 Blazewicz et al(1996)和 Jain & Meeran(1999)有詳細說明。

表 2.1 演算法分類

主概念	類別	演算法	論文/年代
Optimization algorithms		B&B Branch and Bound	Carlier J, Pinson E (1989)
		IP Integral Programming	Manne (1960)
Approximation algorithms (meta-heuristics)	Constructive	DPR Dispatching Priority Rules	Brucker et al.(1992)
		SB Shifting Bottleneck Procedure	Adams et al. (1988)
	Iteration (Meta-Heuristics)	GA genetic algorithm	Croce et al. (1995)
		SA simulated annealing	Van Larrhoven et al. (1992)
		ACO ant colony optimization	Dorigo & Stutzle (2004)
		TS Taboo search	Nowicki & Smutnicki (1996)

可惜的是雖然有這樣多的演算法，但是每一個演算法都有其缺點，如螞蟻演算法設定的參數會強烈的影響到求解的品質，禁忌演算法則是會強烈的受到初始解的影響，由於單一的演算法都有其缺點，所以近年來許多學者結合兩種以上的演算法來求解，稱之為混合式演算法(meta-heuristic algorithm)，如 Dorndorf & Pesch (1995)的 SB-GA，Nowicki & Smutincki (1996)的 TSAB，Pezzella & Merelli (2000)的 TSSB。

2.3 本研究的概念來源與標竿演算法

共識因子(consensus operator)是吳政翰(2006)所提出的方法，主要概念在結合母體間的共識，並利用此共識來產生新一代的母體，以此達到世代循環的觀念(使用方法我們在 3.3 會說明)，不過當時他主要應用於專案排程(Distributed Heterogeneous Computing ; DHC)的問題，當時在專案排程問題上他有很優異的結果，所以我們希望可以用以延伸應用於工式排程問題(JSSP)。期待在 JSSP 問題上也能有所表現。

田口方法是 1949 年，由日本田口玄一(Genichi Taguchi)博士發明的，當初他發明的原因是因為他於工作時發現傳統的實驗設計方法在實務上並不適用，為了發展「品質工程」，他竭力於發展出一套新的實驗設計方法(Taguchi, 1986, 1987)。田口方法通常用來改善品質工程，不過近年來也有人用來使用在排程上面發展，有相當不錯的績效。

禁忌演算法(Taboo search；TS)又稱為塔布演算法，最早由 Glover (1986)所提出的，為主要使用在「微調解」(local search)的機制，由於禁忌演算法的求解品質十分受到母體的影響，所以通常都會搭配不同的演算法一起使用，如搭配螞蟻演算法或是模擬退火法，此外本論文所使用的禁忌演算法的架構為 Nowicki & Smutnicki (1996)所使用的快速禁忌演算法(fast taboo search；FT；TSAB)。

第三章　本研究演算法

本論文為應用「共識因子」(Consensus operator)和「田口方法」(Taguchi Methods)於 JSSP 問題，此外我們使用瓶頸飄移法(Shifting Bottleneck Procedure；SB)的分解機制來作為求解基礎，並加上禁忌演算法(Taboo Search；TS)來做修正解的動作。

根據瓶頸飄移法的概念，我們把 JSSP 問題拆解成許多子問題$\prod(m)$，每一台機器就算是一個子問題(sub-problem)，分別求解各個子問題的解，最後把所有的子問題結合起來就是一組完整的解$\prod$。

3.1 解的表達和計算

在說明本論文演算法之前，必須要先說明如何表達一組解，以及演算法原先計算出來的應該是工作排序的結果，但這種產生解的方式並不能保證產生出來的工件排序是合理解(feasible solution)，因此需要使用另一種機制把產生出來的工作排序轉換成合理解，本研究演算法使用 Dauzere-Peres & Lasserre (1993)提出的序限制(delayed precedence constraints；DPC)來解決此問題。

3.1.1 分離網路圖

解的概念我們使用圖 3.1 來表示，需要兩種元素，第一種就是演算法計算出來的一組解$\prod$，代表每個機器中的工件加工順序，第二種就是一開始已經是先給定的工件途程(routing)；通常我們會使用分離網路圖來表示一個 JSSP 問題，如圖 3.2 就是一組擁有兩個作業，三個機台的分離網路圖，其中工件一的作業途程為機器 1→2→3，加工時間分別是 3/3/2，而工件二的作業途程為機器 1→3→2，加工時間分別是 4/4/2，框框左上角的代表作業編號，也就是作業代號 1 的是σ_1^1代表工件 1 在機器 1 的作業，其中 $PT(\sigma_1^1)$=3；我們演算法的目標就是要找出不同作業彼此的先後關係限制，如圖 3.3 即為一組完整的解(包含原先的已知的工件途

程，和演算法計算出來的作業限制)，其中機器一的加工順序為作業 1→2，機器二的加工順序為作業 1→2，機器三的加工順序為作業 2→1。

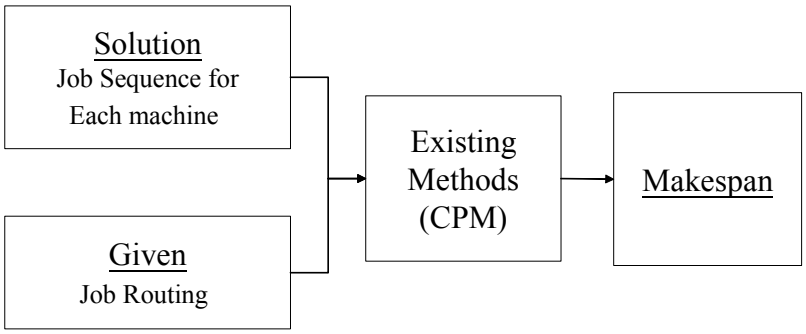

圖 3.1　解的概念圖

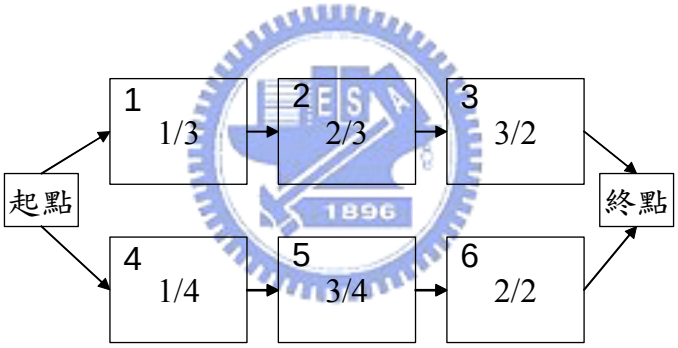

圖 3.2　起始已知的分離網路圖

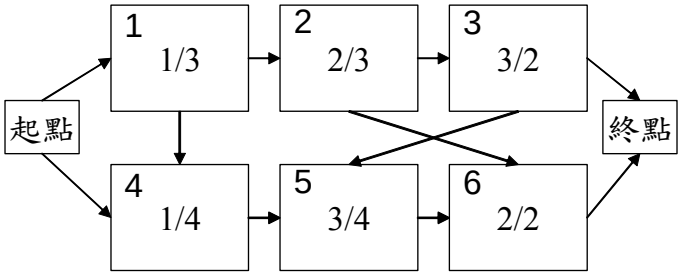

圖 3.3　已知的工件途程和演算法計算出來的作業限制

我們使用簡單的要徑法(critical path method；CPM)來計算 JSSP 問題的完工時間(makespan)，概念就如同之前提到的圖 3.1。其中要徑法為根據工件先後關係圖(有向分離網路圖)，分別計算各個作業的最早開始時間(early start time；EFT)、最早完工時間(early finish time；EFT)、最晚開始時間(late start time；LST)、和最晚完工時間(late finish time；LFT)，這樣一來就可以得到寬裕時間(slack time；ST)也就是 ST = EFT-EST = LFT-LST，其中寬裕時間為 0 的就是要徑(critical path；CP)，要徑時間就是我們所需要的完工時間(makespan)，如圖 3.4 所計算出來的要徑長度就是 14，要徑作業就是 1→2→3→5→6。

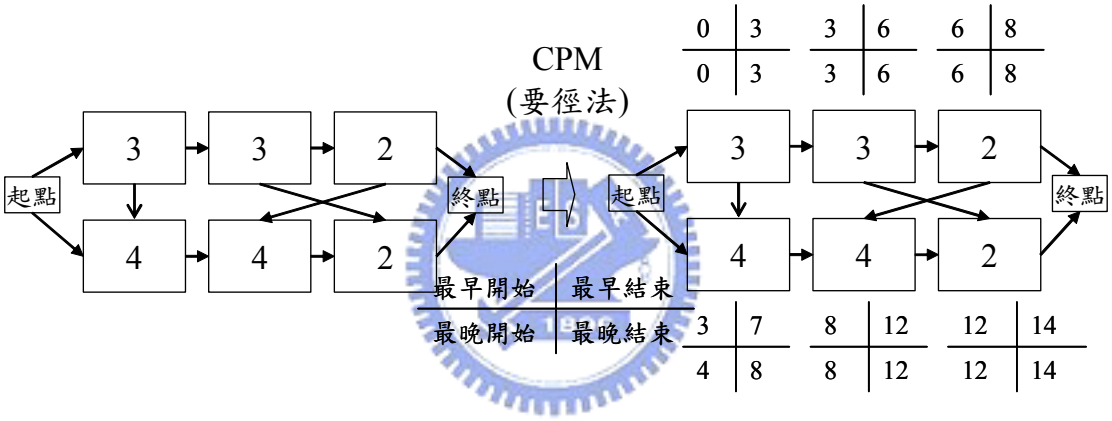

圖 3.4 CPM 的計算法(框左上的為作業編號，框中間的為作業時間)

3.1.2 延遲作業順序(delayed precedence constraints)

在 JSSP 這個問題之中，由於本論文使用的演算法都是產生工件在機器裡面的順序，這種產生方式並不能保證每次產生的解都會是合理解(feasible solution)於是 Dauzere-Peres & Lasserre (1993)使用了一種轉換機制，稱之為延遲順序限制(delayed precedence constraints ; DPC)，簡單來說，會額外設定一個作業的限制式來確保工件排定的先後關係，用以避免產生不合理解。如圖 3.5 就是一個不合理的狀態，四個工件彼此間受到牽制。

為了讓人容易了解，本文以一個小例子來說明使用 DPC 的程序步驟：

假設已知一組解，M1={1→2}，M2={2→1}並需把原先排定的工件排程轉換成工件權重，權重數字越小的，有越先排定的能力，比如原先演算法計算出來的工件排序 M1{1→2}，代表在機器 1 裡面工件的作業順序是 1→2，而現在轉換為機器 1 裡面工件 1 的權重為 1，工件 2 的權重為 2(權重值即為原先計算出的工件順序)，所有的機器排序都需要如此的轉換。

假設已知機器 2 的工件已排入到分離網路圖，如圖 3.6。圖 3.6 表示的 X/Y 代表，工件 Y 在機器 X 上作業，也就是之前所提到的 σ_X^Y 。

下面是我們的排定程序：

1. 根據 TML 值(3.3.3 會仔細說明)決定要排入的機器 M(此例為機器 1)
2. 把該機器 M(此例為機器 1)中所有的作業(此例為 J_1、J_2)，分別找尋出該作業的後續作業集合。
   i. σ_1^1 的後續作業集合為：$\sigma_2^1 \rightarrow \sigma_2^2 \rightarrow \sigma_1^2$ 。
   ii. σ_1^2 的後續作業集合為：無。
3. 如果有兩相同的作業屬於同一個集合之中，便額外給定一作業順序關係(此例，由於 σ_1^1 和 σ_1^2 在同一個作業集合裡，且 σ_1^2 為 σ_1^1 的後續作業，因此我們便增加一限制式，限制作業 σ_1^2 要在 σ_1^1 之後完工，如圖 3.7 所示。
4. 開始排定作業，現在機器 1 有兩個作業可以排定(σ_1^1 、 σ_1^2)，去除後續作業(2/1)，只剩作業 1/1 可以排，
5. 如果還有作業尚未排入，則根據剛剛排序的作業修改作業順序關係，並回到步驟 4。
6. 如果所有機器都已排序完畢，則結束程序，否之回到步驟 1。

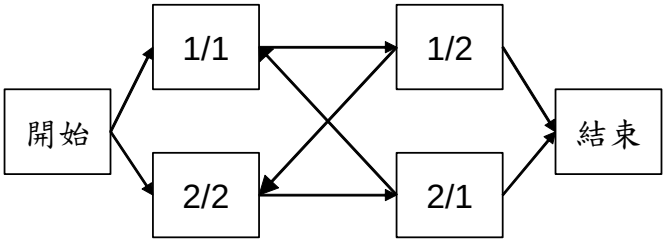

圖 3.5　不合理的解

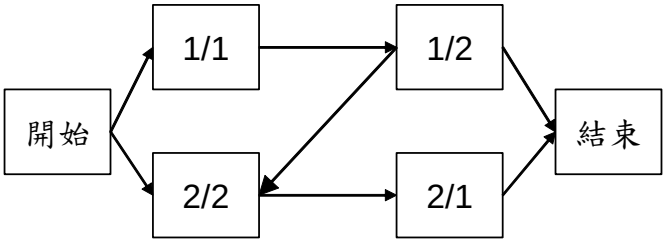

圖 3.6　機器 2 的工件已經排序完畢

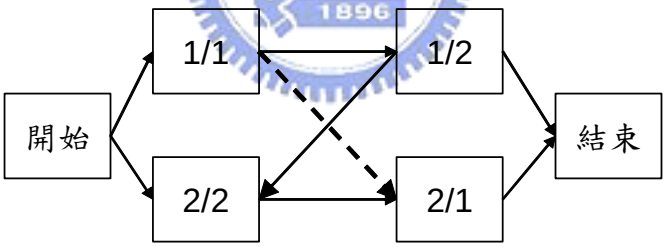

圖 3.7　增加 DPC 限制

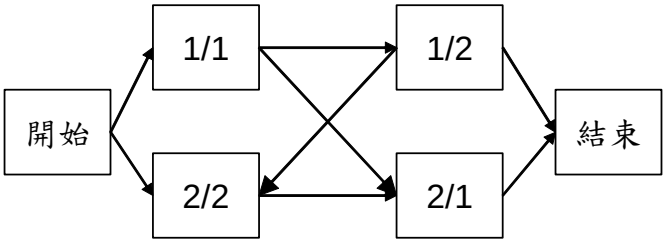

圖 3.8　合理的一組解

3.2 本論文演算法架構

下圖 3.9 為本論文的整體架構圖，由於本論文演算法的組合太多，光是基本組合表就有 18 組，此外加上一些參數調整或是流程改變更有 47 組(第四章會詳細說明)，因此本論文把基本的組合情況放在表 3.1，在此種情況之下如果針對所有的問題去做實驗的話，不但花時間，也很沒有效益，為了快速的搜尋最好的配方組合，本論文使用一個簡單的想法來挑選最佳配方組合，被挑選出來的配方組合才開始針對有代表性的問題來實驗求解，比較的部分則是和跟過去的歷年文獻做比較，用以判斷是否本論文演算法具有優異的表現。

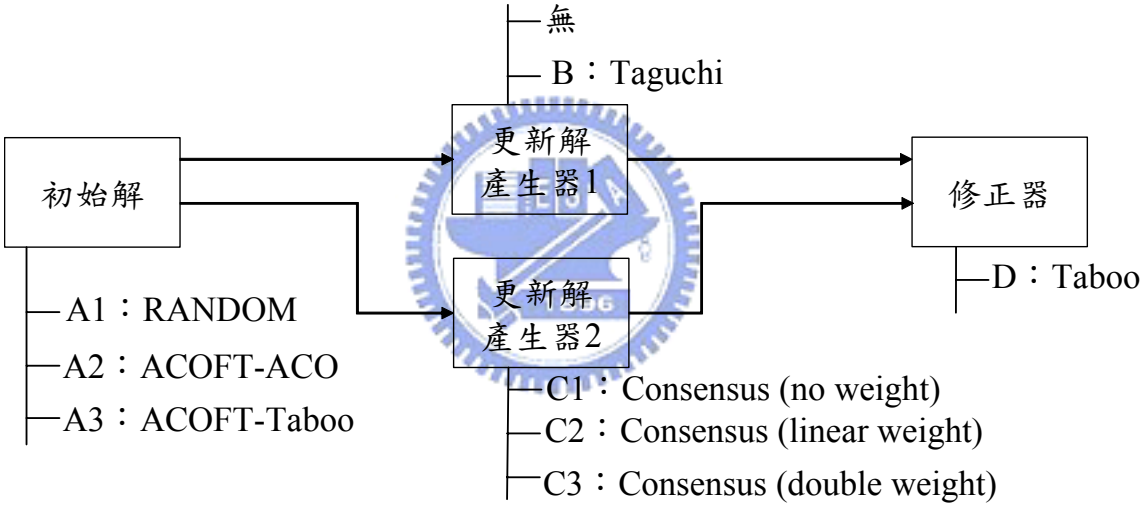

圖 3.9 整體架構圖

表 3.1 基本組合表

初始解	更新解 1	更新解 2	修正解
(A1)Random	無	(C1)consensus normal	(D)Taboo
(A2)ACOFT-MWR(ACO)	(B)Taguchi	(C2)consensus linear	
(A3)ACOFT-MWR(Taboo)		(C3)consensus double	

本論文決定最佳配方的想法相當簡單，先由過去的文獻之中找出一個具代表性的題目，在本論文使用的是 Lawrence(1984)提出的 La29 這個題目，由於此題目至今尚無任何演算法能找到最佳解，且需要大量的計算時間，適合做為標竿性測試的題目；把本論文所有的演算法組合(包含參數改變的組合)，都針對 La29 這個題目做 5 次實驗，且最後依據這 5 次實驗找到的平均完工時間(Mean makespan)來判斷這個演算法是否具有良好的發展性，並設定表現最好的演算法為我們的最佳配方組合。

由於本論文使用許多不同的演算法機制，如果比較固定計算時間(CPU times)產生的解品質，來判斷各種演算法的優劣，非常的不合情理；畢竟要比較的是求解的品質，而不是求解的速度，所以採用另外一種停止條件，也就是使用產生的「總新解數目」當作我們的停止條件，根據實驗結果，平均採用產生 2000 萬組新解為 La29 這個題目的停止條件(不同的題目有不同的停止條件)，此外這個停止條件是根據最主要的指標性方法，Huang & Liao (2006)發展的 ACOFT-MWR 當初在設計時候是使用 320 個迭代為停止條件，本論文概括的計算平均產生新解個數約略為 2000 萬。

3.2.1 初始解

本論文演算法一共使用了三種不同的初始解，分別是 RANDOM，也就是隨機產生 N 組母體，並挑選其中最佳的 n 組當做初始母體，根據實驗發現，母體數越多所計算出來的解品質會越好，但過多則是無意義的浪費新解個數；第二種產生方法為 ACOFT-MWR(ACO)，也就是採用 ACOFT-MWR 這個方法產生出來的 ACO 解當作初始母體，第三種方法則是 ACOFT-MWR(Taboo)，也就是採用 ACOFT-MWR 這種方法產生的 Taboo 解當作初始母體，其中所謂的 ACOFT-MWR 法，是由 Huang & Liao (2006)所提出的一種混合性螞蟻塔布演算法，演算邏輯如圖 3.10 所示。

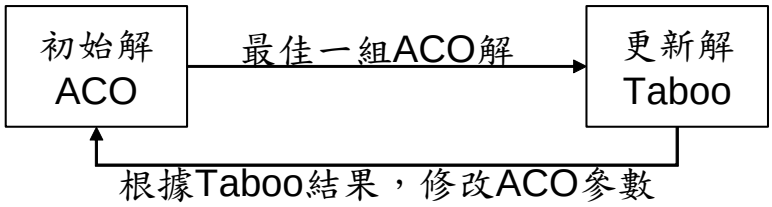

圖 3.10　混合螞蟻塔布演算法邏輯

ACOFT-MWR 的演算想法如下，先由螞蟻演算法(ACO)來產生初始解(大約 3-5 隻螞蟻解)，再由這些螞蟻解中，挑選完工時間最短的一組螞蟻解，做禁忌演算法的計算，之後再把禁忌演算法求得的參數用來更新螞蟻演算法的費洛蒙(pheromone)矩陣，之後便重複以上步驟，直到找到最佳解或是達到停止迭代 320 次為止。其中每次迭代裡面所產生的螞蟻演算法解，就是本論文所謂的 ACOFT-MWR(ACO) 解，而禁忌演算法所得到的解就是本論文所謂的 ACOFT-MWR(Taboo)解。

3.3 共識因子(consensus operator)

共識因子是吳政翰(2006)所提出的一種方法，當時他使用在專案排程(Distributed Heterogeneous Computing; DHC)的問題，基於他的結論有很優秀的表現，所以本論文希望可以應用到 JSSP 的問題上面，期待也能有不錯的發展。以下將針對共識因子的概念和計算方法做詳細解說。

3.3.1 共識因子架構

共識因子(consensus operator)是根據上一代解的工件排序「共識」來產生新一代的解，好處是擁有優秀排列順序的解可以用來產生新一代解，不好的解自然淘汰無法產生新解，在這樣的引導機制之下優秀排列順序的解會被保留起來，不好的解自然淘汰，導致求解會慢慢的往好的解空間移動。相較於其他演算法如：螞蟻演算法為最傑出解的更新方式，所以難免有時會落入區域最佳；基因演算法為盲目的產生新解，缺乏整體的導引性。但本文使用的共識因子是屬於群眾智慧，相較於單一傑出解的力量，理應會有更好的發展。

由於共識因子的導引機制是由母體所粹取的，所以對於母體的依賴性非常的大，故在產生共識因子之前，必須要有一套好的方法來產生母體，但經過實驗發現，光是增加母體數目(隨機產生)，並不能很明確的提升共識因子的解品質，所以借用別人的演算法作為初始母體，也就是之前所提到的 ACOFT-MWR (Huang & Liao, 2006)這個方法。其中 ACOFT-MWR(ACO)代表由這此方法抽取出 ACO 所產生的解當作母體初始解，另外 ACOFT-MWR(Taboo)則是抽取出這個演算法裡面每代 taboo 的最優秀解為母體初始值。

不過可惜的是即使改變了母體，共識因子所產生解依然無法令人滿意，於是本論文在共識因子加入禁忌演算法(Taboo Search)來提升解的品質，關於如何使用禁忌演算法的細節於 3.5 會仔細說明。

共識因子產生解的過程主要可以分成兩個步驟，首先為建立導引機制，此導引機制稱之為共識矩陣(consensus matrix)，並根據此引導機制來產生新解。

共識因子的使用程序步驟如下：

1. 取得 $N_{initial}$ 組解，把 $N_{initial}$ 的解放到母體 P(t)裡面，此時 t=0。
2. 把解由母體 *P(t)*抽出，並放入 *setS*。
3. 利用 *setS* 裡面的解，產生導引機制(3.3.3 會詳細說明)。
4. 根據導引機制，產生出 N_{new} 組新解(3.3.4 會詳細說明) 。
5. 把 *P(t)*和 N_{new} 的解放到 *SSset* 裡面，再由 *SSset* 裡面抽取出 *N* 組解放入 *P(t)*，且 $t = t + 1$ (3.3.5 會詳細說明)。
6. 如果已達停止條件則停止程序，否之回到步驟 2。

通常停止條件有兩種方式，第一種方法為固定的迭代數，此種方法較為方便，且計算時間較容易控制，缺點是有時候明明解空間已經收斂，卻還盲目的計算浪費時間，或是解空間還未收斂，就匆匆停止；避免上述的情況就使用第二種方式為設定一固定數目 *N*，連續 *N* 代最佳解都沒有更新時，才停止演算程序，這種演算法可以避免收斂的情況發生，但由於停止的代數不固定，所以計算時間無法掌握，只能用估計的方式進行。

3.3.2 導引機制

建立導引機制：共識矩陣(Consensus Matrix)如圖 3.11 所示，必須要注意的是，由於本文基於瓶頸飄移法的觀念把 JSSP 問題拆解為|M|個小型子問題，所以共會有|M|組共識矩陣，稱 M_i 為機器 i 的共識矩陣。

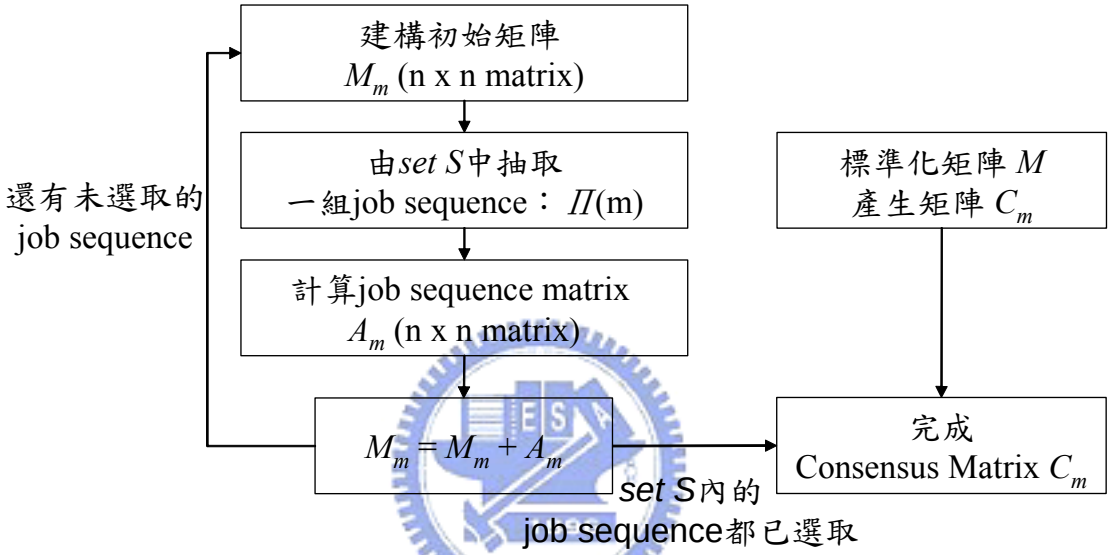

圖 3.11　共識矩陣產生流程

本研究提出的導引機制(consensus matrix , C_m，其中 m：第 m 台機器)，產生步驟如下：初始為一個 0 矩陣 M_m ($n \times n$ matrix，n 為工件個數)，更新方法由 *setS* 中抽取每組解的排列(job sequence)，其中所計算出此機器 m 裡每個工件的先後關係矩陣 A_m(job sequence matrix)，並利用此矩陣 A_m(下面會提到)更新矩陣 M_m $(M_m=M_m+A_m)$ ，重複上述步驟直到 *setS* 為空集合，最後再標準化 M_m 就是本文所要求的導引機制 C_m。

其中產生 A_m 矩陣，如圖 3.12 所示，先產生一個「初始工件優先關係矩陣」稱之為：$A = [a_{ij}] = 0$ ($1 \le i \le n$, $1 \le j \le n$, $n \times n$ matrix)，此外在工件排序 X_i 中，若有任兩工件 J_i、J_j 有順序關係，也就是工件 J_i 在工件 J_j 之前，a_{ij} 就加 1，用來表示 J_i 在工件 J_j 前面出現過一次。另外 rank(J_i)為工件排序內工件 i 的優先順序。必須要注意的是如果使用的是具有權重(weight)的共識因子法則，如：consensus linear 或 consensus double 這兩種方法的時候，並不是加上 1，而是加上不同的權重值，權重值設定為：

$$權重值 = \frac{A_{Makespan} - Best}{Worst - Best} \qquad (3)$$

$A_{Makespan}$ 為被抽取出來的解 $\prod(m)$ 的完工時間，*Best* 為 *Sset* 裡面最好一組解的完工時間，*Worst* 為 *Sset* 裡面最差一組解的完工時間。

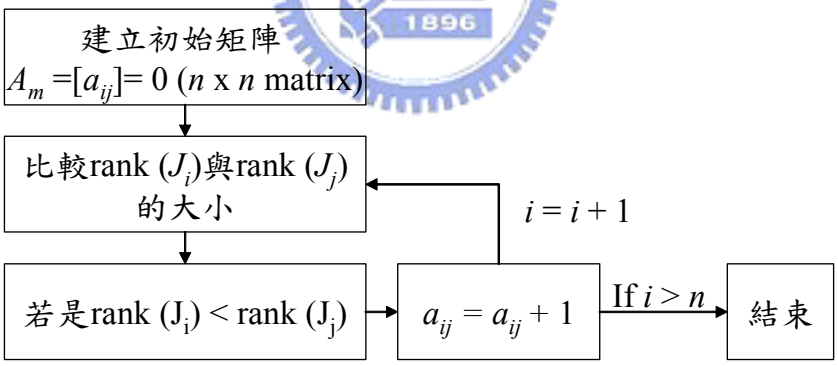

圖 3.12 產生 m 機器的工件優先關係矩陣

優先關係矩陣釋例：給定一組工件排序 ，藉由圖 3.11 的流程，就可以產生如表 3.2 的 m 機器的工件優先關係矩陣 A_m 。

表 3.2　m 機器的工件優先關係矩陣 A_m

A_m	後面(j)			
		J1	J2	J3
前面(i)	J1		1	1
	J2	0		0
	J3	0	1	

標準化矩陣則是把 M 矩陣裡面所有的元素都除以一個數值，通常是累加上的總值，原先 M 矩陣裡面的元素代表的是工件 i 在工件 j 出現的次數，如果都除上一個數值的話，這個累積值就會變成機率，也就是本文要的共識矩陣 $C_{m,i,j}$=0.9 的意思是機器 m 的共識矩陣之中，工件 i 在工件 j 出現的機率是 0.9，也就是工件 i 在工件 j 後面出現的機率是 0.1。

產生共識矩陣的釋例，如果已知有一個|J|=3，|M|=2 的 JSSP 問題，假定有兩組解 A,B 分別是：

$$\prod_A = \{\prod_A(1), \prod_A(2)\}, \prod_B = \{\prod_B(1), \prod_B(2)\}$$

其中

$$\prod_A(1) = \{1,2,3\}, \prod_A(2) = \{1,3,2\}, \prod_B(1) = \{3,2,1\}, \prod_B(2) = \{2,1,3\}$$

使用上述的手法，就可以得到累積矩陣 M_m(表 3.3)，和標準化矩陣 C_m(表 3.4)

表 3.3　累積矩陣 M_m, m=1,2

m=1	後面(j)			
		J1	J2	J3
前面(i)	J1		1	1
	J2	1		1
	J3	1	1	

m=2	後面(j)			
		J1	J2	J3
前面(i)	J1		1	2
	J2	1		1
	J3	0	1	

表 3.4　標準化矩陣 C_m, m=1,2

m=1	後面(j)			
		J1	J2	J3
前面(i)	J1		0.5	0.5
	J2	0.5		0.5
	J3	0.5	0.5	

m=2	後面(j)			
		J1	J2	J3
前面(i)	J1		0.5	1
	J2	0.5		0.5
	J3	0	0.5	

3.3.3 產生新解

先定義一個空集合 $Q=\phi$，和一個所有工件的集合 $Z=\{J_i;\ 1\le i\le n\}$，主要為不斷抽取 Z 集合裡面的東西放入 Q 集合中(依工件編號的由小到大)，直到 Z 為空集合的時候停止，且使用「插入法」來決定 J_i 在 Q 內工件的執行順序。完成這個步驟就等於完成一組解裡面的一台機台，也就是 Q 裡面的工件排序為 $\prod(m)$，直到所有機台都產生了工件排序之後，稱之為一組解：

$$\prod = \{\prod(1),\prod(2),...\prod(|M|)\} \qquad \text{M} = 1, ,2 \ldots|\text{M}| \qquad (4)$$

至於同一組解 $\prod$ 裡面，哪一個機器要先做工件排序也是很重要的，因為先排入的工件會影響到後面工件的排序狀況，所以這時候本論文使用參數 TML(total machine loading)參數來決定順序，一般來說 TML 參數越大的代表此機器越重要，因此要越先排列。TML 的參數如下所示，稱之為 $\pi(m)$。

$$\pi(m) = \sum_{j=1}^{|J|} PT(\sigma_m^j) \qquad \forall m = 1,...,|M| \qquad (5)$$

本論文演算法裡用共識矩陣產生一組新解的步驟程序如下：

1. 計算所有機器的 TML 值，並建立一個機器集合 *Mset*。
2. 抽取出 Mset 裡面 TML 權重最大的機器 m，若 Mset 為空集合則停止程序。
3. 為機器 *m* 建立一個工件集合 *Jset* 和一個空集合 *Q*。
4. 抽取 *Jset* 裡面編號最小的工件，並依照「插入法」將工件排入到集合 *Q* 裡面。
5. 如果 *Jset* 不為空集合，回到步驟 4。
6. 如果 *Mset* 不為空集合，回到步驟 2。

由於插入法的說明過於繁雜，因此本文解釋程序步驟，並同時使用一個簡單的例子來說明：

以下是「插入法」的程序步驟：

已知條件為：共識矩陣、要插入的工件 J 和 Q 集合中的排序狀況，，最後會得到的為：工件 J 插入到集合 Q 裡。

1. 先決定合理區域，以此例來說「開始→終點」之間的區域就是合理區域
2. 由合理區域中的「工件」(不包含起始工件和結尾工件)，挑選機率最大的當作「參考指標」如圖 3.13(此例 0.8 最大，因此挑選 J2 為參考指標，機率值由共識矩陣可知)。
3. 針對參考指標 J2 隨機抽取一參數 P~[0,1]，以此例若 P 值小於 0.2，則更新合理區域為「開始→J2」，反之若 P 值大於 0.2，則更新合理區域為「J2→終點」。
4. 若合理區域中間已經沒有工件，則插入工件 J3 於合理區域中間(如上述 P 小於 0.2 的狀況)並結束程序，否之回到步驟 2。

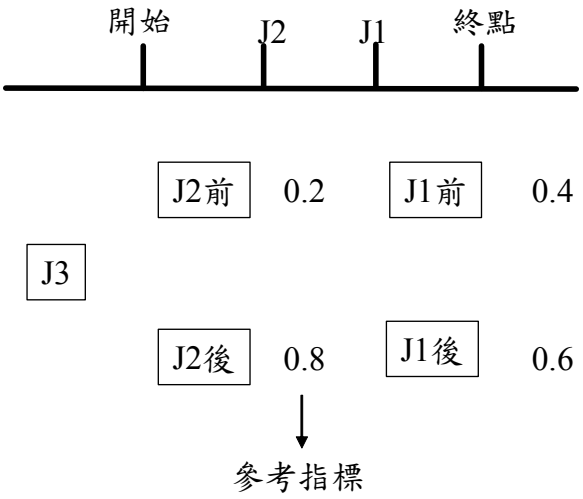

圖 3.13　參考指標

3.4 田口方法(Taguchi Methods)

田口方法是 1949 年，由日本田口玄一(Genichi Taguchi)博士發明的，當時剛好第二次世界剛結束，日本進行戰後的重建工作，面臨成本高昂，以及缺乏生產設備和工程師。在如此惡劣的環境之下，如何生產利用最低的成本生產最高品質的產品是一項具有挑戰和重點的課題。當初田口玄一博士發明的動機是因為於工作時發現傳統的實驗設計方法在實務上並不適用，為了發展「品質工程」，他竭力於發展出一套新的實驗設計方法。

田口方法通常用來改善品質工程，所發展的方法為為透過實驗來最佳化系統參數的方法，而非傳統的實驗設計使用統計參數，此方法的原先目的為改善品質工程，在日本又稱之為品質工程(quality engineering)，田口博士發展的方法就簡稱為田口方法(Taguchi methods)，不過近年來也有人用來使用在排程上面發展，但大多為應用於設計參數部分，只有少數的學者使用在演算法的架構上，特別是 JSSP 問題幾乎無人使用過。

田口方法的優點：假設染色體內含 5 個基因的超小型問題，目的為由兩條染色體中做實驗產生出「最佳配方」染色體，如果用傳統的實驗設計方法，需要做的實驗次數為 2^5=32 次，但使用田口方法的話只需要做 8 次實驗，相差四倍的差距；放大到更大的 JSSP 問題來看，如果是一個|M|=10、|J|=10 的中小型問題，這樣一條染色體裡面含有 100 個基因，也就是說由兩條染色體產生一條最佳組合的話，需要做 2^{100} 的實驗次數才能找到這最佳組合，但使用田口方法的話，只需要 128 次實驗就可以得到，由此可知田口方法具有大量的減少實驗次數的效果。

3.4.1 田口方法架構

田口方法應用再排程問題的主要概念在於「降低實驗次數」，主要期望在使用一種有系統的實驗方法來產生新母體，簡單的基本流程如下：

1. 由母體 *P* 隨機抽取兩條染色體。
2. 使用直交表(orthogonal array)做實驗。
3. 根據直交表的實驗結果，挑選出最佳的基因配方組成一條新的染色體。

3.4.2 直交表

田口方法是如何減少實驗次數？，答案就在於它使用了一種實驗設計的方法，也就是利用直交表(orthogonal array)，此種直交表就是經過統計方法設計出來的一種特定實驗方法。田口博士提出了 18 種基本直交表，Phadke (1989)，稱之為標準直交表(standard orthogonal arrays)。為了要能夠直接使用標準直交表，我們的實驗水準數必須要和直交表中的水準數配合才行(水準數本論文採用機器數目)，且為了減少實驗次數，最好是盡量使用最小的直交表來做實驗，其中本研究方法的實驗方法裡面採用 L_{16} 這個標準直交表(表 3.5)。

表 3.5　L_{16} 標準直交表

L16	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	基底a	基底b	基底c	基底d	a+b	a+c	a+d	b+c	b+d	c+d	a+b+c	a+b+d	a+c+d	b+c+d	a+b+c+d
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1
3	0	0	1	0	1	1	0	1	0	1	1	0	1	1	1
4	0	0	1	1	1	1	1	1	1	0	1	1	0	0	0
5	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
6	0	1	0	1	1	0	1	1	0	1	1	0	1	0	0
7	0	1	1	0	0	1	0	0	1	1	0	1	1	0	0
8	0	1	1	1	0	1	1	0	0	0	0	0	0	1	1
9	1	0	0	0	0	1	1	0	0	0	1	1	1	0	1
10	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0
11	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
12	1	0	1	1	1	0	0	1	1	0	0	0	1	0	1
13	1	1	0	0	1	1	1	1	1	0	0	0	1	1	0
14	1	1	0	1	1	1	0	1	0	1	0	1	0	0	1
15	1	1	1	0	0	0	1	0	1	1	1	0	0	0	1
16	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0

3.4.3 田口實驗

所謂的田口實驗，就是由一組解(包含兩條不相同的解)，經過交直表實驗產生一組最佳配方的新解，方法如下：假設已知兩組解 A、B 且這兩組解分別是 $\prod_A = \{\prod_A(1), \prod_A(2), \prod_A(3)\}$ 和 $\prod_B = \{\prod_B(1), \prod_B(2), \prod_B(3)\}$，於是可以找到最小值交表配合有 3 基因，也就是 3 水準的標準直交表 L_8，如表 3.6 所示

表 3.6　L_8 標準直交表

L8	1	2	3	4	6	5	7
	基底a	基底b	基底c	a+b	c+a	b+c	a+b+c
1	0	0	1	0	1	1	1
2	0	1	0	1	0	1	1
3	0	1	1	1	1	0	0
4	1	0	0	1	1	0	1
5	1	0	1	1	0	1	0
6	1	1	0	0	1	1	0
7	1	1	1	0	0	0	1

表中的 0 代表使用解 A 裡面的基因，表中的 1 代表使用解 B 裡面的基因由此表看來，令 E_i 為實驗的基因，其中 i 代表第幾組實驗，於是第一組實驗解果可以表示為 ,第二組實驗解可以表示為 ，以此類推到第 8 組解，在分別使用要徑法計算每一組實驗解的完工時間，會得到下表 3.7：

表　3.7　完工時間

L8	1	2	3		
	基底a	基底b	基底c	完工時間	完工時間的倒數
1	0	0	1	1	1
2	0	1	0	5	0.2
3	0	1	1	5	0.2
4	1	0	0	10	0.1
5	1	0	1	5	0.2
6	1	1	0	10	0.1
7	1	1	1	5	0.2

最後計算，每個基因的最佳配方，用每一列的累積完工時間倒數當作權重值，最後權重最大的就是我們要的基因，以此為例的話，新解的第一個基因使用解 A 的基因權重為 1.4，使用第二個基因的權重為 B，由於 1.4>0.6，所以最佳配方解的第一個基因使用解 A 的基因，以此類推可求得最後新解的基因值 $\prod_{new} = \{\prod_A(1), \prod_B(2), \prod_B(3)\}$ 為 A 和 B 這兩組解的最佳配方新解。

3.4.4 田口方法實際應用在 JPPS 問題

田口方法的實際應用於 JSSP 問題，下面是本論文演算法所使用的程序步驟如下：

1. 隨機產生 N_P 組初始解，$t = 0$，$STOP = 0$。
2. 把 N_P 組解放入到母體 $P(t)$，$t = t+1$，$STOP = STOP + 1$，$X = 0$。
3. 隨機挑選 $P(t)$ 中的兩組解為一組合(同一個世代 t 中，不會抽取到同樣的組合)。
4. 根據被挑選出來的兩組解，做一次田口實驗，$X = X+1$。
5. 產生出一組新的解 $S_{t,x}$，如果 $X<N_{new}$，並回到步驟 3。
6. 如果已達停止條件($STOP>N_{stop}$)，則停止程序。
7. 由母體 P(t) + X 中挑選出最佳的 N_P 組解當作下一代的初始母體，回到步驟 2。

其中 N_P 代表初始母體數目，N_{new} 代表新解數目，N_{stop} 為停止條件累積次數，t 為世代數，$S_{t,x}$ 為第 t 個世代，所產生的第 X 組新解

3.5 禁忌演算法(Taboo Search ; TS)

禁忌演算法(Taboo Search ; TS)屬於近似求解法裡面的迭代搜尋機制，最早由Glover (1986)提出，主要為使用可行解(feasible solution)的鄰近解來作為尋優的方向，慢慢的朝向良好的解區域移動，且禁忌演算法並不像模擬退火法是來自於「物理現象」的改變，也不像是螞蟻演算法來自「自然界生物體」的搜尋現象，而是使用「人工智慧」和記憶來進行尋優動作。舉個例子來說就好像是人類天天嚐試不同的事物已達到更好的生活品質為目標，其中嚐試過的事物會被放入在大腦記憶之中裡面也就是稱之為「禁忌名單」(taboo list；TS)的東西，此外為了提升更好的品質，我們又會記住人生中特別重要的抉擇，也就是稱之為「長期記憶」的東西，在此章我們會描述禁忌演算法的概念和使用方法，但基本上本論文和ACOFT-MWR 中使用的禁忌演算法都是來自於 Nowicki & Smutnicki (1996)所提出的 FT(fast taboo)又稱為 TSAB。

3.5.1 禁忌演算法架構

基本運作邏輯如下：首先要定義「移步」(move)，通常我們稱之為由一個解轉換到另一個解的過程，也就是說我們可以使用移步的方式對目前的解產生一群「鄰近解」，並由鄰近解中挑選最佳的解來改變目前的解，於是乎在每一個迭代裡面都會往更好的解和解空間移動；此外為了避免「循環」(cycle)，設立了一個叫做「禁忌名單」(taboo list)這個機制來紀錄最近曾經使用過的移步，以避免重複走過的路徑(也就是使用過的移步)，禁忌名單也稱為短期記憶(short-term memory)；為了要強化特別良好解空間的搜尋，建立了一個機制稱之為長期記憶(long-term memory)，主要在紀錄程經表現過特別良好的解，期待原先搜尋路徑部分結束後，可用於重新把曾紀錄良好的解重新作為搜尋的初始解，直到所有在長期記憶的解空間都已經搜尋完畢即停止演算法。

3.5.2 禁忌演算法演算流程

禁忌演算法有幾個重要的元件：移步、鄰近解、禁忌名單、渴望準則、長期記憶，下面我們會針對這些元件詳細的說明：

移步指的是由目前解(current solution)改變到另一個解的過程，其中移步的方式會根據問題的特性來設計，通常在排程上會使用兩兩交換法 (swap)、插入法(insert)與消去法(add/drap)，不過在本演算法只有使用到 swap，也就是兩兩交換的動作。

本論文裡面的移步使用 Nowicki & Smutnicki (1996)的定義方式：將要徑(critical path)分為數個區塊(block)，且每個區塊表示連續工件在相同的機器上作業，換句話說就是每一區塊中包含 1 到多個作業不等，且這些作業都是要徑上的連續作業並加工於同一機台；移步的方式為交換最前面區塊的最後面兩個作業，最後面區塊的最前面兩個作業，其他區塊則交換最前面的兩個作業和最後面的兩個作業，也就是說如果要徑可以分為 n 個 block，則可採用的移步數目最多為 $(2n-2)$，這個數目也就是鄰近解的最大值。

本研究演算法定義鄰近解為：目前解經過移步改變之後的解集合。我們使用要徑法計算出所有鄰近解的完工時間，用以判斷目前的解要往哪一個鄰近解移動會提升更好的解品質，通常我們會挑選品質最好的解做為移動的方向，但是必須特別注意的是，如果所有鄰近解品質都比目前的解還要「差」時，仍必須挑選鄰近解裡面最好品質的解做為移動的方向，也就是即使知道此改變只會讓目前的解品質變差，我們仍要改變以求的更大的突破，就像人生有時必需忍受一時的低潮，用來創造出另一個高峰。

禁忌名單又稱為短期記憶(short-term memory)，禁忌名單是一個佇列(queue)的資料結構(先進先出；first in first out；FIFO)，禁忌裡面儲存的元素就是最近移步的紀錄，用來避免不小心採用過去曾用過的移步而回到改變前的狀態。

簡單說如果現在有一組解確定採用移步 M1(2,3)，則把此移步反轉為 M1(3,2)放到禁忌名單的最後面，並根據採取的移步修正解的狀態；禁忌名單，用以避免再次使用移步 M1(3,2)而回到之前的解狀態。必須要注意的是禁忌名單通常都會有一長度限制，若加入新的移步造成禁忌名單過長，則必須去除禁忌名單中存在時間最久的一筆資料。

禁忌名單的長度也是一門很大的學問，一般而言長度越長越大代表移步受到的限制越多，可以避免重複搜尋相同解空間的狀況，但也因此限制住解空間的移動導致容易陷入區域最佳解，此外也要付出龐大的記憶體空間來儲存禁忌名單，此外還會增加計算時間；禁忌名單長度太短的話則是代表受到的限制不多，解空間很容易往不同的方向去移動，容易跳脫區域最佳解，計算時間縮短，但卻會造成解收斂速度過慢，並很容易陷入循環(cycle)之中，所以通常排程上使用 Glover (1986)所提出的魔術數字 7 為禁忌名單的長度，這也是本論文演算法所使用的長度。

此機制主要用來打破移步在禁忌名單中所受到的限制狀況；通常在一般的情況下，被紀錄在禁忌名單的移步是不會被採用的，但如果這組解滿足一個特別的條件的話，就會不管禁忌名單的限制而採用此移步，稱此條件為渴望準則；在本論文演算法之中，這個特別的條件為：採用此移步後的完工時間，小於目前最佳解的完工時間。此時更新禁忌名單的方法是，把此移步由禁忌名單中抽取出來並同時放入禁忌名單的最後面。

long-term memory (長期記憶)，長期記憶屬於 TS 裡面的強化機制，主要目的為再次搜尋之前有找到，但是未搜尋的良好解空間，經此強化中堂可以找到更好的解，使用方法為先建立一個長期記憶名單(可自由設定一門檻值，本論文使用的是當解品質超越此迭代裡面所找到的最佳解)，並搜尋所有長期記憶名單的裡的鄰近解。仔細的搜尋過長期記憶名單裡面的所有解，這種方式可以大大的提升找到解的品質，雖然也會大大的增加演算時間，所以本論文設定的長期記憶名單長度為 3，且當長度超過 3 的時候，會移除長期記憶名單裡面存在最久的一筆資料。所以我們的長期記憶名單會包含三種元素，第一種是扣除目前選取的移步以外的鄰近解集合，第二種為目前的禁忌名單，第三種為目前的工件排序狀態。

本論文所使用的 TS 步驟程序如下：

1. 取得一組初始解，並把此解放入長期記憶名單 LM 之中，此時 $LM = 1$。
2. 抽出 LM 裡面最新的一筆資料，做為往後的起始解更新解，且迭代次數 $N = 0$，若 $LM = 0$ 則停止程序。
3. 根據移步，找出此目前解的鄰近解。
4. 依據所挑選的移步解，修改禁忌名單和目前此迭代最佳解(current solution)。
5. 如果所挑選出來的移步解完工時間小於此迭代最佳解，把此狀態放入 LM，並更新迭代裡的最佳解，$LM = LM + 1$。
6. 進行迴圈測試(cycle test)，如果陷入迴圈則回到步驟 2。
7. 如果已經達到迭代次數 N>3000 次，則回到步驟 2，否之 $N = N + 1$，並回到步驟 3。

本論文所使用的迴圈測試(cycle test)為紀錄過去連續 30 個迭代解的完工時間，並計算是否有超過 6 個迭代出現相同的完工時間，如果發生此種狀況，會視為目前的搜尋方向已經陷入迴圈，並需要停止目前的搜尋方向。

第四章　實驗結果

本論文使用 Visual C++ 6.0 來撰寫程式，所有的程式都在 Windows XP 系統下進行測試，而實驗的電腦為 Intel(R) Pentium(R) D CPU 3.4GHz，記憶體為 512MB RAM。

4.2 節為本論文演算法 Huang & Liao (2006)所提出的 ACOFT-MWR 的比較，在 4.3 節為本論文演算法和過去著名方法比較：Dorndorf & Pesch (1995)的 SB-GA，Nowicki & Smutnicki (1996)的 TSAB(FT)，Balas & Vazacopoulos (1999)的 SB-RGLS2，Pezzella & Merelli (2000)的 TSSB、Schultz et al. (2004)SHKT-720min 和 Huang & Liao (2006)的 ACOFT-MWR，必須注意的是，由於無法取得這些論文的平均解品質，故 4.3 的比較方法為比較所能找到的最佳解。

測試的題目可以在標竿題庫 OR-Library(http://mscmga.ms.ic.ac.uk/info.html)上面找到，以下是本論文挑選出來的標竿題目

1. 9 個來是於 Lawrence(1984)，分別是 LA24-LA25、LA29 和 LA36-LA40。
2. 5 個來自於 Applegate & Cook (1991)，分別是 ORB1-ORB5。
3. 3 個來自於 Adams et al. (1988)，分別是 ABZ7-ABZ9。
4. 2 個來自於 Fisher & Thompson (1963)和，分別是 TF10 和 FT20。

4.1 前置實驗(Pilot Study)

本論文的前置實驗主要測試 Lawrence(1984)所提出的 La29 這個問題，表 4.1 為本論文演算法「基本架構組合」(同表 3.1)。

表 4.1　基本組合表

初始解	更新解 1	更新解 2	修正解
(A1)Random	無	(C1)consensus normal	(D)Taboo
(A2)ACOFT-MWR(ACO)	(B)Taguchi	(C2)consensus linear	
(A3)ACOFT-MWR(Taboo)		(C3)consensus double	

由於本論文演算法的組合大多在於參數和流程的不同，為了達到實驗的公平性，我們固定初始解所產生的數值，也就是本論文先分別計算 5 次三種初始值並取得 15 組的參數，之後無論使用怎樣的更新解和修正解組合，都是由這 15 組參數取得，以建立公平性，因此演算法「基本組合」有 21 種，列於表 4.2。

表 4.2　基本演算法組合表

初始解	A1							A2							A3						
更新解 1	無			B				無			B				無			B			
更新解2	C1	C2	C3	無	C1	C2	C3	C1	C2	C3	無	C1	C2	C3	C1	C2	C3	無	C1	C2	C3
修正解	D																				

特別需要注意的是，因為本論文的初始解分成三大部分，第一種為隨機產生初始解(A1)，用意在於希望可以和其他的初始解比較出，共識因子和母體的相依特性，希望能在實驗中確認共識因子在優秀的母體之下可以表現出更好的解品質；此外第二種初始解(A2)和第三種初始解(A3)在於已經具有良好的初始解特性之下，故使用不同組合參數的共識因子和田口方法來實驗，但是必須要注意的是，第二種初始解(A2)是可以在更新解的部分做世代循環的動作，但是第三種(A3)卻無法單純的做世代循環，原因在於第三種的初始解為 ACOFT-MWR(Taboo)，這個解已經是經過禁忌演算法搜尋過的最佳解，單單使用共識因子所產生的新解是無法超越的，無論是產生多少組共識因子，解的品質都無法超越(ACOFT-MWR(Taboo))，也就無法達到更新母體，除非我們也在世代循環的時候也加入禁忌演算法來提升共識因子。基於以上的原因本文設計出三種實驗架構來判斷那種演算法組合較為優秀，以下介紹這三種實驗的特性和實驗結果

第一種實驗流程為圖 4.1 實驗結果在表 4.3，只針對初始解為 A3 的部分，原因如同之前所述，由於無法直接世代更新，所以只好先大量的產生更新解，再抽取出最好的 200 組解過修正的動作。在此實驗有 16 種組合，實驗代號為 V1-V16。

圖 4.1　第一種實驗流程概念

第二種實驗流程如圖 4.2 結果在表 4.4，主要針對「A2」實驗，由於 A2 這種方法不需要加入禁忌演算法就可以世代更新，所以我們採用先世代更新的，之後再挑選出最優秀的 200 組做修正的動作。在此實驗有 10 種組合，實驗代號為 V17-V26。

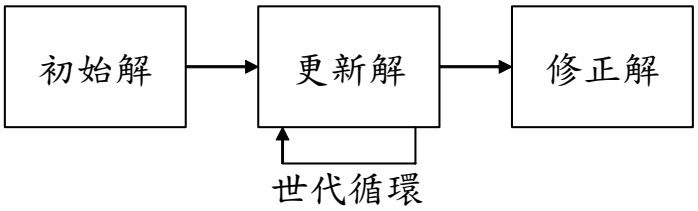

圖 4.2　第二種實驗流程概念

第三種實驗流程如圖 4.3 結果在表 4.5，這種實驗適合於三種不同的初始解，主要想法為在更新解世代更新的時候，加入禁忌演算法來提升解，以便於有效的利用更好的母體來提升共識因子或是田口方法。在此種實驗裡有 21 種不同的組合，實驗代號為 V27-V47。

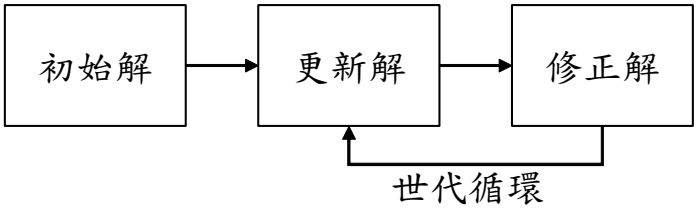

圖 4.3　第三種實驗流程概念

如同之前所提到的，把每一種組合都針對 LA29 這個問題做 5 次實驗，再分別抽取出產生組數 500 萬組、1000 萬組和 2000 萬組的結果來做比較，比較的結果如同表 4.3、表 4.4 和表 4.5 所表現的。

由這三個實驗結果發現很多的組合的差異幾乎都不大，但是有兩種演算法的組合的表現較為優秀，故挑選出來當作本論文最佳配方組合，分別是「V13」和「V41」這兩組組合，在 4.2 和 4.3 節會針對這兩個組合在有指標性的題目做實驗。

其中實驗一所採用的初始解救是 ACOFT-MWR 此方法演算過程中的第 11-110 代的禁忌演算法的最佳解，共 100 組為本研究方法採用的初始母體；實驗二所採用的為 ACOFT-MWR 此方法演算法中的第 21-60 代的螞蟻演算法的最佳解，由於 40 代共會產生 120 組螞蟻解，挑選最佳的 100 組螞蟻解當作本研究方法的初始母體；實驗三則是隨機產生 10000 組和 ACOFT-MWR 的前 21-60 代取 30 組最佳螞蟻解，以及 ACOFT-MWR 的前 11-110 代的前 30 組最佳禁忌演算法解。

表 4.3　實驗一的結果

初始解	更新解 1	更新解 2	修正解	產生多少組 更新解	比例 共識因子 / 田口方法	挑選多少組 做修正	100 萬	500 萬	2000 萬	代號
A3	無	C1	D	1000	100% / 0%	200	1178.6	1173.0	1173.0	V1
				10000			1177.8	1174.0	1174.0	V2
		C2		1000			1176.4	1176.0	1173.0	V3
				10000			1177.0	1174.4	1171.0	V4
		C3		1000			1175.6	1175.6	1171.4	V5
				10000			1176.4	1172.2	1171.6	V6
	B	無		100	0% / 100%		1181.0	1177.0	1173.0	V7
				1000			1180.0	1175.0	1174.0	V8
		C1		1000	90% / 10% 或 50% / 50%		1175.6	1171.4	1170.2	V9
							1176.2	1171.2	1169.8	V10
				10000			1175.5	1171.6	1170.0	V11
							1176.0	1172.0	1171.0	V12
		C2		1000			1178.8	1173.0	1169.0	V13
							1177.8	1174.0	1174	V14
		C3		1000			1179.0	1172.8	1170.4	V15
							1180.0	1171.8	1170.2	V16

表 4.4　實驗二的結果

初始解	更新解 1	更新解 2	修正解	產生多少組 更新解	比例 共識因子 / 田口方法	挑選多少組 做修正	100 萬	500 萬	2000 萬	代號
A2	無	C1	D	1000	100% / 0%	200	1184.4	1175.0	1177.0	V17
		C2		1000			1180.2	1173.0	1171.2	V18
		C3		1000			1184.8	1175.8	1174.0	V19
	B	無		100	0% / 100%		1188.8	1177.4	1176.0	V20
		C1		1000	90% / 10% 或 50% / 50%		1182.8	1177.0	1173.2	V21
							1185.2	1175.4	1172.0	V22
		C2		1000			1182.4	1177.0	1177.0	V23
							1180.4	1173.2	1176.0	V24
		C3		1000			1183.6	1178.0	1173.8	V25
							1177.4	1172.2	1172.6	V26

表 4.5　實驗三的結果

始解	更新解 1	更新解 2	每代產生的解數量	比例 共識因子/田口方法	每代挑出多少更新解做 D 的修正動作	100 萬	500 萬	2000 萬	代號
A1	無	C1	1000 組	100% / 0%	30 組	1183.2	1175.4	1173.6	V27
		C2				1178.8	1175.2	1173.0	V28
		C3				1179.4	1176.3	1172.8	V29
	B	無		0% / 100%		1180.8	1175.8	1173.4	V30
		C1		70% / 30%		1184.4	1178.8	1172.0	V31
		C2				1180.0	1175.4	1171.2	V32
		C3				1177.0	1173.2	1171.4	V33
A2	無	C1		100% / 0%		1176.8	1174.2	1171.6	V34
		C2				1177.8	1173.6	1169.2	V35
		C3				1177.0	1172.4	1170.0	V36
	B	無		0% / 100%		1177.8	1175.4	1171.0	V37
		C1		70% / 30%		1176.0	1174.8	1172.6	V38
		C2				1177.8	1177.8	1171.2	V39
		C3				1174.0	1170.6	1169.2	V40
A3	無	C1		100% / 0%		1178.6	1167.6	1167.2	V41
		C2				1177.2	1168.2	1168.0	V42
		C3				1175.8	1169.6	1169.2	V43
	B	無		0% / 100%		1178.6	1175.2	1175.2	V44
		C1		70% / 30%		1175.8	1175.8	1167.8	V45
		C2				1181.8	1170.6	1168.6	V46
		C3				1177.2	1171.2	1170.6	V47

4.2 本論文演算法最佳配方實驗

在 4.1 節本研究已由所有的實驗組合裡面挑出最佳配方組合，也就是 V13 和 V41 這兩種組合，之後在把這兩組最佳配方組合對有指標性的題目做實驗，由於 Lawrence(1984)雖然有 40 組題目，由於無論是使用本論文演算法或是標竿演算法(ACOFT-MWR)都能在設定停止條件內找到最佳解，所以不具代表性，因此本論文不予列出比較，所以共有 19 種具指標性的題目，如表 4.6

表 4.6　指標性題目

Lawrence(1986)	LA24-LA25、LA29和LA36-LA40
Applegate & Cook (1991)	ORB1-ORB5
Adam et al. (1988)	ABZ7~ABZ9
Fisher & Thompson (1983)	FT10、FT20

根據上面所挑選的題目對本論文在 4.1 節所得到的最佳配方組合 V13 和 V41 做實驗，每組題目實驗 5 次，且都有不同的停止條件(表 4.7)，比較的方法為計算平均最佳解的百分誤差(MRE)，RE 為目前找到的解減去此問題最佳解，再除上此問題的最佳解，最後再乘上 100%就是 RE 值，MRE 則是把 RE 值再除以總例子數目，就可以得到平均最佳解百分比誤差 MRE 值。表 4.7 表示 V13 和 V41 和 Huang & Lain (2006)年所提出的 ACOFT-MWR 方法比較。

$$MRE = \frac{solution \ - \ OPT}{OPT} \times 100\% \qquad (6)$$

其中 *solution* 就是演算法計算出的完工時間，*OPT* 為該題目的最佳解，如果尚無最佳解的話，則使用下標 LB(low bound) 來取代之。

表 4.7　指標性題目內容

問題	工件數 |J|	機器數 |M|	最佳解	最大停止條件(萬)
La24	10	10	935	1700
La27	10	10	1235	2000
La29	20	10	(1142,1153)	2000
La37	15	15	1397	5000
La40	15	15	1222	5000
Abz7	20	15	656	5000
Abz8	20	15	(645,665)	5000
Abz9	20	15	(661,679)	5000
Orb1	10	10	1059	1000
Orb2	10	10	888	1000
Orb3	10	10	1005	1000
Orb4	10	10	1005	1000
Orb5	10	10	887	1000
FT10	10	10	930	1000
FT20	20	5	1165	1000

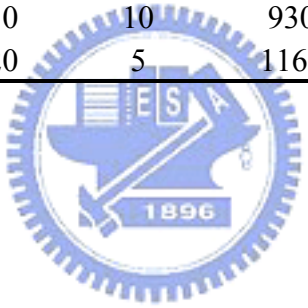

表 4.8　最佳配方演算法實驗結果比較

		ACOFT-MWR			V13			V41		
問題	OPT (LB,UB)	Best	Average	RE	Best	Average	RE	Best	Average	RE
LA24	935	935	938.4	0.3636	935	938.2	0.3422	935	937.8	0.2995
LA25	977	977	977.0	0.0000	977	977.0	0.0000	977	977.0	0.0000
LA27	1235	1235	1236.6	0.1296	1235	1241.6	0.5344	1235	1237.2	0.1781
LA29	(1142,1153)	1158	1165.6	2.0140	1163	1169.0	2.3643	1167	1168.0	2.2767
LA36	1268	1268	1268.0	0.0000	1268	1268.0	0.0000	1268	1268.0	0.0000
LA37	1397	1397	1398.8	0.1288	1397	1398.2	0.0859	1397	1397.2	0.0143
LA38	1196	1196	1196.0	0.0000	1196	1196.0	0.0000	1196	1196.0	0.0000
LA39	1233	1233	1233.0	0.0000	1233	1233.0	0.0000	1233	1233.0	0.0000
LA40	1222	1224	1226.4	0.3601	1224	1227.6	0.4583	1224	1229.0	0.5728
			小計RE=	2.9961		小計RE=	3.7851		小計RE=	3.3415
ORB1	1059	1059	1059	0.0000	1059	1059	0.0000	1059	1059	0.0000
ORB2	888	888	888.5	0.0563	888	889.0	0.1126	888	888.0	0.0000
ORB3	1005	1005	1007.4	0.2388	1005	1005	0.0000	1005	1005	0.0000
ORB4	1005	1005	1005	0.0000	1005	1005	0.0000	1005	1005	0.0000
ORB5	887	887	888.9	0.2142	887	889.0	0.2255	887	889.0	0.2255
			小計RE=	0.5093		小計RE=	0.3381		小計RE=	0.2255
FT10	930	930	930	0.0000	930	930	0.0000	930	930	0.0000
FT20	1165	1165	1165	0.0000	1165	1165	0.0000	1165	1165	0.0000
			小計RE=	0.0000		小計RE=	0.0000		小計RE=	0.0000
			MRE=	0.2191		MRE=	0.2577		MRE=	0.2229
ABZ7	656	658	663.2	0.7903	672	675.4	2.9573	670	675.4	2.9573
ABZ8	(645,665)	670	670.8	0.1194	679	688.2	6.6977	673	687.4	6.5736
ABZ9	(661,679)	683	687.4	0.6442	699	705.6	6.7474	691	706.0	6.8079
			小計RE=	1.5539		小計RE=	16.4024		小計RE=	16.3388

由表 4.8 可以發現，本論文演算法的求解品質在前三類的指標性題目和 ACOFT-MWR 並無很大的差異，在 19 個問題裡面 V13，贏了 3 個、平手 8 個和輸了 8 個，輸率大約為 42%；在 V41 裡贏了 5 個、平手 7 個和輸了 7 個，輸率為 36%。

其中在 ABZ7-ABZ9 這 3 個題目有很大的落差，故無法說本論文最佳演算法求解品質相似於 Huang & Liao (2006)年所提出的 ACOFT-MWR 方法，只能說再部分問題上相似，但在少數問題上則是輸的。

4.3 本論文最佳配方和過去著名方法比較

在 4.1 節已由所有的實驗組合裡面挑出最佳配方組合，也就是 V13 和 V41 這兩種組合，之後我們在把這兩組最佳配方組合對過去著名的方法做比較，在 LA 的 9 個問題上面，取最佳解的 MRE 值比較，比較的結果在表 4.9。

由表 4.9 可發現，雖然本論文演算法輸給 Huang & Liao (2006)年所提出的 ACOFT-MWR，但是和過去著名的方法比較，並無太大差異性，幾乎 MRE 都超過這些著名演算法，特別是 V13 這個方法僅敗給 ACOFT-MWR，而 V41 這個方法則是敗給 SB-RGL2、SHFT-720min 和 ACOFT-MWR 這三種方法。

表 4.9　本論文演算法和過去著名方法比較

年份		1995		1996		1999		2000		2004		2006		2008		2008	
方法		SB-GA		TSAB(FT)		SB-RGLS2		TSSB		SHKT-720min		ACOFT-MWR		V13		V41	
問題	OPT	最佳解	RE	最佳解	RE	最佳解	RE	最佳解	RE	最佳解	RE	最佳解	RE	最佳解	RE	最佳解	RE
LA24	935	957	2.35	939	0.43	935	0.00	938	0.32	938	0.32	935	0.00	935	0.00	935	0.00
LA25	977	1007	3.07	977	0.00	977	0.00	979	0.20	977	0.00	977	0.00	977	0.00	977	0.00
LA27	1235	1269	2.75	1236	0.08	1235	0.00	1235	0.00	1238	0.24	1235	0.00	1235	0.00	1235	0.00
LA29	(1142,1153)	1210	5.95	1160	1.58	1164	1.93	1168	2.28	1161	1.66	1158	1.40	1164	1.93	1167	2.19
LA36	1268	1317	3.86	1268	0.00	1268	0.00	1268	0.00	1268	0.00	1268	0.00	1268	0.00	1268	0.00
LA37	1397	1446	3.51	1407	0.72	1397	0.00	1411	1.00	1397	0.00	1397	0.00	1397	0.00	1397	0.00
LA38	1196	1241	3.76	1196	0.00	1196	0.00	1201	0.42	1196	0.00	1196	0.00	1196	0.00	1196	0.00
LA39	1233	1277	3.57	1233	0.00	1233	0.00	1240	0.57	1233	0.00	1233	0.00	1233	0.00	1233	0.00
LA40	1222	1252	2.45	1229	0.57	1224	0.16	1233	0.90	1224	0.16	1224	0.16	1224	0.16	1229	0.57
MRE			3.48		0.37		0.23		0.63		0.27		0.17		0.23		0.31

4.4 實驗分析結論

本論文在實驗的過程之中，發現的一些關於 JSSP 問題和「共識因子」與「田口方法」的實驗心得，分別如下，本論文使用不同的實驗去說明這些結論。

1. 共識因子和母體之間的關係，越好的母體越能提升共識因子的解。
2. 越好的更新解，不一定能創造越好的修正解。
3. 求解效益的不同。

4.4.1 母體的差異性

關於共識因子本研究發現，共識因子使用越好的母體，計算出來的解會越好，但卻無法找到更好的解，以之前的第三種實驗實驗為例子，抽出 V27、V34 和 V41 這四個方法來說明，數據如表 4.10 所示。

表 4.10　母體的差異性

	初始解平均	100萬	500萬	2000萬	3000萬	5000萬	
RANDOM	5000	1183.2	1175.4	1173.6	1171.2	1167	V27
ACOFT-MWR(ACO)	2000	1176.8	1174.2	1171.6	1169.4	1167	V34
ACOFT-MWR(Taboo)	1210	1178.6	1167.6	1167.2	1167	1166.4	V41

可以發現的是，在這三個實驗裡面，除了母體不同以外，其他的實驗因子都相同，但是可以很明確的發現，在三種不同的母體效果之下(品質差異)，初始解的比例分別為 5000：2000：1210，彼此間的差異相當的明顯，但是在產生解數目增加的情況之下會慢慢趨近相同。故可以得知「共識因子」需要一個有導引性的方法來產生新母體，並因此快速的找到收斂解，但是儘管收斂速度不同，最後會所找到的收斂解卻都大致相差不遠。

4.4.2 母體的結構性

關於共識因子本研究發現，共識因子的母體結構性越強的，共識因子所能產生的更新解「品質」越好，以之前的第兩種實驗實驗為例子，抽出第一種實驗和第三種實驗的平均數字做比較，數據如表 4.11 所示。

表 4.11 母體的結構性

母體來源	解平均	共識因子產生的解	
ACO + Taboo	1180-1200	2000-2700	第一種實驗
Consensus + Taboo	1180-1190	1180-1210	第三種實驗

由以上的兩種不同的母體來源可以發現，如果共識因子的母體來源為 ACO + Taboo，此時母體的解品質平均約為 1180-1200，使用共識因子產生的更新解品質約為 2000-2700；若把共識因子的母體來源變更為 Consensus + Taboo，此時母體的解品質平均約為 1180-1190，但是卻可以讓共識因子的更新解品質提高到 1180-1210 的程度，表示出即使母體的解品質相同，但是根據母體的結構性的差異，會絕對的影響到共識因子所產生的解。也就是說，Consensus + Taboo 這種母體具有強大的結構性，比起 ACO + Taboo 此種方法更能讓「共識因子」產生優秀的子代解。

也就是結構性越強的母體，使用共識因子可以產生越優秀的更新解，相反的如果母體不具有強烈結構性，共識因子便無法產生優秀的更新解(品質部份)。

4.4.3 更新解的品質

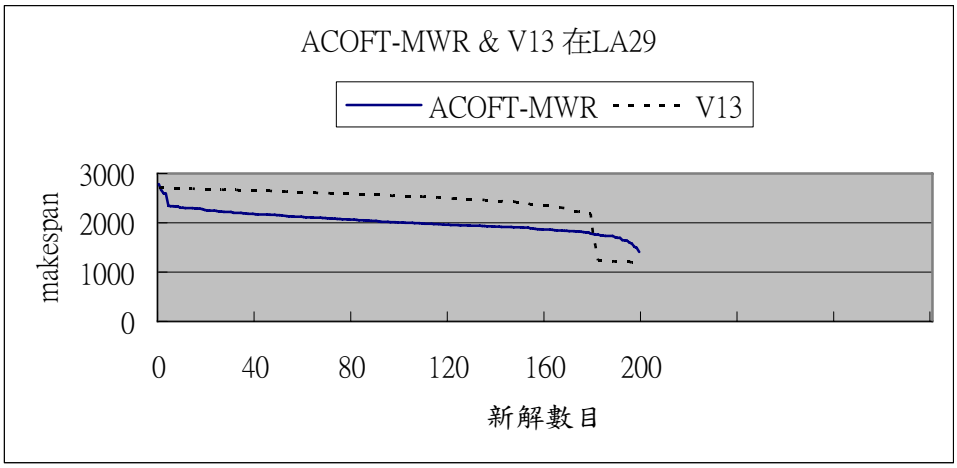

圖 4.4　V13 和 ACOFT-MWR 的方法比較：更新解部分

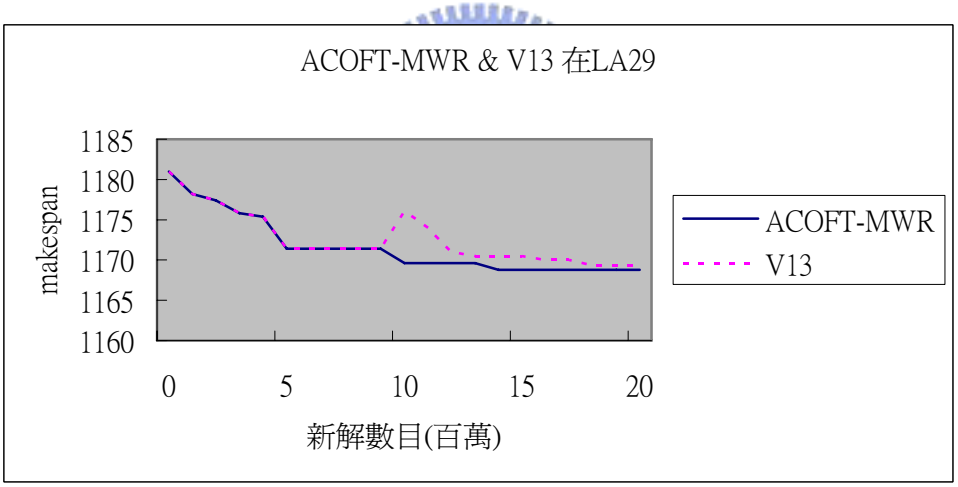

圖 4.4　V13 和 ACOFT-MWR 的方法比較：修正解部分

關於 JSSP 問題和演算法的特性關係，本論文發現「光是提升更新解的品質，不一定能找到最佳的完工時間」。

如圖 4.4 所示，本論文最佳配方演算法 V13 和 ACOFT-MWR 的方法比較，可以發現，在 V13 裡面，左半部分為共識因子產生的更新解，平均約為 2700，右半部分為田口方法產生的更新解，平均為 1200，但是 ACOFT-MWR 所產生的更新解大約在 1900 左右，即使我們利用田口方法產生出來的更新解品質「大大」

的優於 ACOFT-MWR 中的螞蟻演算法的更新解品質，但是經過禁忌演算法之後，三種不同的更新解幾乎就沒有差異性(圖 4.5)。

整體來說在本論文的實驗結果裡三種不同的方法求解的品質大約如下所示：

更新解品質：

$$Taguchi > ACO > Consensus$$

修正解品質：

$$ACO > Taguchi \cong Consensus$$

即使產生了優良的更新解，但是如果此更新解不具有發展性質的話，即使經過強大的修正器禁忌演算法來修正，效益也是有限的，所以與其追求更新解的求解品質，不如轉而追求利用更新解求得良好的解空間，但是如何定義一個演算法來用以尋找良好的解空間是十分困難的。

4.4.4 田口方法的影響

在本論文的三種實驗我們分別找出了三個最佳配方演算法，分別是實驗一的 V13、實驗二的 V18 和實驗三的 V41，但是發現一件很奇怪的事情，那就是為何三種實驗的結果不一致？在實驗一裡面，使用共識因子加上田口方法的實驗結果都優於單一使用共識因子，但是在實驗二和實驗三，表現品質卻都是只有單一共識因子的配方較好，是怎樣的原因造成這樣的結果？

在表 4.12 我們列出實驗的結果來說明此現象：

表 4.12 田口方法的影響

共識因子 / 田口方法	100萬	500萬	2000萬	代號
100 % / 0%	1178.6 1177.2 1175.8	1167.6 1168.2 1169.6	1167.2 1168.0 1169.2	V41 V42 V43
0 % / 100%	1178.6	1175.2	1175.2	V44
70% / 30%	1175.8 1181.8 1177.2	1175.8 1170.6 1171.2	1167.8 1168.6 1170.6	V45 V46 V47

根據我們的推論我們發現「田口方法」產生的解品質雖然優於共識因子，但是由於田口方法所產生的解和共識因子的解在「結構性」並不相似，也就是我們在 4.4.2 所提到的母體結構性問題，因此在混合兩種演算法於世代更新時，田口方法會打亂共識因子所產生的解結構，也就造成了實驗結果並不相符合的現象。

4.4.5 效益的比較

三種不同演算法的效益性，以表 4.13 為例，在 LA29 問題上面，會發現光是一組共識因子解經過禁忌演算法，就會產生約 80000 萬組新解，大約是 1：80000 的新解比例，即使是田口方法產生的解，比例也在 1：10000 左右，這也就是為何本論文演算法的停止條件都是用「百萬」為單位來計算，原因就在於禁忌演算法的演算過程之中，會產生大量的新解。但是如果不使用禁忌演算法，解的品質卻是無比的差，圖 4.4 表現的數據就是尚未使用禁忌演算法之前的數據，也就是說，即使更新解的差異性頗大，但經過禁忌演算法的運算所得結果並無太大差異。

整體而言，更新解越差的情況之下，禁忌演算法的計算時間會增加，進而產生過多的新解，如共識因子和田口方法為 80000：10000 為例，在停止條件為「產生固定新解」的情況之下，這樣的浪費產生解是相當不好的，所以重點還是要放在如何產生一組良好的更新解，此良好的更新解除了要具有良好的發展性以外，更新解的品質也要越優秀越好。

表 4.13　效益的比較(每產生一組解會產生的新解數)

	LA24	LA29	LA40
ACO	1	1	1
Consensus	1	1	1
Taguchi	12+1	12+1	15+1
Taboo (ACO)	80000	50000	150000
Taboo (Consensus)	80000	50000	150000
Taboo (Taguchi)	10000 以下	10000 以下	10000 以下

第五章　結論

本論文演算法，採用瓶頸飄移法(Shifting Bottleneck Procedure；SB)的分解機制來作為「共識因子」和「田口方法」的求解基礎，並加上禁忌演算法(Taboo search；TS)來做修正解的動作。本論文應用了兩種演算法，分別是「共識因子」(consensus operator) 主為利用解的共識，和前代群聚智慧的力量來找到新一代的解，以及「田口方法」(Taguchi methods)經過有效率的實驗來產生新解。

雖然最後本論文演算法的實驗結果不如 ACOFT-MWR 演算法，但本論文仍然超過了許多過去的著名演算法，畢竟過去並無人應用「共識因子」和「田口方法」於 JSSP 問題。

本研究方法的貢獻如下：

1. 成功的把共識因子使用在求解 JSSP 問題上面。
2. 成功的把田口方法使用在求解 JSSP 問題上面。
3. 證明了在更新解部分，不能只追求品質，具有整體的結構性才重要。
4. 使用了三種不同的實驗方法，經過多次的實驗，來找出最佳的配方組合。

本研究方法尚有許多不足之處，以下本文提出幾點後續研究建議：

1. 關於母體的研究仍不夠詳細，畢竟本論文只研究了三種方法。
2. 雖然找出了最佳配方，但是之下的參數並無仔細的研究。
3. 禁忌演算法太浪費產生新解個數，是否能用其他種演算法取代。

參考文獻

1. 蘇朝墩，2002，品質工程，中華民國品質學會
2. 吳政翰，2006，”DHC 系統之任務指派與排程的新型基因演算法“，國立交通大學工業工程研究所，碩士論文
3. Adams, J., Balas, E., and Zawack, D., (1988) “The shifting bottleneck procedure for job shop scheduling,” *Management Science*; 34:391–401.
4. Balas, E. and Vazacopoulos, (1998) “A Guided local search with shifting bottleneck for job shop scheduling,” *Management Science*; 44:262–75.
5. Blazewicz, J., Domschke, W., and Pesch E., (1996) “The job shop scheduling problem,” *European Journal of Operational Research*; 93:1–33.
6. Brucker, P., Jurisch, B., and Sievers, B., (1992) “Job-shop (C codes)”, *European Journal of Operational Research*; 57, 132-133.
7. Carlier, J. and Pinson, E., (1989) “An algorithm for solving the job-shop problem.” *Management Science*; 35: 164–76.
8. Croce, F.D., Tadei, R, and Volta, G., (1995) “A genetic algorithm for the job shop problem.” *Computers and Operations Research*; 22: 15–24.
9. Dauzere-Peres, S. and Lasserre, J., (1993) “A modified shifting bottleneck procedure for job shop problem,” *International Journal of Production Research*, 31, 923-932.
10. Dorigo, M. and Stützle, T. “Ant colony optimization,” (2004) *Cambridge, MA: MIT Press*.
11. Dorndorf, U. and Pesch, E., (1995) “Evolution based learning in a job shop scheduling environment,” *Computers and Operations Research*; 22:25–40.
12. Garey, M.R. and Johnson, D.S. (1979) “Computers and Intractability: a guide to the theory of NP-completeness,” *San Francisco, CA: Freeman and Company*.

13. Glover, F., (1986) “Futher path for integer programming and links to artificial intelligence.” *Computers and Operations Research*; 13: 533–49.
14. Graham, R.L., (1979) “Optimization and Approximation in Deterministic Sequencing and Scheduling: a survey,” Ann. Discrete Math, 5, pp.287-326
15. Huang, K.L. and Liao, C.J., (2006) “Ant colony optimization combined with taboo search for the job shop scheduling problem,” *Computers & Operations Research* 2006, doi: 10.1016/j.cor.07.003.
16. Jain, A.S. and Meeran, S., (1999) “Deterministic job-shop scheduling: past, present and future,” *European Journal of Operational Research*; 113:390–434.
17. Marine, A.S., (1960) “On the job shop scheduling problem,” *Operations Research*, 8, 219-223.
18. Muth, J.F., and Thompson, G.L., (1963), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NJ.
19. Nowicki, E. and Smutnicki, C., (1996) “A fast tabu search algorithm for the job shop problem,” *Management* Science; 42:797–813.
20. Phadke, M. S., (1989) “Quality Engineering Using Robust Design,” *Prentice-Hall*.
21. Pinedo, M., (1995) “Scheduling Theory, Algorithm, and System,” *Prentice Hall*, NEW Jersey.
22. Pezzella, F. and Merelli, E., (2000) “A tabu search method guided by shifting bottleneck for the job shop scheduling problem,” *European Journal of Operational Research*; 120: 297–310.
23. Roy, B. and Sussmann, B., (1964) “Les problems d’ordonnancement avec contraintes disjunctives,” Notes DS N.9 Bis, SEMA, Montrouge.
24. Schultz, S.R., Hodgson, T.J., and King R.E., (2004) “On solving the classic job shop makespan problem by minimizing Lmax,” Raleigh, NC: Department of

Industrial Engineering, North Carolina State University.

25. Taugchi, G., (1986) “Introducion to Quality Engineering: Designing Quality into Products and Processes,” Tokyo: Asian Production Organization.
26. Taguchi, G., (1987) “System of Experimetal design,” vols. 1 and 2. While Plains, New York: UNIPUB/Krauss International.
27. Van Laarhoven PJN, (1992) “Aarts EHL, Lenstra JK. Job shop scheduling by simulated annealing. ” *Operations Research*; 40:113–25.