國立交通大學

資訊科學與工程研究所

碩士論文

運用蒙地卡羅樹狀搜尋於多目標彈性零工式工廠排程問題

Multi-Objective Flexible Job Shop Scheduling Problem Based on Monte-Carlo Tree Search

研 究 生：吳東穎

指導教授：吳毅成教授、李素瑛教授

中 華 民 國 102 年 8 月

運用蒙地卡羅樹狀搜尋於多目標彈性零工式工廠排程問題

Multi-Objective Flexible Job Shop Scheduling Problem

Based on Monte-Carlo Tree Search

研 究 生：吳東穎　　Student： Tung-Ying Wu

指導教授：吳毅成　　Advisor： I-Chen Wu

李素瑛　　Suh-Yin Lee

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

In

Computer Science

August 2013

Hsinchu, Taiwan, Republic of China

中華民國 102 年 8 月

運用蒙地卡羅樹狀搜尋於多目標彈性零工式工廠排程問題

研究生：吳東穎　　指導教授： 吳毅成

李素瑛

國立交通大學 資訊科學與工程研究所

摘要

在生產管理和組合最佳化的領域中，彈性零工式工廠排程(Flexible Job Shop Scheduling Problem, FJSP)是一個非常重要的問題，本論文最佳化 FJSP 是最小化以下三項目標：總時程、總工作量與最大工作量，並採用非凌越解集合(Pareto Solutions)的方式進行最佳化。

近年來，蒙地卡羅樹狀搜尋(Monte-Carlo Tree Search, MCTS) 非常成功地運用於電腦圍棋及其他許多遊戲，本論文將運用 MCTS 來解 FJSP 問題。我們的做法是結合 MCTS 與變鄰域下降演算法(Variable Neighborhood Descent Algorithm)，並且加入一些變異的方法如 Rapid Action Value Estimates Heuristic、換位表(Transposition Table)應用於 FJSP。

我們的 MCTS 方法能夠在 116 秒找出 Kacem 等人提出之基準問題(benchmark)的 17 組解：4x5 共四組、10x7 共三組、8x8 共四組、10x10 共四組、15x10 共兩組，除了 8x8 有一組沒有找到外，其他皆是目前找到最好的，此結果是目前最好的，與蔣宗哲教授等人提出的簡易演化式演算法(Simple Evolutionary Algorithm)與模因演算法(Memetic Algorithm)相同，雖然 Xiaojuan Wang 等人提出的演算法有找到額外的一組 8x8 的解，但他們並沒有找到一些上述提到的解。

本研究是第一個用 MCTS 解出 Kacem 等人提出之基準問題中 17 組目前的最佳解，此結果不亞於其他演算法如基因演算法，在作業數量較多的 Mk 基準問題中所找到的解也與目前的最佳解相近。

Multi-Objective Flexible Job Shop Scheduling Problem Based on Monte-Carlo Tree Search

Student: Tung-Ying Wu Advisor: I-Chen, Wu

Suh-Yin, Lee

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

Flexible job-shop scheduling problem (FJSP) is very important in both fields of production management and combinatorial optimization. This thesis addresses the multi-objective flexible job shop scheduling problem (MO-FJSP) with three objectives which minimizing makespan, maximal workload and total workload respectively. We consider these objectives with Pareto manner.

Monte-Carlo Tree Search (MCTS) is successful in computer Go and many other games. In this paper, we propose an MCTS algorithm for FJSP. Our algorithm also incorporates Variable Neighborhood Descent Algorithm and other variations like Rapid Action Value Estimates Heuristic and Transposition Table are applied to improve this algorithm.

Our algorithm finds Pareto solutions of benchmark problems by Kacem et al.: 4 solutions in 4x5, 3 in 10x7, 4 in 8x8, 4 in 10x10 and 2 in 15x10 within 116 seconds. These solutions are the same as the best found to date. Although one article claimed to find an extra 8x8 solution, that article did not find some of the above solutions. Of the previously attempts to solve the FJSP problem using MCTS, our method yields the best results to date. In Mk benchmark problems, Pareto solutions found by our algorithm are close to the best solutions.

誌謝

此篇論文得以完成，首先要感謝吳毅成教授兩年來的指導，在百忙之中仍然安排足夠的時間討論，提供我更正確的思考方向與更全面性的建議，讓我獲益良多。感謝口試委員們點出本論文需再改善的地方，使本論文能更加完整。

感謝高國元教授與周政緯學長提出將蒙地卡羅搜尋樹運用在最佳化問題上的想法。也感謝蔣宗哲教授在多目標彈性零工式工廠排程問題的領域中為我解惑。

感謝實驗室的學長姊、同學與學弟妹們，讓實驗室充滿了正面的力量，在研究遇到瓶頸時能夠保持愉快的心情面對。感謝 TF，讓我寫的程式架構更加完整，解掉許多不容易處理的錯誤；感謝 Ting，幫我翻譯困難的敘述與修改投影片；感謝包子和阿水，接手其他的專案讓我能專心於此研究中。感謝阿賢與庭築幫忙口試的場佈，過程雖然顛頗但還是準備完善。感謝實驗室的所有夥伴們在這兩年來一起經歷的時光。

感謝我的父母，從小培養我到長大陪伴我，提醒我在研究之餘也要照顧好身體，不要常常熬夜爆肝或飲食不正常，也告訴我學習態度的重要性，讓我有動力努力完成自己的研究。

最後此論文獻給支持我的老師、家人、學長姊同學學弟妹，與所有在研究過程中支持我關心我的朋友們。

民國一百零二年八月 於 新竹交通大學工程三館 EC511 實驗室

目錄

圖表目錄

第一章、介紹

零工式工廠排程(Job Shop Scheduling Problem, JSP)是電腦科學及作業研究領域中的一個重要的最佳化問題，其目標是：在特定的時間分配資源或機器(machine)給一個最適當的作業(Operation)。在 JSP 中，每個作業只能在某台特定的機器上執行，當有一台機器無法正確運作時，就會導致整個工作(Job)停擺，若每個作業能夠讓多個機器運行則可降低此風險，彈性零工式工廠排程(Flexible Job-Shop Scheduling Problem, FJSP)的「彈性」(flexible)意指作業不被限制在某特定機器上運行[4]。

自 2006 年蒙地卡羅樹狀搜尋(Monte-Carlo Tree Search，簡稱 MCTS)被成功地應用在圍棋之後，應用在許多其他遊戲也有不錯的成果。本論文嘗試將 MCTS 應用到其他領域如數學最佳化(Mathematical Optimization)的問題上，以了解 MCTS 的特性以及應用在其他領域上的效果。

演化式演算法被成功地應用在最佳化問題中，本論文將 MCTS 應用在 FJSP，以比較演化式演算法與 MCTS 的特性與其優缺點。

1.1 多目標彈性零工式工廠排程問題

FJSP 定義如下[4]：

1. 設一個 FJSP 問題包含 n 個工作與 m 台機器(machine)，表示為$n \times m$。
2. 每個工作(Job) j 有n_j個作業(Operation)，$\{o_{j,1}, o_{j,2}, \dots, o_{j,n_j}\}$，其中$o_{j,i+1}$在$o_{j,i}$完成後才可開始執行。
3. 作業$o_{j,i}$須被一台機器$m_{j,i}$執行。
4. p_{jik}為作業$o_{j,i}$在機器 k 上的運行時間。

5. 作業$o_{j,i}$的完工時間定義為$C_{j,i}$。

6. O_k為機器 k 所需執行的作業集合。

7. 在同一工作中的作業有優先權的順序。

8. 工作間沒有相依性(dependency)。

9. 一台機器同時只能執行一個作業。

10. 作業不可被中斷。

		M1	M2	M3	M4	M5	M6	M7	M8
J1	$o_{1,1}$	5	3	5	3	3	X	10	9
	$o_{1,2}$	10	X	5	8	3	9	9	6
	$o_{1,3}$	X	10	X	5	6	2	4	5
J2	$o_{2,1}$	5	7	3	9	8	X	9	X
	$o_{2,2}$	X	8	5	2	6	7	10	9
	$o_{2,3}$	X	10	X	5	6	4	1	7
	$o_{2,4}$	10	8	9	6	4	7	X	X
J3	$o_{3,1}$	10	X	X	7	6	5	2	4
	$o_{3,2}$	X	10	6	4	8	9	10	X
	$o_{3,3}$	1	4	5	6	X	10	X	7
J4	$o_{4,1}$	3	1	6	5	9	7	8	4
	$o_{4,2}$	12	11	7	8	10	5	6	9
	$o_{4,3}$	4	6	2	10	3	9	5	7
J5	$o_{5,1}$	3	6	7	8	9	X	10	X
	$o_{5,2}$	10	X	7	4	9	8	6	X
	$o_{5,3}$	X	9	8	7	4	2	7	X
	$o_{5,4}$	11	9	X	6	7	5	3	6
J6	$o_{6,1}$	6	7	1	4	6	9	X	10
	$o_{6,2}$	11	X	9	9	9	7	6	4
	$o_{6,3}$	10	5	9	10	11	X	10	X
J7	$o_{7,1}$	5	4	2	6	7	X	10	X
	$o_{7,2}$	X	9	X	9	11	9	10	5
	$o_{7,3}$	X	8	9	3	8	6	X	10
J8	$o_{8,1}$	2	8	5	9	X	4	X	10
	$o_{8,2}$	7	4	7	8	9	X	10	X
	$o_{8,3}$	9	9	X	8	5	6	7	1
	$o_{8,4}$	9	X	3	7	1	5	8	X

圖 1. 一個 8×8 的 FJSP 問題(Kacem 8×8) [14]

圖 1 為一個 8×8 的 FJSP 問題，其中工作 J1 包含三個作業：$o_{1,1}$、$o_{1,2}$、$o_{1,3}$，執行$o_{1,1}$的時間為{5, 3, 5, 3, 3, X, 10, 9}，M_1執行$o_{1,1}$的時間為 5，M_6執行$o_{1,1}$的時間為 X(表示無法運行此作業)。當每個工作的所有作業都已經被執行，

稱為一個完整的調度(complete schedule)。

在 JSP 問題中，有許多已被定義的目標標準(objective criteria)，三個常見的目標標準是總時程 (makespan)、總工作量(Total Workload)及最大工作量(Maximum Workload)，其計算方法如下[4]：

$$\text{makespan: } C_M = \max_{j=1\ldots n} C_{j,n_j} \tag{1}$$

$$\text{total workload: } W_T = \sum_{k=1\ldots m} \sum_{o_{j,i} \in O_k} p_{jik} \tag{2}$$

$$\text{maximum workload: } W_M = \max_{k=1\ldots m} \sum_{o_{j,i} \in O_k} p_{jik} \tag{3}$$

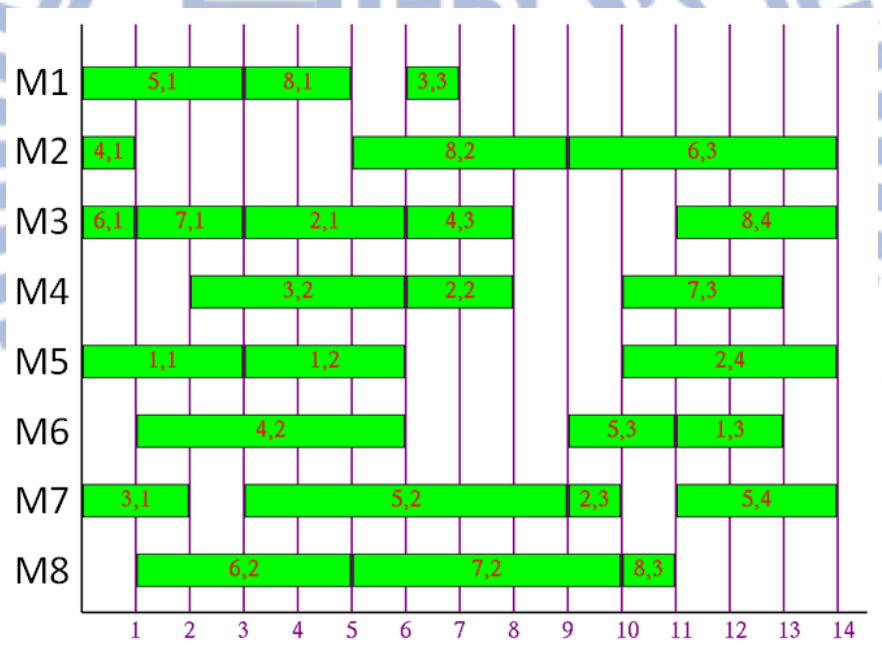


圖 2. FJSP 問題的一個完整調度

圖 2 為 FJSP 問題的一個完整調度，其C_M為 14，W_T為 77，W_M為 12。這三個目標標準皆是愈小愈好，並在最佳化時同時考慮這三樣目標標準，同時考慮多個目標的 FJSP 稱為多目標彈性零工式工廠排程(Multiobjective Flexible Job-Shop Scheduling, MO-FJSP)，有許多種同時考慮多個目標的方

式，以下介紹其中兩種：

1. 對多個目標標準進行線性組合(linear combination)：以C_M、W_T、W_M為例，線性組合按照目標標準的重要性來進行線性組合而得到一個分數：$v = \alpha C_M + \beta W_T + \gamma W_M$，舉例來說，若較重視$C_M$而較不重視$W_T$、$W_M$，則可以給予$\alpha = 0.8$、$\beta = 0.1$、$\gamma = 0.1$。擁有最大/最小的 v 值代表最佳結果。
2. 考慮非凌越解集合(Nondominated set of solutions，又稱 Pareto solutions)：Pareto optimal 的定義是：「對於所有的目標標準(objective criterion)，在不犧牲(將之變差)其他目標標準的情況下，無法再被改善」。如下圖，A、C、D 的所有目標標準都無法在不犧牲其他標準的狀況下被改善，而 B 的C_M可以在不犧牲W_T與W_M的狀況下被改善，因此 B 不是非凌越解集合中的元素。

Solution	C_M	W_T	W_M
A	15	75	12
B	17	76	12
C	16	73	13
D	14	77	12

圖 3. 非凌越解集合的例子

1.2 蒙地卡羅樹狀搜尋

蒙地卡羅方法也稱「統計模擬方法」，使用隨機取樣以統計逼近真實結果，蒙地卡羅評分的準確度可以經由樹狀搜尋來改善[7]，基於蒙地卡羅方法進行樹狀搜尋稱為蒙地卡羅樹狀搜尋(Monte-Carlo Tree Search, MCTS)。2006 年，Kocsis 與 Szepervari 將 Upper Confidence Bounds(UCB)的技術應用在蒙地卡羅搜尋樹，稱之為 Upper Confidence Bounds for Trees (UCT) [3]。

Rapid Action Value Estimates Heuristic (RAVE)是一種用以改善 MCTS 的技術。

MCTS 運用於電腦圍棋非常成功，2006 年，Crazy Stone 將 MCTS 用在電腦圍棋上[8]，獲得奧林匹亞電腦 9x9 圍棋競賽金牌，2007 年 MoGo 將 UCB 加入 MCTS 中[11]，獲得奧林匹亞電腦 19x19 圍棋全勝的紀錄。以下子節將詳細介紹這些方法。

1.2-1 蒙地卡羅樹狀搜尋架構

蒙地卡羅搜尋樹是一個最佳優先搜尋(best first search)樹，其中包含了四個步驟：選點(Selection)、展點(Expansion)、模擬(Playout)、更新(Backpropagation)，此四個步驟做一次稱做一個 Simulation，以下介紹這四個步驟：

1. 選點：從根節點(root)開始，利用之前模擬的資訊選出一個評估值最好的子節點，遞迴地選到葉節點(leaf)。
2. 展點：從選到的葉節點長出一個新節點並進行初始化。
3. 模擬：不斷地隨機選步，直到模擬結束。
4. 更新：往此節點的祖先更新模擬的結果。

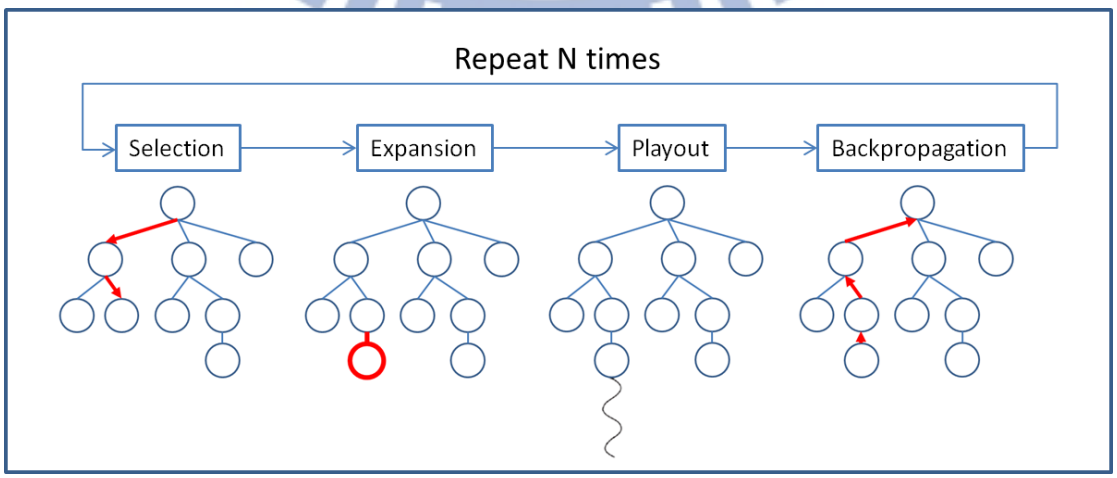


圖 4. 蒙地卡羅樹狀搜尋

1.2-2 Upper Confidence Bounds Applied to Trees

在 MCTS 中，勘探(Exploration)與開發(Exploitation)的平衡是非常重要的，Upper Confidence Bounds (UCB)的計算方式提供勘探與開發平衡的參考。

最佳優先搜尋(Best First Search)在選擇節點時，從某個樹節點 p 去找到它的最佳子節點，計算公式如下：

$$UCB_j = X_j + C * \sqrt{\frac{\log_{10} N}{N_j}} \tag{4}$$

計算所有子節點的 UCB 值以選擇最好的子節點，其參數如下：

- UCB_j：子節點 j 的 UCB 值。
- X_j：目前為止子節點 j 的平均勝率/分數。
- C：UCB 常數。
- N：目前為止 p 的模擬次數。
- N_j：目前為止子節點 j 的模擬次數。

X_j項為平均勝率/分數較好的點加分，即鼓勵勘探；N_j若越小，則第二項越大，為模擬次數少的點加分，即鼓勵開發。

用此公式來做樹狀搜尋的方法就稱作 Upper Confidence Bounds Applied to Trees(UCT)。

1.2-3 Rapid Action Value Estimates Heuristic

UCT 在選點時，用 UCB 來選擇最好的子節點，而 UCB 公式參考了節點的平均勝率，此勝率由模擬(playout)的結果更新，但模擬的結果還可以

提供更多的資訊。在模擬的過程中，有些決定(decision)不論順序對於全局的影響力是相近的，選點時可以不考慮順序地參考這些資訊，此概念稱為 All Move As First。

Rapid Action Value Estimation (RAVE)是利用前述的概念，除了將某節點的模擬結果更新至該節點的祖先之外，若祖先的兄弟的決定有出現在模擬結果中，也將模擬結果的分數更新到此節點。選點除了考慮 UCB 值，也參考 RAVE 的值。如圖 5(圖中的粗框節點被 MCTS 標準更新，黃色實心節點被 RAVE 更新)，若有一個模擬的結果為$D = \{D_2, D_3, D_1, D_5\}$，$D_3$的兄弟$D_1$的 RAVE 值會被更新。

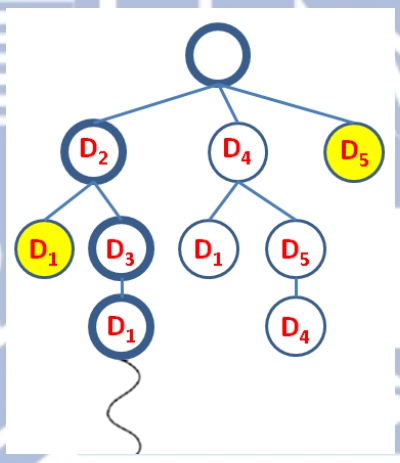


圖 5. RAVE 更新方式

在選點時參考 RAVE 資訊，可以給予 UCB 與 RAVE 不同的比重：

$$Score_j = (1-\beta) \times RAVE_j + \beta \times UCB_j = (1-\beta) \times (X_j + C * \sqrt{\frac{\log_{10} N}{N_j}}) + \beta \times X_{R_j} \quad (5)$$

其中由理論[10]推導得出最佳的$\beta = \frac{N_{R_j} X_{R_j}}{N_{R_j} + N_j + DNR_j N_j}$，其中參數$D$為一常數、$X_{R_j}$為 RAVE 更新的平均勝率、$N_{R_j}$為 RAVE 更新的次數。

1.3 本論文之目標與貢獻

我們嘗試將 MCTS 應用到 FJSP，結合 MCTS 以及變鄰域下降演算法(Variable Neighborhood Descent Algorithm, VND)以改善蒙地卡羅搜尋樹中模擬的品質。並加入一些被成功應用在 MCTS 的 Heuristic 以改善效能如：子樹修剪、RAVE、prior knowledge、換位表(Transposition Table)。

比較以上變異方式的效能，同時使用全部四種變異方式的效率是最好的，在 116 秒找出 Kacem 等人提出之基準問題[14]的解：4x5 共四組、10x7 共三組、8x8 共四組、10x10 共四組、15x10 共兩組，除了 8x8 有一組沒有找到外，其他皆是目前找到最好的。這是目前用 MCTS 解 FJSP 最好的結果，在 Kacem 等人提出之基準問題不亞於其他演算法。至於 Mk 問題中所找到的解也與目前找到的最佳解相近。

1.4 論文組織

本篇論文第二章介紹先前的研究成果，包括三種不同的演算法應用在 FJSP：簡易演化式演算法、變鄰域下降演算法及蒙地卡羅樹狀搜尋。第三章是我們提出的運用蒙地卡羅樹狀搜尋的演算法及四種變異方式。第四章是實驗，比較各種變異方式的效能、本論文的演算法與其他演算法的效能。第五章是結論以及未來展望。

第二章、先前的研究

在這個章節中將介紹過去關於 MO-FJSP 的研究內容，在 2.1 節中介紹簡易演化式演算法(Simple Evolutionary Algorithm, SEA)應用在 MO-FJSP、2.2 節中介紹變鄰域下降演算法(Variable Neighborhood Descent Algorithm, VND)應用在 MO-FJSP、2.3 節介紹蒙地卡羅樹狀搜尋(Monte-Carlo Tree Search, MCTS)應用在 MO-FJSP。

2.1 簡易演化式演算法應用在多目標彈性零工式工廠排程

SEA [4]應用在 FJSP 包含以下步驟：

1. 初始化群體(Initialization)：產生N_{POP}個初始群體，將這些群體做解碼並計算它們的目標分數，設定代數 $t = 1$。
2. 生殖(Reproduction)：利用單淘汰選出N_{POP}交配母群體，在交配母群體中，每兩個個體透過交配產生子個體，共產生N_{POP}個子個體，在當代群體與子群體中選擇最好的N_{POP}個子個體存活到下一代。
3. 平衡地勘探與開發(Balanced Exploration and Exploitation)：透過有效率的突變(mutation)將重複的個體精緻化。
4. 終止條件(Termination)：若$t > T_{GEN}$則終止，否則將t加一，重複步驟 2~4。

2.1-1 染色體編碼與解碼

使用三元組格式(3-tuple scheme)來進行染色體的編碼[1][16][18]，此編碼方式包含 Routing 與 Sequencing 的資訊[4]，routing 資訊決定作業要放在

哪台機器上執行，sequencing 資訊決定同一台機器上作業的執行順序。每個染色體都是由三元組(j,i,k)基因排列而成，其中 j、i 代表工作 j 的第 i 項作業，k 代表$o_{j,i}$由機器 k 執行，越靠左端點的基因擁有較高的優先序。如下圖，$o_{5,1}$、$o_{8,1}$會在機器 1 上按照順序執行。

(5,1,1)	(8,1,1)	[4,1,2]	[3,1,7]	[5,2,7]	...	...

圖 6. 染色體編碼的例子

解碼的部分，使用 GT 演算法進行解碼[13]，解碼步驟如下：

1. 取得擁有最早「可能完成時間」(possible completion time)的作業，其時間為 ρ，目標機器為 m^*。
2. 蒐集所有需要在 m^*上運行並且可開始時間(startable time)不大於 ρ 的作業，形成一個集合O_{start}。
3. O_{start}內的作業在排程時擁有最高的優先權。
4. 對被影響的作業更新 ρ。
5. 若所有作業皆被分派則結束，否則重複 1~5。

在以上步驟中，染色體的基因會按照 GT 演算法處理的作業重新排序。

2.1-2 初始化群體

在初始化群體的過程中，引進領域知識來優化初始群體，分為 Routing 方法與 Sequencing 方法。

Routing 方法：

1. 全域最小工作量(global minimal workload)
2. 隨機排列(random permutation)
3. 區域最小工作量(local minimal workload)

4. 最小處理時間(minimal processing time)
5. 隨機分派(random assignment)

Sequencing 方法：

1. 最多剩餘工作(most work remaining)
2. 最多剩餘作業量(most number of operations remaining)
3. 隨機調度(random dispatching)

初始化群體演算法步驟如下：

1. 以上兩種方法的組合共有 15 種，扣除隨機分派與隨機調度的組合，餘 14 種組合形成一個集合 *RS*，並初始化一個目標分數向量(objective vector)*OV* 來儲存。
2. 從 *RS* 中隨機選擇一個方法 *c* 來產生染色體，若 *RS* 為空，使用隨機分派與隨機調度。
3. 對染色體進行解碼，若目標分數存在於 *OV* 則把 *c* 從 *RS* 中移除，若不存在則將染色體加入群體並把目標分數加入 *OV*。
4. 重複步驟 2 和 3 直到群體數目到達N_{POP}。

2.1-3 個體評估與選擇

每個個體由 NSGA-II 演算法[9]進行評估以得到適應分數，此分數供 SEA 的選擇步驟做參考。演化式演算法包含兩個選擇步驟：交配選擇(mating selection)與環境選擇(environmental selection)，在交配選擇之後，兩個母個體產生兩個子個體。交配選擇與環境選擇方法如下：

- 交配選擇：在群體中選擇兩個個體進行比較，留下較好的個體，重複此步驟直到母群體有N_{POP}個個體。
- 環境選擇：在當代群體與子帶群體共$2 \times N_{POP}$個個體中選擇N_{POP}個存

活至下一代。

2.1-4 交配

當兩個母個體進行交配時，會隨機從 ASX 與 POX [15]方法中隨機選一個。ASX 與 POX 方法如下：

- ASX (assignment crossover)：選擇隨機數量的作業，將其運行的機器做交換。此交配演算法只會改變 routing 的資訊，圖 7 為一個 ASX 的例子，隨機選擇四個作業($o_{1,1}$、$o_{2,3}$、$o_{1,2}$、$o_{1,3}$)後，將其運行的機器做交換。
- POX (Preserving Order-based Crossover)：隨機鎖定一個工作的所有作業後，重新按照作業在另一母個體出現的順序排序基因。此交配演算法改變了 sequencing 資訊，圖 8 為一個 ASX 的例子，鎖定工作 2 的所有作業，其餘的作業按照出現在另一母代的順序重新排序。

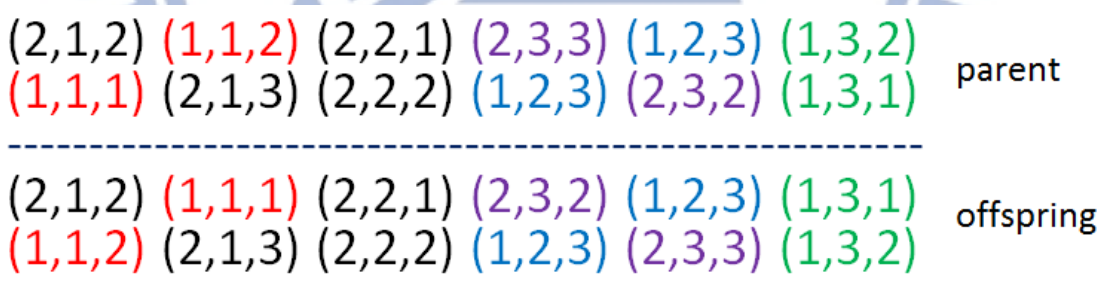


圖 7. 一個 ASX 的例子

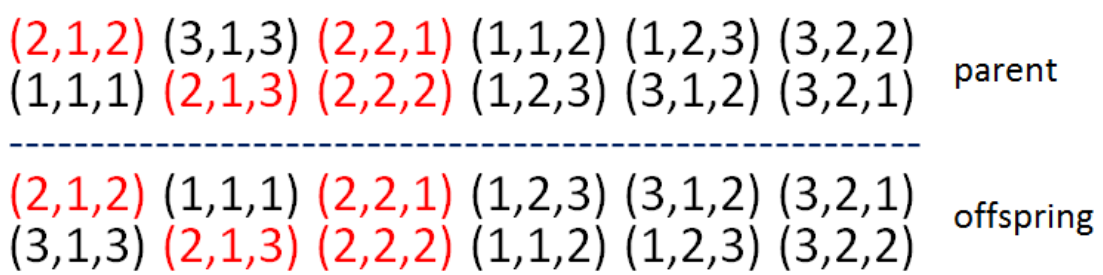


圖 8. 一個 POX 的例子

2.1-5 平衡地勘探與開發

演化式演算法的群體多樣性可以避免過早的收斂[4]，群體中若有兩個個體擁有相同的目標分數，即被定義為「重複的個體」。為了讓群體更有多樣性同時又希望充分利用個體的資訊，作者引進了五個方法來對重複的個體做修正：

1. 以降低最大工作量為目標重新分派機器(machine re-assignment for reduction of maximum workload)
2. 以降低總時程為目標重新分派機器(machine re-assignment for reduction of makespan)
3. 隨機重新分派機器(random machine re-assignment)
4. 交換關鍵作業(critical operation swap)
5. 重新插入關鍵作業(critical operation re-insertion)

2.2 變鄰域下降演算法應用在多目標彈性零工式工廠排程

簡易基因演算法的演化速度相對地慢[17]，區域搜索(local search)是一個可能可以改善其收斂速度的方法[12]，在[12]中，作者使用變鄰域下降演算法（Variable Neighborhood Descent Algorithm, VND）來最佳化基因演算法中的每個個體，其中的 neighborhood function 包含了「移動一個作業的區域搜索」與「移動兩個作業的區域搜索」。我們提出的演算法中運用「移動一個作業的區域搜索」(以下簡稱為 LSONE)來對排程進行最佳化。在下面的章節中會介紹 LSONE 如何被應用在 MO-FJSP 中。

2.2-1 作業相依性

FJSP 問題的其中一項定義：同一工作的所有作業有順序性、每台機器同時只能執行一個作業，形成了作業相依性。每個作業在執行前都要確保「同一工作的前一個作業」與「同一機器的前一個作業」已經完成，在此定義以下章節會用到的符號：

1. 工作前任作業(job predecessor)：作業 r 的工作前任作業為$PJ(r)$。
2. 工作後任作業(job successor)：作業 r 的工作後任作業為$SJ(r)$。
3. 機器前任作業(machine predecessor)：作業 r 的機器前任作業為$PM(r)$。
4. 機器後任作業(machine successor)：作業 r 的機器後任作業為$SM(r)$。

2.2-2 最早事件時間與最晚事件時間

事件時間包含作業的開始時間與結束時間，以下為四種事件時間的定義與符號：

1. 最早開始時間：一個作業可以被運行的時間點。$s^E(r)$代表作業 r 的最早開始時間。
2. 最晚開始時間：一個作業最晚運行但不會拖延到總時程(makespan)的時間點。$s^L(r)$代表作業 r 的最晚開始時間。
3. 最早完成時間：$c^E(r) = s^E(r) + t(r)$，其中 $t(r)$代表作業 r 的運行時間。
4. 最晚完成時間：$c^L(r) = s^L(r) + t(r)$。

由作業相依性定義$s^E(r)$、$c^L(r)$：

$$s^E(r) = max\{c^E[PJ(r)], c^E[PM(r)]\} \quad (6)$$

$$c^L(r) = min\{s^L[SJ(r)], s^L[SM(r)]\} \quad (7)$$

若一個作業的最早開始時間等於最晚開始時間，則該作業稱為「關鍵作業」(critical operation)，由關鍵作業組成且長度為總時程的路徑稱為「關鍵路徑」(critical path)。

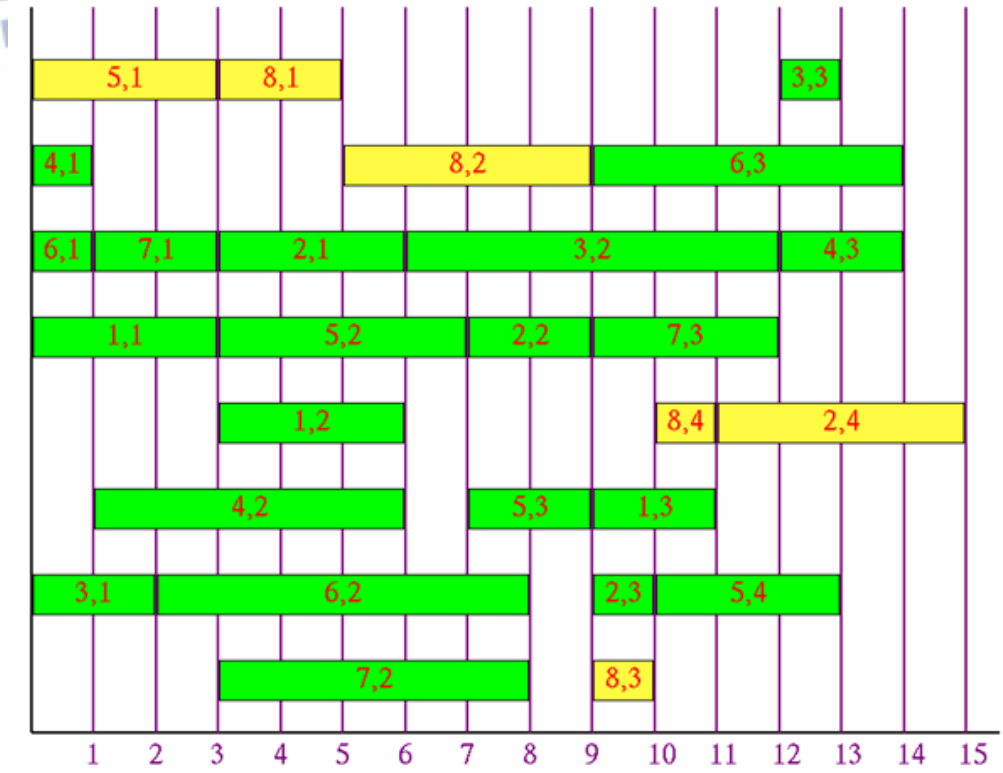


圖 9. 排程的關鍵路徑(標示為黃色)

2.2-3 移動作業

移動作業的動作如下：

1. 把要移動的作業 r 從原始排程中刪除，定義原始排程為G，刪除該作業後為G^-。
2. 計算G^-中所有作業的最早與最晚事件時間，如下的圖 10 與圖 11。
3. 按照步驟 2 的結果，找到一個可分派區間(assignable interval)將 r 插入。
4. 更新排程為G'。

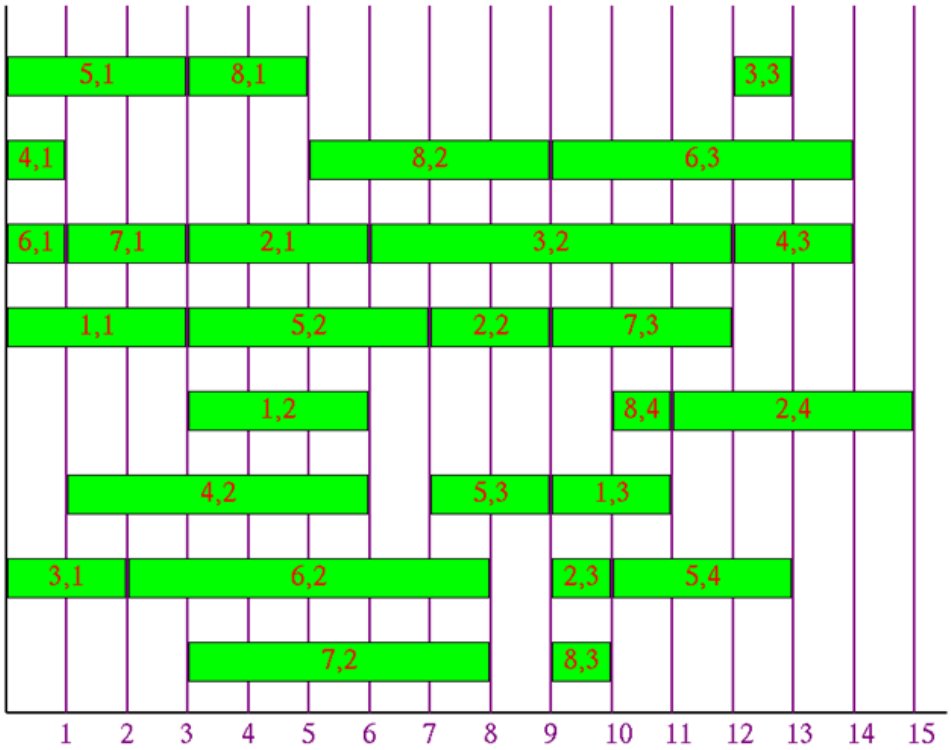


圖 10. 所有作業的最早開始/完成時間

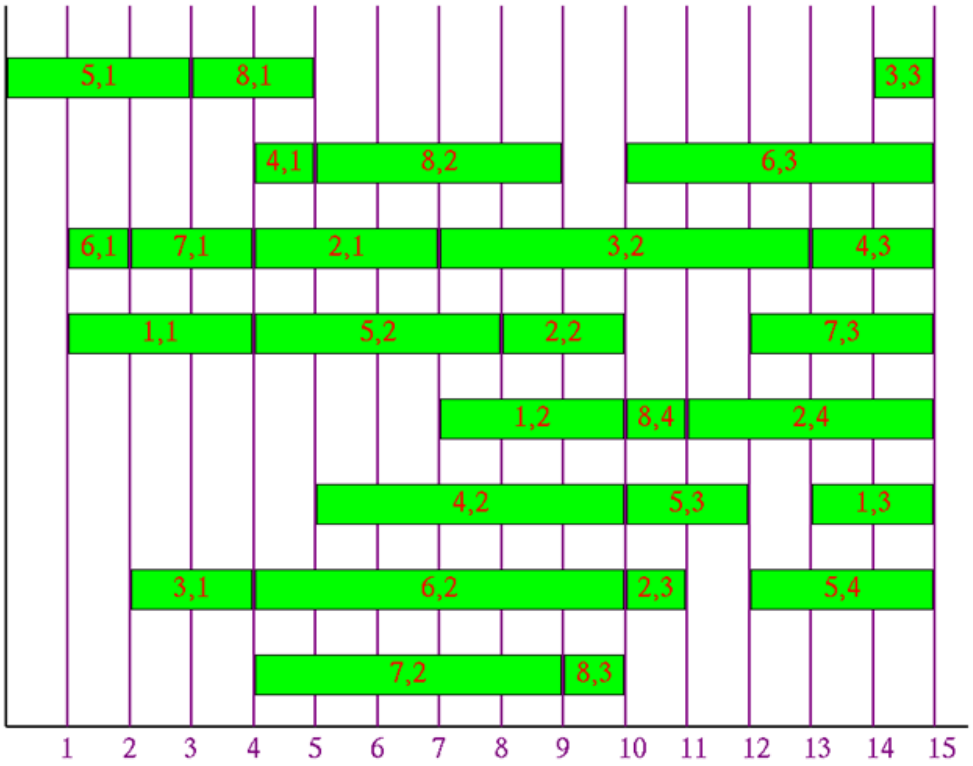


圖 11. 所有作業的最晚開始/完成時間

步驟 2 中，參考G的總時程C_M來計算最晚事件時間，計算方式如下：

- 若一個作業 v 沒有$PJ(v)$，則將$c^{E-}[PJ(v)]$設為 0。
- 若一個作業 v 沒有$SJ(v)$，則將$s^{E-}[SJ(v)]$設為C_M。
- 若一個作業 v 沒有$PM(v)$，則將$c^{E-}[PM(v)]$設為 0。
- 若一個作業 v 沒有$SM(v)$，則將$s^{E-}[SM(v)]$設為C_M。
- 其他狀況將參考 2.2-2 計算。

圖 10 中，$c^{E-}[PM([5,1])]$及$c^{E-}[PJ([5,1])]$皆為 0，因為$o_{5,1}$並沒有工作前任作業與機器前任作業；$s^{E-}[SM([8,4])]$為 11 且$s^{E-}[SJ([8,4])]$為 15，其中 15 為 G 的總時程。

在步驟 3 中，需要計算可分派區間，令一個區間 I 在作業 v 之前，在作業 w 之後。若滿足以下條件，則 I 對待插入的作業 r 是可分派的：

- $max\{c^{E-}[PM(v)], c^{E-}[PJ(v)]\} + p_{rM(v)} < min\{s^{L-}(v), s^{L-}[SJ(r)]\}$
- $c^{E-}(v) \geq c^{E-}[PJ(r)]$
- $s^{L-}(w) \leq s^{L-}[SJ(r)]$

2.2-4 移動一個作業的區域搜索

整個 LSONE 程序不斷地移動關鍵路徑上的作業，步驟如下：

1. 在排程G中找到一條關鍵路徑P，令 r 為P的第一個作業。
2. 重複以下子步驟
   a. 從G中移除 r 得到G^-。
   b. 在G^-中找尋一個可分派 r 的區間 I。
   c. 若有找到可分派區間，進入步驟 3；否則設 r 為P的下一個作業，回到步驟 a，若P沒有其他作業則結束。
3. 將 r 放入G^-的區間 I 中。

以圖 9 為例，找到關鍵路徑P後，將其第一個作業$o_{5,1}$設為 r，把 r 從排程G中刪除，但找不到任何可分派的區間。將 r 設為$o_{8,1}$，把 r 從排程G中刪除，計算其最早/最晚事件時間如下頁圖 12、圖 13，找到可分派的區間 I(在$o_{4,2}$之前)後，將 r 插入如圖 14。

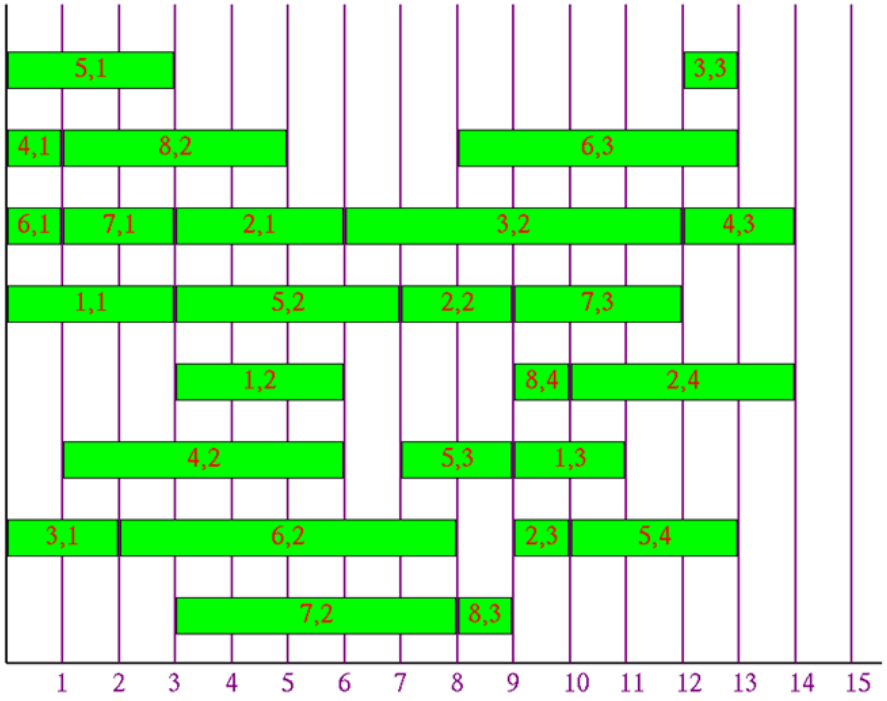


圖 12. 刪除作業[8, 1]後的最早開始/完成時間

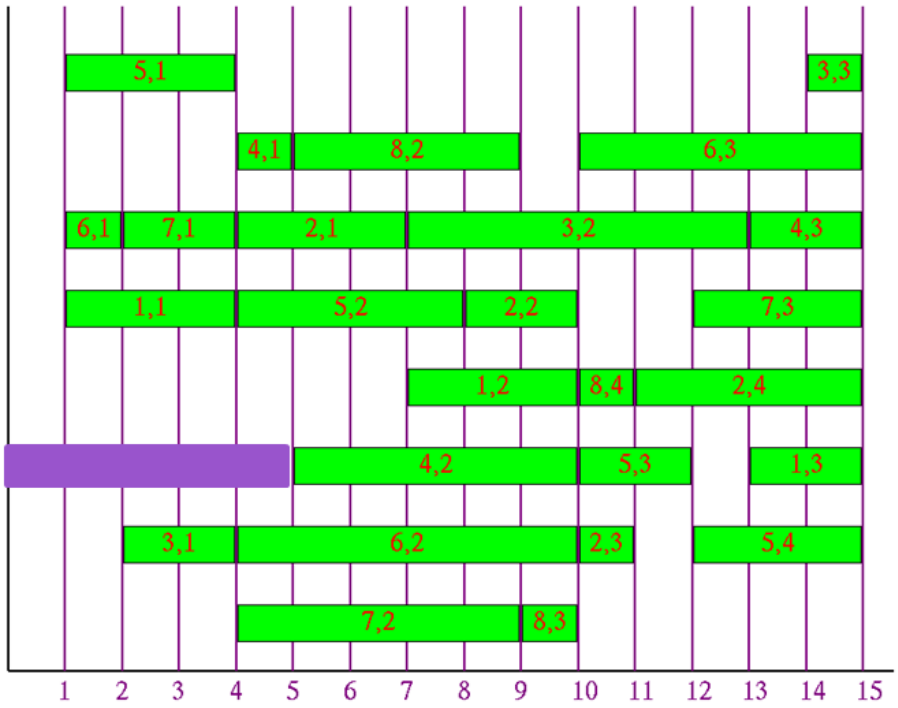


圖 13. 刪除作業[8, 1]後的最晚開始/完成時間

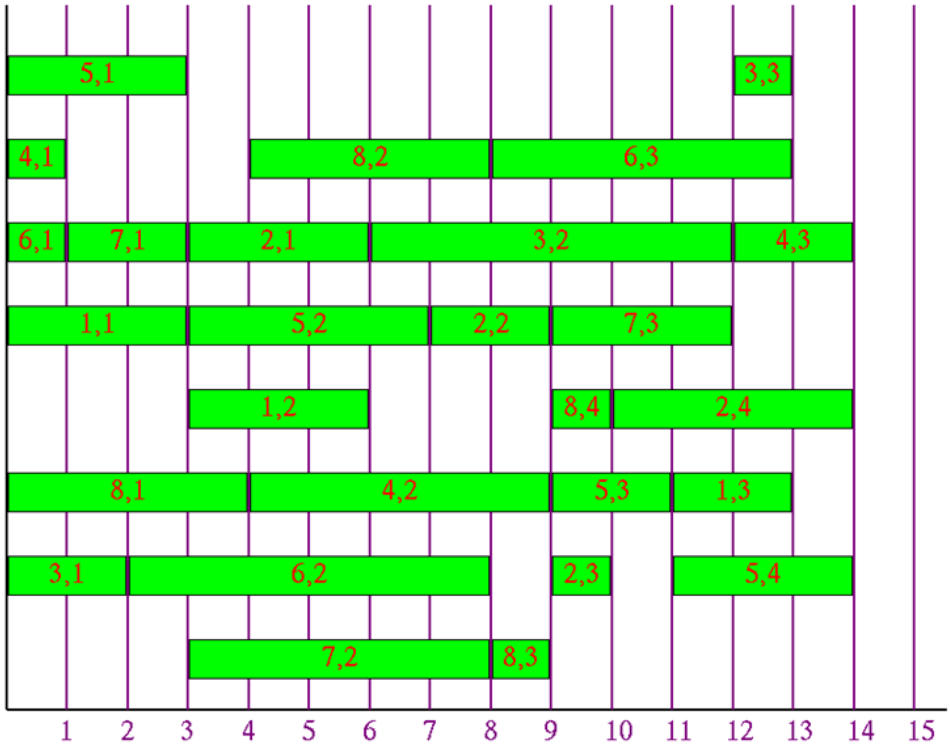


圖 14. 插入[8, 1]後，更新排程

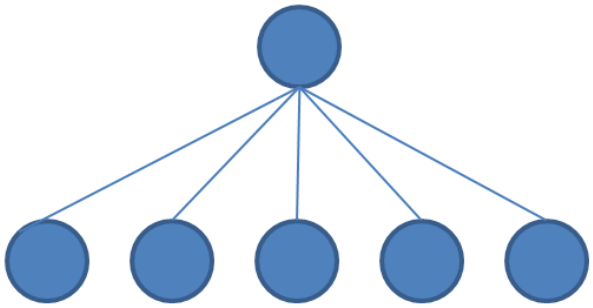
2.3 蒙地卡羅樹狀搜尋應用在多目標彈性零工式工廠排程

蒙地卡羅樹狀搜尋應用在多目標彈性零工式工廠排程[6]是以基本的 MCTS 來解 FJSP。基本的 MCTS 不斷重複選點、展點、模擬與更新四個步驟。在此節中將一一介紹。

2.3-1 選點(Selection)

與 1.2 節中提到的選點相同，按照 Upper Confidence Bounds (UCB)公式選出最好的子節點，遞迴走到目前蒙地卡羅搜尋樹的葉節點。在 MO-FJSP 中會同時考慮多個目標，因此選最佳子節點時，需要依照三個目標的資訊來進行選點。在[6]中，作者使用 Hydra Method 來進行選點：對

於所有目標分別選出分數最高的子節點，再從中隨機選擇一個。



	Node1	Node2	Node3	Node4	Node5
C_M	0.68	0.64	0.62	0.72	0.72
W_T	0.7	0.65	0.62	0.68	0.66
W_M	0.65	0.7	0.45	0.65	0.5

圖 15. 以 Hydra Method 選擇最佳子節點

如圖 15，假設分數愈高愈好，考慮C_M時，Node4 與 5 是最好的、考慮W_T則 1 最好、考慮W_M則 2 最好，因此隨機由這些節點擇一當作最佳子節點。

2.3-2 展點(Expansion)

依照 MCTS 的架構，展點時會從所有可走步中選擇一步來展開新的子節點。在[6]中，作者藉由領域知識(domain knowledge)來決定可走步的優劣，留下較佳的可走步進行展點，其方法如下：

1. 初始化一個陣列A，用以紀錄待展開的候選步。
2. 對每個工作選出第一個可被執行的作業$o_{j,i}$進行以下步驟：
   a. 從所有機器中選執行時間(p_{jik})最短的N_{best}台機器形成集合M_{best}。
   b. 對M_{best}中的機器k，將「$o_{j,i}$放在k上執行」當作一候選步放入A。

3. 從A中擇一進行展點。

	M1	M2	M3	M4
[1, 1]	10	9	8	7
[1, 2]	6	7	5	4
[2, 1]	1	2	3	4
[2, 2]	7	10	3	2

圖 16. 選擇展點候選步的例子

如圖 16，$o_{1,2}$在$o_{1,1}$結束前無法被執行，$o_{2,2}$在$o_{2,1}$結束前也無法被執行，因此不會對$o_{1,2}$與$o_{2,2}$進行展開。假設$N_{best} = 2$，對於$o_{1,1}$而言，選取前兩個執行時間最短的機器，即機器 3 與 4，對於$o_{2,1}$而言則是機器 1 與 2，因此在展點時將由這四種分派方式選擇一種。

2.3-3 模擬(Playout)

一個部份排程(partial schedule)可以藉由模擬而得到一個完整排程(complete schedule)。模擬的詳細步驟如下：

1. 對於目前節點的部分排程狀況，隨機選擇一個可被執行的工作。
2. 從該工作中選擇第一個尚未被分派的作業。
3. 隨機選擇一台機器來執行此作業。
4. 重複步驟 1~3，直到所有作業都已被分派。

2.3-4 更新(Backpropagation)

在進行 MCTS 前，準備一個全域陣列A_{Pareto}，記錄搜尋到排程的非凌越解集合。蒙地卡羅搜尋樹的一個葉節點透過模擬得到一個完整排程後，

進行以下步驟：

1. 計算此排程的目標標準，更新至A_{Pareto}。
2. 由評估函數(evaluation function)對此排程進行評分，獲得分數e。
3. 將e更新至該節點的所有祖先。

以上的e為三個不同的目標標準(C_M、W_T、W_M)轉換而成的分數，更新時分別對三項勝率做更新。

第三章、運用蒙地卡羅樹狀搜尋於多目標彈性零工式工廠排程問題

在這章中，我們提出運用蒙地卡羅樹狀搜尋演算法，3.1 節定義了之後章節用到的參數與符號、3.2 節為本論文考慮的目標標準與分數計算、3.3 節中介紹演算法架構、3.4 節介紹修改的蒙地卡羅樹狀搜尋、3.5 節中包括了四種應用於 MCTS 的變異方法。

3.1 符號定義

以下列出演算法用到的參數與符號：

1. N_{sim}：分派一個作業的決定所需的 simulation 次數。
2. N_{op}：所有作業的數量。
3. $[j,i]$：作業$o_{j,i}$。
4. $[j,i,k]$：作業$o_{j,i}$ (工作 j 的第 i 項作業)被分派在機器 k 上。
5. $[j,i,k,seq]$：作業$o_{j,i}$ (工作 j 的第 i 項作業)被分派在機器 k 上，並且是機器 k 上第 seq 個被執行的作業，以下稱為「四元組決定」或「一步(move)」。
6. (a,b,c)：三個目標標準，其中a為C_M、b為W_T、c為W_M。
7. A_{Pareto}：程式運行到目前為止所找到的非凌越解。

3.2 目標標準與分數計算

3.2-1 節介紹本論文考慮的目標標準、3.2-2 節介紹分數計算的方式。

3.2-1 目標標準

與[6]及部分文獻相同，考慮三個目標標準：總時程(Makespan)、總工作量(Total Workload)與最大工作量(Max Workload)，並且記錄非凌越解集合，其定義於 1.1 節。

3.2-2 分數計算

在 Upper Confidence Bounds 公式中，參考了節點的平均勝率(或分數)：X_j，為了評估一個完整排程的好壞，在運行過程中，分別對三個目標標準紀錄到目前為止找到最好的結果$best(C_M)$、$best(W_T)$、$best(W_M)$，再使用線性對映(linear mapping)的方式將目標標準分別對映出三個分數$v(C_M)$、$v(W_T)$、$v(W_M)$，其公式如下：輸入$(C_M, W_T, W_M) = (a, b, c)$，則：

$$v(C_M) = 2 - \frac{a}{best(C_M)} \tag{8}$$

$$v(W_T) = 2 - \frac{a}{best(W_T)} \tag{9}$$

$$v(W_M) = 2 - \frac{a}{best(W_M)} \tag{10}$$

此分數計算的方式會動態地調整分數評估，隨著A_{Pareto}的品質愈好，評分的標準也會變得愈嚴苛，下頁圖 17 為總時程C_M分數計算的線性對映。

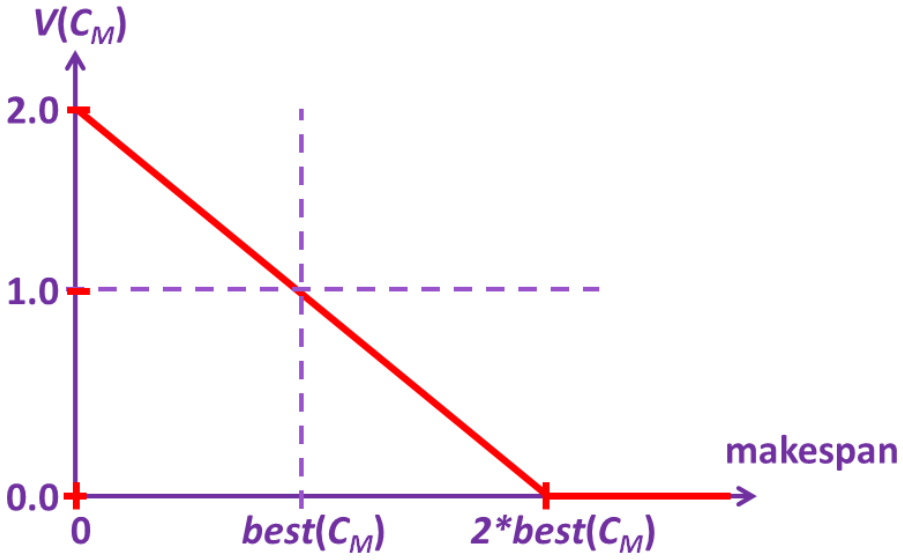


圖 17. 總時程的目標分數計算對映方式

3.3 演算法架構

完整的運用蒙地卡羅樹狀搜尋演算法架構如下：

1. 設S為一個空排程。
2. 重複以下步驟N_{op}次以得到完整排程：
   a. 將S設為蒙地卡羅搜尋樹的根節點，進行N_{sim}個 simulations。
   b. 由目前的蒙地卡羅搜尋樹的根節點，找尋最佳(拜訪次數最多) 的子節點。
   c. 將該子節點的決定$[j, i, k]$加入S中(把$o_{j,i}$分派到機器 k)。

在演算法中，不斷地用一棵蒙地卡羅搜尋樹來決定一個作業的分派。步驟 2 會將一個作業分派到某台機器上，因此重複N_{op}次代表所有的作業都已經被分派，而得到一個完整排程。3.4 節中將會詳細說明步驟 2 的演算法。

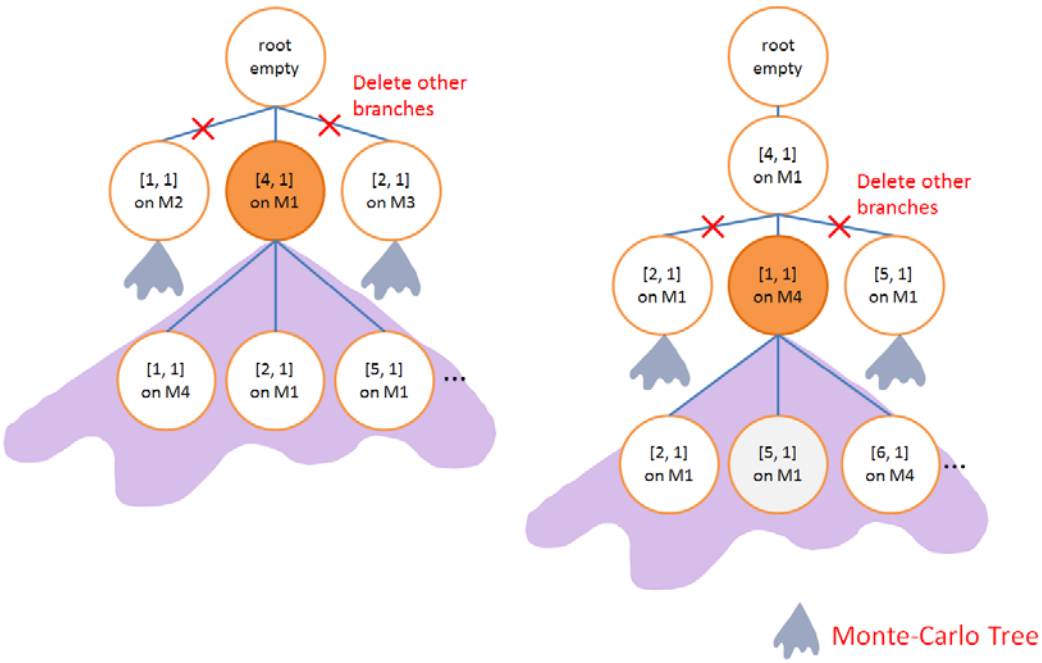


圖 18. 實作時，分派的動作藉由砍掉其他子樹達成

步驟 2 的 c 項會將一個作業分派到一台機器中，在實作時，此分派的動作由砍掉其他子樹來達成。如圖 18 的左圖，從 root 開始進行N_{sim}次 simulations 後，root 最好的子節點是[4,1,1]，因此保留[4,1,1]的子樹，砍掉[1,1,2]與[2,1,3]的子樹。右圖接續左圖，當已經決定了[4,1,1]，剩下的 MCTS 都會由[4,1,1]開始，N_{sim}次 simulations 後，選出[4,1,1]最好的子節點，即[1,1,4]留下。依此類推。

3.4 修改蒙地卡羅樹狀搜尋

基本的 MCTS 包含選點、展點、模擬與更新四個步驟，在本論文提出的演算法中，在模擬後還有一個階段是「區域搜索」(Local Search)，引用了 2.2 節中提到的 LSONE。以下子節會介紹本論文演算法中所用到的 MCTS。

3.4-1 選點(Selection)

如基本的 MCTS 架構，選點時從根節點開始，利用之前模擬的資訊不斷選出平均分數最佳子節點直到葉節點。在[6]中，作者使用 Hydra Method 來選擇最佳的子節點，但在本論文的演算法選點時，只考慮總時程(C_M)，而不考慮總工作量(W_T)與最大工作量(W_M)，其原因為：(1)在工廠調度中，總時程是最常被考慮的[5]。(2)在我們提出的 MCTS 中，在模擬後有增加一個區域搜索的階段，對隨機產生的排程最佳化總時程，若排程的總時程太大，會增加區域搜索的時間。

選擇最佳子節點的公式按照 Upper Confidence Bounds 公式計算，其中X_j為該節點C_M的平均分數，分數依 3.2-2 分數計算節中提到的公式計算。

3.4-2 展點(Expansion)

同 2.3-2 節，展點時只為每個可執行的作業選擇前N_{best}好的機器，在後面的實驗中，皆設定N_{best}為 2，執行時間相同的機器則同時考慮。如圖 19，{M2, M4, M5}為執行$o_{1,1}$前兩好的機器、{M1, M3}為執行$o_{2,1}$前兩好的機器、{M7, M8}為執行$o_{3,1}$前兩好的機器，因此展點時將[1,1,2]、[1,1,4]、[1,1,5]、[2,1,1]、[2,1,3]、[3,1,7]、[3,1,8]展開。

	M1	M2	M3	M4	M5	M6	M7	M8
$o_{1,1}$	5	3	5	3	3	X	10	9
$o_{2,1}$	5	7	3	9	8	X	9	X
$o_{3,1}$	10	X	X	7	6	5	2	4

圖 19. 展點候選步數量大於N_{best}的狀況

3.4-3 模擬(Playout)

在模擬步驟中，我們加入一項領域知識來改善模擬的品質：分派作業$o_{j,i}$時，傾向選擇執行該作業需時最短的機器。此方法可以大幅減少模擬產生排程的平均總工作量($avg(W_T)$)，亦可降低平均總時程($avg(C_M)$)與平均最大工作量($avg(W_M)$)。

為了使模擬產生的排程更具多樣性，我們套用ε-*greedy* 的方式，在選擇機器時，有$1-\varepsilon$的機率會選擇執行此作業所需時間最短機器的其中一台，ε的機率選擇其他執行時間較長的機器。

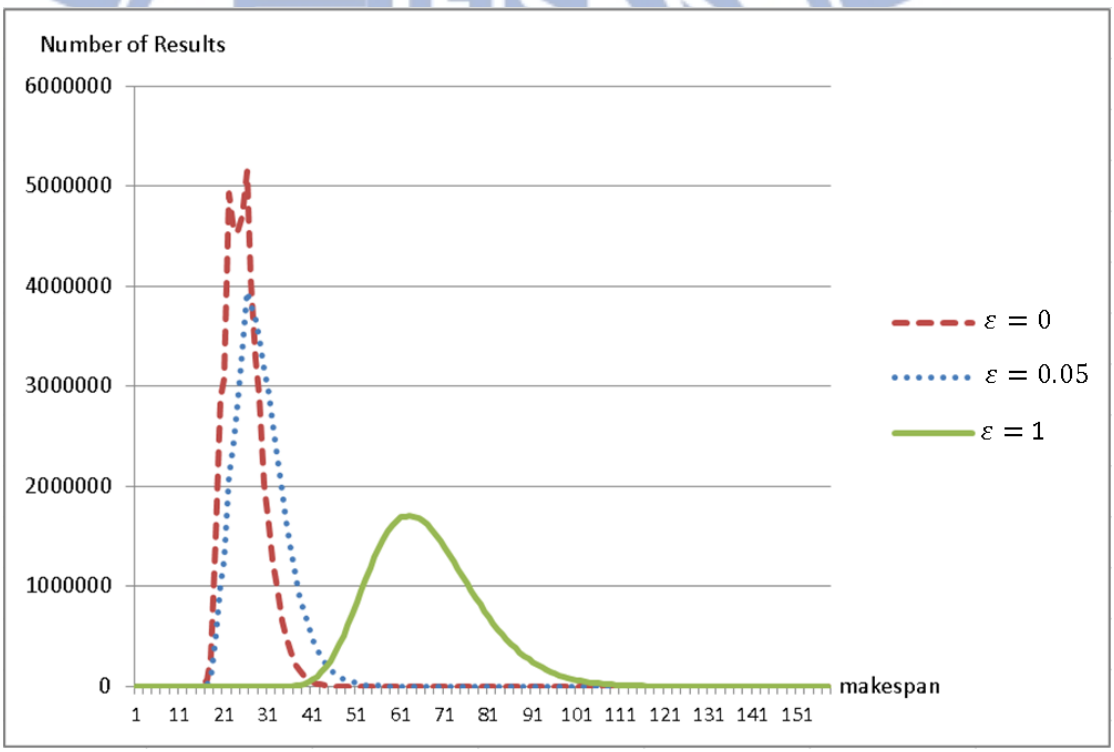


圖 20. ε -*greedy* 與隨機排程的比較

圖 20 為ε-*greedy* 模擬與隨機模擬的$avg(C_M)$比較，基準問題為 Kacem 8x8 [14]，橫軸為模擬產生排程的C_M，縱軸為模擬 5000 萬次所得到不同C_M的次數，三條曲線的設定分別為$\varepsilon = 0$ (完全選擇最好的機器)、$\varepsilon = 0.05$、

$\varepsilon = 1$ (隨機選取機器)，由此圖可以看出在此基準問題中，greedy 模擬可以得到較好的$avg(C_M)$。不同的基準問題中最佳的ε值不盡相同，在第四章的實驗中皆採用$\varepsilon = 0.02$。

3.4-4 區域搜索(Local Search)

在修改的 MCTS 中，模擬的結果會進行區域搜索以最佳化總時程。本論文修改了 2.2 節所提到的 LSONE 來進行區域搜索，其步驟如下：

1. 在排程G中找到所有關鍵路徑P_{all}，令P為P_{all}的第一條關鍵路徑。
2. 令r為P的第一個作業。
3. 重複以下子步驟
   a. 從G中移除r得到G^-。
   b. 在G^-中找尋一個可分派 r 的區間 I。
   c. 若有找到可分派區間，進入步驟 4；否則設 r 為P的下一個作業，若沒有下一個作業則進入步驟 5。
4. 將 r 放入G^-的區間 I 中，得到新的完整排程，設此排程為G，回到步驟 1。
5. 令P為P_{all}的下一條關鍵路徑，若P_{all}沒有其他關鍵路徑則結束。

此演算法不斷地重複使用 LSONE 來移動關鍵路徑上的作業，直到排程G中的每條關鍵路徑上的每個作業都找不到可分派區間才結束區域搜索。

3.4-5 更新(Backpropagation)

更新方法同 2.3-4 更新(Backpropagation)，將目標分數更新至祖先節點並且更新A_{Pareto}，但因為本論文的演算法在選點時並不會考慮W_T和W_M，所以不需要對這兩項目標分數進行更新，只需更新C_M的目標分數即可。

3.5 其他的變異方法

以下子節將介紹四種應用於 MCTS 的變異方法：子樹修剪的方法、Rapid Action Value Estimates Heuristic (RAVE)的更新方式、prior knowledge (Prior)的更新與使用、換位表(Transposition Table)。

3.5-1 子樹修剪(subtree pruning)的方法

在樹狀搜尋的過程中，若可以確定某個子樹的搜尋資訊對考慮的目標毫無幫助，則可以將該子樹砍掉，將計算量集中於其他子樹。在 MCTS 中亦是如此，以下介紹 MCTS 應用在 MO-FJSP 的修剪方法。

Lemma 1. 設節點p的排程為G，從節點p加入一作業r而得到的排程為G'，則$C_M(G) \leq C_M(G')$、$W_T(G) < W_T(G')$、$W_M(G) \leq W_M(G')$。

證明：

i. $C_M(G) \leq C_M(G')$：G是一個部份排程，由已被分派的作業集合O_m形成，$C_M(\mathrm{G}) = \max_{o \in O_m}(C_o)$。因此$G'$由$\{O_m, r\}$組成，下列不等式證明$C_M(\mathrm{G}') \geq C_M(\mathrm{G})$：

$$C_M(\mathrm{G}') = \max_{o \in \{O_m, r\}}(C_o) = \max\left(max_{o \in O_m}(C_o),\ C_r\right) \geq \max_{o \in O_n}(C_o) = C_M(\mathrm{G})$$ 。

ii. $W_T(G) < W_T(G')$：同 i，將一作業 r 加入部分排程G中得到G'，可以得到：$W_T(\mathrm{G}') = W_T(\mathrm{G}) + p_{rk} \geq W_T(\mathrm{G})$。其中 k 為執行作業 r 的機器。

iii. $W_M(G) \leq W_M(G')$：同 i，將一作業 r 附加在機器 k 上得到 k’，顯而易見地 $W_M(\mathrm{G}') = W_M(\mathrm{G}) + \sum_{o_{ji} \in o_{k'}} p_{jik} \geq W_M(\mathrm{G})$。

Lemma 2. 設節點 p 的排程為G，從節點 p 進行模擬而得到排程G'，則

$C_M(G) \leq C_M(G')$、$W_T(G) < W_T(G')$、$W_M(G) \leq W_M(G')$。

證明：

將模擬的動作視為不斷地由部分排程G加入作業r_1得到G_1，再由G_1加入作業r_2得到G_2，...直到G_n為一個完整排程。由 Lemma1 可知：

$C_M(G) \leq C_M(G_1)$、$W_T(G) < W_T(G_1)$、$W_M(G) \leq W_M(G_1)$，且

$C_M(G_1) \leq C_M(G_2)$、$W_T(G_1) < W_T(G_2)$、$W_M(G_1) \leq W_M(G_2)$，

…

$C_M(G_{i-1}) \leq C_M(G_i)$、$W_T(G_{i-1}) \leq W_T(G_i)$、$W_M(G_{i-1}) \leq W_M(G_i)$。

因此：

$C_M(G) \leq C_M(G_1) \leq C_M(G_2) \leq .. \leq C_M(G_i) \leq C_M(G')$、

$W_T(G) < W_T(G_1) < W_T(G_2) < .. < W_T(G_i) < W_T(G')$、

$W_M(G) \leq W_M(G_1) \leq W_M(G_2) \leq .. \leq W_M(G_i) \leq W_M(G')$。

由 Lemma1 可知，子節點的三項目標標準分別比母節點來的大，若在展點或選點時已經確定某個展開的子節點其目標標準被A_{Pareto}中的其中一組解所主宰(三項目標標準皆大於或等於該解)，則可以砍掉該子節點，因為由 Lemma2 可知，由該子節點模擬出來的結果無法更新A_{Pareto}。

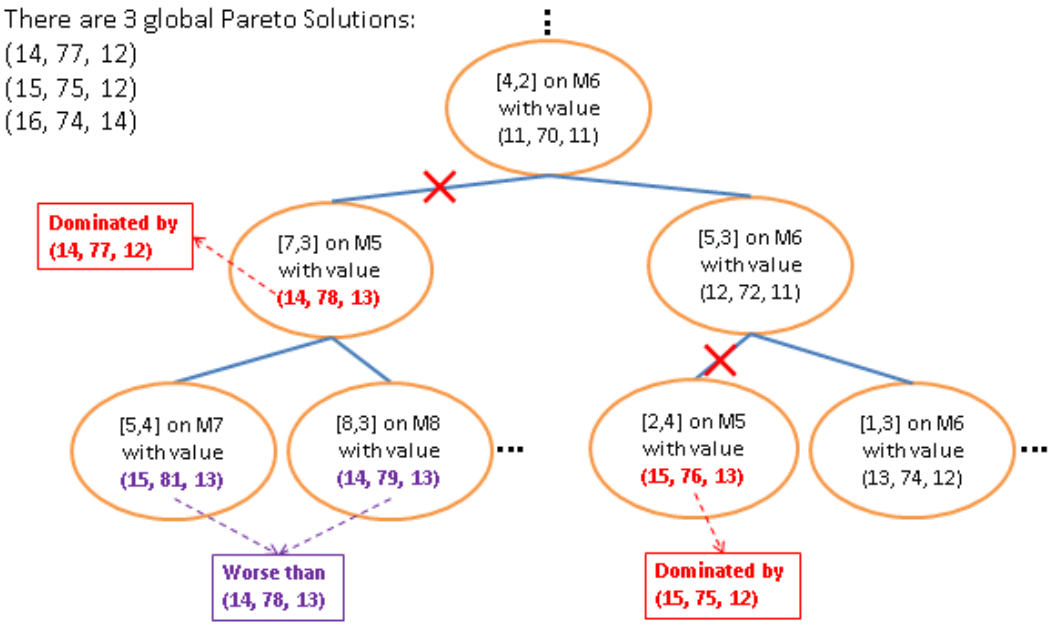


圖 21. 套用子樹修剪的蒙地卡羅搜尋樹

如圖 21，若現在找到的A_{Pareto} = {(14, 77, 12), (15, 75, 12), (16, 74, 14)}，而分派$o_{7,3}$至機器 5 後，部分排程的目標標準為(14, 78, 13)，此目標標準比A_{Pareto}中的(14, 77, 12)還差，因此該節點與它子樹中的節點皆比(14, 77, 12)差，這棵子樹就可以被砍掉(紅色叉叉)。

3.5-2 RAVE

在 MCTS 的更新步驟中，會將目標分數更新至祖先。RAVE 除了更新祖先之外，若祖先兄弟的「決定」(decision)有出現在模擬的結果中，也將模擬的結果更新到此節點。

本論文的演算法定義「決定」為 3.1 節中提到的四元組決定(4-tuple)：$[j, i, k, seq]$。假設經過選點、展點、模擬、區域搜索後得到一個完整排程G，若一個祖先兄弟節點 p 的決定是將$o_{j,i}$放在機器 k 的第seq順位執行，而在G中$o_{j,i}$也是在機器 k 上第seq執行的，則將G的目標分數更新至 p 的 RAVE 平

均勝率與 RAVE 拜訪次數。

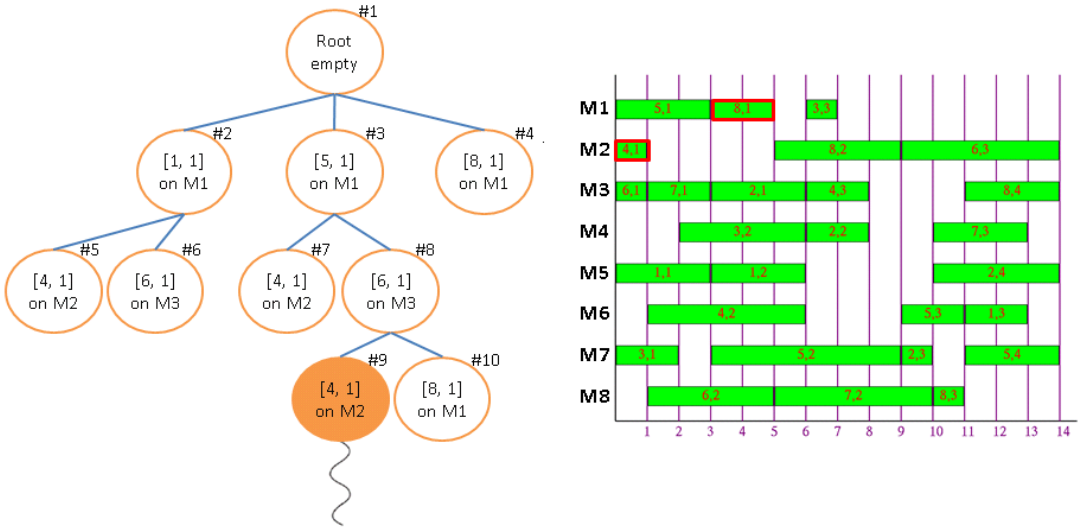


圖 22. 由節點 9 開始執行模擬與區域搜索得到右圖的完整排程

假設目前蒙地卡羅搜尋樹如圖 22 的左圖，展開節點 9(實心節點)之後，進行模擬與區域搜索得到右圖的完整排程G，計算G的目標分數得到e。基本的 MCTS 只更新節點 9 的祖先，即節點 8 與節點 3；令祖先的兄弟節點為$N_{as} = \{\#2, \#4, \#7, \#10\}$ (節點 9 的兄弟為節點 10、節點 8 的兄弟為節點 7、節點 3 的兄弟為節點 2 與節點 4)，RAVE 會判斷N_{as}中元素的決定是否出現在G中來進行 RAVE 更新。以下為N_{as}的 RAVE 更新狀況：

1. #2：#2 的決定將$o_{1,1}$放在機器 1，與G不同，因此不更新。
2. #4：$o_{8,1}$在G中是放在機器 1 的第二順位，而#4 是放在機器 1 的第一順位，因此不更新。
3. #7：#7 的四元組決定為$o_{4,1}$放在機器 2 的第一順位，有出現在G中，因此更新此點的 RAVE 值。
4. #10：#10 的四元組決定為$o_{8,1}$放在機器 1 的第二順位(#3 已將$o_{5,1}$放在機器 1 的第一順位)，有出現在G中，因此更新此點的 RAVE 值。

3.5-3 Prior Knowledge

為了充分利用模擬與區域搜索得到的結果(以下稱為G)，除了 RAVE 更新之外，還將出現在G中的所有四元組決定記錄在全域的表A_{prior}中。A_{prior}紀錄了四元組決定的拜訪次數與平均勝率，是一張大小為$N_{op} \times m \times N_{op}$的陣列($N_{op}$為全部作業的數量，$m$ 為機器數量，圖 1 中的N_{op}為 27、m為 8)，索引是$([j,i],\ k,\ seq)$。第三維度的大小必須設定為N_{op}的原因是所有作業可能都會放在同一台機器上執行。

4-tuple	value	visit count
1,1,1,1	0.5	1
1,1,1,2	0.5	1
…	…	…
1,1,5,1	0.5	1
1,1,5,2	0.5	1
1,1,5,3	0.5	1
…	…	…
4,1,2,1	0.5	1
4,1,2,2	0.5	1
…	…	…
7,2,8,1	0.5	1
7,2,8,2	0.5	1
…	…	…

4-tuple	value	visit count
1,1,1,1	0.601	19
1,1,1,2	0.553	17
…		
1,1,5,1	**0.610**	**18**
1,1,5,2	0.538	13
1,1,5,3	0.445	8
…		
4,1,2,1	**0.630**	**15**
4,1,2,2	0.613	13
…		
7,2,8,1	0.487	12
7,2,8,2	**0.540**	**16**
…		

4-tuple	value	visit count
1,1,1,1	0.601	19
1,1,1,2	0.553	17
…		
1,1,5,1	**0.615**	**19**
1,1,5,2	0.538	13
1,1,5,3	0.445	8
…		
4,1,2,1	**0.634**	**16**
4,1,2,2	0.613	13
…		
7,2,8,1	0.487	12
7,2,8,2	**0.549**	**17**
…		

圖 23. A_{prior}的初始化(左表)與更新(中表、右表)

在整個演算法運行前建立此表，初始化每個四元組的平均勝率為 0.5、拜訪次數為 1，如圖 23 的左表。

每次由模擬與區域搜索得到一個完整排程G與其目標分數e後，就對G中的每個四元組進行A_{prior}的更新，其步驟如下：

1. 設o為G中的第一個作業，其四元組決定d。

2. 令$A_{prior}(d)$的平均分數為avg、拜訪次數為$visit$。

3. 更新$A_{prior}(d)$

$$avg = \frac{avg \times visit + e}{visit + 1} \tag{11}$$

$$visit = visit + 1 \tag{12}$$

4. 設o為G的下一個作業，若G沒有其他作業則結束。

圖 23 的中表為一定數量更新之後的A_{prior}，此時獲得了一完整排程G如圖 22 的右圖，並獲得目標分數e為 0.7，將A_{prior} 的([5,1], 1, 1)、([8,1], 1, 2)、([3,3], 1, 3)、([4,1], 2, 1)…等四元組以分數 0.7 更新而得到圖 23 的右表。

在 MCTS 展點時，若沒有任何新節點的評估，只能給予初始分數一個固定的值(如 0.5)，紀錄A_{prior}是為了讓先前模擬的資訊可以在展點時運用。初始化新展開的節點時，以此節點的四元組決定d到表中查詢平均勝率$A_{prior}(d)$，依下列公式進行初始化：

$$Initial\ Value = \beta \times A_{prior}(d) + (1 - \beta) \times 0.5 \tag{13}$$

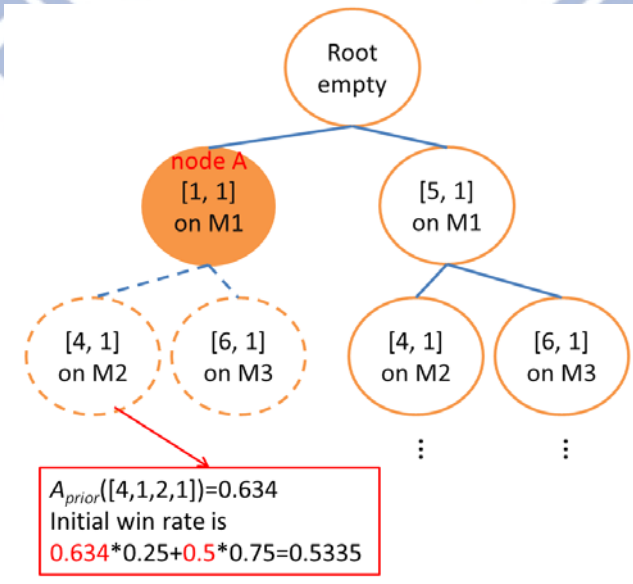


圖 24. 利用A_{prior}給予新展開的節點初始分數

其中β為參考 prior knowledge 的比重，如圖 24，$\beta = 0.25$，在#1 展出新節點#2 時，依照#2 的四元組決定[4,1,2,1]到A_{prior}表中查詢，得到$A_{prior}([4,1],\ 2,\ 1) = 0.634$，再按照公式給予初始分數 0.5335。

3.5-4 換位表(Transposition Table)

在建構蒙地卡羅搜尋樹時，可能會有不同的節點代表相同排程的狀況，如下頁圖 25，決定「$o_{1,1}$放在機器 5」與決定「$o_{5,1}$放在機器 1」的先後順序並不會影響到排程結果，所以#2 和#3 代表的排程是相同的；決定「$o_{1,1}$放在機器 5」與決定「$o_{5,1}$放在機器 5」的先後順序卻會影響到$o_{1,1}$和$o_{5,1}$的執行順序，因此#1 和#4 代表不同的排程。

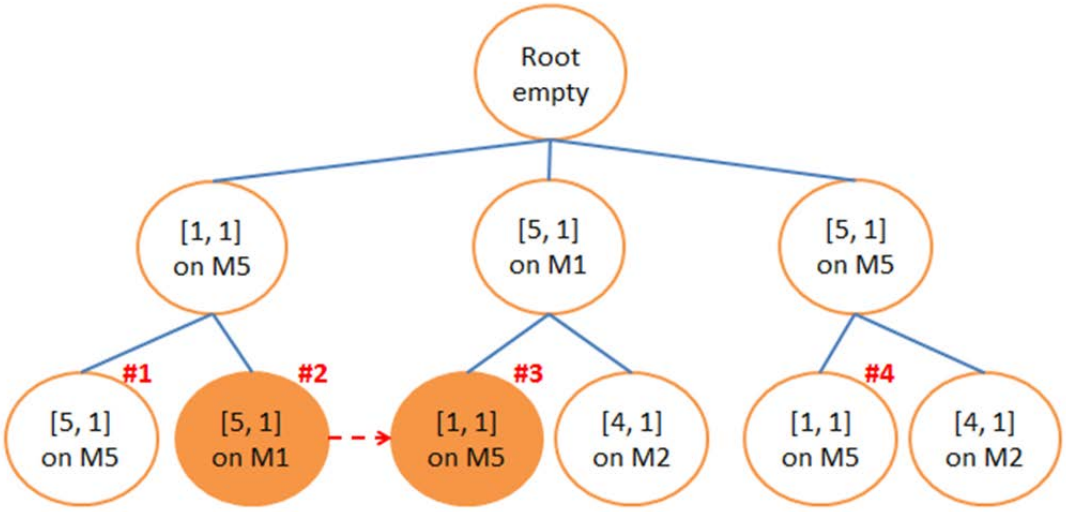


圖 25. 不同樹節點代表相同排程的狀況

我們希望讓代表相同排程的節點共享同一子樹，藉此充分利用模擬的資訊並省下相似的計算，換位表(Transposition Table，以下簡稱為 TT)即是用以偵測搜尋樹中代表重複排程的節點。以下為 TT 在本論文演算法中的設計方式：

1. 進行蒙地卡羅搜尋前，在全域宣告一個大小為2^{25}的陣列A_{TT}，每一列都存放V_{hash}(一個 64 位元的整數)與 *link*(一個指標)。
2. 在展點時，對新節點 p 代表的排程進行 zhash [21]計算得到一個 64 位元的整數h。
3. 取h的後 25 位元得到整數h_{25}。
4. 若 $A_{TT}[h_{25}].V_{hash}$ 等於 h，則將 p 連結至 $A_{TT}[h_{25}].link$；否則設 $A_{TT}[h_{25}].V_{hash}$為h、$A_{TT}[h_{25}].link$為 p。

在 MCTS 的選點、展點與更新步驟，若走到的節點有連結至具有相同h的節點，則要以連結到節點的資訊進行處理。值得一提的是，在本論文提出的演算法中，當選擇了最好的子節點c_{best}並將它的四元組決定加入排程中，會將c_{best}的兄弟子樹砍掉，若c_{best}之下的子樹節點有連結到被砍掉的子樹節點，則會發生連結上的問題，因此在實作時需要將這些被連結到的子樹搬移到c_{best}之下。

第四章、實驗

本章介紹實驗，4.1 節實驗不同的參數設定對運用蒙地卡羅樹狀搜尋演算法的效率影響，4.2 比較運用蒙地卡羅樹狀搜尋演算法與其他演算法所找到的非凌越解集合。

4.1 參數設定

在此節中實驗對不同 UCB 常數、是否使用 RAVE、prior knowledge、TT、子樹修剪的效率，因為參數的數量較大，先固定使用子樹修剪來進行 4.1-1 的實驗：UCB 常數與 RAVE， 4.1-2 節以 4.1-1 實驗中效率最好的參數設定，實驗 prior knowledge 與 TT 對於演算法效率的影響，4.1-3 節再依前兩個子節中最好的參數設定來實驗子樹修剪的重要性。

4.1-1 UCB 常數與 RAVE

以基準問題 Kacem 15x10 [14]為標準進行此實驗，在[4]中作者蒐集了多篇論文找到的非凌越解集合，其中 15x10 的非凌越解集合有兩組，分別為(11, 91, 11)與(11, 93, 10)，稱為A^*_{Pareto}。

		Number of Simulation			
		10000	30000	50000	80000
UCB weight	0	0.42/2	0.94/2	1.52/2	1.86/2
	0.3	0.00/2	0.12/2	0.28/2	1.34/2
	0.7	0.06/2	0.26/2	0.28/2	1.16/2
	1.2	0.02/2	0.06/2	0.18/2	1.30/2

圖 26. 使用 RAVE，不同的 UCB 常數與模擬次數的結果比較

		Number of Simulation			
		10000	30000	50000	80000
UCB weight	0	0.00/2	0.00/2	0.00/2	0.00/2
	0.3	0.02/2	0.52/2	0.84/2	1.38/2
	0.7	0.08/2	0.30/2	0.98/2	1.18/2
	1.2	0.08/2	0.24/2	0.98/2	1.26/2

圖 27. 不使用 RAVE，不同的 UCB 常數與模擬次數的結果比較

圖 26 是使用 RAVE，不同的 UCB 常數與模擬次數N_{sim}的實驗結果、圖 27 是不使用 RAVE 的實驗結果，其中前項數字為平均每次演算法得到的A_{Pareto}有幾組屬於A^*_{Pareto}，後項數字為A^*_{Pareto}中有幾組非凌越解。

每組設定都跑 50 次取平均值，由上圖可知，使用 RAVE 並且 UCB 常數設為 0 的時候，不論N_{sim}的多寡，平均找到解的數量都較其他參數設定多(效率較佳)，而找到解的數量會隨著N_{sim}的增加而變多。N_{sim}達到 80000 次時，對於基準問題 15x10，接近每次都可以找到全部的非凌越解(1.86 組)。以下實驗皆設定使用 RAVE、UCB 常數為 0。

4.1-2 使用 Prior Knowledge、TT 與否

在本子節實驗中固定使用子樹修剪、RAVE、UCB 常數為 0，比較使用 Prior knowledge、TT 與否的四種組合在 Kacem 等人提出的五個基準問題[14]：4x5、10x7、8x8、10x10、15x10 的效率比較。

	4x5				10x7			8x8					10x10				15x10	
C_M	11	11	12	13	11	11	12	14	15	15	16	16	7	7	8	8	11	11
W_T	32	34	32	33	61	62	60	77	75	81	73	77	42	43	42	41	91	93
W_M	10	9	8	7	11	10	12	12	12	11	13	11	6	5	5	7	11	10

圖 28. 不同演算法對於五個基準問題所找到的非凌越解[4]

	4x5	10x7	8x8	10x10	15x10
NONE	3.18/4	3.00/3	4.00/5	4.00/4	0.92/2
TT	3.20/4	3.00/3	3.72/5	3.94/4	1.86/2
PRIOR	3.12/4	3.00/3	4.00/5	4.00/4	1.04/2
TT+PRIOR	3.30/4	3.00/3	3.80/5	4.00/4	1.84/2

圖 29. N_{sim}=30000 時，兩種變異組合的比較

	4x5	10x7	8x8	10x10	15x10
NONE	2.88/4	3.00/3	3.72/5	3.96/4	0.16/2
TT	2.42/4	3.00/3	3.28/5	3.14/4	0.58/2
PRIOR	3.00/4	3.00/3	3.88/5	4.00/4	0.06/2
TT+PRIOR	2.80/4	3.00/3	3.58/5	3.92/4	**0.86/2**

圖 30. N_{sim}=5000 時，兩種變異組合的比較

圖 28 為這五個基準問題目前找到的非凌越解、圖 29、圖 30 分別為N_{sim}設為 30000 與 5000 的平均結果，表示方式如 4.1-1 節，前項數字為平均每次演算法得到的A_{Pareto}有幾組屬於A^*_{Pareto}，後項數字為A^*_{Pareto}中有幾組非凌越解。在較小的基準問題(4x5、10x7、8x8、10x10)中差別較不明顯，而在 15x10 問題中，可以看到$N_{sim} = 30000$時，只用 TT 與 TT+prior knowledge 的效率是差不多的，但在N_{sim}較小的狀況下，就顯現出 prior knowledge 對效率的影響力，因為 prior knowledge 重複利用了模擬與區域搜索得到的結果，在模擬次數較少的狀況下，選點會更準確。

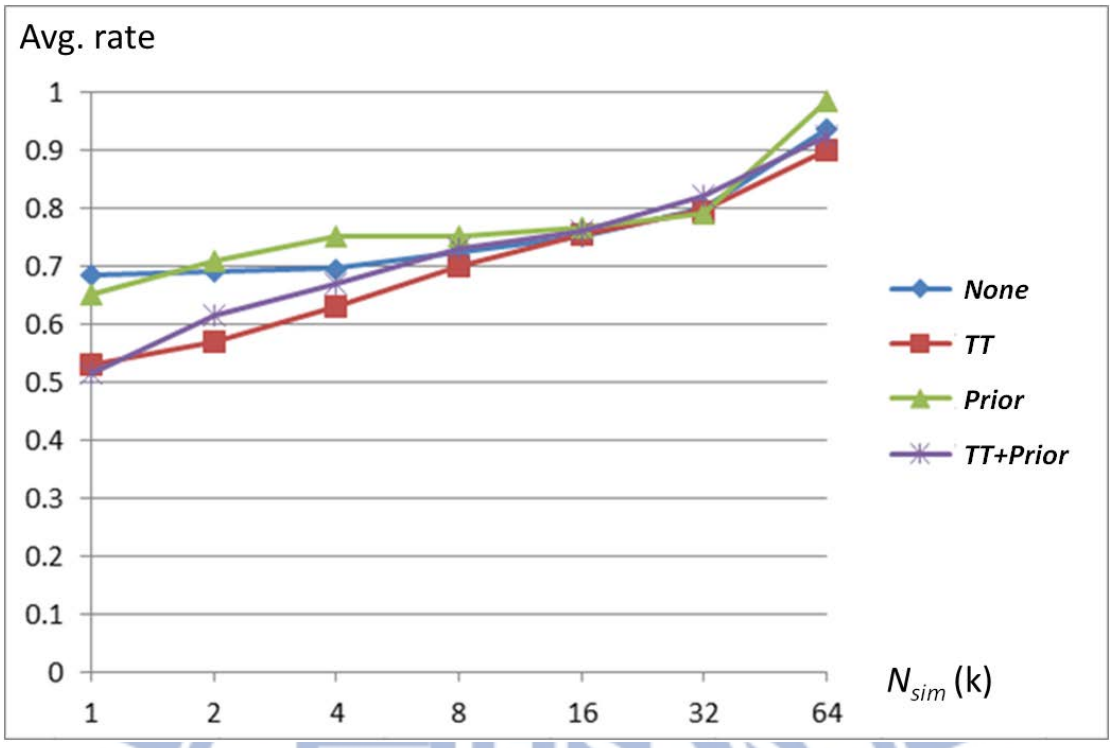


圖 31. 不同模擬次數的效能比較(Kacem 4x5)

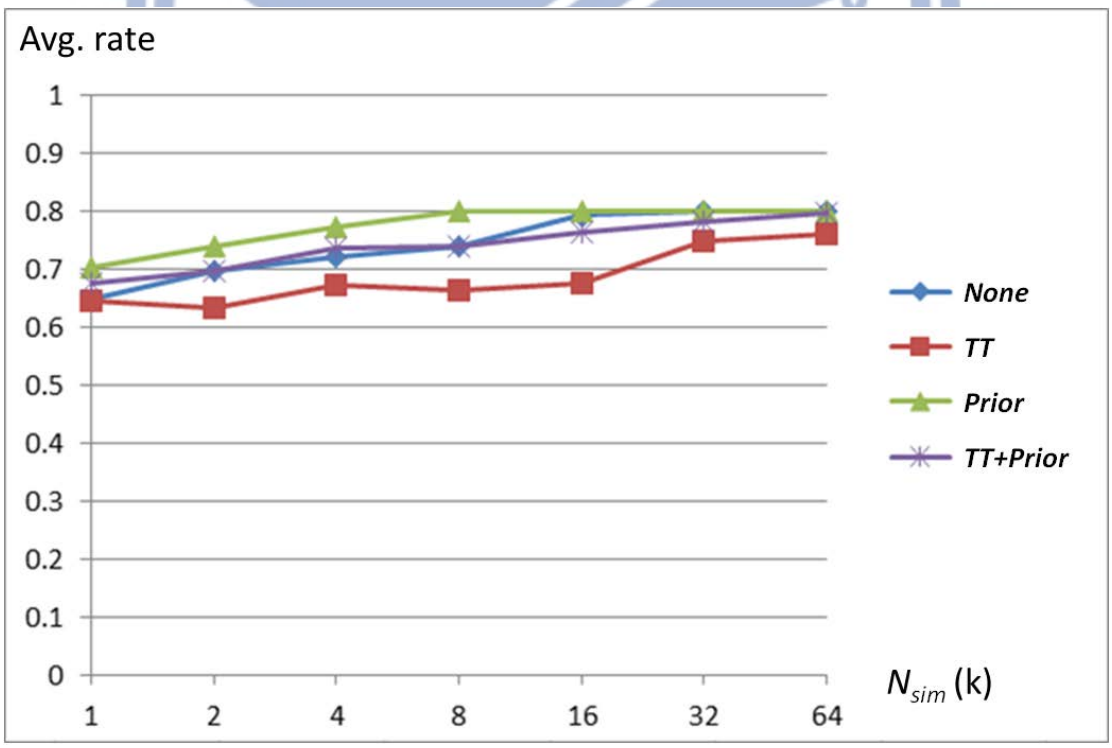


圖 32. 不同模擬次數的效能比較(Kacem 8x8)

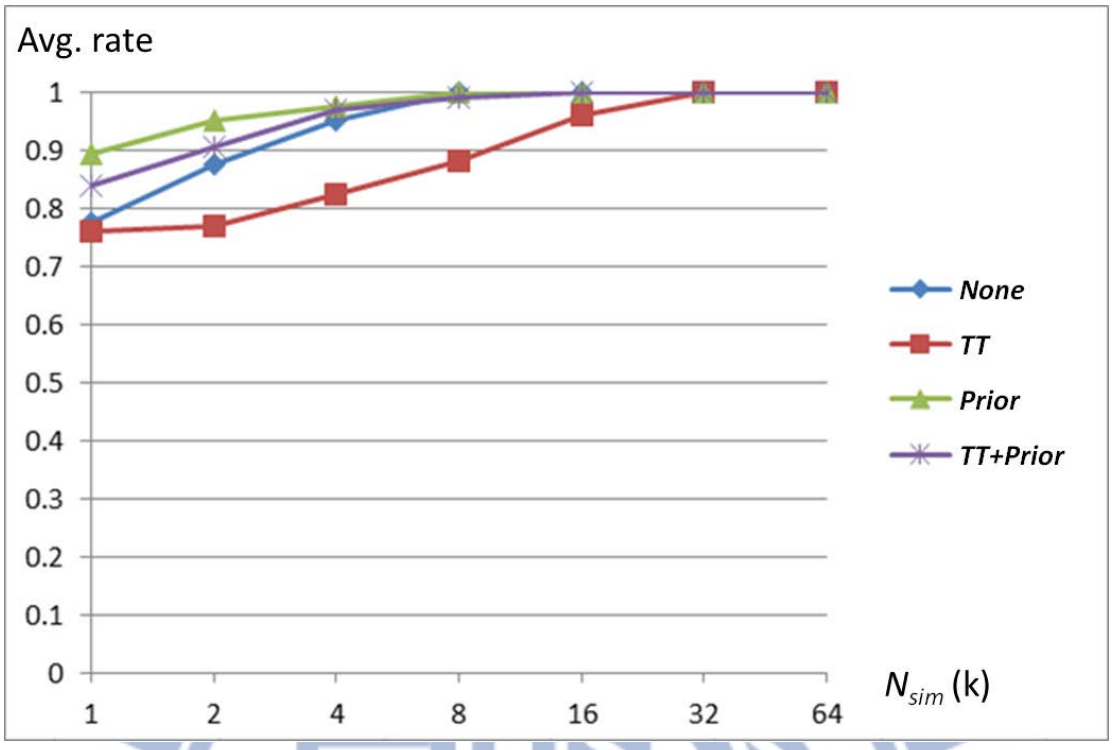


圖 33. 不同模擬次數的效能比較(Kacem 10x10)

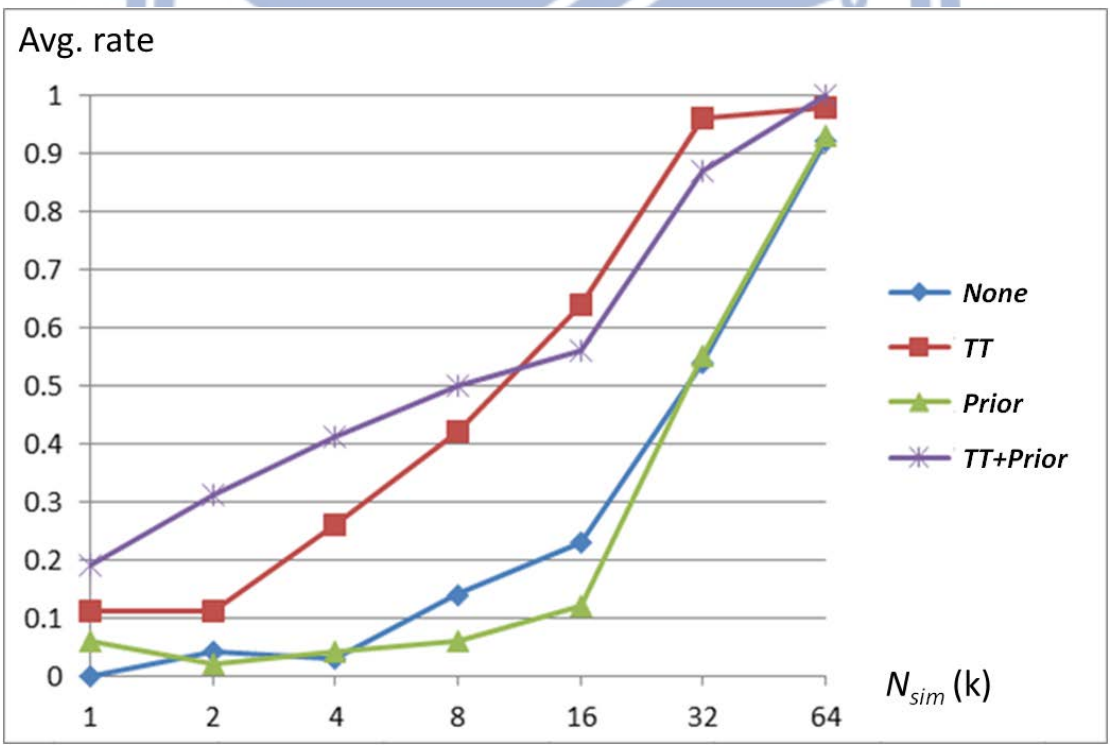


圖 34. 不同模擬次數的效能比較(Kacem 15x10)

圖 31、圖 32、圖 33、圖 34 分別是針對 Kacem 等人提出的 4x5、8x8、10x10、15x10 基準問題，TT 與 prior knowledge 的四種組合在不同模擬次數下的平均結果，橫軸為模擬次數，縱軸為A_{Pareto}在A^*_{Pareto}中的比例。在較小的基準問題中，TT 的效能較差，但在較大的基準問題如 15x10，有無 prior knowledge 與 TT 就會對效能有極大的影響。

4.1-3 使用子樹修剪與否

在本子節實驗中固定使用 RAVE、prior knowledge、TT、UCB 常數為 0，比較是否使用子樹修剪對演算法的效率影響。

	4x5	10x7	8x8	10x10	15x10
PRUNING	3.30/4	3.00/3	3.80/5	4.00/4	**1.84/2**
NONE	3.40/4	3.00/3	3.82/5	4.00/4	1.70/2

圖 35. N_{sim}= 30000 時，子樹修剪在五個基準問題的比較

	4x5	10x7	8x8	10x10	15x10
PRUNING	2.80/4	3.00/3	3.58/5	3.92/4	**0.86/2**
NONE	2.78/4	3.00/3	3.62/5	3.80/4	0.74/2

圖 36. N_{sim}= 5000 時，子樹修剪在五個基準問題的比較

	4x5	10x7	8x8	10x10	15x10
PRUNING	2.30/4	3.00/3	3.44/5	3.62/4	**0.62/2**
NONE	2.60/4	3.00/3	3.44/5	3.66/4	0.54/2

圖 37. N_{sim}= 2000 時，子樹修剪在五個基準問題的比較

圖 35、圖 36、圖 37 分別為N_{sim}等於 30000、5000、2000 的實驗結果比較，其中 ***PRUNING*** 是使用子樹修剪的結果、***NONE*** 是不使用子樹修剪的結果，表示方式如 4.1-1 節，前項數字為平均每次演算法得到的A_{Pareto}有

幾組屬於A^*_{Pareto}，後項數字為A^*_{Pareto}中有幾組非凌越解。可以發現不論模擬次數的多寡，子樹修剪都可以稍微增進演算法的效率。

4.2 演算法比較

本節中比較了其他論文、MCTS 應用於 MO-FJSP 演算法、本論文提出的演算法所找到的非凌越解集合，使用 Kacem 等人提出的基準問題[14]進行比較。圖 38 的第一張表格為[4]中作者蒐集多種演算法的非凌越解、第二張表格為 MCTS 應用在 MO-FJSP 演算法[6]所找到的非凌越解、第三張表格為本論文提出的演算法找到的非凌越解。在[6]中，作者沒有進行 4x5 與 10x7 的實驗，因此標記為 NA。比較中表與下表，MCTS 應用在 MO-FJSP 演算法沒有找到 10x10 的(7, 43, 5)與 15x10 的兩組非凌越解，而本論文的演算法皆有找到；對於 4x5、10x7 與 8x8，本論文的演算法也有找到與其他演算法相同的非凌越解集合，除了 8x8 的(15, 81, 11)則只有論文[19]找到[4]。

	4x5				10x7			8x8					10x10				15x10	
C_M	11	11	12	13	11	11	12	14	15	15	16	16	7	7	8	8	11	11
W_T	32	34	32	33	61	62	60	77	75	81	73	77	42	43	42	41	91	93
W_M	10	9	8	7	11	10	12	12	12	11	13	11	6	5	5	7	11	10
MOGA	v	v	v		NA				v	v	v		v		v	v	v	
SEA	v	v	v	v	v	v	v	v	v		v	v	v	v	v	v	v	v
MA	v	v	v	v	v	v	v	v	v		v	v	v	v	v	v	v	v
Chou	NA				NA			v	v		v	v	v		v	v		
Our MCTS	v	v	v	v	v	v	v	v	v		v	v	v	v	v	v	v	v

圖 38. 不同演算法解 Kacem 等人提出之基準問題的比較

以下為另一組基準資料集合 Mk [2]的比較，此資料集合共有十個基準問題，因為同時考慮三個目標基準的比較結果較為龐大，所以在此只考慮總時程(makespan)。

	List of Pareto Sol.	hGA	EA	MA	Our MCTS
Mk01	40	40	40	40	40
Mk02	26	26	26	26	26
Mk03	204	204	204	204	204
Mk04	60	60	61	61	60
Mk05	172	172	173	172	173
Mk06	58	58	65	62	73
Mk07	139	139	143	140	143
Mk08	523	523	524	523	523
Mk09	307	307	311	307	317
Mk10	197	197	225	208	224

圖 39. 比較不同演算法解 Mk 基準問題的所得到的最佳總時程

第二欄的 List of Pareto Sol.是[4]中作者蒐集至今最好的結果，hGA 是提出 VND 的論文[12]所找到的非凌越解集合，EA 為論文[4]的結果，MA 為論文[5]的結果，Our MCTS 是本論文提出演算法的結果。由圖 39 可觀察到本論文的演算法在 Mk06、Mk09、Mk10 這三組基準問題所找到的總時程明顯較差，這三個基準問題的總工作、作業、機器數量都較多(Mk09 與 Mk10 有 240 個作業)，在作業與機器數量較多的狀況之下，本論文演算法的效率較不明顯，原因是模擬的品質在沒有足夠領域知識的情況下品質下降，總時程的改善主要依靠區域搜索，而在總時程較高的情況下區域搜索運行的時間較長，因此無法進行太多次模擬，MCTS 在模擬數量不足的情況下，有較高的機率選到不好的四元組決定。

第五章、結論與未來展望

過去 MCTS 被廣泛運用在遊戲中如圍棋，將 MCTS 運用在 FJSP 是一個新的嘗試，由東華大學的周政緯提出[6]，使用單純的 MCTS，但該研究目前找到 Kacem 等人提出之基準問題的非凌越解僅有 8x8 四組、10x10 三組，15x10 則尚未找到目前的最佳解。

本論文是由 MCTS 加入區域搜索與其他變異方法：子樹修剪、RAVE、prior knowledge、TT，改善了模擬的品質，並盡可能地將區域搜索的結果在搜尋樹中重複運用。在第四章的實驗中顯示，同時使用這些變異方式的結果是最好的，能夠在 116 秒找出 Kacem 等人提出之基準問題[14]的 17 組解：4x5 共四組、10x7 共三組、8x8 共四組、10x10 共四組、15x10 共兩組，除了 8x8 有一組沒有找到外，其他皆是目前找到最好的。

本研究是第一個用 MCTS 解出 Kacem 等人提出之基準問題[14]中 17 組目前的最佳解，此結果也不亞於其他演算法如基因演算法，然而仍有改善的空間：在作業數量較多的 Mk [2]基準問題如 Mk09、Mk10，因為區域搜索的時間過久導致 MCTS 的效率較不明顯，找到的目標值相對其他演算法也較高。其原因可能有以下幾點：

1. 模擬的品質依賴區域搜索以最佳化，使用的 heuristic 較少，其它論文[4]提到的一些 heuristic 尚未使用。
2. 仍有許多 MCTS 之變形或參數尚未調整。
3. 本論文的演算法對於 sequencing 的排程能力較不完善，當該四元組決定被鎖定(其他子樹被砍掉)，該作業就只能依靠區域搜索來調整分派位置。
4. 一棵蒙地卡羅搜尋樹只決定一個四元組決定較慢。

對於上述提到的幾點，分別提出可能可以改善的方法：

1. 在 MCTS 的模擬中加入更多的領域知識。
2. MCTS 之變形與參數
   a. 調整 MCTS 的參數：UCB 常數、RAVE 權重、prior knowledge 比重。
   b. 調整 MCTS 的變異：改變 RAVE 判斷更新的依據、在選點與更新時增加 variance。
3. 利用較好的排程結果擷取 routing 與 sequencing 的特徵。
4. 調整演算法的架構，可以依 MCTS 節點的平均分數同時決定多個四元組決定。

其他可嘗試的方法如下：

1. 在 MCTS 選點時同時考慮其他目標標準。
2. 嘗試各種不同給予完整排程分數的方式(目前只有動態線性對映，可以採用非線性的方式給予分數)。

目前的演算法在較大的基準問題中所找到的解較不理想，希望未來在實作上述改善方式之後能夠將效率提升，找到更理想的解。

參考文獻

[1] Bagheri, A., Zandieh, M., Mahdavi, I., and Yazdani, M., An artificial immune algorithm for the flexible job-shop scheduling problem, *Future Generation Computer Systems 26*, pp. 533–541, 2010.

[2] Brandimarte, P., Routing and scheduling in a flexible job shop by tabu search, *Annals of Operations Research*, vol. 41, pp. 157–183, 1993.

[3] Browne, C., Monte Carlo Tree Search, http://mcts.ai/.

[4] Chiang, T.C., and Lin, H.J., A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling, *International Journal of Production Economics*, vol. 141, no.1, pp. 87–98, 2013.

[5] Chiang, T.C., and Lin, H.J., Flexible job shop scheduling using a multiobjective memetic algorithm, *Lecture Notes in Artificial Intelligence*, vol.6839, pp.49–56, August, 2011.

[6] Chou, C.W., Monte-Carlo Tree Search for Flexible Job-Shop Scheduling, *French-Taiwanese Workshop on Energy Management*, 2012.

[7] Coulom, R., Efficient selectivity and backup operators in monte-carlo tree search. In P. Ciancarini and H. J. van den Herik, editors. *Proceedings of the 5th International Conference on Computers and Games*, Turin, Italy, 2006.

[8] Coulom, R., Monte-Carlo tree search in crazy stone, in *Proc. Game Prog. Workshop*, pp. 74–75, Tokyo, Japan, 2007.

[9] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactionson Evolutionary Computation 6*, pp. 182–197, 2002.

[10] Gelly, S., and Silver, D., Monte-Carlo tree search and rapid action value

estimation in computer Go, *Artificial Intelligence*, pp. 1856–1875, 2011.

[11] Gelly, S., and Wang, Y., Exploration Exploitation in Go: UCT for Monte-Carlo Go, In: *Twentieth Annual Conference on Neural Information Processing Systems*, NIPS, 2006.

[12] Gao, J., Sun, L.Y., and Gen, M.T., A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, *Computers & Operations Research 35* (2008), pp. 2892–2907, 2007.

[13] Giffler, B., and Thomspon, G.L., Algorithms for solving production-scheduling problems. *Operations Research 8*, pp. 487–503, 1960.

[14] Kacem, I., Hammadi, S., and Borne, P., Pareto-optimality approach for flexible job shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic, *Mathematics and Computers in Simulation*, vol. 60, pp. 245–276, 2002.

[15] Lee, K.M., Yamakawa, T., and Lee, K.M., Genetic Algorithm for General Machine Scheduling Problems, In: *Proceedings of International Conference on Knowledge-based Intelligent Electronic Systems*, pp.60–66, 1998.

[16] Li, J.Q., Pan, Q.K., and Chen,J., A hybrid Pareto-based local search algorithm for multi-objective flexible job shop scheduling problems, *International Journal of Production Research*, 2011.

[17] Moscato, P., and Norman, M., A memetic approach for the traveling salesman problem: implementation of a computational ecology for combinatorial optimization on message-passing systems. In: *Proceedings of the international conference on parallel computing and transputer applications*, Amsterdam; 1992.

[18] Pezzella, F., Morganti, G., and Ciaschetti, G., A genetic algorithm for the flexible job-shop scheduling problem, *Computers & Operations Research 35*, pp. 3202–3212, 2008.

[19] Wang, X., Gao, L., Zhang, C., and Shao, X., A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology 51*, pp. 757–767, 2010.

[20] Xia, W.J., and Wu, Z.M., An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, *Computers & Industrial Engineering 48*, pp. 409–425, 2005.

[21] Zobrist A., A New Hashing Method with Application for Game Playing, *ICCA Journal*, Vol. 13, No. 2, 1990.