國 立 交 通 大 學

運輸科技與管理學系

碩 士 論 文

確定性成本擾動法求解旅行售貨員問題之研究

A Study on Deterministic Noising Method for Solving Traveling Salesman Problem

研 究 生： 簡 輝 鵬

指導教授： 謝 尚 行 博士

中 華 民 國 九 十 五 年 六 月

確定性成本擾動法求解旅行售貨員問題之研究

A Study on Deterministic Noising Method for Solving Traveling Salesman Problem

研 究 生：簡輝鵬　　Student：Hui-Peng Chein

指導教授：謝尚行　　Advisor：Shang-Hsing Hsieh

國 立 交 通 大 學

運輸科技與管理系

碩 士 論 文

A Thesis

Submitted to Institute of Transportation Technology and Management

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master of Business Administration

in

Transportaiton Management

June 2006

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 五 年 六 月

確定性成本擾動法求解旅行售貨員問題之研究

學生: 簡輝鵬　　指導教授: 謝尚行

國立交通大學運輸科技與管理學系

摘要

本研究的目的主要是在求解旅行售貨員問題(Traveling Salesman Problem, TSP)，利用傳統鄰域搜尋方法(Local Research)為主要核心，例如 2-Opt 和 Or-Opt 等發展成熟的方法，結合確定性成本擾動法(Deterministic Noising Method)擾動成本的方法，能在搜尋的過程中陷入區域最佳解時，提供一個跳脫的機制；確定性成本擾動法主要是引用成本擾動法(Noising Method)擾動成本的觀念加以修正和測試，將原本隨機性擾動成本的方式修正為確定性擾動成本的方式；確定性成本擾動法保留成本擾動法在擾動節線成本的觀念，並在擾動上提出具有靈活性且可重現最終結果的方法，希望藉由此研究提供一套穩定、精確且有效的方法。

研究中所提出的確定性成本擾動法針對成本擾動比率值、增加成本比率值、成本擾動比率值長度、增加成本比率值長度等參數，以及起始解解法、核心交換法及擾動公式等執行架構組件的設計與測試。在所建議的組件組合方式與數值範圍下得到的之解題績效為：16 個測試例題在總擾動次數 30 的總平均誤差為 1.01%、標準差為 1.22%；總擾動次數 45 的總平均誤差為 0.88%、標準差為 1.11%；總擾動次數 60 的總平均誤差為 0.80%、標準差為 1.16%。若從整個測試過程中所能找到 16 個例題的最佳個案來看：DNM 法共找到 13 題的已知最佳解，而平均個案誤差為 0.19%。

關鍵詞：旅行售貨員問題、確定性成本擾動法、成本成本擾動法

A Study on Deterministic Noising Method for Solving Traveling Salesman Problem

Student: Hui-Peng Chein Advisor: Shang-Hsing Hsieh

Institute of Transportation Technology and Management
National Chiao Tung University

Abstract

Traveling salesman problem is a well-known NP-hard combinatorial optimization problem. It has been one of the problems in some academic fields such as mathematics, computer science and management science want to resolve. Due to the NP-hard complexity of TSP, it seems to be no way to find out an algorithm which can provide an optimal solution for the TSP without suffering from exponentially growing complexity. If we can find out a more efficient methods to solve the problem, it will help many industries to improve their operational costs.

This research presents an implementation of the Deterministic Noising Method, a combinational optimization meta-heuristics, for solving traveling salesman problem. The DNM revises the Noising Method form a stochastic method to a deterministic method. In this paper, we provide a new formula to disturb the cost matrix in order to escape from a local solution. We also consider different parameter settings to test the performance of the DNM. Sixteen problems from TSPLIB library are used to test the quality of the DNM. The average accuracy of the best solutions of the sixteen problems computed by the DNM is 0.19 % above the performance of the current best known solution. Overall, the heuristics should be a tool for real-world application of TSP.

Keywords: Traveling salesman problem, Deterministic noising method, Noising method

誌謝

本論文得以順利付梓完成，首先要感謝指導教授謝尚行老師，老師不棄個人天資駑鈍，兩年來的悉心指導和諄諄教誨，使學生在論文寫作期間獲益良多。在生活上，老師總是掛在嘴邊「enjoy your life,enjoy your research」的生活態度，影響學生至極，使學生每每在寫作的瓶頸時，仍然保持樂觀且正面的思考。師恩浩浩，永銘於心！

口試期間，口試委員中華大學張靖副教授、系上王晉元副教授於百忙之中不辭辛勞地提供寶貴意見，使本論文的內容更臻完備，在此由衷感謝。在學期間，系上老師在學業上的授業和解惑，使學生能在堂課上均獲益匪淺；特別是韓復華老師不僅在課業上的教導，在平時生活上的鼓勵和器重，也使學生能在遇到困難時，能夠滿懷信心，不畏艱難。感謝之情，片語難抒！

論文研究寫作期間，感謝在同實驗室中一起打拼的子長、承軒、思慧和博士班世鴻學長，因為有你們使我能順利完成這篇論文，能夠跟各位同師門一起成長學習，真的很幸運。還有要感謝十樓韓Lab、許Lab和高Lab等一起努力的好朋友們，謝謝你們陪我度過這個豐富的碩士生活。另外還有彥宏，陪我一起運動、吃飯和發牢騷，這一段同學之情我會永遠記在心中。

最後，謹以本論文獻給我最親愛的家人和女友姿雅，感謝你們同我攜手渡過此一漫長歲月，讓我在生活上無後顧之憂，你們的關懷與鼓勵是我前進的動力，希望你們和我一起來分享這份成果和喜悅。

簡輝鵬 謹誌

風城交大 2006 夏

目錄

圖目錄

表目錄

第一章 緒論

1.1 研究動機

旅行售貨員問題(Traveling Salesman Problem , TSP)的命名是 Hassler Whitney 在 1934 年普林斯頓大學所舉辦的研討會中所提出(Flood, 1956)[18]。長久以來，一直是數學、電腦科學、管理科學等學術領域所欲解決的問題之一，而在產業應用上也相當頻繁，例如物流實體配送、大眾通勤車輛的路線安排、電路印刷版鑽孔設計、繪圖機繪圖順序的安排、網路線的佈置、IC 版零組件安插等等問題，都可以轉換成旅行售貨員問題，其應用非常廣泛，若能穩定有效的方法能求解此問題，對於改善上述各種作業的成本和提高各種企業組織的競爭力上必能有很大的助益。

旅行售貨員問題(TSP)是一種組合最佳化的問題，同時也被Stephen A. Cook (1971)[14]證明至少是NP-complete難度的問題，計算時間會隨著節點數的增加而呈現指數成長，雖然不能證明沒有多項式時間(Polynomial time)的絕對最佳解(Exact Solution)演算法，但發展至今日仍然沒有一種演算法可以在多項式時間內找到最佳解，學者仍致力於突破和解決此問題。現有的最佳解演算法對於規模大的問題，受限於演算法及電腦運算的機制無法在有效時間內求取到絕對最佳解，所以在實際應用上皆採用啟發式解法(Heuristic Method)，希冀在合理的時間內求取準確度高的近似解(Approximate Solution)。

由近年來的文獻回顧得知，部份學者在求解 TSP 這一類 NP-hard 的問題時，轉向於啟發式解法的研究和修正應用，以期能在合理時間內求得準確的近似解，目前文獻上對於應用啟發式解法求解 TSP，已有不少研究成果。隨著新近發展巨集啟發式解法的觀念不斷的被提出，改進了傳統交換型啟發式解法的缺點，但其理論架構與實際執行仍有很大的探索和改善空間，若能利用學者所提出之觀念，修正原始方法理論架構與實際執行的缺點，進而提出更加可行且符合需求之方法；另外對於修正之理論架構與實際執行中的組件和參數亦有詳加探討之必要性，不同的組件和參數的組合嚴重影響到求解品質與準確度。

1.2 研究目的與範圍

本研究在文獻回顧中發現 1993 年 Charon & Hudry[12]首先提出成本擾動法(Noising Method)的解題架構，並將 NM 應用在 Clique Partitioning Problem 之組合最佳化問題上，之後韓復華等人(1996)[6]將 NM 應用於求解 TSP 問題，發現在所選取的 15 個例題中 NM 比 TA 和 SSS 有較佳的解題能力，隨後 Charon &

Hudry(2000)[13]又將此方法應用於求解各種不同型態的 TSP 問題，並與模擬降溫法(Simulation Annealing)做作比較，發現成本擾動法(N M)亦有良好的解題績效。

有鑑於成本擾動法(N M)藉由擾動成本跳脫區域最佳解的觀念在求解 TSP 問題上有穩定的績效，所以本研究擷取此擾動成本的觀念，並提出不同擾動成本的方式，稱之為確定性成本擾動法(Deterministic Noising Method)，以改進原始成本擾動法用隨機方式來擾動成本，會有結果不能重現的缺點。確定性成本擾動法亦結合傳統發展成熟的交換型啟發式解法(Exchanged Heuristics)為本研究之核心，加以進行深入之探索、測試和改良，以確立有效之執行架構；最後將對執行架構中的各個參數進行探討，以建立適當的參數範圍。希望藉由此研究提供一套穩定且有效的方法，能在合理可容忍的時間內，求得準確度高的近似解。

本研究的範圍是將確定性成本擾動法應用於求解完全、對稱、無方向性路網、單一場站和單一路線的 TSP 問題為主，並選取目前國際上已經發表文獻中的旅行售貨員問題，選取適當節點大小的題庫進行測試，以確定本研究所設計之演算法之求解準確度(Accuracy)和時間效率(Efficiency)。

1.3 研究方法與流程

本研究之研究流程與執行步驟，如下頁圖 1.1 所示，分述如下：

(1) 相關文獻蒐集與回顧

蒐集與回顧國內外對於旅行售貨員問題(Traveling Salesman problems)相關文獻，了解 TSP 定義及各種現有啟發式演算法的觀念和發展情況，並進行觀念的探討和分析。

(2) 測試題庫之建立

蒐集目前國內外文獻中已經發表的旅行售貨員問題測試例題及其最佳結果，建立績效評估之測試題庫。

(3) 求解執行架構之建立

建立確定性成本擾動法(DNM)的各種功能組件和模組，分述如下：

(a) 起始解模組構建：利用傳統路線構建法來構建較佳的起始解。

(b) 核心交換改善模組構建：利用發展成熟的交換型啟發式解法來改善起始解，例如 K-Opt 交換法和 Or-Opt 交換法等，並以這些交換型啟發式當成求解之核心交換法。

(c) 跳脫區域最佳解機制模組構建：以確定性擾動成本的觀念以跳脫區域最佳解的機制構建，以搜尋更佳的路線解。

(4) 電腦執行程式之撰寫

將上述所建立之各種模組以 Visual C++ 軟體撰寫程式，以進行解題的準確性與效率之評估。

(5) 確定性成本擾動法之測試與參數設定

以 Visual C++軟體所撰寫程式進行測試，並對參數進行有系統的測試實驗，找出參數的最適組合，以增加此方法的求解穩定性。

(6) 求解結果之綜合比較與分析

針對求解出之路線解，與國際已發表之測試例題進行各項之指標比較與分析，探討本研究模式求解之優劣，作為求解模組的修正依據。

(7) 結論與建議

根據前述各步驟所得結果，提出具體之結論與建議，並研擬未來後續研究方向與重點項目。

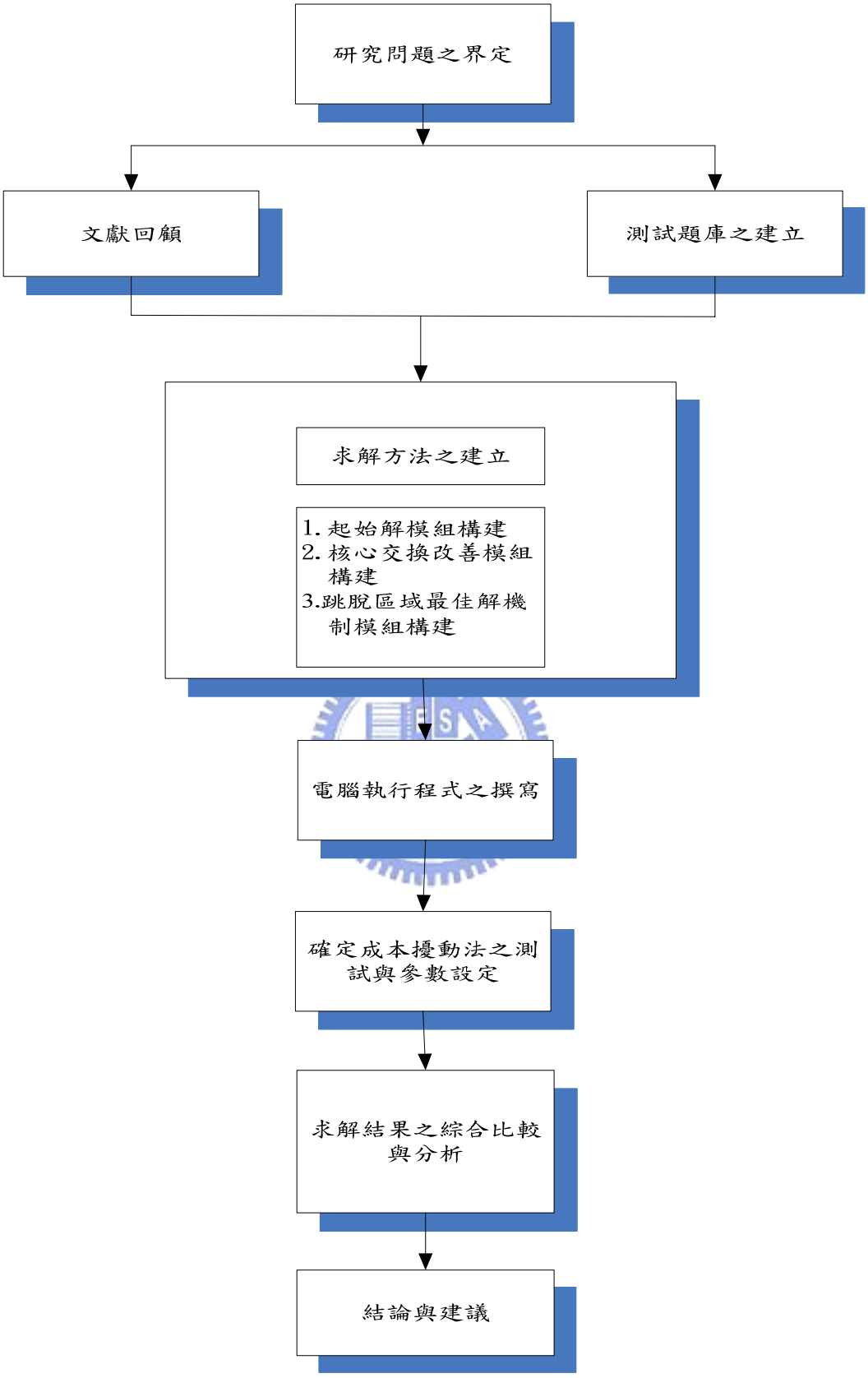

研究問題之界定
文獻回顧
測試題庫之建立
求解方法之建立
1. 起始解模組構建
2. 核心交換改善模組構建
3.跳脫區域最佳解機制模組構建
電腦執行程式之撰寫
確定成本擾動法之測試與參數設定
求解結果之綜合比較與分析
結論與建議

圖 1.1　研究流程圖

第二章 文獻回顧

2.1 旅行售貨員問題(Traveling Salesman Problem ,TSP)

旅行售貨員的問題是指有一位售貨員從某一個城市出發，要到所有指定的城市去推銷貨物，他必須經過所有城市恰一次，最後回到出發的城市，試問他要如何規劃造訪城市的次序，使得總旅行成本最小？

旅行售貨員問題以路網定義為：「在一個路網 G=(N,A)中，N 代表路網所有節點(nodes)的集合，A 代表路網所有節線(links)的集合；目標是在路網上求得一條成本最小的路徑(tour)，此路徑從路網中某一點出發經過 N 中的節點恰一次，最後再回到出發點」。

上述的旅行售貨員問題是一種基本的問題型態，另外在學術研究及實際應用上，韓復華、張靖等人 (1996)[6]曾歸納出許多種變化的型式。

(1) 依路網特性的不同，可分成：「完全(Complete)/不完全路網 TSP」、「對稱(Symmetric)/非對稱路網 TSP」及「無方向(Undirected)/有方向/混合式路網 TSP」等。

(2) 依場站數目多寡，可分成：「單一場站(Single-depot)TSP」與「多場站(Multi-depot)TSP」。

(3) 依路線數目的多寡，可分成：「單一路線(1-)TSP」與「多路線(m-)TSP」。

(4) 依目標函數的不同，可分成：「最小成本(Minimum Cost)TSP」、「最大長度(Maximum Length)TSP」及「最小瓶頸路段長度(Bottleneck)TSP」。

(5) 依節線成本的型式不同，可分成：「固定成本(Fixed Cost)TSP」、「變動成本(Time Dependent Cost)TSP」與「隨機成本(Stochastic Cost)TSP」。

(6) 若有時間或容量的限制，則可衍生成：「最大時間限制(Time Constraints)TSP」、「時間窗限制(Time Windows)TSP」及「車輛路線問題(Vehicle Routing Problem, VRP)」。

(7) 若考慮節點的利潤或成本，則可衍生成：「收集獎金問題(Prize Collecting

Problem, PCP)」、「Orienteering Problem, OP」、「Orienteering Tour Problem, OTP」與「旅行採購員問題(Traveling Purchaser Problem, TPP)」等。

(8) 若將「經過所有節點恰一次」改為「經過所有節線恰一次」，則形成另一個著名的網路問題：「中國郵差問題(Chinese Postman Problem, CPP)」。

2.2 模式介紹

2.2.1 旅行售貨員問題模式

傳統上所說的 TSP 問題，若未加指明，係指：完全、對稱、無方向性路網，單一場站，單一路線，最小成本目標，固定節線成本，無時間及容量限制之 TSP。

傳統的 TSP 問題可用數學規劃模式表示如下[6]：

Minimize $\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}$ $(2-1)$

subject to

$\sum_{i=1}^{n} x_{ij} = 1 \ (j = 1,2,\ldots\ldots\ldots,n)$ $(2-2)$

$\sum_{j=1}^{n} x_{ij} = 1 \ (i = 1,2,\ldots\ldots\ldots,n)$ $(2-3)$

$X = (x_{ij}) \in S$ $(2-4)$

$x_{ij} = 0,1$ $(2-5)$

d_{ij}：表示從節點 i 到節點 j 的距離

x_{ij}：表示連接節點 i 到達節點 j 的節線

其中(2-2)式，(2-3)式限制每個節點只能進出一次，(2-5)式為雙元整數限制式，(2-4)式中的 S 則為避免產生子迴路(Subtours)的限制式。S 的形式一般有以下三種：

(1) $S = \{(x_{ij}) : \sum_{i\in Q}\sum_{j\notin Q} x_{ij} \geq 1 \text{ for every nonempty proper subset Q of N}\}$

意指在 TSP 之解集合 X 中，Q 代表節點子集合，而每個節點子集合 Q 內至少要

有一個節點與 Q 以外的某一節點相連，以避免產生子迴路。

(2) $S = \{(x_{ij}) : \sum_{i \in R} \sum_{i \in R} x_{ij} < |R| \text{ for every nonempty subset R of } \{2,3,.....,n\}\}$

其中，$|R|$ 代表解集合 X 之子集合 R 的節點個數。由於 R 中所有節點若要形成一個迴路，需要有 $|R|$ 條節線，因此該式限制 R 中的節線個數不得超過 $|R|-1$ 條，以避免子迴路的產生。

Dantzig, Fulkerson & Johnson(1954)[15]提出(1)、(2)避免產生子迴路限制式，這兩個形式需要 $2^n - 2$ 條限制式。

(3) $S = \{(x_{ij}) : u_i - u_j + nx_{ij} \le n-1, \text{ for } 2 \le i \ne j \le n, \ u_i \text{為實數}\}$

Miller, Tucker & Zemlin(1960)[22]提出(3)避免產生子迴路限制式，比起(1)、(2)的形式是一個很大突破，此種形式只需要 $(n-1)*(n-2)$ 條限制式，其中 $u_i = j$ 表示節點 i 在 TSP 路線中第 j 個次序被經過。

以上三種形式為線性的整數規劃模式，若不考慮(2-4)式避免產生子迴路限制式，則上述的 TSP 數學規劃模式就變成「指派問題(Assignment Problem, AP)」或「運輸問題(Transportation Problem, TP)」的數學規劃模式。

2.2.2 二次指派問題模式

Koopmans & Beckmann (1957)[24]提出二次指派問題的數學規劃模式，數學規劃模式表示如下：

Minimize

$$\sum_{i=1}^{n}\sum_{k=1}^{n} b_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{p=1}^{n} f_{ij}d_{kp}x_{ik}x_{jp} \qquad (2-6)$$

Subject to :

$$\sum_{i=1}^{n} x_{ik} = 1, \qquad k = 1,.....,n \qquad (2-7)$$

$$\sum_{k=1}^{n} x_{ik} = 1, \qquad i = 1,.....,n \qquad (2-8)$$

$$x_{ik} \in (0,1) \qquad i,k = 1,.....,n \qquad (2-9)$$

b_{ik} ：表示設施 i 設置於位置 k 時的固定成本

x_{ik} ：表示設施 i 設置於位置 k

f_{ij} ：表示設施 i 與設施 j 間的互動流量

d_{kp} ：表示位置 k 與位置 p 間的距離

目標函數(2-6)表示總成本最小，其中二次項部份表示設施間的互動成本，一次項部份表示設施安裝成本，(2-7)表示每一個位置只會有一個設施設置，(2-8)表示每一個設施只會被設置在一個位置上，(2-9)表示當設施 i 設置於位置 k 時 $x_{ik}=1$，否則 $x_{ik}=0$，經過這數十年來，此模式已經被運用在非常多的領域上。

Lawler (1963)[21]提到旅行售貨員問題是二次指派問題的特殊例子，可將旅行售貨員問題轉換成二次指派問題的數學規劃模式表示如下：

Minimize

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\sum_{k=1}^{n} x_{ki}x_{k+1\,j} \qquad (2-10)$$

Subject to :

$$\sum_{i=1}^{n} x_{ki} = 1, \qquad k = 1,.....,n \qquad (2-11)$$

$$\sum_{k=1}^{n} x_{ki} = 1, \qquad i = 1,.....,n \qquad (2-12)$$

$$x_{ki} \in (0,1) \qquad i,k = 1,.....,n \qquad (2-13)$$

d_{ij} ：表示從節點 i 到節點 j 的距離

x_{ki} ：表示順序第 k 次到達節點 i

目標函數(2-10)表示總成本最小，(2-11)表示每一次只會到達一個節點，(2-12)表示每一個節點只會被經過一次，(2-13) 表示順序第 k 次到達節點 i 時 $x_{ki}=1$，否則 $x_{ki}=0$。

此模式排除原來線性模式下的消除子迴路限制式，大大減少了限制式的個數，二次指派模式設計可以排除考慮消除子迴路限制式的優點；但二次指派模式具有離散(Discrete)、非線性(Nonlinear)及非凸性(Non-convex)的特性，是一個非線性整數規劃(Nonlinear Integer Programming)問題，因為二次指派模式具有離散(Discrete)、非線性(Nonlinear)及非凸性(Non-convex)的這些特性，所找到的解只能保證是一個區域(local，relative，extremum)最佳解，而非全域的最佳解。

線性與非線性數學規劃模式(Mathematical Programming Model)在求解上各有其優缺點，以線性模式來說，若沒有消除子迴路限制式則為模式為線性指派問題，在求解上並無太大難度，但加入此限制式後則大幅增加求解難度；而非線性模式雖然沒有消除子迴路的問題，但是目標函數型態變得非常複雜，此種非線性且非凸性(Non-convex)整數規劃問題在求解上亦非常困難。上述兩種是屬於常見的數學規劃模式，若要求得確切解，則通常以數學規劃形式為基礎的方法來求解；然而當問題規模變大時，數學規劃模式所需的解題時間會急遽增加。

2.3 旅行售貨員問題之求解演算法

陳建緯(2001)[5]所整理文獻提到求解 TSP 的演算法一般分為兩大類：一為確切解演算法(Exact Algorithms)，TSP 之求解複雜度被證實是 NP-hard 的問題，到目前為止還無法找出一個多項式時間(polynomial time)的確切解演算法；另一為啟發式演算法(Heuristic Algorithms)，啟發式解法主要是根據問題的特性所設計出有效率的近似值解法。以下將針對 TSP 問題之精確解演算法和啟發式演算法做整理。

2.3.1 精確解演算法

窮舉法(Enumerative Approach)：TSP 問題為一個組合最佳化的問題，若以窮舉法來列舉所有可能的可行解，在計算是非常繁重且沒有效率。

分枝定限法(Branch-and-Bound)/分枝切割法(Branch-and-Cut)：其原理是利用逐漸縮小上限(Upper Bound)與下限(Lower Bound)間的差距來搜尋最佳解，對於像 TSP 這種求最小化問題而言，其最佳解之上限通常是以求解到的當前最佳解(Incumbent solution)來代表，下限通常是一個放鬆某些限制式的不可行解來代

表。如何設計找到下限值來使得上下限差距能快速收斂，成為一個很重要的設計重點。

鬆弛法(Relaxation)：是一種藉由放鬆原問題的某些限制條件來產生上限值或下限值的方法。以 TSP 的問題來說，可藉由最簡單的「線性鬆弛(Linear Relaxation)法」來求得下限值，即是將 TSP 問題中雙元整數限制放鬆成實數整數。「拉氏鬆弛(Lagrangian Relaxation)法」是利用拉氏乘數將某些限制式移至目標式，以簡化原問題。另外也可以將原問題的某些限制式刪除，得到較簡單的問題型式，例如將進出節點一次的限制式放鬆即是「最小伸展樹(Minimal Spanning Tree)問題」和將消除子迴路的限制式放鬆即是「指派問題(Assignment Problem)」，這兩個問題的最佳解都可做為 TSP 問題的下限值。

受到「多面體理論(Polyhedral Theory)」的影響，使得 TSP 最佳解方法之發展有所突破。如：「切割平面(Cutting Plane)法」、「不等式截面(Facet-defining Inequalities)」及「變數產生(Column Generation)法」等，使得最佳解方法之效率大為提昇。而在處理大規模問題時，亦可結合「問題分解(Separation)」的觀念，將原本複雜的模式分解成數個較簡單的模式，然後分別求解。此外，「動態規劃(Dynamic Programming)」亦可用以求解 TSP，其觀念係將求解過程分解成一連串的階段(Stages)，每階段有多種可能狀態(Stages)，最佳解受前階段各狀態之值所影響。

在這麼多的確切解的演算法中，最常被使用的演算法為分枝定限法。分枝定限法的鬆弛方法通常是放鬆 0-1 整數變數的限制，使得子問題變成放鬆的線性規劃問題，並以簡捷(Simplex Method)法求得子問題的最佳解，此為以線性規劃為基礎的分枝定限法(LP-Based Branch and Bound)；其它常見的鬆弛方法有將進出節點一次的限制式放鬆的「最小伸展樹(Minimal Spanning Tree)問題」和將消除子迴路的限制式放鬆的「指派問題(Assignment Problem)」，這些放鬆的方法都是在線性的模式下，再去求解子問題。

2.3.2 啟發式演算法

傳統啟發式解法的較具有代表性的發展，主要分為三大類：

(1) 路線構建法(Tour Construction Procedures)：路線構建式是直接根據已知的成本資料矩陣或路網中節點的距離，直接構建出的較佳的TSP的起始可行解，常見的方法有，鄰點法(Neighbor Procedure)、插入法(Insertion Method)、節省法（Saving Method）、貪心法(Greedy Algorithm)和最小擴張樹法(Minimal Spanning Tree Approach)等，2.4.1節將對此進行部分重點式回顧。

(2) 路線改善法(Tour Improvement Procedures)：路線改善法是先求任意的一個起使解，再以鄰近交換路線或交換節點的搜尋法，以求得較佳的 TSP 解，常見的方法有，節線交換法如 K-Opt 交換法、Lin-Kernighan 交換法、Or-Opt 交換法，節點交換法如 λ-interchange 等，2.4.2 節將對此進行部分重點式回顧。

(3) 綜合型解法(Composite Procedures)：是將路線構建法和路線改善法合併執行，常見的方法有，「路線構建法 + 2-Opt」、「路線構建法 + 2-Opt + 3-Opt」、CCAO 法(Convex Hull + Cheapest Insertion + Largest Angle+ Or-Opt)、一般化插入解繫法(GENeralized Insertion / Unstring and String，GENIUS)等。

2.4 傳統啟發式演算法

求解 TSP 可以利用路線構建起始的 TSP 可行解，再利用鄰域搜尋法(Neighborhood Search Approaches)，進行鄰域路線的搜尋，希望藉以求得更佳的問題解。本小節將先針對幾個路線構建法進行回顧，再對 K-Opt 交換法、Or-Opt 交換法、Lin-Kernighan 交換法等路線改善法進行回顧。

2.4.1 路線構建法(Tour Construction Procedures)

(1) 最近插入法(Nearest Insertion)[27][28][31]

最近插入法是任選三個節點形成原始路線，在其它還未插入路線的節點中，選擇插入現有路線後增加成本最小的節點，形成新的未完成路線，接著反覆進行此一動作，直到所有節點加入路線為止。

(2) 最遠插入法(Farthest Insertion)[27][28][31]

最遠插入法是任選三個節點形成原始路線，在其它還未插入路線的節點中，選擇插入現有路線後增加成本最大的節點，形成新的未完成路線，接著反覆進行此一動作，直到所有節點加入路線為止。

(3) 最近鄰點法(Nearest Neighbor)[31]

最近鄰點法是任選一個起始點，將距離起始點最近的節點相連接，再找出與前一次加入之節點距離最近的節點與之相連，接著反覆進行此一動作，直到所有節點加入路線為止，最後將路線的頭尾相連形成一個封閉的網路。

2.4.2 路線改善法(Tour Improvement Procedures)

本小節將針對 K-Opt 交換法、Or-Opt 交換法及 Lin-Kernighan 交換法等方法進行回顧。

(1) K-Opt 節線交換法[25]

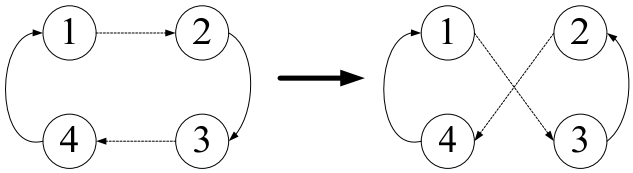
K-Opt 節線交換法是由 Lin 在 1965 年所提出，其中 K 表示每次交換的節線數，一般常被使用的 K 為 2 或 3，以圖 2.1 為例，說明 K-Opt 的交換概念。

K=2 時的交換概念：令圖 2.1(1-a)為一 TSP 之解，若換掉其中(1,2)及(3,4)兩條節線，然後連接(1,3)及(2,4)兩條節線而成為圖 2.1(1-b)封閉而完整的路線。

K=3 時的交換概念：令圖 2.1(2-a)為一個 TSP 之解，若換掉其中(1,2)、(3,4)及(5,6)三條節線，然後再以另外三條節線加以連接，其可能交換組合有十四種，但真正符合封閉而完整路線的交換型態只有四種，以圖 2.1(2-b)來表示其交換概念。

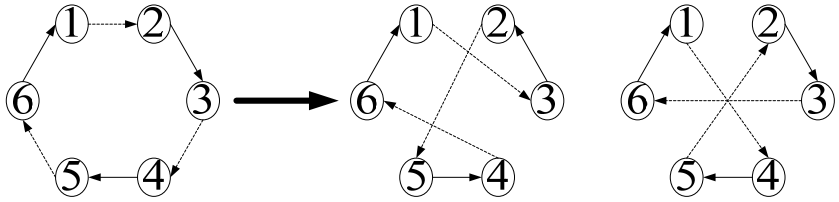
K-Opt 節線交換法：是對任一條 TSP 可行路線，交換路線上任 K 條不相鄰的節線，交換後的路線必須是封閉而完整路線，然後檢查交換後的解是否優於交換前的解。若是則更新 TSP 解；否則維持原解，繼續交換其它 K 條節線，當檢查 K 條節線所有可能交換的路線都無法更新原解，則找到一個區域最佳解。K-Opt 節線交換法，其運算時間及複雜度會隨著 K 條交換的節線數增大而增加，可行性也隨之降低。

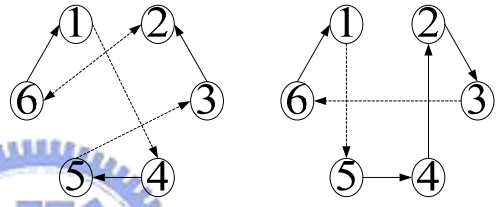
(1) 2-Opt：



(a) (b)

(2) 3-Opt：





(a) (b)

圖 2.1 K-Opt 節線交換法的交換概念

(2) Or-Opt 節線交換法[29]

Or-Opt 節線交換法是由 Or 在 1976 年所提出[29]，其中 p 表示在路線中被取出的節點個數，以圖 2.3 為例，說明 Or-Opt 的交換概念。

Or-Opt 節線交換法的交換概念：是將路線中任意相鄰的 p 個節點抽出，再插入剩餘路線中，以形成新的封閉路線。當抽取節點數 p=3 時，令圖 2.2(1-a)為一個 TSP 之解，先抽取相鄰的三個節點 1、2 及 3，在插入(6,7)的節線中，即換掉其中(9,1)、(3,4)及(6,7)三條節線，然後連接(9,4)、(6,3)及(1,7)三條節線而成為圖 2.2(1-b)封閉而完整的路線。同理 p=2 和 p=1 時可由圖 2.2 中得知。

Or-Opt 節線交換法：是對任一可行 TSP 路線，先將路線中任意相鄰的三個節點抽出，再插入剩餘路線中，然後檢查交換後的解是否優於交換前的解若是則更新 TSP 解；否則維持原解，繼續抽出其它任意相鄰的三個節點，再插入剩餘路線中，檢查完相鄰的三個節點，再依序檢查相鄰兩個節點和一個節點，當檢查完所有交換組合後都無法更新原解，則找到一個區域最佳解。

(3) Lin-Kernighan 節線交換法[26]

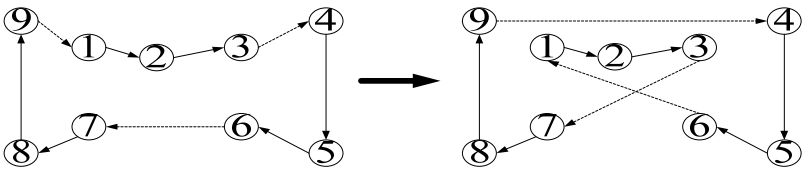
Lin-Kernighan 在 1973 年提出一種稱之為變動深度(variable depth)的節線交換法[26]，因為每次移動時所交換的節線數並不是固定的值，故其產生鄰解的機制更加複雜。

Lin-Kernighan 節線交換法的交換概念在於利用實際計算的經驗進行更有彈性的交換，在此利用 2-opt 的修正過程狀況進行說明，如圖 2.3 所示，假設圖 2.3(a)為一組 16 個點的路網連結情形，利用 2-opt 進行交換後，我們將(1,16)、(9,10)兩條節線刪除，新增(9,16)及(10,1)兩條節線，成為圖 2.3(b)的路網型態，2-opt 演算法將持續依照此原則進行交換，重複新增與刪除兩條節線的動作。但當節點數目很多的時候，此種交換方法將難以對整體路線進行大幅度交換，也降低了尋找到更佳解的可能性。而 Lin-Kernighan 節線交換法則採用不同之方式，將對圖 2.3(b)之(10,1)連結重新進行考慮，尋找另一個可能的交換，假設新增(10,6)，因此刪除(6,7)、(10,1)並連結(7,1)形成圖 2.3(c)之路線；再繼續由圖 2.3(c)之(7,1)連結重新考慮，假設新增(7,13)，因此刪除(12,13)、(13,14)，新增(12,14)、(1,13)等節線，成為圖 2.3(d)的型態。

傳統啟發式解法，先以路線構建方法來構建起始解，再利用交換節點以達到起始解獲得尋優改善之目的。這些傳統的方法都面臨了無法突破的重要難題，即求解過程會落入「區域最佳解(Local Optimum)」而無法求得更佳的精確解，且求出的解無法保證是最佳解。

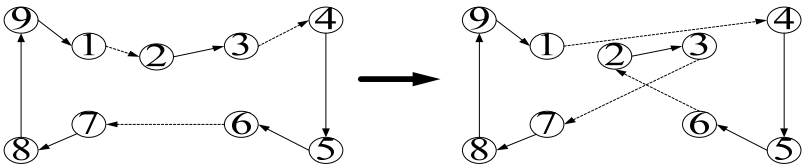
為克服傳統啟發式解法的缺點，自 1980 年代開始，一種新的「巨集啟發式（Meta-heuristic）」解題概念便逐漸成形。所謂的巨集啟發式方法，乃是以傳統啟發式方法為核心架構，並結合高階的搜尋策略（Meta-strategies），使得求解過程中得以跳脫區域最佳解的束縛，一方面並擴大搜尋的空間以便找到更好的解。

(1) Or-Opt (p=3)



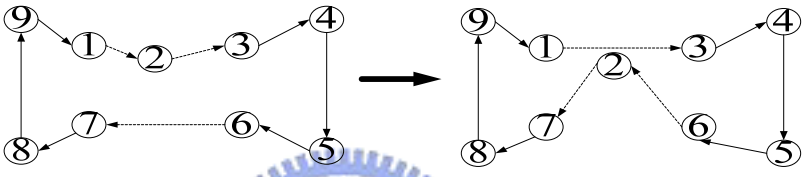
(a) (b)

(2) Or-Opt (p=2)



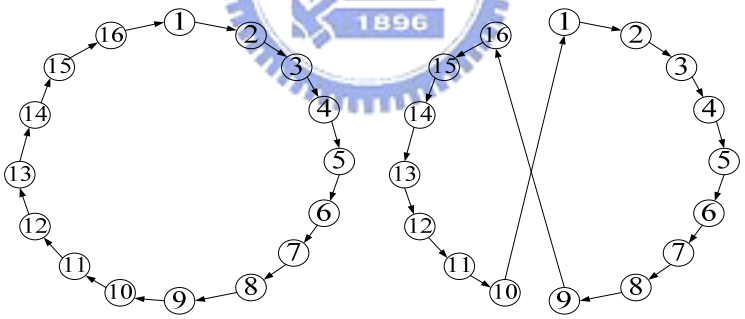
(a) (b)

(3) Or-Opt (p=1)

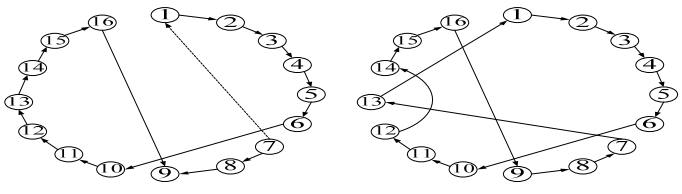


(a) (b)

圖 2.2 Or-Opt 節線交換法的交換概念



(a) (b)



(c) (d)

圖 2.3 Lin-Kernighan 節線交換法的交換概念

2.5 巨集啟發式演算法

巨集啟發式解法著重在如何能跳脫區域最佳解，主要的概念是在上述的傳統啟發式解法落入區域最佳解，結合高階搜尋策略，以跳脫區域最佳解的束縛，並增廣和加深搜尋空間範圍，期能求得更好的目標解。著名的方法有，模擬降溫法(Simulation Annealing, SA)、基因演算法(Genetic Algorithm, GA)、禁制搜尋法(Tabu Search, TS)、類神經網路(Neural Network, NN)、門檻接受法(Threshold Accepting, TA)、大洪水法(Great Deluge Algorithm, GDA)、記錄更新法(Record-to-Record Travel, RRT)、成本擾動法(Noising Method, NM) 、噪音擾動法(Noising Method, NM)、兩極跳躍法(Flip Flop Method, FF)及搜尋空間平滑法(Search Space Smoothing, SSS)等；其中模擬鍛鍊法(SA)、門檻接受法(TA)、大洪水法(GDA)與紀錄更新法(RRT)等屬於門檻型演算法(Threshold Algorithms)；成本擾動法(NM)、兩極跳躍法(FF)與搜尋空間平滑法(SSS)等是屬於擾動型演算法(Perturbation Algorithms)。本小節後續將介紹數種與本研究較相關的著名巨集啟發式方法，並概述其應用情形。

2.5.1 門檻型演算法(Threshold Algorithms)

模擬鍛鍊法(SA)、門檻接受法(TA)、大洪水法(GDA)與紀錄更新法(RRT)皆屬於門檻型演算法。這一類方法的基本觀念乃是在鄰域搜尋陷入局部最佳解時，採取較寬鬆的接受法則(通常為一門檻值)接受劣於現解之鄰解，以便脫離局部最佳解的束縛而繼續搜尋下去。SA、TA、GDA與RRT等方法的執行架構與傳統鄰域搜尋法之架構相似，差異之處僅在於使用的接受法則不同。傳統的鄰域搜尋法僅接受較佳的鄰解，門檻型演算法則可接受暫劣之鄰解。

(1) 模擬降溫法(Simulation Annealing, SA)[23]

模擬降溫法的基本觀念源Metropolis等人於1953年所建立的簡單蒙地卡羅方法(Monte-Carlo)，用以模擬一組原子由一特定高溫逐漸冷卻後所產生的行為，而最早應用於求解組合最佳化問題則是 Kirkpatrick 等人(1983)[23]。SA 法的執行關鍵在於接受法則(Acceptance Rule)與降溫過程(Cooling Process)的機制設計。SA 法的接受法則為機率性(Stochastic)接受暫劣解。利用一個隨機產生的數值與門檻值做比較，此門檻值是鄰解與當前解之目標值差額及溫度的函數；此處所謂「溫度」對 SA 而言是一個抽象的觀念，僅做為控制門檻值高低的參數；降溫則是為了使 SA 能夠逐漸收斂。

SA法用於尋找TSP最佳解的過程，若令 X_i 代表在時間 t 的當前解，其成本為 $C(X_i)$，令 X_j 為搜尋到的下一個解，其成本為 $C(X_j)$，則SA法會接受 X_j 取代 X_i

變成$t+1$時間新解的機率是：

$$PR(A)=\min\{1,\exp(\frac{C(X_i)-C(X_j)}{T_t})\}$$

$$PR(A)=\begin{cases}1 & if\ \ C(X_j)<C(X_i)\\ \exp[\frac{C(X_i)-C(X_j)}{T_t}] & if\ \ C(X_j)\geq C(X_i)\end{cases}$$

其中，T_t稱為時間t的溫度，用以決定接受機率之高低，適當的控制溫度可使 SA 法有效地跳脫區域最佳解，又能在有限時間內收斂，對 SA 法來說是一項非常重要的參數。最後，再利用亂數產生器(Random number generator)產生一個介於 0,1 之間的亂數R。若$R>PR(A)$則不接受X_j，若$R\leq PR(A)$則接受X_j，圖 2.4 表示 SA 法搜尋概念。

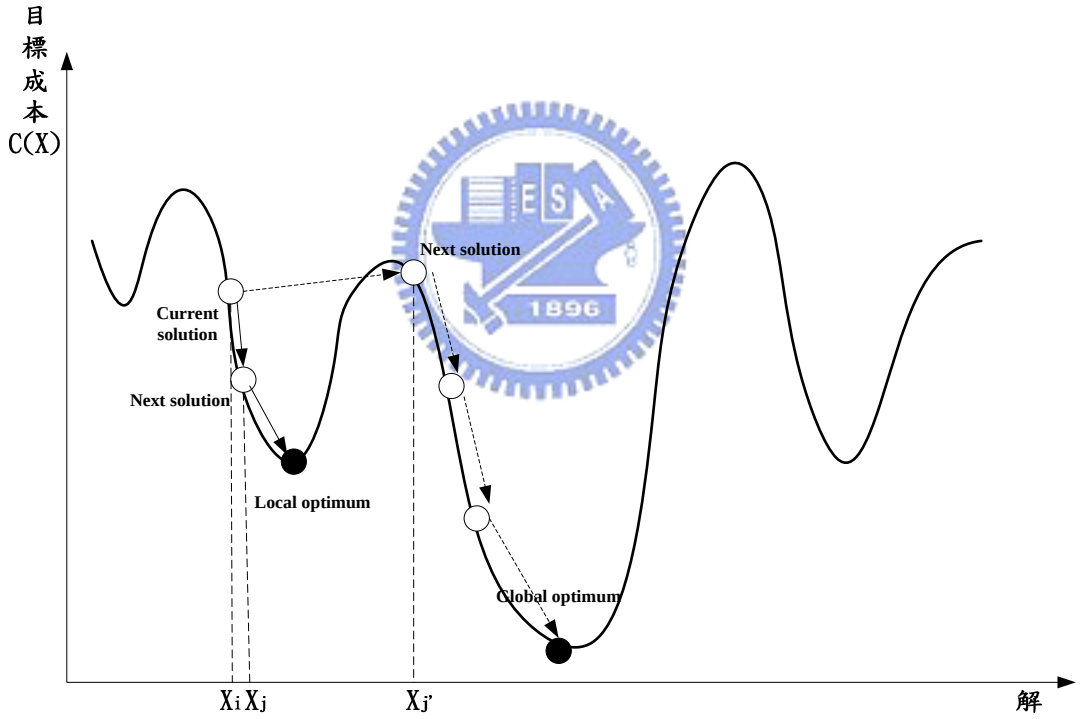


圖 2.4　模擬降溫法之解題概念示意圖

(2) 門檻接受法(Threshold Accepting, TA)[16]

門檻接受法是Dueck & Scheuer在1990年所提出[16]，而TA法的觀念源自於SA法，其搜尋步驟大致都相似。TA法是以門檻數列來決定是否接受新的交換，凡是交換後變動成本在門檻值內的解都予以接受，為使求解過程能達到收斂效果，一般是將門檻值漸次遞減，直到所有門檻數列使用結束為止；惟TA法和SA

法最大不同之處在於接受法則，TA法是採用確定性(Deterministic)的接受法則，而SA法則採用機率性(Stochastic)的接受法則。

TA法用於尋找TSP最佳解的過程，若令 X_i 代表在時間 t 的當前解，其成本為 $C(X_i)$，令 X_j 為搜尋到的下一個解，其成本為 $C(X_j)$，則TA法會接受 X_j 取代 X_i 變成 $t+1$ 時間新解的法則是：

$$PR(A)=\begin{cases}1 & if\ C(X_j)-C(X_i)<T_t\\ 0 & if\ \ C(X_j)-C(X_i)\geq T_t\end{cases}$$

其中，T_t 稱為時間 t 的門檻值。圖 2.5 表示 SA 法和 TA 法接受暫劣解的機率比較。

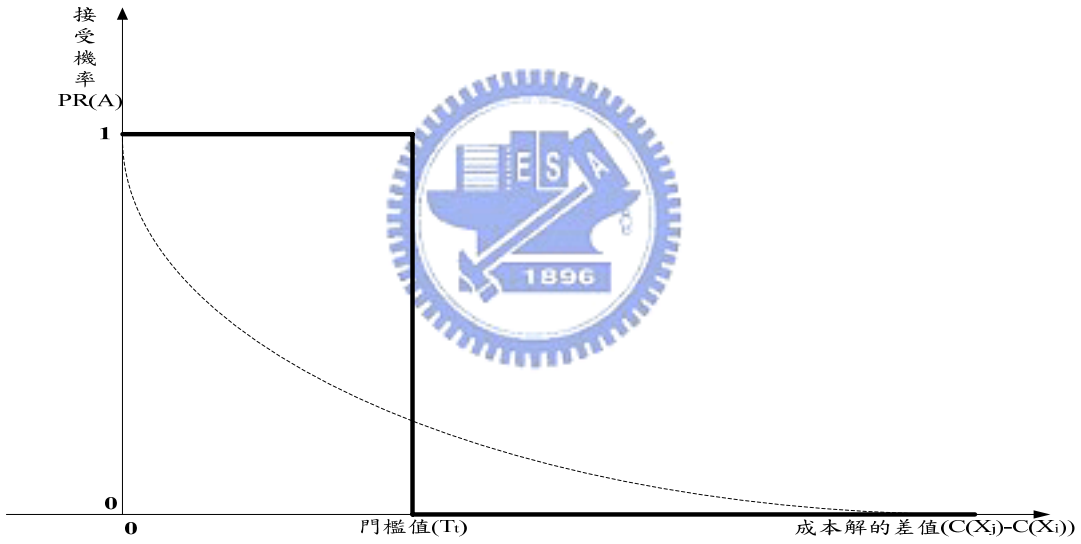


圖 2.5　SA 法和 TA 法接受暫劣解的機率比較

(3) 大洪水法(Great Deluge Algorithm, GDA)[17]

Dueck在1993年根據TA法的測試經驗所提出大洪水法和紀錄更新法[17]，所以GDA法和TA法的觀念和流程大致上都相同。最大不同之處在於接受法則，TA法是事先設定一組門檻數列，隨著搜索過程依次使用數列中的門檻值，最後門檻值降低到零即停止；而GDA法是先設定一個起始的水位，以固定的消退速度下降。

GDA法用於尋找TSP最佳解的過程，若令 X_i 代表在時間 t 的當前解，其成本為 $C(X_i)$，令 X_j 為搜尋到的下一個解，其成本為 $C(X_j)$，則GDA法會接受 X_j 取代 X_i 變成 $t+1$ 時間新解的法則是：

$$PR(A)=\begin{cases}1 & if\ C(X_j)<C(X_i)+L_t\\ 0 & if\ \ C(X_j)\geq C(X_i)+L_t\end{cases}$$

其中，L_t 表示時間 t 的水位。

對一個TSP的問題而言：可以想像解集合空間中有很多「窪地」，而水面則如水庫洩洪一般由高處往低處下降。圖2.6為大洪水法之解題概念示意圖。假設我們現在的位置為圖中之A點，並將水位設於 L_1。接著開始找尋「窪地」，設找到圖中B點，便將水位下降D的高度至 L_2。此時水面 L_2 下的所有範圍皆為吾人找尋並可接受「窪地」的範圍，包括不是很低的C點。如此不斷地搜尋「窪地」及下降水面(至 L_n)，直至找到水面下最低之山谷(E點)或無法再找到更低點為止。

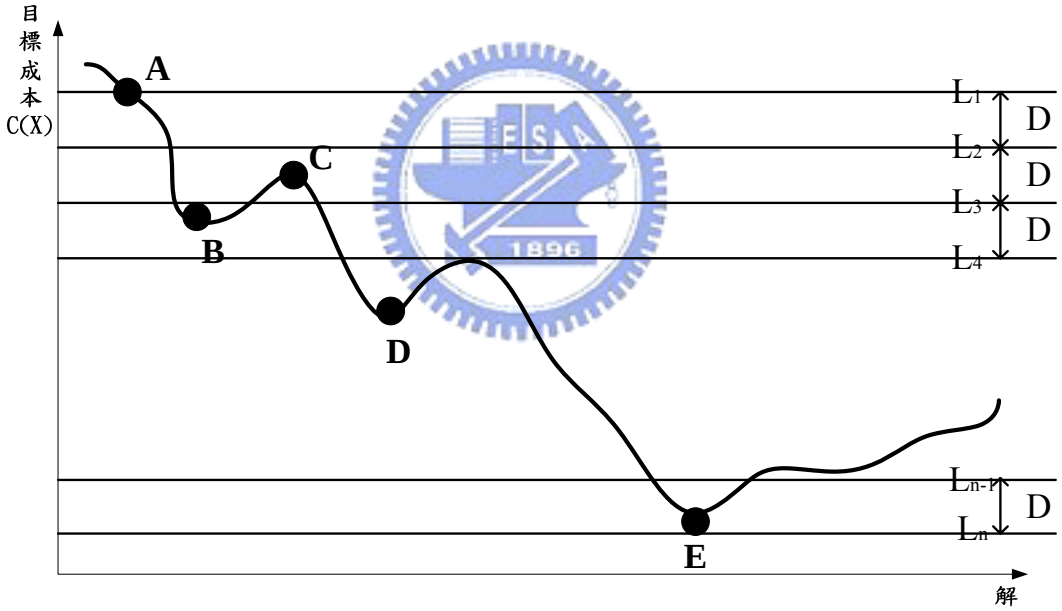


圖 2.6　大洪水法之解題概念示意圖

(4) 記錄更新法(Record-to-Record Travel, RRT)[17]

記錄更新法是Dueck在1993年所提出兩種類似TA法執行架構的方法之一[17]。基本上，如同TA法和GDA法，皆以傳統的局部搜尋法為其核心架構，並具有跳出區域解之機制的啟發式解法。最大不同之處也是在於接受法則，RRT法之接受法則為紀錄值的固定百分比例做為偏差值，此偏差值隨著紀錄值的下降而逐漸變小。

RRT法用於尋找TSP最佳解的過程，若令 X_i 代表在時間 t 的當前解，其成本為 $C(X_i)$，令 X_j 為搜尋到的下一個解，其成本為 $C(X_j)$，則RRT法會接受 X_j 取代 X_i 變成 $t+1$ 時間新解的法則是：

$$PR(A)=\begin{cases}1 & if\ C(X_j)<C(X_i)+R_t\times p\% \\ 0 & if\ \ C(X_j)\geq C(X_i)+R_t\times p\%\end{cases}$$

其中，R_t 表示時間 t 的記錄值，$p\%$ 為固定偏差率。

圖2.7為RRT法之解題概念示意圖。假設目前找到的記錄(Record)為A點，則目前可接受新鄰域解之值為低於目前記錄值加上目前偏差值之值，而目前的偏差值為目前記錄值乘上固定偏差率($p\%$)。在圖中包括D點在內之鄰域解，其值皆低於可接受範圍值，則接受這些較劣鄰域解，而此時之偏差值一直為一固定值，直到找到另一個新的記錄B點時，偏差值始隨著記錄值之減少而變小。RRT法和GDA法最大差異即在於GDA法是以一固定之值慢慢下降，而RRT法之偏差值則會隨著新的記錄值而改變。而RRT法之停止法則唯一經過一段時間後仍無法再找到新的記錄值時就停止。

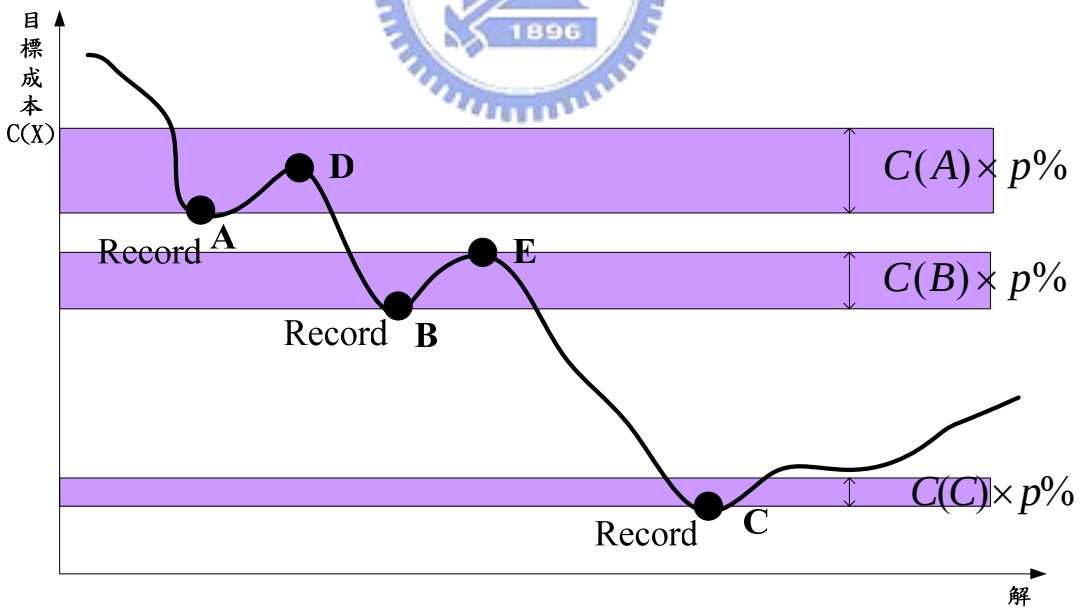
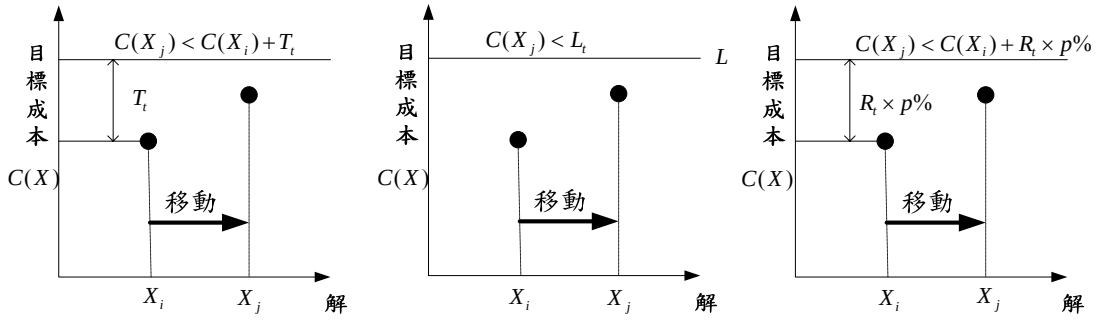


圖 2.7　記錄更新法之解題概念示意圖

TA法、GDA法和RRT法等三種門檻型演算法的觀念和流程大致上都相同，最大不同在於接受法則，圖2.8表示TA法、GDA法與RRT法接受法則的不同[3]。門檻型演算法又稱為包容性搜尋法，門檻型的四種方法應用於求解TSP問題時，

其控制參數、接受法則、收斂法則及停止法則分別說明如表2.1，其中 $C(X_i)$ 為現解 X_i 的目標成本值；$C(X_j)$ 為鄰解 X_j 的目標成本值。



(a) TA　　(b) GDA　　(c) RRT

圖 2.8　TA、GDA 與 RRT 接受法則示意圖

表 2.1　SA、TA、GDA 與 RRT 四種方法比較

方法	SA	TA	GDA	RRT
控制參數	溫度 (T_t) 機率值 $(0 \le r \le 1)$ 次數 (K)	門檻 (T_t) 次數 (K)	水位 (L_t) 速度 (D)	偏差率 $(p\% < 1)$ 記錄值 (R_t) 次數 (K)
接受法則	機率性接受： $R \le \exp[\frac{C(X_i) - C(X_j)}{T_t}]$	確定性接受： $C(X_j) \le C(X_i) + T_t$	確定性接受： $C(X_j) \le L_t$	確定性接受： $C(X_j) \le C(X_i) + R_t \times p\%$
收斂法則	T_t 遞減	T_t 遞減	$L_t = L_{t-1} - D$	更新 R_t 值
停止法則	完成 K 次迴圈	完成 K 次迴圈	當所有 $C(X_j) > L_t$	完成 K 次迴圈

註：(1) 資料來源：[3]

2.5.2　擾動型演算法(Perturbation Algorithms)

成本擾動法(NM)、兩極跳躍法(FF)與搜尋空間平滑法(SSS)等是屬於擾動型演算法，擾動型演算法的基本觀念是「改變搜尋空間」。這類方法仍以傳統鄰域搜尋法為其移動之機制；當鄰域搜尋陷入局部最佳解時，藉由暫時性的成本擾動改變其原有的搜尋空間，然後藉由傳統鄰域搜尋跳脫出原始成本空間之局部最佳

解的束縛。擾動型演算法之設計重點在於成本擾動機制與執行架構；NM、FF與SSS的執行架構較傳統鄰域搜尋架構複雜，其中，NM與FF屬於原始搜尋空間與成本擾動空間兩者交替搜尋之雙層架構，而SSS則屬於一連串平滑空間持續搜尋之單層架構。

(1) 成本擾動法(Noising Method, NM)[12]

1993年Charon & Hudry首先提出成本擾動法(NM)的解題架構，並將NM應用在Clique Partitioning Problem之組合最佳化問題上。NM法的解題觀念為：已陷入局部最佳解的情況下，使用隨機產生的噪音量(Noise)來擾動原來的成本函數，然後再對擾動後的問題重新求解，期望藉由成本的擾動使其跳離局部最佳解的束縛。NM法跳出局部最佳解的概念可由圖2.9進行說明。

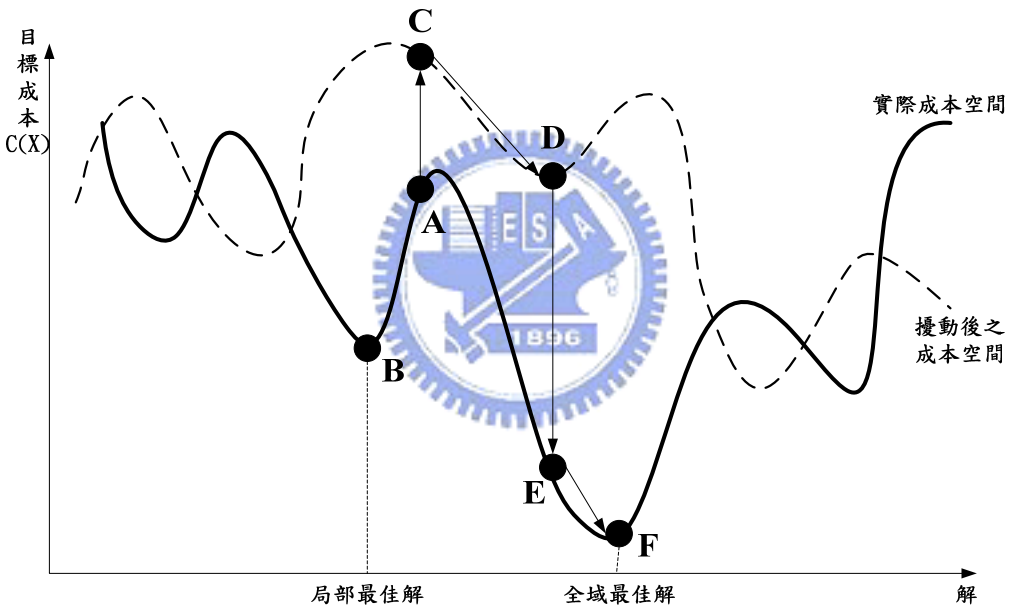


圖 2.9　成本擾動法之解題概念示意圖

對於一個TSP的問題而言：假設A點為實際成本空間上所求得的起始解，若以傳統的鄰域搜尋法來改善此起始解，最後一定會陷入局部最佳解B點。此時若利用隨機產生具有正負值的噪音量加於原來的成本之上，即將圖2.9中實線所示之實際成本空間改變成虛線所示的擾動成本空間，則可將B點投射到擾動後的成本空間上之C點，再於此擾動成本空間上繼續執行局部搜尋法，即可到達擾動成本空間上之局部最佳解D點。此時相對於實際成本空間而言，目前的解E已跳出了原B點鄰近之局部最佳解A之限制。因此若再將D點轉換回實際成本空間上的E點，並繼續執行鄰域搜尋法，便有可能找到全域最佳解之F點。

從表 2.2 的求解步驟可知，NM 法在執行時必須先設定四個主要的控制參數，分別為：最大噪音率($R_{\max}$)、最小噪音率($R_{\min}$)、週期次數(N_k)和每週期內重複次數(N_i)。為使噪音擾動趨於平緩，在每一次擾動後進行下降噪音率的動作，噪音率的下降步距為μ，μ為一固定值，其計算如下(2-14)公式所示。此外，用噪音率隨機擾動成本的方式如(2-15)公式所示：

$$u = (R_{\max} - R_{\min}) \Big/ (N_k \times N_i) - 1 \qquad (2\text{-}14)$$

$$c_{ij}^{p} = c_{ij} + (r \times R \times c_{ij}(\max)) \qquad (2\text{-}15)$$

其中，c_{ij}^{p}：為擾動後之成本；

c_{ij}：為原始之成本；

r：為均勻分配於-1 到 1 之間的隨機值；

R：為噪音率；

$c_{ij}(\max)$：為c_{ij}中之最大值。

最早將 NM 法應用在 TSP 上是國內韓復華等人(1996)[6]的研究，之後有 Charon & Hudry(2000)[13]等相關文獻。

表 2.2　NM 演算法的基本求解步驟

參數

最大噪音率 $R_{\max}$ 、最小噪音率 $R_{\min}$ (實數)；

週期次數 N_k 、每週期內重複次數 N_i (整數)。

變數

現有解變數 X_n 、暫優解變數 X_t ；

現有目標成本值 $C(X_n)$ 、最佳目標成本值 $C(X_t)$ 、噪音率 R 、下降步距 μ (實數)。

步驟一：設定參數

設定：最大噪音率 $R_{\max}$ 、最小噪音率 $R_{\min}$ (實數)；

設定：週期次數 N_k 、每週期內重複次數 (N_i) (整數)。

計算步距：$u := (R_{\max} - R_{\min}) \Big/ (N_k \times N_i) - 1$ ；

設定噪音率：$R := R_{\max}$ 。

步驟二：求起始解

構建可行的起始解 (X_0) ，並令：

現有解:=起始解，即 $X_n := X_0$ ；

暫優解:=起始解，即 $X_t := X_0$ 。

步驟三：執行

執行迴圈：$k = 1 to N_k$ ，

執行迴圈：$i = 1 to N_i$ ，

對原始成本空間 $C(X)$ 擾動，得到新的成本空間 $C^p(X)$ ；

在 $C^p(X)$ 上對 X_n 進行鄰域搜尋，得到區域最佳解 X' ，

並令：現有解=區域最佳解，即 $X_n = X'$ ；

在 $C(X)$ 上對 X_n 進行鄰域搜尋，得到區域最佳解 X'' ，

並令：現有解=區域最佳解，即 $X_n = X''$ ；

若 $C(X_n) < C(X_t)$ ，則更新暫優解，即 $X_t = X_n$ ；

降低噪音率，令：$R = R - \mu$ ；

完成迴圈 i ；

更新現有解，即 $X_n = X_t$ ；

完成迴圈 k ；

停止，輸出最終結果：X_t 與 $C(X_t)$ 。

(2) 兩極跳躍法(Flip Flop Method, FF)[4]

兩極跳躍法(FF)為韓復華與陳國清於1996所提出[4]，其主要概念在於當目前解已陷入局部最佳解而難以改善時，將所有的成本乘上-1，然後繼續鄰域搜尋，使其往反方向搜尋，如此能跳脫此區域並求得一個區域最大成本解後，再往原始空間的另一個區域解搜尋。此外，為避免搜尋過程重複原先搜尋途徑而落入原先的局部最佳解無法跳脫，FF法在成本擾動前後採用不同的鄰域搜尋方法。FF法解題概念可由圖2.10進行說明。

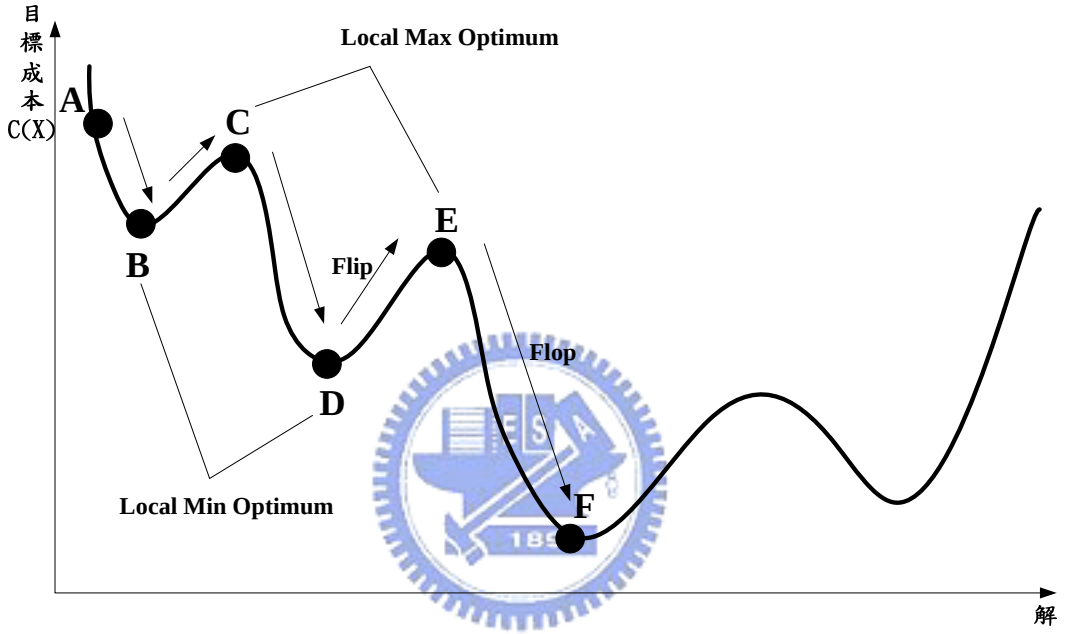


圖 2.10　兩極跳躍法之解題概念示意圖

對於一個TSP的問題而言：假設在求解時所面臨的解空間如圖中曲線所示，令起始解為A點，以成本最小化目標來改善(flop)，則可求得局部最小解(Local Min Optimum)B點。接著再以反向的改善(flip)方式來求解，即將目標式改為成本最大化，使目前解完全跳出區域解而找到反向的局部最大解(Local Max Optimum)C點。如此反覆的求解，使求解過程跳躍在兩極的區域解之間，最後將可以找到最佳解F點。

(3) 搜尋空間平滑法(Search Space Smoothing, SSS)[19]

搜尋空間平滑法(SSS)是由 Gu & Huang 於 1994 年所提出，SSS 法的基本概念是先透過一個平滑函數的轉換機制，將原本高低起伏的解集合空間使其平滑，使一部分局部最佳解能消失，期望在搜尋過程中不會有落入局部最佳解而不能跳脫的困擾。SSS 法的執行關鍵在於平滑函數與平滑因子遞減數列之機制設計。SSS

法解題概念可由圖 2.11 進行說明。

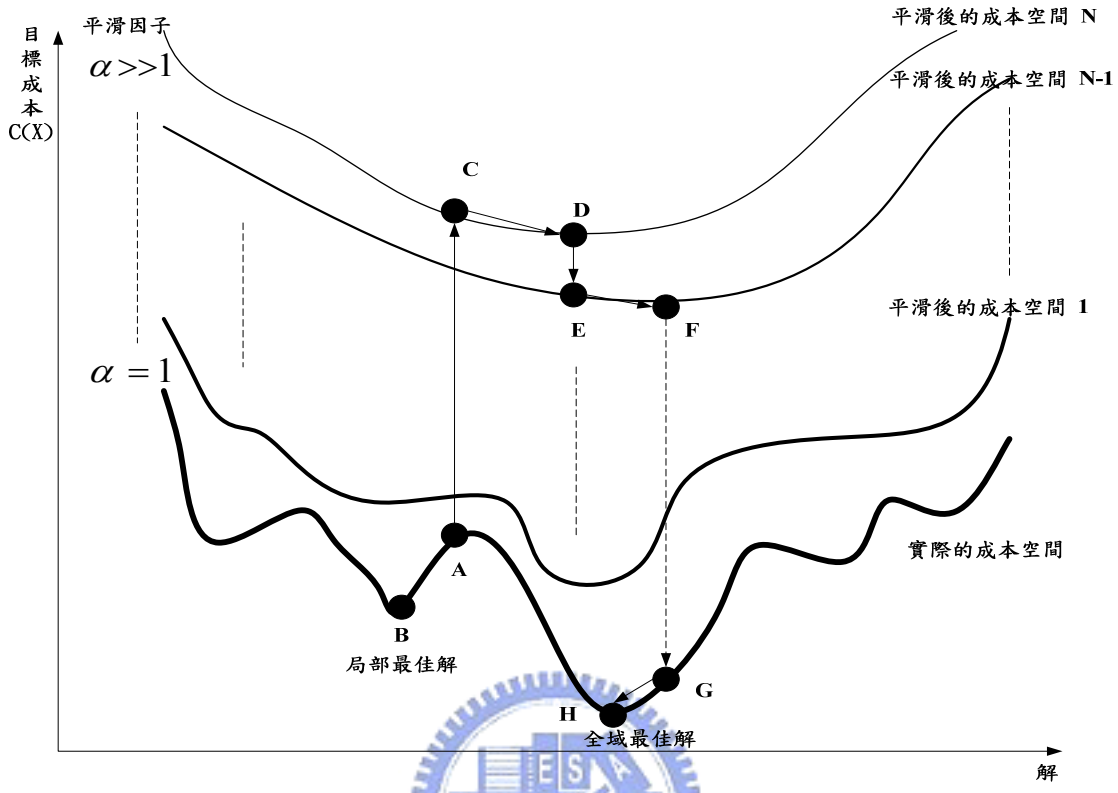


圖 2.11　搜尋空間平滑法之解題概念示意圖

對於一個 TSP 的問題而言：假設 A 點為實際成本空間上所求得的起始解，最後一定會陷入局部最佳解 B 點，若經過平滑因子為$\alpha >> 1$的平滑函數來轉換成一個新的平滑成本空間 N，將 B 點投射到平滑成本空間 N 之 C 點，再於此平滑空間上繼續搜尋可達平滑成本空間 N 的局部最佳解 D 點。若縮小平滑因子可產生另一個平滑成本空間 N-1，接著將 D 點投射到此平滑成本空間上之 E 點，並繼續搜尋可達平滑成本空間 N-1 的局部最佳解 F 點，如此重覆縮小α值及投上述步驟直到$\alpha = 1$，此時的平滑成本空間將收斂成實際成本空間；將上一個平滑成本空間所求得的局部最佳解投射到實際成本空間之 G 點上，再執行搜尋動作，便可能找到全域最佳解之 H 點。

NM 法、FF 法和 SSS 法等三種擾動型演算法的執行機制如圖 2.8 所示[3]，其中$C(X)$為原始成本空間之目標成本值，$C_p(X)$為擾動成本空間之目標成本值，$C_p^{\alpha}(X)$為第α次平滑成本空間之目標成本值。擾動型演算法的三種方法應用於求解 TSP 問題時，其控制參數、接受法則、收斂法則及停止法則分別說明如表 2.2，其中c_{ij}為各節線之實際成本值，c_{ij}^{p}為各節線之擾動成本值，$c_{ij}(\max)$為實際成本最大項之值，$c_{ij}^{s}(\alpha)$為各節線之平滑成本值，$c_{ij}(n)$為實際成本之正規

化值 $(0 \le c_{ij}(n) \le 1)$ ，$c(a)$ 為 $c_{ij}(n)$ 之平均值。

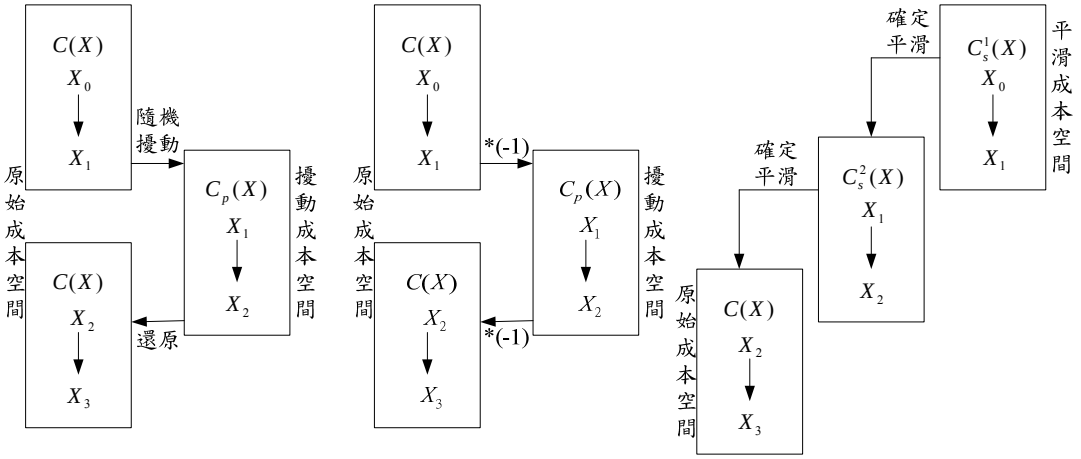


圖 2.12　MN、FF 與 SSS 執行機制示意圖

表 2.3　NM、FF 與 SSS 三種方法比較

方法	NM	FF	SSS
控制參數	擾幅：$R \in [R_{\max}, R_{\min}]$ 次數：$(N_k \times N_i)$ 機率值 $(-1 \le r \le 1)$	次數 (K)	平滑因子 $(\alpha \ge 1)$ 次數 (K)
擾動法則	機率性擾動： $c_{ij}^p = c_{ij} + (r \times R \times c_{ij}(\max)$	確定性擾動： $c_{ij}^p = -c_{ij}$	確定性平滑： $c_{ij}^s(\alpha) = \begin{cases} c(a) + (c_{ij}(n) - c(a))^\alpha \\ c(a) - (c(a) - c_{ij}(n))^\alpha \end{cases}$
接受法則	$C(X_j) < C(X_i)$ 或 $C_p(X_j) < C_p(X_i)$	$C(X_j) < C(X_i)$ 或 $C_p(X_j) < C_p(X_i)$	$C_s^\alpha(X_j) < C_s^\alpha(X_i)$
收斂法則	擾幅自 R_x 遞減至 R_n	無	α 遞減至1
停止法則	完成 $N_k \times N_i$ 次迴圈	完成 K 次迴圈	完成 K 次迴圈

註：(1) 資料來源：[3]

2.5.3　包容性深廣搜尋法(Generic Intensification and Diversification Search, GIDS)

卓裕仁(2001)[3]提出 GIDS 方法，結合多種巨集啟發式方法的特性與優點，將接受劣解、變換鄰域、擾動成本與多重起點等巨集策略融合在深度搜尋與廣度搜尋的概念中，發展出一套「包容性深廣度搜尋(Generic Intensification and

Diversification Search, GIDS)」的巨集啟發式方法。GIDS 法共包含：(1)多起始解構建(Multiple Initialization Constructor, MIC)、(2)深度化包容搜尋(Generic Search for Intensification, GSI) 與 (3) 廣度化擾動搜尋 (Perturbation Search for Diversification, PSD)三個策略群組。整套 GIDS 法係以傳統鄰域搜尋為實際執行求解之工具，以深度搜尋之 GSI 群組為核心，再搭配廣度搜尋之 PSD 與 MIC 群組。此外，並設計了五種模組來執行 GIDS 之策略群組：再 MIC 群組構建有加權起始(Weighted Initialization, WI)模組與鄰域搜尋(Neighborhood Search, NS)模組；在 GSI 群組設計有 G1 與 G2 兩種包容搜尋(Generic Search)模組；在 PSD 群組中則購建有成本擾動(Cost Perturbation, CP)模組。GIDS 之解題概念如圖 2.13 所示。

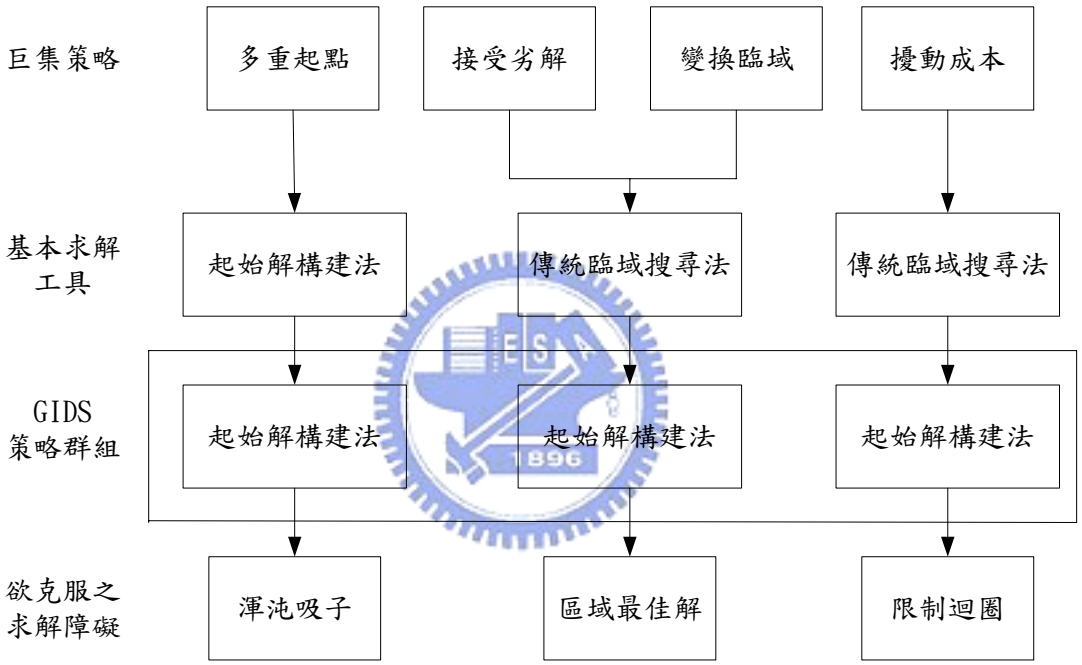


圖 2.13　包容性深廣搜尋法之解題概念示意圖

2.6 TSP 測試例題之蒐集與最佳解結果整理

TSP 早在 1934 年被提出後，後續被廣泛的引用研究，此種問題型態所發展出來的測試例題的數量非常多，由德國海德堡大學的 Gerhard Reinelt 教授為首的研究群，專門研究組合最佳化問題，此研究團隊彙整各種不同型態問題，建立標竿測試例題及當前所知的最佳結果的題庫，隨著網際網路的發達，此研究團隊將此題庫連結上網際網路，作為全世界從事此方面研究的學者一個共同比較的基準和相互討論研究的地方。

Gerhard Reinelt[30]在 1991 年發表文章成立 TSPLIB，並將所有蒐集到的所有相關資源和測試例題題庫，以網頁方式免費提供學術性研究使用，主要的包含了幾個和 TSP 相關的議題，像是對稱性 TSP、漢米爾頓迴圈(Hamiltonnian Cycle)問題、非對稱性 TSP、優先順序的 TSP(Sequential Ordering Problem)和容量限制的 VRP 問題等等，並且定期由 Gerhard Reinel 領導的團隊予以更新，目前最近更新的日期為 30/8/2005。在 TSPLIB 下載的方式細節如下：

以 WWW Server 方式下載

(1). 利用 Internet Explorer 瀏覽軟體連接到 TSPLIB 的位址上，http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/。

(2). 在網頁的連結文字 TSP data 上以滑鼠游標擊點，即自動進入測試例題 Index of /groups/comopt/software/TSPLIB95/tsp 的存取畫面，http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/。

(3). 在所要擷取的例題上以滑鼠游標擊點，再指定所要存取的個人電腦位置路徑，即可將測試例題存入所指定硬碟位置中。

(4) 回到 TSPLIB 的進入畫面，在網頁的連結文字 best know solution for symmetric TSPs 上以滑鼠游標擊點，即自動進入測試例題目前所知最佳解畫面上，http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/STSP.html

本研究主要所要測試的例題來源是擷取自 TSPLIB，TSPLIB 題庫共有 112 題對稱性(Symmetric)TSP 例題，本研究挑選 16 個例題進行測試和分析，所挑選的原則是酌以往文獻的測試報告，儘量選擇使用率較高的測試例題[6][9][13][30]。所選擇的例題名稱、題目規模、資料型態和當前文獻研究所發現

的已知最佳解成本如表 2.4 所示。此外，不同資料型態在轉成成本計算時，會因為電腦在實數歸整(round-off)而產生誤差，可能有相同的路徑結果，但卻有不同數值解，本研究根據 Gerhard Reinelt 在文章中所採取實數歸整的方式，將成本資料轉換成整數數值，最後輸出的總成本解也是整數，可以避免實數歸整計算誤差累積所造成的偏誤，同時避免造成比較上的困擾，讓求出的總成本解能有相同的比較基準。

表 2.4　本研究使用之 16 個 TSP 測試例題

題號	例題名稱	題目規模*	資料型態**	已知最佳解成本
1	gr24	24	Matrix	1272
2	swiss42	42	Matrix	1273
3	gr48	48	Matrix	5046
4	hk48	48	Matrix	11461
5	eil51	51	EUC_2D	426
6	brazil58	58	Matrix	25395
7	st70	70	EUC_2D	675
8	pr76	76	EUC_2D	108259
9	kroA100	100	EUC_2D	21282
10	kroC100	100	EUC_2D	20749
11	lin105	105	EUC_2D	14379
12	gr120	120	Matrix	6942
13	u159	159	EUC_2D	42080
14	ts225	225	EUC_2D	126643
15	lin318	318	EUC_2D	42029
16	pcb442	442	EUC_2D	50778

註：(1) * 題目規模係指網路的節點數目

(2) ** Matrix 為成本矩陣，EUC_2D 為平面空間座標

(3) 資料來源：[30]

2.7 名詞定義

本研究是針對啟發式解法做探討，因為牽涉到 TSP 測試例題的結果並不一定有絕對最佳解(Exact Solution)，評估時多以現有文獻最佳解(Best Known Solution)為基準。為避免本研究後續說明上名詞的混淆，本小節參考文獻[5] [6]做出以下的名詞定義：

(1) 絕對最佳解(exact solution)：理論上真正存在而且使目標值為最小的解與成本。

(2) 已知最佳解(best known solution)：目前已發表的文獻上，所能找到最接近絕對最佳解的解與成本。

(3) 最佳尋獲解(best found solution)：指本研究過程中，所能找到最接近絕對最佳解與成本。

(4) 當前解(current solution)：演算法在搜尋的過程中，最近一回合的結果就是整個程序至該步驟時的當前解；當前解會不斷更新，直到演算法停止為止。

(5) 暫優解(incumbent solution)：演算法在搜尋最佳解的過程中，曾經找到的解中，最接近絕對最佳解的解與成本。

(6) 最終解(final solution)：演算法完成後，整個程序得到的最好結果；通常是演算法結束時的暫優解。

(7) 成本誤差：對某例題之單次測試而言，該測試之最終解成本與該例題之已知最佳解成本的誤差百分比。計算公式如下：

$$\frac{\text{最終解成本} - \text{已知最佳解成本}}{\text{已知最佳解成本}} \times 100\%$$

(8) 平均誤差：對某例題而言，在某組參數組合或執行次數之測試下，多個成本誤差的平均值；或某組參數組合下，所有例題成本誤差的平均值。

(9) 總平均誤差：對整組測試例題或整組參數之平均誤差的總平均值。

第三章 DNM 求解執行架構之建立

3.1 核心方法執行架構之建立

本研究執行架構是先採用傳統啟發式解法，即採用路線構建法來求得起始解，再以交換型鄰域搜尋法來改善路線，當傳統交換型鄰域搜尋方法落入區域最佳解中，可利用本研究所改良的智慧型演算法和交換型鄰域搜尋方法之結合，來跳脫區域最佳解。所以綜合來說，交換型鄰域搜尋法是整個研究方法的核心方法，對本研究的解題績效有重大的影響力，所以3.1節將詳細說明交換型鄰域搜尋法執行架構。

3.1.1 起始解構建法

在執行交換型鄰域搜尋法之前，必須先有一個起始解，然後再以此起始解進行交換改善。根據文獻的建議，TSP 可採用多重起始解，不同的起始解可以達到不同的效果，以避免搜尋過程侷限在某個特定範圍裡，而有效率的起始解構建方法相當多，例如 2.4.1 節中所述：最近插入法(NI)、最遠插入法(FI)及最近鄰點法(NN)等等。本研究主要不在於探討起始解之構建方法的優劣，所以採用楊智凱(1995)[8]之研究結果所提之建議，以最遠插入法當成起始解之構建方法，期望能得到較好的結果。

3.1.2 鄰域搜尋法 (Neighborhood Search Method)

本研究採用鄰域搜尋法中的交換型啟發式為核心方法，即在當前解中利用特定的交換程序產生鄰域解，在交換過程中所產生的鄰域解中，符合接受法則就入選考慮的方案，利用某個選擇策略，在入選的方案中選取一個方案，更新當前解和暫優解，重複此搜尋動作，直到落入區域最佳解後便停止。交換型鄰域搜尋法的執行架構如圖 3.1 所示，步驟分述如表 3.1：

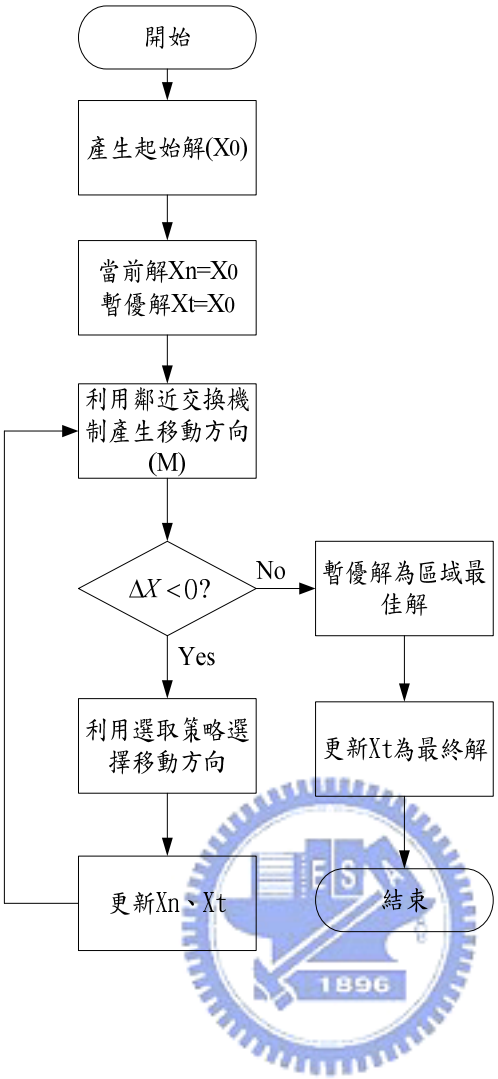


圖 3.1　交換型鄰域搜尋法的執行架構

表 3.1　交換型鄰域搜尋法的基本求解步驟

步驟一：起始步驟

構建可行的起始解(X_0)，並令：

當前解≔起始解，即 $X_n := X_0$；

暫優解≔起始解，即 $X_t := X_0$。

步驟二：區域最佳解

M 為所有移動方向的集合，ΔX 為可改善目標成本值的方向，ΔX 屬於 M，若 ΔX 中有可以改善當前解 X_n 的目標成本值，即執行步驟三；若 ΔX 中沒有可以改善當前解 X_n 的目標成本值，X_t 為最終解。

步驟三：移動

若 ΔX 可以改善當前解 X_n 的目標成本值，根據選取策略來選擇 ΔX 中的某一方案。

步驟四：更新

更新當前解和暫優解為，$X_n = X_n + \Delta X$、$X_t = X_n$，回到步驟二。

由上面的步驟流程可歸納出幾個特點：

(1) 產生鄰域移動方向的機制

所謂鄰域移動方向即是利用特定的鄰域交換機制所產生新的鄰解和當前解的差異，利用特定的鄰域搜尋機制可以形成多個鄰域移動方向，進而形成一個集合。不同產生鄰域移動方向的機制會產生不同的鄰域移動方向，例如 2.4.2 節中提到 K-Opt 交換法、Or-Opt 交換法及 Lin-Kernighan 交換法等等，其交換機制不同所產生的鄰域移動方向也會不同。

(2) 接受法則

鄰域移動方向有很多，必須透過接受法則來篩選對目標值有改善之移動方向。鄰域搜尋法採用嚴格的接受法則，即對目標值有改善才入選；某些巨集演算法採用寬鬆的接受法則，即允許接受對目標值沒有改善的方向，例如 2.5.1 節提到的門檻型演算法就是採用寬鬆的接受法則。

(3) 選取策略

若利用接受法則可篩選多個對目標值有改善之移動方向，必須透過選取策略

來選擇移動方向，選取策略一般來說可分為三種：

(a) 首先改善法(Method of First Improvement)

按順序進行交換，一旦發現目標值有所改善時，即刻進行交換，並且從頭再重新進行交換。

(b) 最佳改善法(Method of Best Improvement)

先將所有可能改善的交換都紀錄下來，然後再對其中最佳改善來進行實際交換，然後在重新開始交換。

(c) Heider法(Heider's Method, Heider[20])

以固定順序進行交換，並且對有改善的部分即刻進行交換，繼續進行此次的交換動作，不必重新開始。

不管採用何種產生鄰域移動方向的機制，在一個完整的交換週期中所採取的選取策略均沒有可選取的方案，可使得目標成本值有改善的情況發生，則停止。

本研究以鄰域搜尋法中的交換型啟發式為核心方法，其中發展較成熟的方法有K-Opt、Or-Opt和節點交換法(City Swap)等。

3.2 確定性成本擾動法(Deterministic Noising Method)的觀念探討

雖然本研究考慮採取 K-Opt、Or-Opt 和節點交換法(City Swap)等方法當成 TSP 的核心交換法，然而鄰域搜尋這一類相關的方法，皆會有落入區域最佳解的情況，所以必須搭配巨集啟發式解法以跳脫區域最佳解，期望能找到更優的解。

由 2.5.2 節可知，成本擾動法(NM)對成本空間的噪音擾動並不是固定單一值的擾動，而是分別對每一個節線成本值以隨機產生的 r 值來擾動，擾動後的成本空間和原始成本空間是完全不同的空間，其可能的缺點是無法重現，亦即在相同的參數下，兩次執行的結果不一定會相同。

再者，觀察鄰域搜尋法的解題過程中，節點常會考慮與節點本身較近的節點相連接，但對整個求解目標成本值來講，節點可以考慮與節點本身距離較遠的節點相連接，反而對整體目標成本值有較大的貢獻。以國際例題 st70 為例，在執行完鄰域搜尋法時會陷入區域最佳解，如圖 3.2(a)所示，節點 6 和節點 41 分別會選擇較近的節點 18 和節點 42 連結，再觀察已知最佳解，如圖 3.2(b)所示，節點 6 和節點 41 分別與距離比節點 18 和節點 42 遠的節點 53 和節點 43 連結，則有機會找到較佳的解。圖 3.3 所示，若用單一節點 A 來看，將與 A 相鄰 C_{α}^{i} 長度範圍內的節線增加特定長度 H_{β}^{i}，則節點 A 比較有機會和距離較遠的節點 B 或 C 連結，藉由此種成本的干擾，使得原本落入區域最佳解的搜尋，能有機會跳脫。

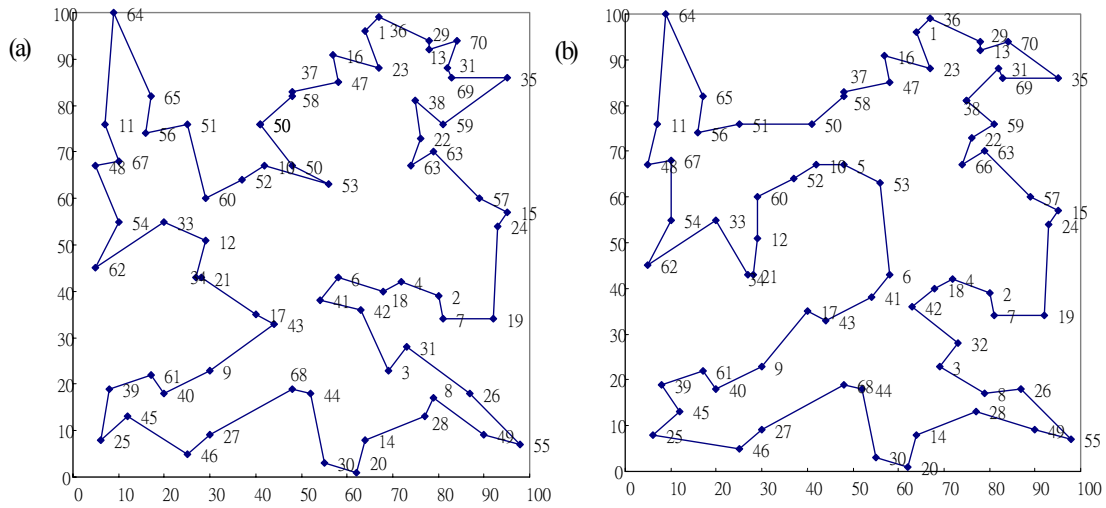


圖 3.2　例題 st70 區域最佳解與全域最佳解的路線圖

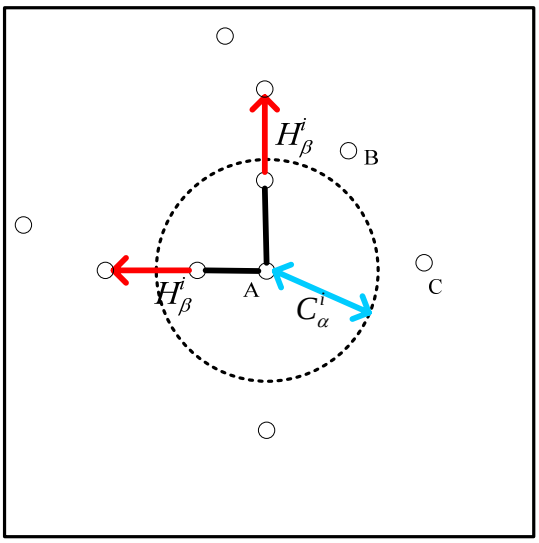


圖 3.3　節點擾動示意圖

所以本研究除了考慮 NM 法求解時最終結果無法重現的缺點，亦考慮節點的靈活性使其有機會連結附近的節點，進而修改 NM 法的擾動成本機制，提出確定性的成本擾動法。確定性成本擾動法的觀念是考慮經過核心搜尋法後會陷入區域最佳解，對於某個比例下的線段成本做擾動；而擾動的方式是在某個比例下的線段成本加上一個固定成本，使得節點有機會考慮附近的節點，藉以跳脫區域最佳解，加入的固定成本必需有一個適當的大小，否則加入的固定成本太大對於成本擾動過於劇烈，徒增計算時間上的負擔，且不容易收斂到較佳的解上；加入的固定成本太小對於成本的擾亂沒有影響，無法利用跳脫區域最佳解的束縛。

綜合而言，確定性成本擾動法(DNM)和2.5.2節中所提到NM法的解題觀念均相同，均是擾動成本以跳出區域最佳解，所不同在於擾動的方式。DNM應用在求解TSP的觀念在於：對於已經陷入區域最佳解的情況，使用增加某部份線段成本長度的方式來擾動，然後再對擾動後的問題重新求解，期望藉由成本的擾動使其跳脫區域最佳解，圖2.9亦可用來說明DNM法跳出區域最佳解的概念。

3.3 DNM 之求解步驟、執行架構與細部設計

本節將針對DNM法的基本求解步驟、執行架構與控制參數之設計加以說明。

3.3.1 DNM 之基本求解步驟

本研究提出 DNM 的求解步驟如表 3.2，從表 3.2 的基本求解步驟可知：DNM 在應用時必須先有一個起始解，並執行核心交換法，可預期此求解過程會陷入區域最佳解；依據最大成本擾動比率值C_α^i、成本擾動比率值下降型態、成本擾動比率長度K、最大增加成本比率值H_β^i、增加成本比率值下降型態和增加成本比率長度L等參數，設計k(外)和l(內)兩個迴圈，在此兩個迴圈之下，利用成本擾動比率值和增加成本比率值來擾動成本。首先在擾動後的成本空間進行核心交換法的鄰域搜尋，可找到一個區域最佳解X'，接著以此區域最佳解X'為起點，回到原始成本空間再做一次核心交換法的鄰域搜尋，可找到另一個區域最佳解X''，此區域最佳解為當前解$X_n = X''$。如果改善後的當前解成本$C(X_n)$比暫優解成本$C(X_t)$更佳$(C(X_n) < C(X_t))$，則更新暫優解；否則降低參數值，再繼續進行成本擾動和核心交換法的鄰域搜尋改善。直到l迴圈執行完畢後$(H_\beta^i = 0)$，再繼續執行下一個k迴圈內的成本擾動和核心交換法的鄰域搜尋改善。若k迴圈執行完畢後$(C_\alpha^i = 0)$，則停止。最終整個過程的最佳解為最後所記錄的暫優解。

表 3.2　DNM 演算法的基本求解步驟

參數

成本擾動比率值 C_α^i 、增加成本比率值 H_β^i(實數)；

成本擾動比率值長度 K 、增加成本比率值長度 L(整數)。

變數

當前解變數 X_n 、暫優解變數 X_t ；

當前目標成本值 $C(X_n)$ 、暫優目標成本值 $C(X_t)$(實數)。

步驟一：設定參數

設定：最大成本擾動比率值 (C_α^i) 、最大增加成本比率值 (H_β^i) ；

設定：成本擾動比率值長度 K 、成本擾動比率值下降型態；

設定：增加成本比率值長度 L 、增加成本比率值下降型態。

步驟二：求起始解

構建可行的起始解 (X_0) ，並令：

現有解≔起始解，即 $X_n := X_0$ ；

暫優解≔起始解，即 $X_t := X_0$ 。

步驟三：執行

執行鄰域交換法，當鄰域交換法陷入區域最佳解；

執行迴圈： $k = 1 to K$ ，

執行迴圈： $l = 1 to L$ ，

對原始成本空間 $C(X)$ 擾動，得到新的成本空間 $C^p(X)$ ；

在 $C^p(X)$ 上對 X_n 進行鄰域搜尋，得到區域最佳解 $X^{'}$ ，

並令：現有解=區域最佳解，即 $X_n = X^{'}$ ；

在 $C(X)$ 上對 X_n 進行鄰域搜尋，得到區域最佳解 $X^{''}$ ，

並令：現有解=區域最佳解，即 $X_n = X^{''}$ ；

依據增加成本比率下降型態和長度，下降所增加成本比率 (H_β^i) ；

完成迴圈 l ；

若 $C(X_n) < C(X_t)$ ，則更新暫優解，即 $X_t = X_n$ ；

依據成本擾動比率下降型態和長度，下降成本擾動比率 (C_α^i) ；

完成迴圈 k ；

停止，輸出最終結果： X_t 與 $C(X_t)$ 。

3.3.2　DNM 之執行架構

從表 3.3 的基本求解步驟可知：DNM 法中有四個控制參數，分別為成本擾動比率值C_α^i、增加成本比率值H_β^i、成本擾動比率值長度K、增加成本比率值長度L。C_α^i與H_β^i用以控制成本擾動的方式和大小，本研究擾動成本方式的說明如下：

(1) 標準化：將所有成本矩陣中的節線成本c_{ij}予以標準化，得到$\hat{c}_{ij}$。

$$\hat{c}_{ij} = \frac{c_{ij}}{c_{ij}(\max)}; \quad 0 \le \hat{c}_{ij} \le 1 \qquad (3\text{-}12)$$

其中，$\hat{c}_{ij}$：為標準化後成本；

$c_{ij}(\max)$：為原成本矩陣中之最大數值。

(2) 成本擾動公式：

$$\hat{c}_{ij}^p = \begin{cases} \hat{c}_{ij} & if\ \hat{c}_{ij} > C_\alpha^i \\ \hat{c}_{ij} + H_\beta^i & if\ \hat{c}_{ij} \le C_\alpha^i \end{cases} \qquad (3\text{-}13)$$

其中，$\hat{c}_{ij}^p$：為標準化後再經過擾動的成本；

C_α^i：最大成本擾動比率值；

H_β^i：最大增加成本比率值。

C_α^i所表示第i題為所要測試的題目中，在α機率下的數值；以 st70 例題為例，$\alpha = 10\%$的數值為 0.156，即$C_{0.1}^{st70} = 0.156$。同理，H_β^i第i題為所要測試的題目中，在β的機率下的數值；以 st70 例題為例，$\beta = 10\%$的數值為 0.156，即$H_{0.1}^{st70} = 0.156$。在每個不同例題中α或β的機率值相同，所代表的意義都是在所有節線中的具有相同的比例，但會因為節線長度的分佈不同，C_α^i和H_β^i有不同的數值。

由(3-12)與(3-13)可知，本研究所用的擾動成本的方式並非全面性的擾動，僅在於成本矩陣某特定比例以下的成本做擾動，而擾動的方式是採取增加固定的成本。圖 3.5 表示若參數為$C_{0.3}^i$與$H_{0.1}^i$時的擾動後成本示意圖。

K與L用以控制k與l兩個迴圈的執行次數，並且與下降型態一起控制C_α^i與

H_β^i下降後數值。以(3-14)與(3-15)說明直線下降型態在每一迴圈下降後的數值。

$$\hat{C}_\alpha^i = C_\alpha^i(1-\frac{k}{K}) \quad k = 0,1,2.....K \qquad (3\text{-}14)$$

$$\hat{H}_\beta^i = H_\beta^i(1-\frac{l}{L}) \quad l = 0,1,2.....L \qquad (3\text{-}15)$$

本研究並所建立之確定性成本擾動法的執行架構如圖 3.4 所示。

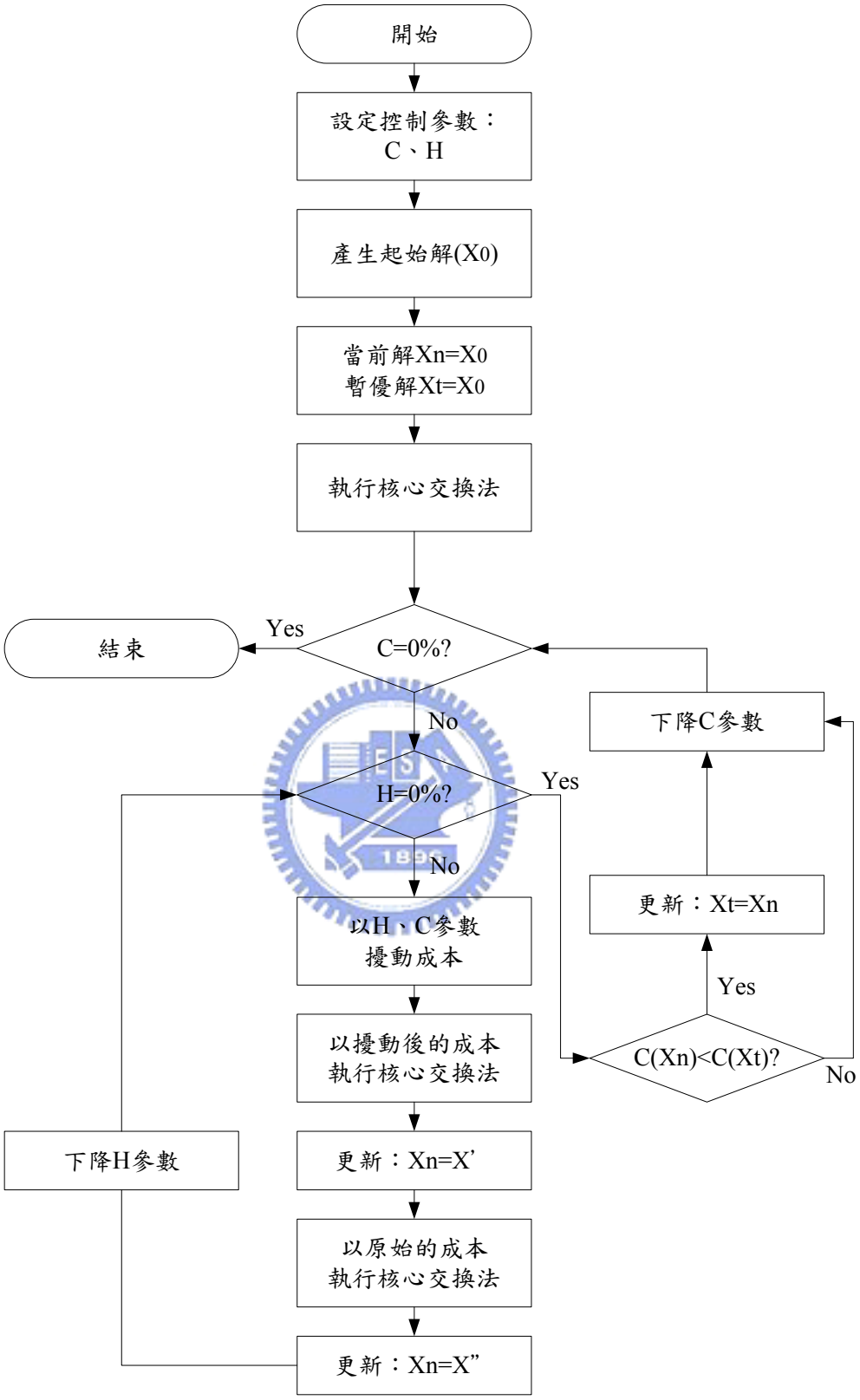


圖 3.4　確定性成本擾動法之執行架構

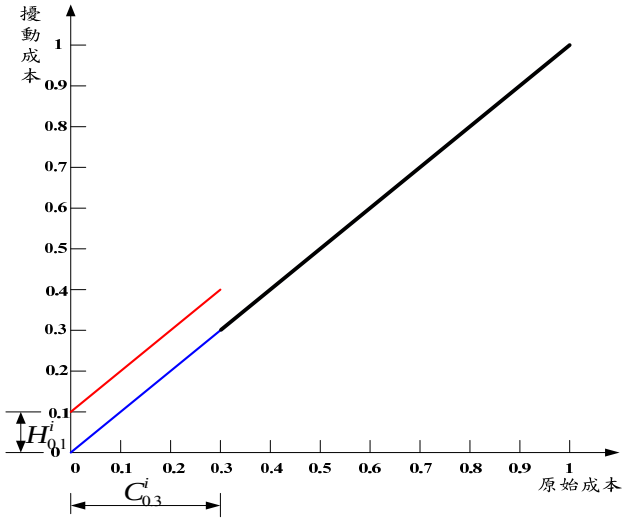


圖 3.5　成本擾動公式的擾動結果示意圖

3.3.3　DNM 之細部設計

由執行架構可知，DNM 的執行績效會受到各個組件項目的影響，其組件包括：起始解解法、核心交換方法、C^i_α、H^i_β、K、L、C^i_α 的下降型態和 H^i_β 的下降型態等等。以下將討論之後本研究在執行時所可能採用的控制參數和細部設計。

(1)　起始解構建法

本研究將採用在 3.1.1 節所提到的最遠插入法(NI)作為 DNM 法的起始解。

(2)　核心方法

本研究考慮K-Opt、Or-Opt和節點交換法(City Swap)等方法當成核心交換方法。但由於3-Opt以上的節線交換很複雜，且計算時間太長，所以本研究暫時不考慮3-Opt以上的節線交換法。另外Gu & Huang(1994)曾對這三種方法做過實驗發現，節點交換法(City Swap)有較差的績效，所以本研究排除此種方法，僅考慮2-Opt和Or-Opt為核心交換法。考慮DNM法需要在擾動後的成本空間和原始成本空間上搜尋新解，在不同成本空間使用不同的搜尋方法以避免搜尋過程走回頭路，落入原先的區域最佳解；另外為了節省搜尋時間，在擾動後的成本空間的搜尋方法將採用Heider法[20]的選取策略且只執行一次完整的交換機制即停止，而在原始成本空間則採用最佳改善法(Method of Best Improvement)。

所以本研究將進行兩種核心方法的實驗，即擾動後的成本空間上採用 Or-H，原始成本空間上採用 2-Opt；和在擾動後的成本空間上採用 2-H，原始成本空間上採用 Or-Opt。

(3) C_α^i 與 H_β^i

首先將整個成本矩陣中的節線成本標準化，使所有長度的介於 0 到 1 之間。本研究初步測試的 C_α^i 和 H_β^i 中之 α 和 β 比率介於整體節線的 5%到 100%之間，以 5%為一個增加單位。

(4) C_α^i 與 H_β^i 的下降型態

C_α^i 與 H_β^i 的下降型態上，本研究初步以直線型的下降型態為主。

(5) K 與 L

K 與 L 的相乘的數值越大，表示執行的次數越多，所花費的時間也會越長。本研究初步實驗 $K \times L = 30$ 、45 和 60 的原則，因此 K 與 L 的組合可以很多。在執行時的不同組合代表不同的意義，若以組合 (15×3) 來說明，表示內迴圈的 H_β^i 值以直線型態 3 次下降完畢，再開始下降外迴圈的 C_α^i 值，每下降一次外迴圈，則重複進行內迴圈的下降動作，如此外迴圈分為 15 次的直線型態下降完畢，(15×3) 的總次數為 45 次。

將上述的 DNM 執行架構之組件項目和測試範圍整理如表 3.3，並分別在下一章節進行實驗設計與參數測試。

表 3.3　DNM 演算法的組件項目與測試範圍

組件項目	測試範圍
起始構建法	最遠插入法(FI)
擾動成本之核心交換法	Or-H 或(2-H)
原始成本之核心交換法	2-Opt 或(Or-Opt)
成本擾動比率值(C_α^i)	α=5%~100%
C_α^i 的下降型態	直線下降型態
增加成本比率值(H_β^i)	β=5%~100%
H_β^i 的下降型態	直線下降型態
K 與 L	$K \times L = 30$ 、45 和 60

第四章 DNM 應用於 TSP 之實驗設計與測試

本章將應用DNM法來求解TSP問題，並以2.6節的16個TSP測試例題，進行DNM法的組件搭配測試和參數測試。本研究的測試是以軟體Mathematica5.0和Visual C++撰寫解題程式，並在個人電腦Pentium(R)4 3.0GHz、1Gb RAM之WindowsXP環境下執行。

4.1 DNM 法核心交換法執行方式之測試

4.1.1 傳統啟發式解法

本研究將以 2.6 節中所蒐集到的 16 個例題進行測試，首先測試 3.3.3 節所提之傳統啟發式解法以作為後續比較的基準，執行方式為「最遠插入法+2-Opt+Or-Opt」和「最遠插入法+Or-Opt+2-Opt」。其測試結果如表 4.1。

表 4.1 傳統啟發式解法之測試結果

題號	例題名稱	已知最佳解	FI+2-Opt+Or-Opt error(%)	FI+Or-Opt+2-Opt error(%)
1	gr24	1272	1.10	2.99
2	swiss42	1273	2.91	1.96
3	gr48	5046	2.38	2.38
4	hk48	11461	6.01	6.01
5	eil51	426	1.64	1.64
6	brazil58	25395	0.00	1.29
7	st70	675	5.33	5.33
8	pr76	108159	2.96	2.96
9	kroA100	21282	4.46	4.46
10	kroC100	20749	1.22	1.22
11	lin105	14379	3.87	3.87
12	gr120	6942	2.05	4.51
13	u159	42080	12.50	12.50
14	ts225	126643	7.33	7.33
15	lin318	42029	8.21	8.21
16	pcb442	50778	7.81	7.81
總平均誤差			4.36	4.66
16 個例題中得到已知最佳解之題目比例			1/16	0/16

在表 4.1 中，第四欄為用傳統方法「FI+2-Opt+Or-Opt」所得到的誤差百分比，其總平均誤差百分比為 4.36%；第五欄為用傳統方法「FI+Or-Opt+2-Opt」所得到的誤差百分比，其總平均誤差百分比為 4.66%。

4.1.2 DNM 法初步測試

此小節將測試 DNM 法的有效性，本研究將測試兩種不同的執行方式，第一種執行方式是先進行「FI+2-Opt」，接著在擾動後的成本空間上採用 2-Opt，原始成本空間上採用 2-Opt，為了後續說明的方便性，以代號「FI+2-Opt+DNM(2-Opt+2-Opt)」表示；另一種執行方式為先進行「FI+Or-Opt」，接著在擾動後的成本空間上採用 Or-Opt，原始成本空間上採用 Or-Opt，為了後續說明的方便性，以代號「FI+Or-Opt+DNM(Or-Opt+Or-Opt)」表示。參數上先固定為 $C_{0.1}^{i}$、$H_{0.1}^{i}$、$K \times L = 15 \times 3$ 和直線下降型態。其測試結果如表 4.2。在表 4.2 中，第四欄為「FI+2-Opt+DNM(2-Opt+2-Opt)」所得到各例題的誤差百分比，其總平均誤差百分比為 1.58%；第五欄為「FI+Or-Opt+DNM(Or-Opt+Or-Opt)」所得到各例題的誤差百分比，其總平均誤差百分比為 2.02%。比較 4.1.1 節傳統的啟發式解法，總平均誤差百分比分別從 4.36%和 4.66%下降到 1.58%和 2.02%，表示 DNM 法具有改善效果。

表 4.2　DNM 法初步測試結果

題號	例題名稱	已知最佳解	FI+2-Opt+ DNM(2-Opt+2-Opt) error(%)	FI+Or-Opt+ DNM(Or-Opt+Or-Opt) error(%)
1	gr24	1272	0.00	0.00
2	swiss42	1273	1.41	2.91
3	gr48	5046	0.63	0.00
4	hk48	11461	1.03	0.44
5	eil51	426	1.64	1.64
6	brazil58	25395	0.00	0.76
7	st70	675	1.63	0.74
8	pr76	108159	0.82	0.76
9	kroA100	21282	0.41	2.59
10	kroC100	20749	2.03	1.04
11	lin105	14379	0.15	1.32
12	gr120	6942	2.30	1.77
13	u159	42080	2.01	1.63
14	ts225	126643	0.86	4.76
15	lin318	42029	5.37	5.20
16	pcb442	50778	4.91	6.79
總平均誤差			1.58	2.02
16 個例題中得到已知最佳解之題目比例			2/16	2/16

4.1.3　修正 DNM 法

有鑑於 4.1.2 節擾動後成本空間和原始成本空間搜尋均採用相同的方式，有可能使搜尋過程走回頭路，落入原先的區域最佳解；另外為了節省搜尋時間，在選取策略上亦做以下修正，在擾動後的成本空間的搜尋方法將採用 Heider 法[20]的選取策略且只執行一次完整的交換機制即停止，簡稱為 2-H 和 Or-H，而在原始成本空間則依然採用最佳改善法。所以本研究將進行兩種核心方法的實驗，即擾動後的成本空間上採用 Or-H，原始成本空間上採用 2-Opt；和在擾動後的成本空間上採用 2-H，原始成本空間上採用 Or-Opt。以上兩種實驗測試，在參數上是先固定為 $C_{0.1}^{i}$、$H_{0.1}^{i}$、$K \times L = 15 \times 3$ 和直線下降型態。其測試結果如表 4.3。

表 4.3　修正 DNM 法之測試結果

題號	例題名稱	已知最佳解	FI+2-Opt+ DNM(Or-H+2-Opt) error(%)	FI+Or-Opt+ DNM(2-H+Or-Opt) error(%)
1	gr24	1272	0.00	0.00
2	swiss42	1273	0.00	0.00
3	gr48	5046	0.18	0.00
4	hk48	11461	0.41	0.08
5	eil51	426	1.41	1.64
6	brazil58	25395	0.00	0.02
7	st70	675	0.00	0.74
8	pr76	108159	0.00	0.07
9	kroA100	21282	0.33	0.79
10	kroC100	20749	1.74	0.50
11	lin105	14379	0.00	0.48
12	gr120	6942	1.18	1.74
13	u159	42080	1.13	1.66
14	ts225	126643	0.00	2.00
15	lin318	42029	2.26	4.08
16	pcb442	50778	4.72	5.10
總平均誤差			0.84	1.18
16 個例題中得到已知最佳解之題目比例			7/16	3/16

表 4.2 中，第四欄為「FI+2-Opt+DNM(Or-H+2-Opt)」所得到各題的誤差百分比，其總平均誤差百分比為 0.84%；第五欄為「FI+Or-Opt+DNM(2-H+Or-Opt)」所得到各題的誤差百分比，其總平均誤差百分比為 1.18%。比較 4.1.2 節未修正之 DNM 法，總平均誤差百分比分別從 1.58%和 2.02%下降到 0.84%和 1.18%，特別是「FI+2-Opt+DNM(Or-H+2-Opt)」的架構，可以在 16 個測試例題中找到 7 題已知最佳解，表示經過修正後的 DNM 法更具有改善效果，本研究後續將以「FI+2-Opt+DNM(Or-H+2-Opt)」的架構進行參數測試。

4.2 DNM 法之參數測試

本小節將針對DNM法的四個控制參數，分別為C_α^i、H_β^i、K和L，進行一連串的測試分析；其中可將C_α^i與H_β^i可視為一組參數，K與L則視為另一組參數。測試的例題仍然是2.6節所蒐集的16個TSP測試例題。

4.2.1 K與L參數組合之測試

K與L是控制DNM法測試長度的參數，即控制擾動的總次數。本研究初步設計以K乘L分別為總次數30、45及60次為原則，因此在總次數為30次原則下，可有下面六種組合，$(K\times L)=(15\times 2)$、(10×3)、(6×5)、(5×6)、(3×10)及(2×15)；在總次數為45次原則下，可有下面四種組合，$(K\times L)=(15\times 3)$、(9×5)、(5×9)及(3×15)；在總次數為60次原則下，可有下十種組合，$(K\times L)=(30\times 2)$、(20×3)、(15×4)、(12×5)、(10×6)、(6×10)、(5×12)、(4×15)、(3×20)及(2×30)。

在此先固定另一組參數$(C_{0.05}^i, H_{0.05}^i)$，以了解在不同擾動次數的組合方式對DNM法的解題績效影響。對於每個例題開始皆先以FI+2-Opt的方式先搜尋，並在擾動成本空間使用Or-H的方式進行鄰域搜尋，在原始成本空間使用2-Opt進行鄰域搜尋。測試結果如表4.4所示。

在表4.4中，第三、四欄為參數K與L之不同組合；第五、六欄為16個例題執行後，各題的平均誤差和標準差。由表4.4的結果可發現：不同的參數K與L組合，對執行結果有所差異性存在，如果參數組合為K較L長，其平均誤差會優於參數為L較K長的組合，例如$(K\times L)=(15\times 2)$的平均誤差為1.64%，而$(L\times K)=(2\times 15)$的平均誤差為1.99%。另外在總次數為30的所有組合，以組合編號2(10×3)表現較好；在總次數為45的所有組合，以組合編號1(15×3)表現較好；在總次數為60的所有組合，以組合編號4(12×5)表現較好。就總平均誤差而言，總次數為30的六種組合方式，其總平均誤差為1.66%；總次數為45的四種組合方式，其總平均誤差為1.64%；總次數為60的十種組合方式，其總平均誤差為1.20%。

表 4.4　DNM 法之 K 與 L 參數組合測試結果(Ⅰ)

固定參數($C^{i}_{0.05}, H^{i}_{0.05}$)

總次數	組合編號	K	L	平均誤差(%)	標準差
30	1	15	2	1.64	1.52
	2	10	3	**1.25**	1.32
	3	6	5	1.75	1.39
	4	5	6	1.57	1.24
	5	3	19	1.79	1.41
	6	2	15	1.99	1.69
	總平均誤差			1.66	
45	1	15	3	**1.14**	1.28
	2	9	5	1.40	1.33
	3	5	9	1.77	1.46
	4	3	15	1.51	1.54
	總平均誤差			1.46	
60	1	30	2	1.18	1.16
	2	20	3	1.06	1.05
	3	15	4	1.35	1.16
	4	12	5	**0.95**	1.19
	5	10	6	1.24	1.25
	6	6	10	1.13	1.33
	7	5	12	1.21	1.25
	8	4	15	1.55	1.52
	9	3	20	1.16	1.18
	10	2	30	1.18	1.37
	總平均誤差			1.20	

再以另一組參數$(C_{0.1}^{i}, H_{0.1}^{i})$對上述的所有組合再進行測試，以了解上述結果對另一組參數是否具有敏感性。測試結果如表4.5所示。

表 4.5　DNM 法之 K 與 L 參數組合測試結果(Ⅱ)

固定參數$(C_{0.1}^{i}, H_{0.1}^{i})$

總次數	組合編號	K	L	平均誤差(%)	標準差
30	1	15	2	1.17	1.25
	2	10	3	**0.89**	1.17
	3	6	5	1.22	1.42
	4	5	6	1.42	1.32
	5	3	19	1.82	1.54
	6	2	15	1.22	1.07
	總平均誤差			1.29	
45	1	15	3	**0.84**	1.27
	2	9	5	0.89	1.07
	3	5	9	0.85	1.18
	4	3	15	0.85	1.19
	總平均誤差			0.94	
60	1	30	2	0.90	1.13
	2	20	3	**0.81**	1.38
	3	15	4	0.97	1.09
	4	12	5	0.96	1.28
	5	10	6	0.95	1.01
	6	6	10	1.17	1.21
	7	5	12	1.13	1.13
	8	4	15	1.04	1.07
	9	3	20	1.20	1.35
	10	2	30	1.80	0.75
	總平均誤差			1.21	

由表4.5的結果可發現：在總次數為30的所有組合，以組合編號2 (10×3) 表現較好；在總次數為45的所有組合，以組合編號1 (15×3) 表現較好；在總次數為60的所有組合，以組合編號2 (20×3) 表現較好。就總平均誤差而言，總次數為30的六種組合方式，其總平均誤差為1.29%；總次數為45的四種組合方式，其總平均誤差為1.94%；總次數為60的十種組合方式，其總平均誤差為1.21%。

綜合以上結果可發現：在總次數為30的組合中，上述兩種情況下皆以組合編號2 (10×3) 有較好的結果，因此選擇 $(K\times L)=(10\times3)$ 做為測試另一組參數組合 (C_α^i, H_β^i) 的標準；在總次數為45的組合中，上述兩種情況下皆以組合編號1 (15×3) 有較好的結果，因此選擇 $(K\times L)=(15\times3)$ 做為測試另一組參數組合 (C_α^i, H_β^i) 的標準；在總次數為60的組合中，兩次所得到之較佳結果不一，推測可能是測試情況較少，故無法推論出較佳的組合，所以採取兩次之綜合較佳結果 $(K\times L)=(20\times3)$ 做為測試另一組參數組合 (C_α^i, H_β^i) 的標準。由上述兩種不同的參數組合 (C_α^i, H_β^i) 可發現：不同的 (C_α^i, H_β^i) 組合對結果平均誤差的差異性頗大，(C_α^i, H_β^i) 間的變化具有敏感性，顯示 (C_α^i, H_β^i) 參數組合測試之必要性。

4.2.2 總次數 30 次之 C_α 與 H_β 參數組合測試

接著本研究將對另一組參數進行測試，C_α^i 所代表的是例題 i 在所有節線成本中的 α 比率下節線成本的數值和 H_β^i 所代表的是例題 i 在 C_α^i 的數值內的所有節線成本所要增加的長度數值，H_β^i 的數值可包含所有節線成本中的 β 比率下節線成本，測試範圍初步訂為：$\alpha=5\sim100\%$，$\beta=5\sim100\%$，分別以 5%遞增的方式，所以共有 400 種參數組合。當總次數 30 時，K 與 L 的組合採用 $(K\times L)=(10\times3)$ 的組合方式；其他組件維持不變：對於每個例題開始皆先以 FI+2-Opt 的方式先搜尋，待落入區域最佳解後，在擾動成本下使用 Or-H 的方式進行鄰域搜尋，在原始成本下使用 2-Opt 進行鄰域搜尋。

表 4.6 為總次數 30 次擾動的 C_α 與 H_β 參數測試組合，橫向的項目為 C_α 的參數範圍，縱向的項目為 H_β 的參數範圍；表中每格數值表示 16 個例題在執行完所對應的參數後，16 個例題的平均誤差百分比(%)。

表 4.6　DNM 法在總次數 30 下 C_α 與 H_β 參數組合測試結果(Ⅰ)

總次數 30，固定參數 $(K \times L) = (10 \times 3)$

		(H_β)參數範圍																				
		5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%	
(C_α)參數範圍	5%	1.25	1.48	1.38	1.02	1.05	1.00	1.68	1.42	0.98	1.51	1.58	0.94	1.60	2.01	1.51	1.30	1.74	1.46	1.51	1.56	1.40
	10%	1.19	0.89	1.16	1.06	0.93	1.21	1.53	1.53	1.28	1.21	1.11	1.82	1.75	1.23	1.33	1.32	1.50	1.56	1.28	1.51	1.32
	15%	1.06	1.18	1.21	1.27	1.38	1.27	1.38	1.27	1.26	1.47	1.50	1.10	1.62	1.33	1.14	1.63	1.41	1.47	1.13	1.62	1.33
	20%	1.15	1.04	0.99	1.21	0.95	1.27	1.30	1.21	1.28	1.01	1.43	1.37	1.53	1.67	1.42	1.51	1.05	1.26	1.37	1.23	1.26
	25%	0.85	1.15	0.86	1.45	0.97	1.16	1.52	1.47	1.39	1.23	1.45	1.32	1.49	2.02	1.60	1.24	1.49	1.45	1.37	1.59	1.35
	30%	1.23	**0.81**	**0.99**	**1.10**	**0.93**	**1.11**	0.97	1.13	0.94	1.43	1.33	1.33	1.60	1.32	1.36	1.46	1.65	1.68	1.31	1.26	1.25
	35%	1.12	**1.26**	**0.82**	**0.93**	**0.91**	**1.30**	1.10	1.25	1.12	1.08	1.23	1.19	1.48	1.30	1.76	1.23	1.04	1.19	1.18	1.38	1.19
	40%	1.52	**0.86**	**0.84**	**1.12**	**1.02**	**0.74**	1.32	1.36	1.39	1.09	1.55	1.41	1.51	1.26	1.27	1.61	1.17	1.69	1.62	1.37	1.29
	45%	1.46	**0.89**	**1.11**	**1.14**	**0.91**	**1.21**	1.24	0.98	1.57	1.30	1.16	1.44	1.28	1.35	1.09	1.35	1.14	1.55	1.58	1.78	1.28
	50%	1.26	**1.04**	**1.05**	**1.01**	**0.95**	**1.08**	0.86	0.92	1.15	1.37	1.49	1.19	1.64	1.38	1.27	1.05	1.13	1.47	1.32	1.92	1.23
	55%	1.23	1.19	0.95	1.39	1.12	1.15	1.09	0.94	1.60	1.41	1.81	0.91	1.01	1.39	1.42	1.37	1.59	1.41	1.37	1.40	1.29
	60%	1.30	1.11	1.20	1.08	1.21	0.82	1.20	0.91	1.34	1.10	1.43	1.10	1.22	1.37	1.26	1.49	1.45	1.44	1.37	1.54	1.25
	65%	1.55	1.41	1.29	1.29	1.32	0.86	0.87	1.15	1.34	1.32	1.30	1.21	1.33	1.22	1.56	1.28	1.59	1.61	1.35	1.37	1.31
	70%	1.69	1.22	1.19	1.22	1.48	1.36	1.61	1.33	1.35	1.29	1.39	1.06	1.31	1.25	1.56	1.32	1.29	1.21	1.42	1.62	1.36
	75%	1.89	1.26	1.15	1.60	1.33	1.42	1.22	1.31	1.57	1.25	1.28	1.36	1.47	1.22	1.13	1.51	1.29	1.45	1.44	1.31	1.37
	80%	1.73	1.56	1.76	1.27	1.37	1.19	1.34	1.12	1.26	1.31	1.05	1.42	1.16	1.31	1.35	1.48	1.16	1.19	1.21	1.21	1.32
	85%	1.57	1.32	1.75	1.04	1.35	1.36	1.26	1.41	1.40	1.13	1.25	1.01	1.17	1.31	1.40	1.51	1.20	1.13	1.29	1.41	1.31
	90%	1.87	1.41	1.40	1.34	1.28	1.41	1.18	1.29	1.50	1.63	1.23	1.10	1.26	1.39	1.40	1.13	1.00	1.22	1.48	1.32	1.34
	95%	1.63	1.80	1.27	1.45	1.48	1.33	1.32	1.47	1.47	1.36	1.47	1.69	1.53	1.12	1.14	1.27	1.03	1.34	1.41	1.19	1.39
	100%	1.88	1.80	1.43	1.74	1.75	1.76	1.53	1.77	1.67	1.58	1.67	1.90	1.30	1.66	1.12	1.23	1.29	1.52	1.37	1.32	1.56
		1.42	1.23	1.19	1.24	1.18	1.20	1.28	1.26	1.34	1.30	1.39	1.29	1.41	1.41	1.35	1.37	1.31	1.41	1.37	1.45	1.32

由表 4.6 可知：此測試範圍的總平均誤差為 1.32%，平均誤差介於 0.74%~2.02%之間。最佳參數為$(C_{0.40}, H_{0.30})$，其最佳平均誤差為 0.74%。所建議之C_α與H_β參數組合，C_α為 30%~50%，H_β為 10%~30%，其總平均誤差為 1.01%，標準差為 1.22%，如表 4.7 所示。

表 4.7　DNM 法總次數 30 之建議參數組合(C_α, H_β)範圍執行結果

$C_\alpha \setminus H_\beta$	10%	15%	20%	25%	30%
30%	0.81 (0.98)	0.99 (1.27)	1.10 (1.31)	0.93 (1.02)	1.11 (1.35)
35%	1.26 (1.15)	0.82 (1.20)	0.93 (1.20)	0.91 (1.38)	1.30 (1.19)
40%	0.86 (1.09)	0.84 (1.26)	1.12 (1.20)	1.02 (1.32)	0.74 (1.00)
45%	0.89 (1.29)	1.11 (1.37)	1.14 (1.37)	0.91 (1.33)	1.21 (1.38)
50%	1.04 (1.10)	1.05 (1.17)	1.01 (1.19)	0.95 (1.27)	1.08 (1.08)
總平均誤差		1.01	標準差	1.22	

表 4.8 中，第三欄為現今文獻中所提出的最佳解；第四、五欄為 16 個例題在四百個不同的參數組合下個別執行結果，各題的平均誤差及標準差；第六欄為各題在四百次測試中所得到已知最佳解之次數；第七欄為各題執行四百次的總時間。

表 4.8　在總次數 30 下 C_α 與 H_β 參數組合測試結果(Ⅱ)

題號	例題名稱	已知最佳解	平均誤差(%)	誤差標準差(%)	得到已知最佳解比例	總執行時間(秒)
1	gr24	1272	0.13	0.31	340 / 400	6.36
2	swiss42	1273	0.25	0.71	357 / 400	21.58
3	gr48	5046	0.42	0.35	73 / 400	30.47
5	hk48	11461	0.68	0.66	53 / 400	30.08
4	eil51	426	1.30	0.72	31 / 400	35.09
6	brazil58	25395	0.02	0.07	325 / 400	49.86
7	st70	675	1.08	0.66	51 / 400	80.44
8	pr76	108159	0.89	0.66	74 / 400	103.73
9	kroA100	21282	0.62	0.52	22 / 400	213.89
10	kroC100	20749	1.09	0.63	20 / 400	212.63
11	lin105	14379	0.89	0.77	56 / 400	245.36
13	gr120	6942	2.08	0.58	0 / 400	327.72
12	u159	42080	1.94	0.96	10 / 400	821.25
14	ts225	126643	1.97	1.52	33 / 400	2887.55
15	lin318	42029	3.35	0.58	0 / 400	9280.75
16	pcb442	50778	4.41	0.65	0 / 400	22982.90
總 平 均			1.32	0.65		93.32
16 個例題中得到已知最佳解之題目比例					13 / 16	
6400 次測試中得到已知最佳解之次數比例					1445 / 6400	

由表 4.8 可知：在每道題目中隨著點數越大，其求解的平均誤差會有越來越大趨勢，以 pcb442 例題來說其四百次的解題平均誤差將達到 4.41%；16 個例題的總平均為 1.32%；總平均之誤差標準差為 0.65%；就以得到已知最佳解的題目數而言，16 個例題有 13 個例題可以得到已知最佳解；若以得到已知最佳解的次數來看，6400 次測試中有 1445 次可得到已知最佳解；在執行速度上，16 個例題在固定某組參數加以測試，所花費的平均時間為 93.32 秒。

4.2.3 總次數 45 次之 C_α 與 H_β 參數組合測試

當總次數 45 時，K 與 L 的組合採用 $(K \times L) = (15 \times 3)$ 的組合方式；其他組件仍維持不變：對於每道例題開始皆先以 FI+2-Opt 的方式先搜尋，待落入區域最佳解後，在擾動成本下使用 Or-H 的方式進行鄰域搜尋，在原始成本下使用 2-Opt 進行鄰域搜尋。

表 4.9 為總次數 45 次擾動的 C_α 與 H_β 參數測試組合，橫向的項目為 C_α 的參數範圍，縱向的項目為 H_β 的參數範圍；表中每格數值表示 16 個例題在執行完所對應的參數後，16 個例題的平均誤差百分比(%)。

由表 4.9 可知：此測試範圍的總平均誤差為 1.09%，平均誤差介於 0.59%~1.64%之間。最佳參數為 $(C_{0.50}, H_{0.40})$，其最佳平均誤差為 0.59%。所建議之 C_α 與 H_β 參數組合，C_α 為 30%~50%，H_β 為 10%~30%，其總平均誤差為 0.88%，標準差為 1.11%，如表 4.10 所示。

表 4.9　DNM 法在總次數 45 下 C_α 與 H_β 參數組合測試結果(Ⅰ)

總次數 45，固定參數 $(K \times L) = (15 \times 3)$

		(H_β)參數範圍																				
		5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%	
(C_α)參數範圍	5%	1.14	1.40	1.28	0.99	1.28	0.98	0.96	1.12	1.00	1.40	1.31	1.05	1.21	1.64	1.38	1.30	1.43	1.62	1.30	0.98	1.24
	10%	0.90	0.75	0.98	0.96	0.99	0.65	1.40	1.05	1.23	1.21	0.83	1.36	1.30	1.22	1.10	1.23	0.85	1.32	1.34	1.03	1.09
	15%	0.75	0.90	1.26	0.93	1.11	1.01	0.94	0.95	1.36	1.18	1.01	1.04	1.33	1.23	1.14	1.31	1.00	1.19	1.12	1.15	1.10
	20%	1.13	0.91	0.82	0.91	1.08	0.98	0.91	0.98	1.12	1.13	0.92	1.15	1.20	1.22	1.18	1.11	1.05	1.14	1.10	1.23	1.06
	25%	0.85	0.77	0.82	1.00	1.12	0.86	1.08	1.09	1.09	1.06	1.13	0.95	1.13	1.22	1.33	1.47	1.25	1.11	1.09	1.36	1.09
	30%	1.06	**0.83**	**0.90**	**0.76**	**0.89**	**0.94**	0.76	1.14	1.15	0.89	0.81	1.05	1.14	1.43	1.14	1.17	1.06	1.31	1.53	0.92	1.04
	35%	1.11	**0.89**	**0.66**	**0.81**	**0.86**	**0.90**	1.06	0.89	0.87	1.20	1.12	0.66	1.35	0.96	0.99	1.05	1.16	1.26	1.00	1.29	1.00
	40%	0.85	**0.93**	**0.85**	**0.88**	**0.96**	**0.72**	0.90	0.81	1.07	1.07	1.04	1.00	0.87	1.19	1.17	1.03	1.06	0.99	0.98	1.06	0.97
	45%	1.10	**0.82**	**0.82**	**0.83**	**0.89**	**0.84**	0.84	0.89	1.08	1.04	0.96	0.84	1.09	1.05	1.10	1.05	1.10	1.50	1.15	1.28	1.01
	50%	1.20	**0.87**	**1.06**	**0.99**	**1.18**	**0.92**	1.04	0.59	0.95	0.96	1.12	1.09	1.04	1.25	1.14	0.92	1.24	1.00	1.03	1.36	1.05
	55%	1.08	1.05	0.95	1.00	1.04	1.00	1.02	0.91	1.05	1.02	1.41	1.13	1.15	1.35	1.10	1.32	1.43	1.11	1.30	1.35	1.14
	60%	1.18	1.28	1.07	1.09	0.96	0.86	1.10	0.92	1.01	0.78	0.94	1.06	1.25	1.21	1.08	0.93	1.32	1.27	0.99	1.28	1.08
	65%	1.14	1.38	0.84	0.93	0.99	0.93	0.86	1.00	0.99	0.96	1.21	1.27	1.17	1.02	1.22	1.14	1.21	1.03	1.05	1.14	1.07
	70%	1.22	1.02	0.88	1.07	1.12	0.91	1.17	0.85	1.09	0.90	1.11	1.16	1.34	1.19	1.01	1.08	0.96	1.31	1.06	1.15	1.08
	75%	1.11	0.95	1.13	0.99	1.09	1.12	0.81	1.06	0.97	0.86	1.02	0.98	0.94	1.10	0.94	1.11	1.26	0.97	1.09	1.12	1.03
	80%	1.27	1.04	1.48	1.21	1.06	1.11	1.04	1.38	0.92	1.05	0.94	0.87	1.01	0.77	0.93	1.08	1.09	1.00	1.14	1.11	1.07
	85%	1.17	0.80	1.01	1.30	1.14	1.13	1.42	1.20	0.98	1.12	1.29	0.99	1.06	1.10	1.33	1.09	1.10	1.08	0.92	0.91	1.11
	90%	1.35	1.61	1.00	1.02	1.16	1.25	1.01	0.97	1.11	1.10	1.10	1.24	1.20	1.06	1.02	0.99	0.87	1.04	0.86	0.73	1.09
	95%	1.57	1.31	1.45	1.41	1.34	1.36	1.39	1.44	1.51	1.04	1.16	1.15	0.92	0.97	0.86	1.14	1.06	1.02	1.06	0.93	1.20
	100	1.57	1.57	1.32	1.20	1.13	1.45	1.47	1.26	1.59	1.16	1.01	0.97	1.30	1.37	1.06	1.11	0.99	1.14	1.08	1.08	1.24
		1.14	1.05	1.03	1.01	1.07	1.00	1.06	1.02	1.11	1.06	1.07	1.05	1.15	1.18	1.11	1.13	1.12	1.17	1.11	1.12	1.09

表 4.10　DNM 法總次數 45 之建議參數組合(C_α, H_β)範圍執行結果

$C_\alpha \setminus H_\beta$	10%	15%	20%	25%	30%
30%	0.83 (1.23)	0.90 (1.18)	0.76 (1.15)	0.89 (0.80)	0.94 (1.26)
35%	0.89 (1.32)	0.66 (0.80)	0.81 (1.03)	0.86 (0.99)	0.90 (1.09)
40%	0.93 (1.40)	0.85 (1.31)	0.88 (0.98)	0.96 (1.03)	0.72 (1.06)
45%	0.82 (1.41)	0.82 (1.08)	0.83 (1.05)	0.89 (0.99)	0.84 (0.95)
50%	0.87 (1.01)	1.06 (1.37)	0.99 (1.41)	1.18 (1.41)	0.92 (0.85)
總平均誤差		0.88	標準差	1.11	

表 4.11 中，第三欄為現今文獻中所提出的最佳解；第四、五欄為 16 個例題在四百個不同的參數組合下個別執行結果，各題的平均誤差及標準差；第六欄為各題在四百次測試中所得到已知最佳解之次數；第七欄為各題執行四百次的總時間。

由表 4.11 可知：在每道題目中隨著點數越大，其求解的平均誤差會有越來越大趨勢，以 pcb442 例題來說其四百次的解題平均誤差將達到 4.07%；16 個例題的總平均為 1.09%；總平均之誤差標準差為 0.54%；就以得到已知最佳解的題目數而言，16 個例題有 13 個例題可以得到已知最佳解；若以得到已知最佳解的次數來看，6400 次測試中有 1726 次可得到已知最佳解；在執行速度上，16 個例題在固定某組參數加以測試，所花費的平均時間為 135.36 秒。

表 4.11　在總次數 45 下 C_α 與 H_β 參數組合測試結果(Ⅱ)

題號	例題名稱	已知最佳解	平均誤差(%)	誤差標準差(%)	得到已知最佳解比例	總執行時間(秒)
1	gr24	1272	0.05	0.16	370 / 400	7.66
2	swiss42	1273	0.13	0.39	377 / 400	26.81
3	gr48	5046	0.30	0.28	113 / 400	38.36
5	hk48	11461	0.45	0.53	94 / 400	37.22
4	eil51	426	1.05	0.67	45 / 400	43.24
6	brazil58	25395	0.01	0.04	346 / 400	62.92
7	st70	675	0.84	0.57	78 / 400	102.98
8	pr76	108159	0.63	0.58	107 / 400	131.45
9	kroA100	21282	0.46	0.40	24 / 400	280.89
10	kroC100	20749	0.83	0.55	33 / 400	278.39
11	lin105	14379	0.61	0.54	67 / 400	319.94
13	gr120	6942	1.93	0.52	0 / 400	439.75
12	u159	42080	1.59	0.86	24 / 400	1112.03
14	ts225	126643	1.37	1.24	48 / 400	4068.14
15	lin318	42029	3.09	0.57	0 / 400	13517.30
16	pcb442	50778	4.07	0.68	0 / 400	33678.40
總 平 均			1.09	0.54		135.36
16 個例題中得到已知最佳解之題目比例					13 / 16	
6400 次測試中得到已知最佳解之次數比例					1726 / 6400	

4.2.4　總次數 60 次之 C_α 與 H_β 參數組合測試

當總次數 60 時，K 與 L 的組合採用 $(K \times L) = (20 \times 3)$ 的組合方式；其他組件亦維持不變：對於每道例題開始皆先以 FI+2-Opt 的方式先搜尋，待落入區域最佳解後，在擾動成本下使用 Or-H 的方式進行鄰域搜尋，在原始成本下使用 2-Opt 進行鄰域搜尋。

表 4.12 為總次數 60 次擾動的 C_α 與 H_β 參數測試組合，橫向的項目為 C_α 的參數範圍，縱向的項目為 H_β 的參數範圍；表中每格數值表示 16 個例題在執行完所對應的參數後，16 個例題的平均誤差百分比(%)。

表 4.12　DNM 法在總次數 60 下 C_α 與 H_β 參數組合測試結果(Ⅰ)

總次數 60，固定參數 $(K \times L) = (20 \times 3)$

		(H_β)參數範圍																				
		5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%	
(C_α)參數範圍	5%	1.06	1.49	1.40	1.42	0.91	0.80	1.26	1.20	1.22	1.35	1.05	0.96	1.09	1.25	1.15	1.27	1.31	1.51	1.23	1.06	1.20
	10%	0.93	0.85	0.70	0.94	0.93	0.77	0.99	1.17	1.04	1.27	0.87	0.98	1.07	1.37	0.98	1.04	1.06	1.11	1.25	1.18	1.02
	15%	0.88	0.70	0.90	0.85	0.79	1.16	0.81	0.99	1.10	0.87	1.06	0.89	0.90	0.98	0.81	1.22	0.88	1.04	1.27	0.85	0.95
	20%	1.14	0.76	0.72	0.85	0.87	0.81	0.80	0.86	1.05	0.91	0.99	0.85	1.01	1.10	1.15	1.05	1.10	0.82	1.01	0.91	0.94
	25%	0.87	0.61	0.62	0.95	0.84	1.02	0.79	0.82	0.90	1.16	0.83	0.73	1.27	1.14	1.02	1.32	1.16	1.22	1.31	1.19	0.99
	30%	1.05	0.82	0.72	0.96	0.66	0.97	0.89	0.85	0.82	0.87	0.93	0.91	1.05	0.85	1.21	1.27	0.93	1.01	1.01	1.22	0.95
	35%	0.94	0.89	0.75	0.80	0.71	0.97	0.96	0.61	0.89	1.06	0.94	0.78	1.00	0.93	1.42	1.03	0.98	1.15	1.02	1.11	0.95
	40%	1.15	0.74	**0.65**	**0.95**	**0.82**	**0.66**	**0.76**	0.74	1.01	0.77	0.90	0.79	1.05	1.01	0.80	1.05	1.04	1.05	1.20	1.04	0.91
	45%	1.09	0.83	**0.73**	**0.89**	**0.98**	**0.89**	**0.86**	1.01	0.85	0.75	0.82	1.08	0.99	1.06	0.86	1.04	0.85	1.29	1.02	1.41	0.96
	50%	1.00	0.94	**0.87**	**0.83**	**0.93**	**0.69**	**0.69**	0.89	0.79	0.88	0.87	0.96	0.99	0.91	1.06	0.89	0.97	1.03	1.08	1.11	0.92
	55%	1.01	0.87	**0.70**	**0.79**	**0.70**	**0.87**	**0.85**	0.95	1.05	0.69	0.95	1.04	0.94	0.94	0.92	1.04	0.87	0.96	1.00	1.10	0.91
	60%	1.05	0.90	**0.76**	**1.03**	**0.85**	**0.69**	**0.65**	0.88	0.87	0.76	0.82	0.83	0.80	1.07	1.16	1.03	1.32	1.02	1.04	1.14	0.93
	65%	1.23	1.16	0.94	1.03	0.94	0.79	0.85	0.81	0.74	0.85	0.95	0.93	0.88	0.88	0.95	0.95	0.92	1.01	0.95	1.15	0.95
	70%	1.10	0.89	1.06	0.98	1.16	0.62	0.81	1.21	0.90	1.02	0.73	0.68	1.16	1.03	1.12	0.75	0.94	1.10	0.96	1.36	0.98
	75%	1.09	1.20	1.08	1.21	0.84	0.92	0.70	1.12	0.79	0.68	1.02	0.92	0.87	0.82	0.98	1.06	0.94	1.10	1.00	0.85	0.96
	80%	1.14	1.28	0.97	0.90	0.99	1.05	1.26	1.05	1.00	1.08	1.04	0.97	1.09	0.88	0.82	1.12	0.85	0.90	0.91	1.25	1.03
	85%	1.28	1.10	1.24	0.90	1.21	1.09	1.10	0.93	0.92	0.84	0.75	0.64	0.80	1.00	0.89	0.92	0.94	1.09	1.07	1.23	1.00
	90%	1.36	1.06	1.15	1.05	1.09	0.90	1.01	0.87	1.18	0.94	0.83	1.17	1.11	1.02	0.76	0.81	0.86	1.08	0.89	1.22	1.02
	95%	1.26	1.10	1.18	1.24	0.99	0.98	0.99	0.83	0.99	1.01	1.06	0.95	1.07	0.88	0.81	1.10	0.86	0.94	0.83	1.02	1.01
	100%	1.29	1.21	1.37	1.15	1.13	1.14	1.13	0.81	1.18	1.20	1.06	1.32	1.22	1.26	0.96	1.03	1.04	1.12	1.08	0.88	1.13
		1.10	0.97	0.93	0.99	0.92	0.89	0.91	0.93	0.96	0.95	0.92	0.92	1.02	1.02	0.99	1.05	0.99	1.08	1.06	1.11	0.98

由表 4.12 可知：此測試範圍的總平均誤差為 0.98%，平均誤差介於 0.61%~1.51%之間。最佳參數為$(C_{0.25}, H_{0.10})$，其最佳平均誤差為 0.61%。所建議之C_α與H_β參數組合，C_α為 40%~60%，H_β為 15%~35%，其總平均誤差為 0.80%，標準差為 1.16%，如表 4.13 所示。

表 4.13　DNM 法總次數 60 之建議參數組合(C_α, H_β)範圍執行結果

$C_\alpha \setminus H_\beta$	15%	20%	25%	30%	35%
40%	0.65	0.95	0.82	0.66	0.76
45%	0.73	0.89	0.98	0.89	0.86
50%	0.87	0.83	0.93	0.69	0.69
55%	0.70	0.79	0.70	0.87	0.85
60%	0.76	1.03	0.85	0.69	0.65
總平均誤差		0.80	標準差	1.16	

表 4.14 中，第三欄為現今文獻中所提出的最佳解；第四、五欄為 16 個例題在四百個不同的參數組合下個別執行結果，各題的平均誤差及標準差；第六欄為各題在四百次測試中所得到已知最佳解之次數；第七欄為各題執行四百次的總時間。

由表 4.14 可知：在每道題目中隨著點數越大，其求解的平均誤差會有越來越大趨勢，以 pcb442 例題來說其四百次的解題平均誤差將達到 3.95%；16 個例題的總平均為 0.98%；總平均之誤差標準差為 0.46%；就以得到已知最佳解的題目數而言，16 個例題有 13 個例題可以得到已知最佳解；若以得到已知最佳解的次數來看，6400 次測試中有 1918 次可得到已知最佳解；在執行速度上，16 個例題在固定某組參數加以測試，所花費的平均時間為 175.67 秒。

表 4.14　在總次數 60 下 C_α 與 H_β 參數組合測試結果(Ⅱ)

題號	例題名稱	已知最佳解	平均誤差 (%)	誤差標準差(%)	得到已知最佳解比例	總執行時間(秒)
1	gr24	1272	0.02	0.09	390 / 400	8.53
2	swiss42	1273	0.04	0.17	392 / 400	31.83
3	gr48	5046	0.23	0.24	139 / 400	45.84
5	hk48	11461	0.38	0.44	96 / 400	44.30
4	eil51	426	1.02	0.66	44 / 400	51.17
6	brazil58	25395	0.00	0.02	350 / 400	75.89
7	st70	675	0.67	0.56	112 / 400	125.38
8	pr76	108159	0.51	0.52	144 / 400	160.09
9	kroA100	21282	0.36	0.27	35 / 400	348.61
10	kroC100	20749	0.72	0.50	48 / 400	345.55
11	lin105	14379	0.45	0.45	100 / 400	398.39
13	gr120	6942	1.72	0.56	0 / 400	565.81
12	u159	42080	1.42	0.75	19 / 400	1414.95
14	ts225	126643	1.23	1.10	49 / 400	5324.05
15	lin318	42029	3.01	0.53	0 / 400	17741.90
16	pcb442	50778	3.95	0.60	0 / 400	43586.20
總 平 均			0.98	0.46		175.67
16 個例題中得到已知最佳解之題目比例					13 / 16	
6400 次測試中得到已知最佳解之次數比例					1918 / 6400	

4.2.5　小結

綜合 4.2.1 節、4.2.2 節、4.2.3 節及 4.2.4 節對 DNM 法的控制參數進行測試，得到以下幾點結論：

1. 當固定 $K \times L$ 的總次數為 30 次時，以不同組合的 K 與 L 作測試；在六種組合中，K 與 L 組合為 (10×3) 時可以得到較好的結果。當固定 $K \times L$ 的總次數為 45 次時，以不同組合的 K 與 L 作測試；在四種組合中，K 與 L 組合為 (15×3) 時可以得到較好的結果。當固定 $K \times L$ 的總次數為 60 次時，以不同組合的 K 與 L 作測試；在十種組合中，K 與 L 組合為 (20×3) 時可以得到較好的結果。

2. C_α 與 H_β 在(5%~100%)範圍內，對 16 個測試例題做測試，總擾動次數 30 下

總平均誤差為 1.32%；總擾動次數 45 下總平均誤差為 1.09%；總擾動次數 60 下總平均誤差為 0.98%。同樣的參數範圍內，總擾動次數隨著時間增加，總平均誤差會有下降的趨勢，表示隨著擾動次數增加，能夠找到較佳解的機會將變高。

3. 本研究在總擾動次數 30 下所建議之參數組合範圍為：$C_\alpha = 30\% \sim 50\%$、$H_\beta = 10\% \sim 30\%$，建議範圍內的總平均誤差為 1.01%，標準差為 1.22%；在總擾動次數 45 下所建議之參數組合範圍為：$C_\alpha = 30\% \sim 50\%$、$H_\beta = 10\% \sim 30\%$，建議範圍內的總平均誤差為 0.88%，標準差為 1.11%；在總擾動次數 60 下所建議之參數組合範圍為：$C_\alpha = 40\% \sim 60\%$、$H_\beta = 15\% \sim 35\%$，建議範圍內的總平均誤差為 0.80%，標準差為 1.16%。在較中間的 C_α 與 H_β 參數組合有較佳的平均誤差，而兩邊極端 C_α 與 H_β 參數組合的平均誤差較差，顯示擾動太劇烈，使所求到解在解空間裡做大範圍的跳動，一但還原到原始成本空間時，不易收斂到全域最佳解上；若擾動太小，不易跳出區域最佳解。

4. 在執行時間上，將各例題執行的 CPU 平均時間(Time)和題目規模(N)進行迴歸分析和曲線配適，其結果如表 4.15 和圖 4.1 所示，顯示 DNM 法解題架構的複雜度是一多項式時間函數。在相同的執行架構，不同擾動次數的執行時間，擾動次數每增加 15 次，則所有例題的平均執行時間約增加 41 秒。另外圖 4.2 表示在相同的執行架構，不同總擾動次數下，總平均誤差與平均執行時間的結果示意圖。

表 4.15　DNM 法架構下 CPU 平均時間(Time)和題目規模(N)的曲線配適

FI+2-Opt+DNM(Or-H+2-Opt) 執行架構	CPU 平均時間(Time)和題目規模(N) 的曲線配適	R^2
總擾動次數 30	$Time = 2.925025 + 0.000449N^2 - 0.07632$	0.997835
總擾動次數 45	$Time = 4.425265 + 0.000664N^2 - 0.11515$	0.997659
總擾動次數 60	$Time = 5.5771 + 0.000856N^2 - 0.14683N$	0.997898

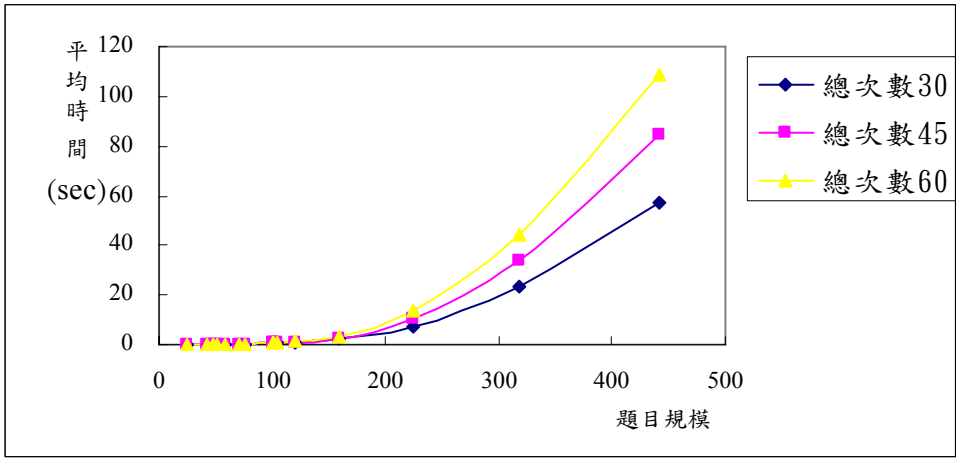


圖 4.1　DNM 法架構下各題的平均執行時間

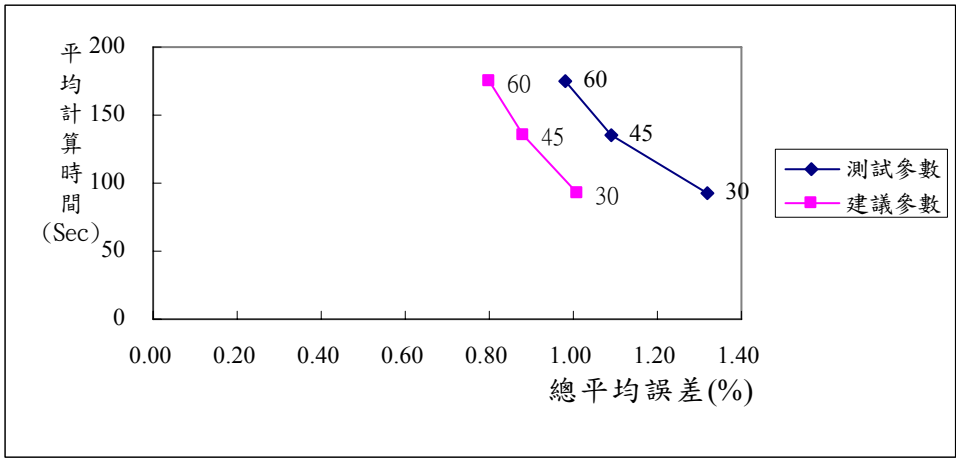


圖 4.2　DNM 法架構下測試結果示意圖

此外，表 4.16 列出 DNM 法應用在 TSP 的測試求解過程中，各題的最佳個案結果。其中 16 個測試例題中得到 13 題之已知最佳解，平均個案誤差為 0.19%，標準差為 0.52%。

表 4.16　DNM 法對 16 個測試例題之最佳個案結果

題號	例題名稱	已知最佳解成本	最佳尋獲解成本	最佳個案誤差(%)	參數組合 (K,L)	參數組合 (C_{α},H_{β})
1*	gr24	1272	1272	0.00%	(10,3)	$(C_{0.05},H_{0.05})$
2*	swiss42	1273	1273	0.00%	(10,3)	$(C_{0.05},H_{0.05})$
3*	gr48	5046	5046	0.00%	(10,3)	$(C_{0.05},H_{0.45})$
4*	hk48	11461	11461	0.00%	(10,3)	$(C_{0.05},H_{0.80})$
5*	eil51	426	426	0.00%	(10,3)	$(C_{0.25},H_{0.05})$
6*	brazil58	25395	25395	0.00%	(10,3)	$(C_{0.05},H_{0.05})$
7*	st70	675	675	0.00%	(10,3)	$(C_{0.05},H_{0.20})$
8*	pr76	108259	108259	0.00%	(10,3)	$(C_{0.05},H_{0.25})$
9*	kroA100	21282	21282	0.00%	(10,3)	$(C_{0.05},H_{0.95})$
10*	kroC100	20749	20749	0.00%	(10,3)	$(C_{0.05},H_{0.60})$
11*	lin105	14379	14379	0.00%	(10,3)	$(C_{0.05},H_{0.05})$
12	gr120	6942	6951	0.13%	(20,3)	$(C_{0.05},H_{0.75})$
13*	u159	42080	42080	0.00%	(10,3)	$(C_{0.05},H_{0.15})$
14*	ts225	126643	126643	0.00%	(10,3)	$(C_{0.10},H_{0.30})$
15	lin318	42029	42458	1.02%	(15,3)	$(C_{0.50},H_{0.40})$
16	pcb442	50778	51750	1.91%	(15,3)	$(C_{0.80},H_{0.70})$
平均個案誤差				0.19%	標準差	0.53%

*表示求解過程中有多組參數可求得已知最佳解

第五章　結論與建議

本研究參考成本擾動法(NM)，藉由擾動成本跳脫區域最佳解的觀念，並提出不同擾動成本的方式，改進了結果不能重現的缺點。另外研究中所提出的確定性成本擾動法針對成本擾動比率值C_α^i、增加成本比率值H_β^i、成本擾動比率值長度K、增加成本比率值長度L等參數，以及起始解解法、核心交換法及擾動公式等執行架構組件的設計與測試，所建議 DNM 法的組件方式和參數組合範圍如下表所示。在所建議的組件組合方式與數值範圍下得到的之解題績效為：16 個測試例題在總擾動次數 30 的總平均誤差為 1.01%、標準差為 1.22%；總擾動次數 45 的總平均誤差為 0.88%、標準差為 1.11%；總擾動次數 60 的總平均誤差為 0.80%、標準差為 1.16%。若從整個測試過程中所能找到 16 個例題的最佳個案來看：DNM 法共找到 13 題的已知最佳解，而平均個案誤差為 0.19%。

組件項目與參數組合	建議之方式與範圍
起始構建法	最遠插入法(FI)
擾動成本之核心交換法	Or-H
原始成本之核心交換法	2-Opt
成本擾動公式	(3-13)式
成本擾動比率值(C_α)	(30~50)、(30~50)、(40~60)
增加成本比率值(H_β)	(10~30)、(10~30)、(15~35)
成本擾動比率值長度與增加成本比率值長度(K,L)	(10,3)、(15,3)、(20,3)
(C_α,H_β)的下降型態	(3-14)、(3-15)式

在測試過程中發現：在測試的範圍內，總執行次數越大則總平均誤差有下降趨勢，惟時間上會有增加的情況。確定性成本擾動法(DNM)的執行架構在各例題的求解上，隨著節點數增加，呈現多項式時間的成長。DNM法經由國際標竿例題之測試，結果顯示有不錯的解題能力，可突破傳統交換型啟發式解法的求解屏障，在求解時間上隨著節點增加亦有合理之成長，表示DNM法有相當之應用潛力，期望此方法能做為求解TSP或相關車輛路線問題之有效工具。

然而，本研究對於DNM法應用在求解TSP上只是提出一個初步的可行架構，且在組件的組合方式之探討和參數範圍的測試上仍有許多地方需要改進和加強。以下提出幾點建議，作為未來後續研究參考之用。

1. 在起始解部分，本研究僅考慮最遠插入法(FI)，未來可考慮其他有效率之起始解構建方法，如鄰點法(Neighbor Procedure)、節省法（Saving Method）、貪心

法(Greedy Algorithm)和最小擴張樹法(Minimal Spanning Tree Approach)等方法。

2. 在核心交換法部分，本研究僅考慮2-Opt和Or-H兩種方法，未來可嘗試不同的交換法，如3-Opt、Lin-Kernighan交換法和GENIUS等方法。

3. 在參數測試上，可進行更大規模和更詳細之測試；值得注意的是隨著上述的新的組件的應用，在參數建議範圍上亦應隨著調整。

4. 本研究所提出之方法亦可結合其他著名之啟發式解法，如Tabu、SA和TA等方法之觀念和優點以加強解題績效。

參考文獻

1. 王國琛(2000)，(指導教授：韓復華)，「巨集啟發式解法在求解大規模旅行推銷員問題之應用」，國立交通大學運輸工程與管理學系，八十九年度大專生參與專題研究計畫研究成果報告 (NSC89-2815-C009-014-E)。

2. 林修竹(1999)，(指導教授：韓復華)，「包容性啟發式解法在VRPTW 問題上之應用」，交通大學運輸工程與管理學系碩士論文。

3. 卓裕仁(2001)，(指導教授：韓復華)，「以巨集啟發式方法求解多車種與週期性車輛路線問題之研究」，交通大學運輸工程與管理學系所博士論文。

4. 陳國清 (1996)，(指導教授：韓復華)，「成本擾動法(NM)與兩極跳躍法(FF)在TSP 問題應用之研究」，國立交通大學，運輸工程與管理學系畢業專題研究報告。

5. 陳建緯(2001)，(指導教授：韓復華)，「大規模旅行推銷員問題之研究：鄰域搜尋法與巨集啟發式解法之應用」，交通大學運輸工程與管理研究所碩士論文。

6. 韓復華、張靖、卓裕仁 (1996)，「車輛路線問題研究：SA、TA、NM、SSS與交換型啟發式方法之綜合應用分析」，國立交通大學運輸工程與管理學系，八十五年度國科會專題研究計畫成果報告(NSC-85-2211-E-009-023)。

7. 韓復華、張靖、卓裕仁 (1997)，「路線與排程問題研究：結合交換型解法與AI 演算法之應用」，國立交通大學運輸工程與管理學系，八十六年度國科會專題研究計畫成果報告(NSC-86-2621-E-009-001)。

8. 楊智凱 (1995)，(指導教授：韓復華)，「以門檻接受法改善TSP 與VRP 路網成本之研究」，國立交通大學，土木研究所運工管組碩士論文。

9. 韓復華、楊智凱 (1996) 「門檻接受法在TSP 問題上之應用」, 運輸計劃季刊, Vol. 25, No. 2, pp. 163-188.。

10. Althofer, I. & K.U. Koschnick (1991), “On the Convergence of Threshold Accepting,” Applied Mathematics and Optimization, Vol.24, pp.183-195.

11. Bodin, L., B.L. Golden, A. Assad & M. Ball (1983), "Routing and Scheduling of Vehicle and Crew: The State of Art," Special Issue of *Computers and Operations Research,* Vol.10, No.2, pp.63-211.

12. Charon, I. & O. Hudry (1993), "The Noising Method: a New Method for Combinatorial Optimization," *Operations Research Letters,* Vol. 14, pp.133-137.

13. Charon, I. & O. Hudry (2000), "Application of Noising Method to the Traveling Salesman Problem," *European Journal of Operational Research,* Vol. 125, pp.266-277.

14. Cook, S.A. (1971), "The Complexity of Theorem-Proving Procedures," Proc. Third Annual ACM Symposium on the Theory of Computing, pp.151-158.

15. Dantzig, G.B., D.R. Fulkerson, & S.M. Johnson (1954), Solution of a large-scale Traveling Salesman problem, *Operations Research,* 2(4), pp.393-410.

16. Dueck, G. & T. Scheuer (1990), "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing," *Journal of Computational Physics*, Vol.90, pp.161-175.

17. Dueck, G. (1993), "New Optimization Heuristics: the Great Deluge Algorithm and the Record-to-Record Travel," *Journal of Computational Physics*, Vol.104, pp.86-92.

18. Flood, M.M. (1956), "The Traveling Salesman Problem, " *Operations Research,* Vol.4, pp.61-75.

19. Gu, J. & X. Huang (1994), "Efficient Local Search With Search Space Smoothing: A Case Study of the Traveling Salesman Problem (TSP)," *IEEE Transaction on Systems*, Man and Cybernetics, Vol. 24, No. 5, pp.728-736.

20. Heider, C.H. (1972), "A Computationally Simplified Pair Exchange Algorithm for the Quadratic Assignment Problem," Pap. No.101 *Center for Naval Analysis*, Arlington, Virginia

21. Lawler, E.L. (1963), "The Quadratic Assignment Problem," *Management Science,* Vol.9, pp.586-599.

22. Miller, C.E., A.W. Tucker & R.A. Zemlin (1960), Integer Programming Formulation of Traveling Salesman Problems, Journal of the ACM, 7(4), pp.326-329

23. Kirkpatrick, S., C.D. Gelatt & M.P. Vecchi (1983), “Optimization by Simulated Annealing”, Science, Vol.220, pp.671-680.

24. Koopmans, T.C. & M. Beckman (1957), “Assignment Problems and the Location of Economic Activities,” *Econometrica ,* Vol.25, pp.53-76.

25. Lin, S. (1965), “Computer Solutions of the Traveling Salesman Problem,” *Bell System Technology Journal,* Vol.44, pp.2245-2269.

26. Lin, S. & B.W. Kernighan (1973), "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research,* Vol.21, pp.498-516.

27. Norback, J.P. & R.F. Love (1977), “Geometric Approaches to Solving the Traveling Salesman Problem”, *Management Science*, Vol.23, pp.1208-1223.

28. Norback, J.P. & R.F. Love (1979), “Heuristic for the Hamiltonian path problem in Euclidean two space”, *Journal of Operations Research Society*, Vol. 30, pp. 363-368.

29. Or, I. (1976), “Traveling salesman-type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking”, Ph.D. Dissertation, Northwestern University, Evanston, IL.

30. Reinelt, G. (1991), “TSPLIB：A Traveling Salesman Problem Library”, *ORSA Journal on Computing*, Vol.3, No.4, pp.376-384.

31. Rosenkrantz, D.J., R.E. Stearns & P.N. Lewis (1977), “An analysis of Several Heuristics for the Traveling Salesman Problem”, *SLAM Journal on Computing*, Vol.6, pp.563-581.

附錄Ⅰ：本研究在 16 個測試例題的最佳個案結果

本研究求得各例題之最佳路線圖如下，其中 gr24.tsp、swiss42.tsp、gr48.tsp、hk48.tsp、brazil58.tsp、gr120.tsp 等例題由於成本資料屬於 Matrix 型態，所以無法繪製其路線圖，僅將其路線的總成本和路線的節點邊編號依序列出。

1. gr24.tsp

＊ 成本：1272

＊ 路線：1 12 4 23 9 13 14 20 2 15 19 22 18 17 10 5 21 8 24 6 7 3 11 16 1

＊ 路線圖：此例題成本資料屬於 Matrix 型態，所以無法繪製其路線圖。

2. swiss42.tsp

＊ 成本：1273

＊ 路線：1 2 7 5 4 28 3 29 30 31 39 23 40 22 25 41 24 42 10 9 11 26 12 13 19 27 6 14 20 15 17 16 38 8 18 32 37 36 21 34 35 33 1

＊ 路線圖：此例題成本資料屬於 Matrix 型態，所以無法繪製其路線圖。

3. gr48.tsp

＊ 成本：5046

＊ 路線：1 13 48 16 11 36 26 6 14 9 32 27 17 21 22 8 33 5 31 12 10 15 24 37 47 43 45 2 40 39 42 35 20 38 30 4 19 3 25 23 34 18 46 41 44 28 7 29 1

＊ 路線圖：此例題成本資料屬於 Matrix 型態，所以無法繪製其路線圖。

4. hk48.tsp

＊ 成本：11461

＊ 路線：1 2 48 15 43 21 33 30 23 9 10 40 36 34 6 8 47 7 38 14 18 12 22 13 28 32 25 3 5 29 26 41 24 35 17 31 20 11 16 42 4 46 45 39 44 27 37 19 1

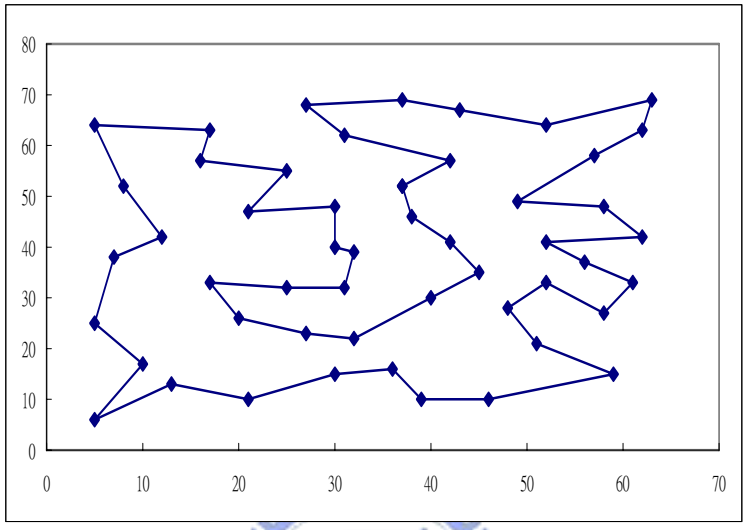
＊ 路線圖：此例題成本資料屬於 Matrix 型態，所以無法繪製其路線圖。

5. eil51.tsp

＊ 成本：426

＊ 路線：1 22 8 26 31 28 3 36 35 20 2 29 21 16 50 34 30 9 49 10 39 33 45 15 44 42 19 40 41 13 25 14 24 43 7 23 48 6 27 51 46 12 47 18 4 7 37 5 38 11 32 1

＊ 路線圖：



6. brazil58.tsp

＊ 成本：25395

＊ 路線：1 18 44 58 24 57 12 23 5 27 43 49 47 51 52 10 35 41 2 54 55 48 3 29 33 45 56 46 34 15 37 14 28 6 19 26 17 36 21 39 11 7 31 38 42 16 22 8 4 50 53 20 32 9 25 40 13 30 1

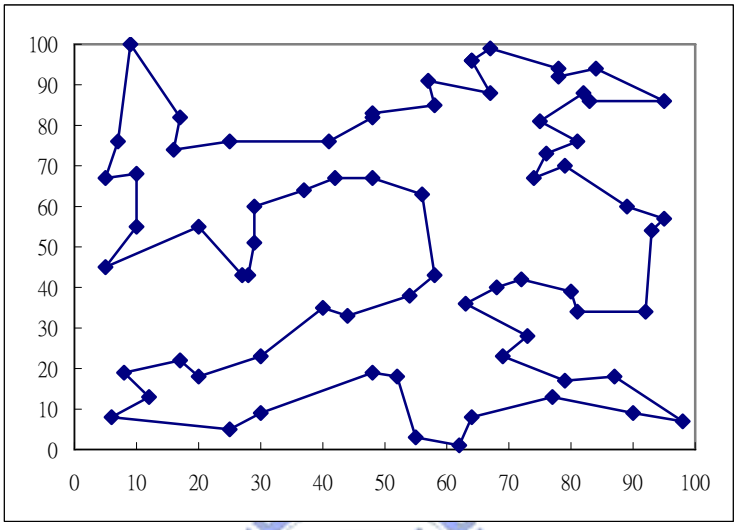
＊ 路線圖：此例題成本資料屬於 Matrix 型態，所以無法繪製其路線圖。

7. st70.tsp

＊ 成本：675

＊ 路線：1 36 29 13 70 35 69 31 38 59 22 66 63 57 15 24 19 7 2 4 18 42 32 3 8 26 55 49 28 14 20 30 44 68 27 46 25 45 39 61 40 9 17 43 41 6 53 5 10 52 60 12 21 34 33 62 54 67 48 11 64 65 56 51 50 58 37 47 16 23 1

＊ 路線圖：

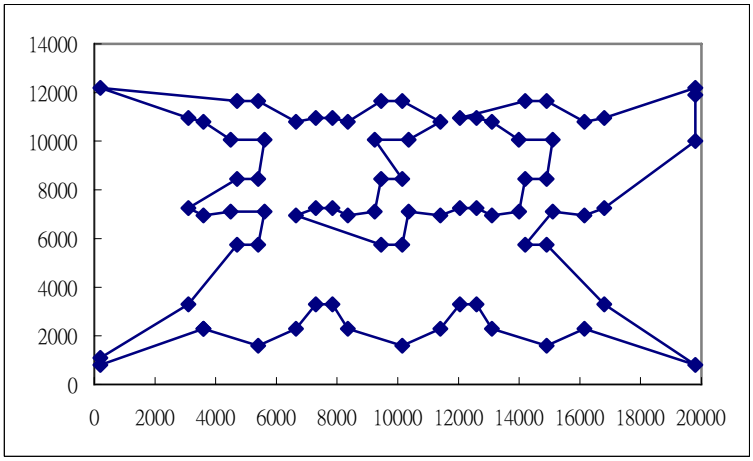


8. pr76.tsp

＊ 成本：108159

＊ 路線：1 23 22 21 25 24 46 45 44 48 47 69 68 70 67 50 49 51 66 65 71 72 73 64 63 62 61 41 60 59 58 57 56 55 52 53 54 42 43 28 27 26 20 19 31 30 29 32 33 35 34 40 39 38 36 37 18 17 16 15 74 14 13 12 11 10 9 8 7 6 5 4 3 2 75 76 1

＊ 路線圖：

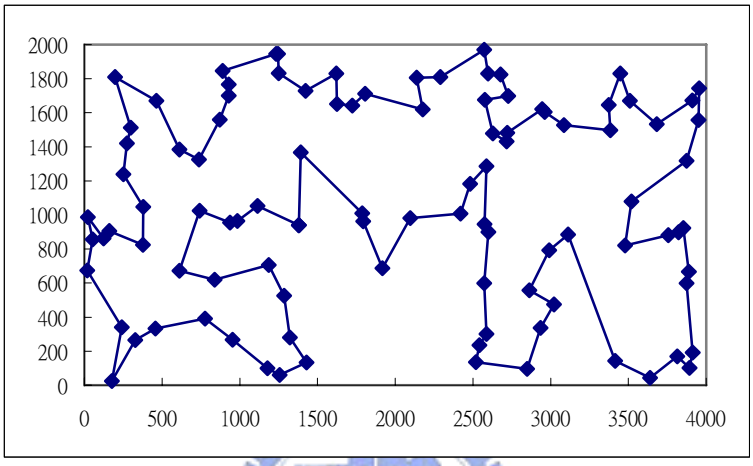


9. kroA100.tsp

＊ 成本：21828

＊ 路線：1 63 6 49 90 19 75 92 8 42 89 31 80 56 97 4 65 26 66 70 22 94 16 88 53 79 18 24 38 99 36 84 10 72 21 74 59 17 15 11 32 45 91 98 23 77 60 62 35 86 27 12 20 57 9 7 55 83 34 29 46 43 3 14 71 41 100 48 30 39 96 78 52 5 37 33 76 13 95 82 85 68 73 50 44 2 54 40 64 69 81 25 87 51 61 58 67 28 93 47 1

＊ 路線圖：

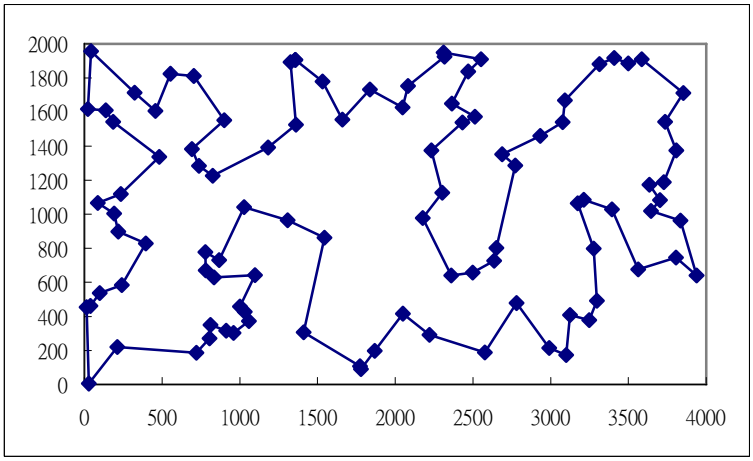


10. kroC100.tsp

＊ 成本：20749

＊ 路線：1 85 27 15 13 79 64 20 42 55 67 47 31 65 80 77 30 68 35 2 54 6 75 22 8 17 25 90 34 58 98 88 28 39 38 71 56 43 5 86 72 83 62 50 95 94 91 76 70 23 21 89 41 59 73 3 69 60 4 93 99 19 92 10 14 36 57 74 100 33 45 81 97 96 87 52 11 84 48 66 44 63 51 16 37 9 78 82 7 26 61 32 24 46 29 18 49 12 40 53 1

＊ 路線圖：

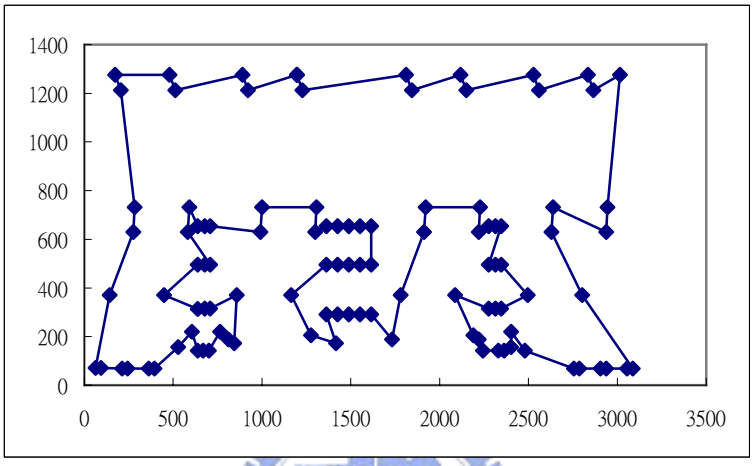


11. lin105.tsp

＊ 成本：14379

＊ 路線：1 2 6 7 10 11 15 103 21 22 29 30 31 32 33 28 23 20 12 19 24 27 16 17 18 25 26 36 37 42 41 43 46 52 53 58 57 54 51 47 44 104 40 49 45 48 50 55 56 59 105 62 63 70 69 74 75 81 73 76 80 86 79 77 72 64 67 68 71 78 82 83 84 85 91 92 96 97 101 102 93 89 90 98 99 100 95 94 88 87 66 65 61 60 39 38 35 34 14 13 4 5 9 8 3 1

＊ 路線圖：



12.gr120.tsp

＊ 成本：6951

＊ 路線：1 76 59 15 30 29 120 32 92 28 45 78 86 94 81 22 66 31 117 85 18 19 25 108 43 79 52 33 100 58 91 68 65 69 113 107 20 46 50 44 75 14 87 74 105 40 72 38 7 56 41 42 98 17 118 49 13 51 11 23 9 103 119 3 82 2 115 21 93 109 88 97 12 95 77 64 53 80 27 5 63 39 83 57 73 114 106 67 37 62 99 104 10 35 84 6 36 112 110 101 102 48 89 55 47 71 26 4 34 116 70 8 54 90 96 111 24 60 16 61 1

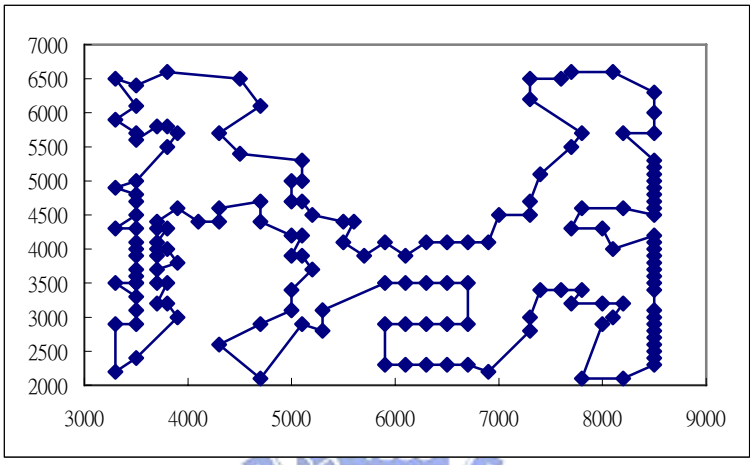
＊ 路線圖：此例題成本資料屬於 Matrix 型態，所以無法繪製其路線圖。

13. u159.tsp

＊ 成本：42080

＊ 路線：1 3 2 4 5 6 153 152 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 37 36 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 97 98 99 100 101 102 103 104 105 106 107 95 96 87 88 94 89 90 91 92 93 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 130 124 129 125 126 127 128 131 132 134 133 135 136 137 138 140 139 141 142 143 144 145 146 147 148 149 150 151 154 155 156 157 158 159 1

＊ 路線圖：

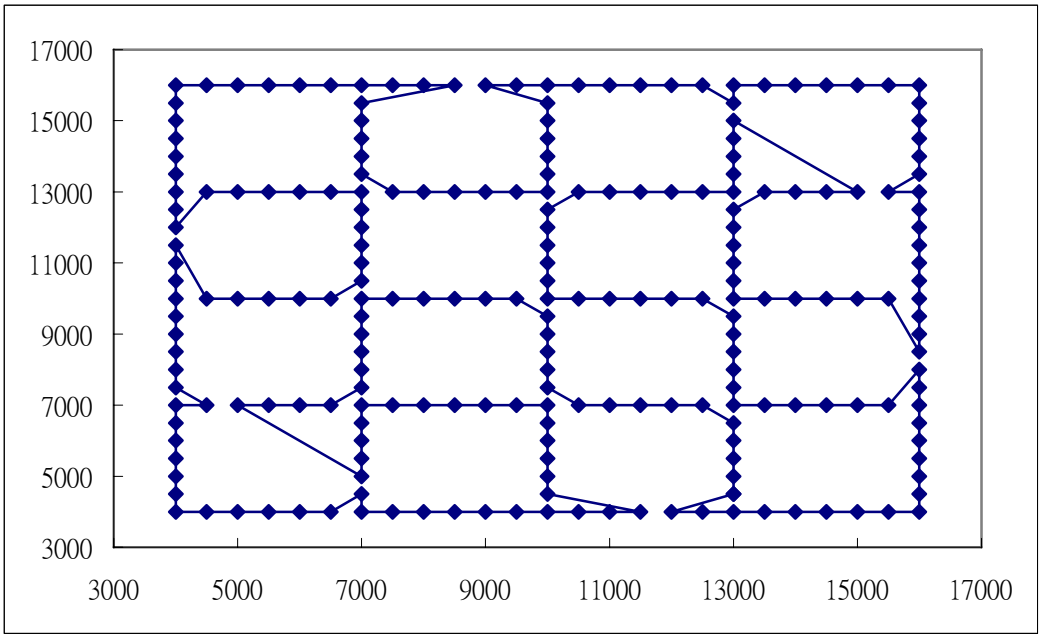


14. ts225.tsp

＊ 成本：126643

＊ 路線：1 2 3 4 5 6 7 131 8 9 10 11 12 13 14 15 16 136 137 138 139 140 39 40 41 42 43 44 145 144 143 142 141 17 18 19 20 21 22 23 24 25 146 147 148 149 150 50 171 172 173 49 48 47 46 45 166 167 168 169 170 69 70 71 72 73 74 174 175 75 196 197 198 199 200 99 100 221 222 223 224 225 125 124 123 122 121 120 220 119 118 117 116 115 114 113 112 111 110 215 214 213 212 211 88 89 90 91 92 93 216 217 218 219 98 97 96 95 94 195 194 193 192 191 68 67 66 65 64 63 186 187 188 189 190 87 86 85 84 83 82 206 207 208 209 210 109 108 107 106 105 104 103 102 101 205 204 203 202 201 76 180 179 77 78 79 80 81 185 184 183 182 181 58 59 60 61 62 165 164 163 162 161 38 37 36 35 34 33 135 134 133 132 28 29 30 31 32 156 157 158 159 160 57 56 55 54 53 52 178 177 176 51 155 154 153 152 151 26 27 130 129 128 127 126 1

＊ 路線圖：

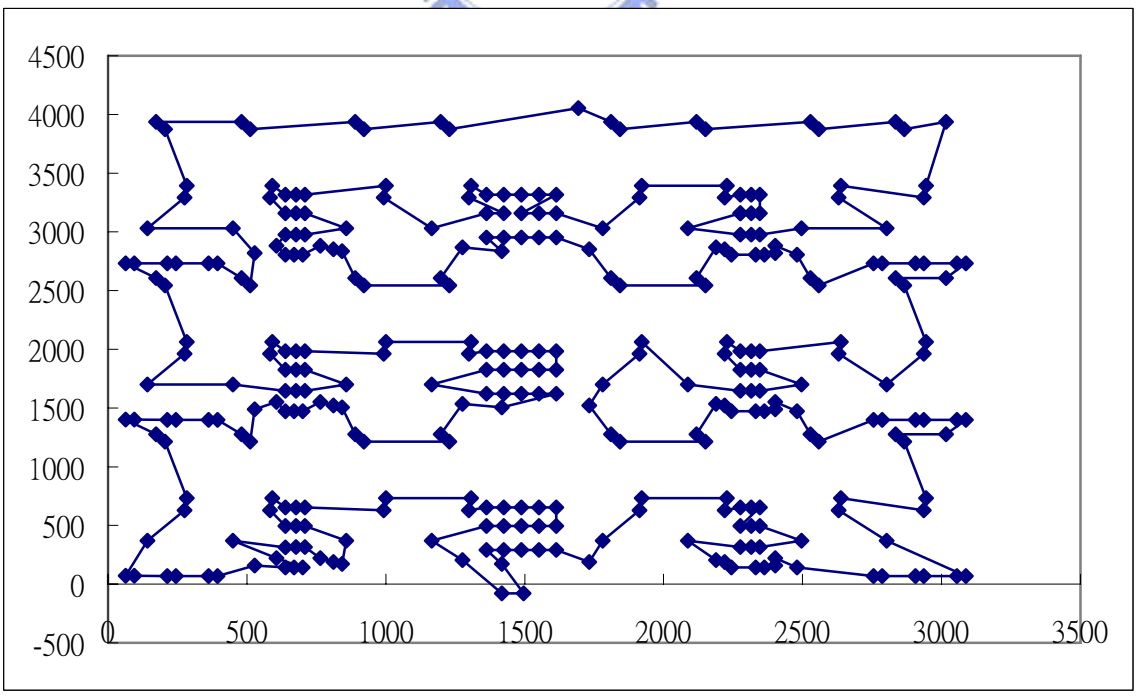


15. lin318.tsp

＊ 成本：42458

＊ 路線：1 2 6 7 10 11 15 21 22 29 103 12 20 23 28 30 31 32 33 27 24 19 16 17 18 25 26 36 37 42 41 43 46 52 53 58 57 54 51 47 44 104 40 316 317 49 45 48 50 55 56 59 105 62 63 70 69 74 75 81 73 76 80 86 79 77 72 64 67 68 71 78 82 83 84 85 91 92 96 97 101 102 93 89 90 98 99 95 94 100 207 206 202 201 197 196 88 87 190 189 188 187 183 176 173 172 65 66 61 60 164 210 167 168 169 177 182 184 191 185 181 178 174 175 179 180 186 195 194 198 203 204 200 199 205 312 311 307 306 302 301 193 192 295 294 293 292 288 281 278 277 170 171 166 165 269 266 265 260 258 255 259 250 143 144 140 139 242 241 240 239 232 231 313 230 233 238 243 237 234 229 226 227 228 235 236 247 246 314 254 257 251 252 253 256 262 263 268 261 264 267 315 272 273 280 279 284 285 291 290 286 283 274 282 287 289 296 303 299 300 308 309 310 305 304 298 297 276 275 271 270 318 249 248 245 244 224 223 214 215 219 218 213 222 225 119 118 221 220 217 216 212 211 109 110 114 113 108 117 125 128 133 138 132 129 124 121 122 123 130 131 141 142 147 146 148 151 157 158 163 162 159 156 152 149 209 150 153 155 160 161 154 145 38 39 35 34 137 136 135 134 127 126 208 120 14 13 116 115 112 111 107 106 4 5 9 8 3 1

＊ 路線圖：

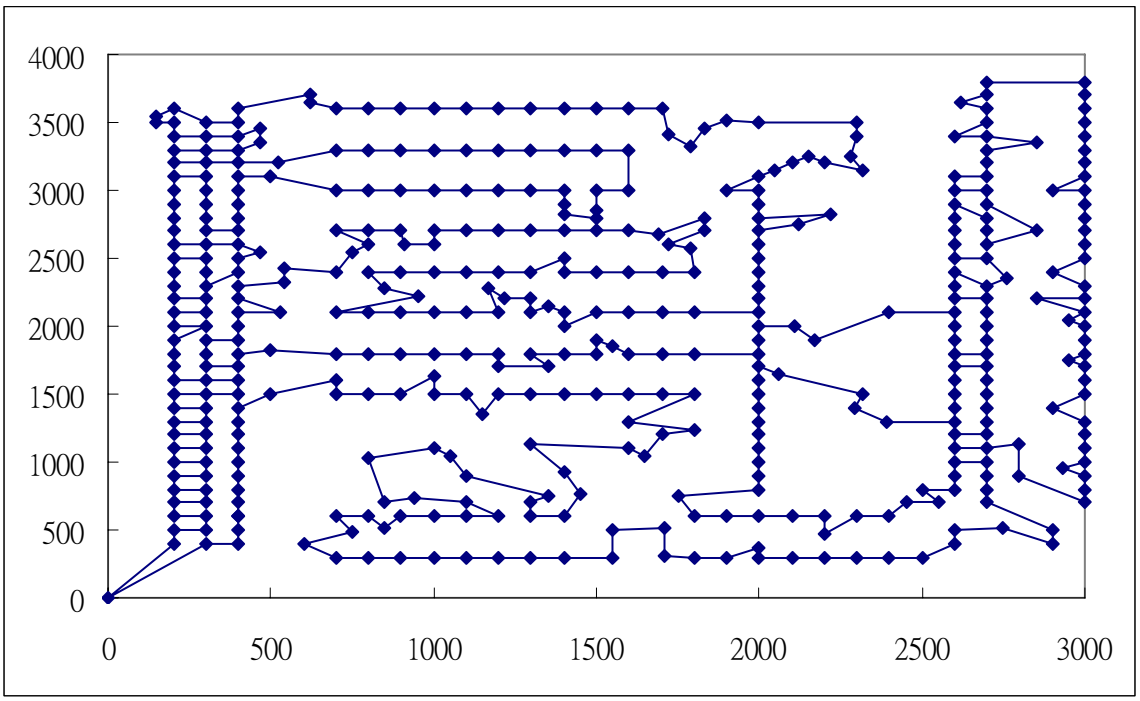


16. pcb442.tsp

＊ 成本：51750

＊ 路線：1 442 34 66 67 68 69 70 71 72 73 74 75 76 99 106 105 117 128 140 139
153 393 163 176 188 200 213 222 230 212 229 221 403 211 175 187
399 186 173 174 396 152 392 138 116 387 389 151 162 150 137 127
386 115 104 441 102 103 114 126 136 149 161 172 185 400 401 406
405 227 234 239 238 266 269 273 276 279 281 282 428 341 342 308
309 310 311 285 284 283 280 440 426 277 274 270 271 267 240 235
228 407 241 242 243 244 245 246 247 248 249 250 415 425 421 424
288 289 290 291 292 318 317 316 315 314 313 287 286 312 340 339
346 347 348 433 349 350 351 352 343 353 354 355 434 356 357 358
435 359 430 360 361 344 362 363 364 365 366 367 345 368 369 370
371 372 373 374 375 338 337 427 336 335 307 334 432 333 332 331
306 305 330 329 431 326 327 328 304 303 302 301 300 325 429 324
299 298 297 323 322 321 320 319 293 294 295 296 278 417 418 253
252 251 231 223 214 402 202 201 189 177 397 164 165 154 141 129
118 107 100 80 79 47 48 49 81 82 83 439 84 85 381 382 109 385 121
110 122 132 391 144 145 157 168 181 194 205 217 404 413 412 409
408 233 225 216 204 192 193 180 167 156 143 131 120 388 390 108
119 130 142 155 166 394 395 179 178 398 191 190 203 215 224 232
254 255 256 257 258 259 260 419 422 261 262 263 236 264 416 268
420 272 438 423 275 437 265 237 414 410 411 226 220 210 199 184
171 160 148 135 125 113 383 384 98 97 65 33 377 376 32 31 64 96 380
379 95 63 30 29 62 94 436 112 124 134 147 159 170 183 198 209 219
218 208 207 206 195 196 197 182 169 158 146 133 123 111 101 93 92
91 90 89 57 58 59 60 61 28 27 26 25 24 23 56 88 378 87 86 53 54 55 22
21 20 19 52 51 18 17 50 16 15 14 13 46 78 77 45 12 11 44 43 10 9 42
41 8 7 40 39 6 5 38 37 4 3 3635 2 1

＊ 路線圖：



附錄Ⅱ：Visual C++和 Mathematica5.0 程式碼

Visual C++

```
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <iomanip>
#include <fstream>
using namespace std;
#define INFI 99999
struct data { long dist; double rdist;char adj;};
void paraMain( double, double, int, int );
void nesrestInsertion( int, data **&, int *&, int *&, long & );
void nesrInit( int, data **&, int, int *&, int *&, long & );
int    closeSele( int, data **, int *, int * );
void insertion( int, data **&, int, int *&, int *&, long & );
void addIN( int, int, int *& );
void removeOUT( int, int, int *& );
void insertAfter( int, int, int *&, int );
void furthestInsertion( int, data **&, int *&, int *&, long & );
void furthInit( int, data **&, int, int *&, int *&, long & );
int    farthSele( int, data **, int *, int * );
void twoOpt( int, data **&, int *&, long &, long );
void revOrder( int, int *&, int, int );
void perturtation( int, data **&, int *&, double, double );
void orOpt( int, data **&, int *& );
void calAdj( int, data **, int * );
long calDist( int, data ** );
void coreExchange( int, data **, int *&, double, double, int, int );
int counts = 0;long totalDist = 0;long tempDist = 0;long origDist = 0;int distMax = 0;ofstream
dataOut( "parameterData.txt", ios::out );int main()
{ifstream para;para.open( "data/parameter.txt", ios::in );double c1[ 10 ];double h1[ 10 ];
int c1Length = 10;int h1Length = 3;int i;for ( i = 0; i < 10; i++ ) {para >> c1[ i ];h1[ i ] = c1[ i ];}
int j;for( i = 0; i < 10; i++ ) {for ( j = 0; j < 10; j++ ) {paraMain( c1[ i ], h1[ j ], c1Length, h1Length );
cout << "c1-" << c1[ i ]<< "(" << c1Length<< ")\th1-" << h1[ j ]<< "(" << h1Length<< ")\tOVER" <<
endl;}}dataOut.close();return 0;}
void paraMain( double c, double h, int cLength, int hLength )
{const int probSize = 48;clock_t start, end;double elapsed;ifstream fin;fin.open( "data/gr48.txt",
```

```
ios::in );ofstream fout;// fout.open( "Ssym_439.txt", ios::out ); data **node = new data *[ probSize +
1 ]; for ( int z= 0; z <= probSize; z++ ) node[ z ] = new data[ probSize + 1 ];int i, j;for ( i = 1; i <=
probSize; i++ ) {for( j = 1; j <= probSize; j++ ) {fin >> node[ i ][ j ].dist; if ( node[ i ][ j ].dist >
distMax )distMax = node[ i ][ j ].dist;node[ i ][ j ].adj = '0';}}
fin.close();
int *inSubtour = new int[ probSize + 1 ]; for ( i = 0; i <= probSize; i++ )inSubtour[ i ] = 0;
int *outSubtour = new int[ probSize + 1 ]; for ( i = 0; i <= probSize; i++ )outSubtour[ i ] = i;/*for ( i =
0; i <= probSize; i++ )cout << setw( 3 ) << inSubtour[ i ] << " ";cout << endl;for ( i = 0; i <= probSize;
i++ )cout << setw( 3 ) << outSubtour[ i ] << " ";cout << endl;*/cout << endl;start =
clock();furthestInsertion( probSize, node, inSubtour, outSubtour, totalDist );tempDist = totalDist;do {if
( tempDist < totalDist )totalDist = tempDist;twoOpt( probSize, node, inSubtour,
tempDist,totalDist );//cout << "temp distance is : " << tempDist << endl;} while ( tempDist <
totalDist );//         cout << "total distance after 2-opt improvement is : " << totalDist << endl;
//       cout << "============================" <<endl;coreExchange( probSize, node, inSubtour,
c, h, cLength, hLength );//      cout << "================================" << endl;
//cout << "total distance after algorithm is : " << totalDist << endl;end = clock();    elapsed = ( (double)
(end - start)) / CLOCKS_PER_SEC;dataOut << "c1-" << c<< "(" << cLength<< ")\th1-" << h
<< "(" << hLength<< ")\tSOLUTION : " << totalDist << endl;for ( i = 0; i <= probSize; i++ )dataOut
<< setw( 3 ) << inSubtour[ i ] << " ";dataOut << endl;dataOut << "The CPU time of this algorithm is :
" << elapsed << endl << endl;/*for ( int x = 1; x <= probSize; x++ ) {for ( int y = 1; y <= probSize;
y++ )fout << setw( 4 ) << node[ x ][ y ].adj << " ";fout << endl;}*/       //       cout << "\ncounts : " <<
counts << endl;}
// ============================== 最遠點插入 ========================== //
void furthestInsertion( int n, data **&work, int *&IN, int *&OUT, long &DIST )
{int insertNode;int ct = 1;int start = 1; addIN( n, start, IN );removeOUT( n, start, OUT );
//       cout << "initial node is : " << start << endl;       furthInit( n, work, start, IN, OUT, DIST );do {
insertNode = farthSele( n, work, IN, OUT ); //     cout << "insert node is : " << insertNode << endl;
insertion( n, work, insertNode, IN, OUT, DIST );ct++;/*for ( int x = 1; x <= n; x++ ) {for ( int y = 1; y
<= n; y++ )cout << setw( 3 ) << work[ x ][ y ].adj << " ";cout << endl;}*/} while ( OUT[ 1 ] !=
0 );IN[ 0 ] = IN[ n ];/*   for ( int i = 1; i <= n; i++ )cout << setw( 3 ) << IN[ i ] << " ";cout << endl;
for ( i = 1; i <= n; i++ )cout << setw( 3 ) << OUT[ i ] << " ";cout << endl;cout << "cout : " << ct <<
endl;*/ //     cout << "Total distance after Furthest Insertion is : " << DIST << endl;}
// ================= 選擇初始最大的 subtour ============================ //
void furthInit( int n, data **&work, int st, int *&IN, int *&OUT, long &DIST ){int tempNode;long
tempDist = 0;for( int i = 1; i <=n; i++ ) {if ( OUT[ i ] != 0 ) {if ( work[ st ][ OUT[ i ] ].dist * 2
>tempDist ) {tempDist = work[ st ][ OUT[ i ] ].dist * 2;tempNode = OUT[ i ];}}else break;}addIN( n,
tempNode, IN );removeOUT( n, tempNode, OUT );DIST += tempDist;//        cout << "insert node is : "
<< tempNode << endl;work[ st ][ tempNode ].adj = '1';work[ tempNode ][ st ].adj = '1';/*for ( i = 1; i
```

```
<= n; i++ )cout << setw( 3 ) << IN[ i ] << " ";cout << endl;*/}
// ========================= 選擇離 subtour 最遠的點========================== //
int farthSele( int n, data **work, int *IN, int *OUT )
{int tempNode;long tempDist = 0;for ( int i = 1; i <= n; i++ ) {if ( IN[ i ] != 0 ) {for ( int j = 1; j <= n;
j++ ) {if ( OUT[ j ] != 0 ) {if ( work[ IN[ i ] ][ OUT[ j ] ].dist > tempDist ) {tempDist =
work[ IN[ i ] ][ OUT[ j ] ].dist;tempNode = OUT[ j ];}}else break;}}else break;}return tempNode;}
// ============================== 插入所選擇的點=========================//
void insertion( int n, data **&work, int insert, int *&IN, int *&OUT, long &DIST )
{int tempI;int tempJ;long tempDist = INFI;for ( int i = 1; i <= n; i++ ) {if ( IN[ i ] != 0 ) {for ( int j = 1;
j <= n; j++ ) {if ( IN[ j ] != 0 ) {if ( i != j ) {if ( work[ IN[ i ] ][ IN[ j ] ].adj == '1' ) {if
( work[ IN[ i ] ][ insert ].dist+ work[ insert ][ IN[ j ] ].dist- work[ IN[ i ] ][ IN[ j ] ].dist < tempDist )
{tempDist = work[ IN[ i ] ][ insert ].dist+ work[ insert ][ IN[ j ] ].dist-
work[ IN[ i ] ][ IN[ j ] ].dist;tempI = IN[ i ];tempJ = IN[ j ];}}}}else break;}}else break;}
insertAfter( n, insert, IN, tempI );removeOUT( n, insert, OUT );DIST +=
tempDist;work[ tempI ][ insert ].adj = '1';work[ insert ][ tempJ ].adj = '1';work[ tempI ][ tempJ ].adj =
'0';/*for ( i = 1; i <= n; i++ )cout << setw( 3 ) << IN[ i ] << " ";cout << endl;*/}
void addIN( int n, int add, int *&IN )
{for ( int i = 1; i <= n; i++ ) {if ( IN[ i ] == 0 ) {IN[ i ] = add;IN[ 0 ] = add;break;}}}
void removeOUT( int n, int out, int *&OUT )
{int i, j;for ( i = 1; i <= n; i++ ) {if ( OUT[ i ] == out ) {//cout << "i : " << i << endl;j = i;}/*for ( int j =
n; j >= 1; j-- ) {if ( OUT[ j ] != 0 ) {OUT[ i ] = OUT[ j ];OUT[ j ] = 0;break;}}*/}do {OUT[ j ] =
OUT[ j+1 ];j++;} while ( j <= n );int z;for ( z = n; z >= 1; z-- ) {if ( OUT[ z ] != 0 ) {//cout << "z : " <<
z << endl;OUT[ z ] = 0;break;}}/*  for ( i = 0; i <= n; i++ )cout << setw( 3 ) << OUT[ i ] << " ";cout
<< endl;*/}
void insertAfter( int n, int add, int *&IN, int I )
{for ( int i = 1; i <= n; i++ ) {if ( IN[ i ] == I ) {IN[ 0 ] = IN[ i + 1 ];IN[ i + 1 ] = add;
I = IN[ i + 1 ];add = IN[ 0 ];}}}
// ======================== 2-opt 改善 ================================== //
void twoOpt( int n, data **&work, int *&IN, long &DIST, long level )
{long tempDist;long temp = INFI;int x, y;
for ( int i = 0; i <= n - 3; i++ ) {for ( int j = i + 2; j <= n - 1; j++ ) {if ( i == 0 && j == n -
1 )continue;tempDist = work[ IN[ i ] ][ IN[ j ] ].dist+ work[ IN[i+1] ][ IN[j+1] ].dist-
work[ IN[ i ] ][ IN[i+1] ].dist- work[ IN[ j ] ][ IN[j+1] ].dist;if ( tempDist == 0 )
continue;else {if ( tempDist < temp ) {temp = tempDist;x = i;y = j;counts++;}}}}if ( DIST + temp <
level ) {swap( IN[ x + 1 ], IN[ y ] ); //if ( ( y - (x+1) ) <= ( (n+(x-1)) - (y+2) ) ) {
if ( x + 2 < y - 1 ) revOrder( n, IN, x+2, y-1 );IN[ 0 ] = IN[ n ]; /*else {if ( ( y+2 ) < ( n+(x-1) ) )
        revOrder( n, IN, y+2, n+(x-1) ); IN[ 0 ] = IN[ n ]; }}*/calAdj( n, work, IN );DIST =
calDist( n,work );}
```

```
void revOrder( int n, int *&work, int st, int end )
{int X, Y;for ( int i = 0; i <= n - 2; i++ ) {if ( st+i < end-i ) {if ( (st+i) % n == 0 )X = ( st + i );
elseX = (st+i) % n;if ( (end-i) % n == 0 )Y = (end - i);elseY = (end-i) % n;swap ( work[ X ],
work[ Y ] );}else break;}}
void calAdj( int n, data **work, int *IN )
{for ( int i = 1; i <= n; i++ ) {for ( int j = 1; j <= n; j++ )work[ i ][ j ].adj = '0';}for ( i = 0; i <= n - 1;
i++ )work[ IN[ i ] ][ IN[ i+1 ] ].adj = '1';}
long calDist( int n, data **work )
{long DIST = 0;for ( int x = 1; x <= n; x++ ) {for ( int y = 1; y <= n; y++ ) {if ( work[ x ][ y ].adj ==
'1' )DIST += work[ x ][ y ].dist;}}return DIST;}
// ============================== 擾動 ================================== //
void perturtation( int n, data **&work, int *&IN, double c1, double h1 ){int i, j;for ( i = 1; i <= n; i++ )
{for( j = 1; j <= n; j++ ) {work[ i ][ j ].rdist = static_cast<double>( work[ i ][ j ].dist ) / distMax;}}for
( i = 1; i <= n; i++ ) {for ( j = 1; j <= n; j++ ) {if ( work[ i ][ j ].rdist < c1    &&    i != j )
{work[ i ][ j ].rdist += h1;}//cout << work[ i ][ j ].rdist << " ";}//   cout << endl;}orOpt( n, work, IN );}
// ========================== or-opt 改善 ================================ //
void orOpt( int n, data **&work, int *&IN )
{//     double tempDist;
// ================================= p = 3 ================================//
int i, j, ii;for ( i = 1; i <= n-3; i++ ) {for ( j = 1; j <= n; j++ ) {if ( j == i-1    ||    j == i    ||    j == i+1    ||
j == i+2 )continue;else { if ( j == n ) {if ( ( work[ IN[ j ] ][ IN[ i ] ].rdist+
work[ IN[i+2] ][ IN[ 1 ] ].rdist- work[ IN[ j ] ][ IN[ 1 ] ].rdist- work[ IN[i-1] ][ IN[ i ] ].rdist-
work[ IN[i+2] ][ IN[i+3] ].rdist+ work[ IN[i-1] ][ IN[i+3] ].rdist ) < -0.000001 ) {int *tempI = new
int[ 3 + 1 ];tempI[ 1 ] = IN[    i    ];tempI[ 2 ] = IN[ i+1 ];tempI[ 3 ] = IN[ i+2 ];
for ( ii = i+3; ii <= j; ii++ )IN[ ii-3 ] = IN[ ii ];IN[ j-2 ] = tempI[ 1 ];IN[ j-1 ] = tempI[ 2 ];IN[    j    ] =
tempI[ 3 ];IN[    0    ] = IN[    j    ];delete[] tempI;}} else {if ( ( work[ IN[ j ] ][ IN[ i ] ].rdist+
work[ IN[i+2] ][ IN[j+1] ].rdist- work[ IN[ j ] ][ IN[j+1] ].rdist- work[ IN[i-1] ][ IN[ i ] ].rdist-
work[ IN[i+2] ][ IN[i+3] ].rdist+ work[ IN[i-1] ][ IN[i+3] ].rdist ) < -0.000001 ) {if ( i < j ) {int *tempI
= new int[ 3 + 1 ];tempI[ 1 ] = IN[    i    ];tempI[ 2 ] = IN[ i+1 ];tempI[ 3 ] = IN[ i+2 ]; for ( ii = i+3; ii
<= j; ii++ )IN[ ii-3 ] = IN[ ii ];IN[ j-2 ] = tempI[ 1 ];IN[ j-1 ] = tempI[ 2 ];IN[    j    ] =
tempI[ 3 ];delete[] tempI;} else {int *tempI = new int[ 3 + 1 ];tempI[ 1 ] = IN[    i    ];tempI[ 2 ] =
IN[ i+1 ];tempI[ 3 ] = IN[ i+2 ];for ( ii = i-1; ii >= j+1; ii-- )IN[ ii+3 ] = IN[ ii ];IN[ j+1 ] =
tempI[ 1 ];IN[ j+2 ] = tempI[ 2 ];IN[ j+3 ] = tempI[ 3 ];delete[] tempI;}}}}}}
// ================================= p = 2 ================================//
for ( i = 1; i <= n-2; i++ ) {for ( j = 1; j <= n; j++ ) {if ( j == i-1    ||    j == i    ||    j == i+1 )continue;else
{if ( j == n ) {if ( ( work[ IN[ j ] ][ IN[ i ] ].rdist + work[ IN[i+1] ][ IN[ 1 ] ].rdist -
work[ IN[ j ] ][ IN[ 1 ] ].rdist- work[ IN[i-1] ][ IN[ i ] ].rdist- work[ IN[i+1] ][ IN[i+2] ].rdist+
work[ IN[i-1] ][ IN[i+2] ].rdist ) < -0.000001 ) {int *tempI = new int[ 2 + 1 ];tempI[ 1 ] = IN[    i    ];
```

```
tempI[ 2 ] = IN[ i+1 ];for ( ii = i+2; ii <= j; ii++ )IN[ ii-2 ] = IN[ ii ]; IN[ j-1 ] = tempI[ 1 ];IN[    j    ] =
tempI[ 2 ];IN[    0    ] = IN[    j    ];delete[] tempI;}} else {if ( ( work[ IN[ j ] ][ IN[ i ] ].rdist+
work[ IN[i+1] ][ IN[j+1] ].rdist- work[ IN[ j ] ][ IN[j+1] ].rdist- work[ IN[i-1] ][ IN[ i ] ].rdist-
work[ IN[i+1] ][ IN[i+2] ].rdist+ work[ IN[i-1] ][ IN[i+2] ].rdist ) < -0.000001 ) {if ( i < j ) {int *tempI
= new int[ 2 + 1 ];tempI[ 1 ] = IN[    i    ];tempI[ 2 ] = IN[ i+1 ];for ( ii = i+2; ii <= j; ii++ )IN[ ii-2 ] =
IN[ ii ];IN[ j-1 ] = tempI[ 1 ];IN[    j    ] = tempI[ 2 ];delete[] tempI;} else {int *tempI = new int[ 2 +
1 ];tempI[ 1 ] = IN[    i    ];tempI[ 2 ] = IN[ i+1 ]; for ( ii = i-1; ii >= j+1; ii-- )IN[ ii+2 ] = IN[ ii ];
IN[ j+1 ] = tempI[ 1 ];IN[ j+2 ] = tempI[ 2 ];delete[] tempI;}}}}}}
// ================================== p = 1==============================//
for ( i = 1; i <= n-1; i++ ) {for ( j = 1; j <= n; j++ ) {if ( j == i-1    ||    j == i )continue;
else {if ( j == n ) {if ( ( work[ IN[ j ] ][ IN[ i ] ].rdist+ work[ IN[ i ] ][ IN[ 1 ] ].rdist-
work[ IN[ j ] ][ IN[ 1 ] ].rdist- work[ IN[i-1] ][ IN[ i ] ].rdist- work[ IN[ i ] ][ IN[i+1] ].rdist+
work[ IN[i-1] ][ IN[i+1] ].rdist ) < -0.000001 ) {int *tempI = new int[ 1 + 1 ];tempI[ 1 ] =
IN[    i    ];for ( ii = i+1; ii <= j; ii++ )IN[ ii-1 ] = IN[ ii ];IN[    j    ] = tempI[ 1 ];IN[    0    ] =
IN[    j    ];delete[] tempI;}} else {if ( ( work[ IN[ j ] ][ IN[ i ] ].rdist+ work[ IN[ i ] ][ IN[j+1] ].rdist-
work[ IN[ j ] ][ IN[j+1] ].rdist- work[ IN[i-1] ][ IN[ i ] ].rdist- work[ IN[ i ] ][ IN[i+1] ].rdist+
work[ IN[i-1] ][ IN[i+1] ].rdist ) < -0.000001 ) {if ( i < j ) {int *tempI = new int[ 1 + 1 ];tempI[ 1 ] =
{int *tempI = new int[ 1 + 1 ];tempI[ 1 ] = IN[    i    ]; for ( ii = i-1; ii >= j+1; ii-- )IN[ ii+1 ] =
IN[ ii ];IN[ j+1 ] = tempI[ 1 ];delete[] tempI;}}}}}}}
calAdj( n, work, IN );
totalDist = calDist( n, work );//        cout << "total distance after perturtation is : " << totalDist <<
endl;}
void coreExchange( int n, data **work, int *&IN, double c, double h, int cLength, int hLength )
{double c1, h1;int i, j;for ( i = 0; i < cLength; i++ ) {origDist = totalDist;int *origIN = new int[ n +
1 ];int x;for ( x = 0; x <= n; x++ )origIN[ x ] = IN[ x ];/*for ( x = 0; x <= n; x++ )cout << setw( 3 ) <<
IN[ x ] << " ";cout << endl;for ( x = 0; x <= n; x++ )cout << setw( 3 ) << origIN[ x ] << " ";cout <<
endl;*/for ( j = 0; j < hLength; j++ ) {c1 = c * ( 1 - static_cast<double>(i)/(cLength) );h1 = h * ( 1 -
static_cast<double>(j)/(hLength-1) );/*cout << "c1 : " << c1 << "\n"<< "h1 : " << h1 << endl;*/
perturtation( n, work, IN, c1, h1 );/*int x;for ( x = 0; x <= n; x++ )cout << setw( 3 ) << IN[ x ] << "
";cout << endl;*/tempDist = totalDist;do {if ( tempDist < totalDist )totalDist = tempDist;
twoOpt( n, work, IN, tempDist, totalDist );//cout << "temp distance is : " << tempDist << endl;} while
( tempDist < totalDist );//cout << "total distance after 2-opt improvement is : " << totalDist << endl;/*
for ( x = 0; x <= n; x++ )cout << setw( 3 ) << IN[ x ] << " ";cout << endl;*/}/*cout << "origDist    :
"<< origDist << "\n"<< "totalDist : " << totalDist << endl;for ( x = 0; x <= n; x++ )cout << setw( 3 )<<
IN[ x ] << " ";cout << endl;for ( x = 0; x <= n; x++ )cout << setw( 3 ) << origIN[ x ] << " ";cout <<
endl;*/if ( origDist < totalDist ) {totalDist = origDist;for ( int x = 0; x <= n; x++ )IN[ x ] = origIN[ x ];}
else continue;delete[] origIN;}}
```

Mathematica5.0

```
ClearAll
<< LinearAlgebra`MatrixManipulation`
n = 70;ReadList["C:\\論文資料\\st70.txt", Number];a = Partition[ReadList["C:\\論文資料\\st70.txt",
Number], 3];dat = MatrixForm[a];b = Array[v1, {n, n}];For[i = 1, i ≤ n, i++, For[j = 1, j ≤ n, j++,
If[i ≠ j,v1[i, j] = Round[Sqrt[(dat[[1, i, 2]] dat[[1, j, 2]])^2 + (dat[[1, i, 3]] - dat[[1, j, 3]])^2]], v1[i, j]
= 0]]];dist = b;dis1 = dist;For[i = 1, i ≤ n, i++, For[j = 1, j ≤ n, j++, If[dis1[[i, j]] == 0, dis1[[i, j]] =
-∞,]]];Do[dis1[[i, 1]] = -∞, {i, n}];seq = {1};a = Take[dis1, 1];p = Position[a, Max[a]];b = p[[1, 1]];
c = p[[1, 2]];seq = Append[seq, c];Do[dis1[[i, c]] = -∞, {i, n}];a = AppendColumns[Take[dis1, {1, 1}],
Take[dis1, {c, c}]];seq = {1, c, 1};For[e = 3, e ≤ n, e++,p = Position[a, Max[a]];b = p[[1, 1]];c = p[[1,
2]];Do[dis1[[i, c]] = -∞, {i, n}]; seq1 = Insert[seq, c, 2];improbj = 0;For[i = 1, i ≤ e, i++,
improbj = improbj + dist[[seq1[[i]], seq1[[i + 1]]]];]For[j = 3, j ≤ e, j++,seq2 = Insert[seq, c,
j];improbj1 = 0;For[i = 1, i ≤ e, i++,improbj1 = improbj1 + dist[[seq2[[i]], seq2[[i + 1]]]];]If[improbj ≤
improbj1, improbj = improbj; seq1 = seq1, improbj = improbj1; seq1 = seq2];];seq = seq1;a =
Take[dis1, {seq[[1]], seq[[1]]}];For[i = 1, i ≤ e - 1, i++, a = AppendColumns[a, Take[dis1, {seq[[i + 1]],
seq[[i + 1]]}]]]; ];
(*=== === === === === === === == Farthest Insert === === === === === = = = == === === ==*)
impr0 = Array[v0, {n, n}];For[j = 1, j ≤ n, j++, For[i = 1, i ≤ n, i++, v0[i, j] = Min[]]];iteration =
0;For[k = 0; i = 1, i < n, i++, k = k + dist[[seq[[i]], seq[[i + 1]]]];]; k = k + dist[[seq[[n]], seq[[1]]]];
For[z = -1, z < 0,For[i = 1, i ≤ n - 2, i++,For[j = i + 2, j ≤ n, j++,temp = dist[[seq[[i]], seq[[j]]]] +
dist[[seq[[i + 1]], seq[[j + 1]]]] - dist[[seq[[i]], seq[[i + 1]]]] - dist[[seq[[j]], seq[[j + 1]]]]; v0[i, j] =
temp]];If[Min[impr0] < 0, z = Min[impr0];k = k + z; iteration = iteration + 1;p = Position[impr0,
Min[impr0]];l = p[[1, 1]];m = p[[1, 2]];a = Take[seq, {l + 1, m}];c = Length[a];b = Drop[seq, {l + 1,
m}];For[j = 1, j ≤ c, j++,b = Insert[b, a[[j]], l + 1];seq = b;, z = 1]];
(*=== === === === === === === ==Two - Opt === === === ==== = = = === = == ===*)
LocalSolution = seq;LocalOpt = k;
(*=== === === === === === === === Setting soltion === === === = == === === === = = = ===*)
impr4 = Array[v4, {n, n}];For[j = 1, j ≤ n, j++, For[i = 1, i ≤ n, i++, v4[i, j] = Min[]]];
(*=== === === === === Matrix setting === === === = == === === === = = = === = ===*)
Tim = Timing[For[K = 0,K ≤ 14, K++, C1 = 0.156; C1 = (C1)*(1 - (K)/15); For[L = 0, L ≤ 2, L++, H1
= 0.156; H1 = H1*(1 - (L)/2);dist2 = dist;a = Max[dist2];dist2 = dist2/a;For[i = 1, i ≤ n, i++, For[j = 1, j
≤ n, j++, If[dist2[[i, j]] ≠0, If[dist2[[i, j]] ≥ C1, dist2[[i, j]] = dist2[[i, j]], dist2[[i, j]] = (dist2[[i, j]] +
H1),]]]];
(*=== === === === === === = ===Change date === === == = = = === = == === === === ==*)
For[k =0; i = 1, i < n, i++, k = k + dist2[[seq[[i]], seq[[i + 1]]]];]; k = k + dist2[[seq[[n]], seq[[1]]]];
For[i = 1, i ≤ n - 3, i++, For[j = 1, j ≤ n, j++, If[i == 1, temp = dist2[[seq[[j]], seq[[i]]]] + dist2[[seq[[
i + 2]], seq[[j + 1]]]] - dist2[[seq[[j]], seq[[j + 1]]]] - dist2[[seq[[n]], seq[[i]]]] - dist2[[seq[[i + 2]], seq[[
i + 3]]]] + dist2[[seq[[n]], seq[[i + 3]]]], temp = dist2[[seq[[j]], seq[[i]]]] + dist2[[seq[[i + 2]], seq[[j +
```

```
1]]]] - dist2[[seq[[j]],seq[[j + 1]]]] - dist2[[seq[[i - 1]], seq[[i]]]] - dist2[[seq[[i + 2]], seq[[i + 3]]]] +
dist2[[seq[[i - 1]], seq[[i + 3]]]]];If[i == 1; j == n, temp = 0,];If[j == i - 1, temp = 0,];If[j == i, temp =
0,];If[j == i + 1, temp = 0,];If[j == i + 2, temp = 0,];If[temp < -0.000001, a = Take[seq, {i, i + 2}]; b =
Drop[seq, {i, i + 2}];If[j < i, For[t =1, t ≤ 3, t++, b = Insert[b, a[[4 - t]], j + 1]], For[t = 1, t ≤ 3, t++, b =
Insert[b, a[[4 - t]], j - 2]]]; seq = b; seq[[n + 1]] = seq[[1]]; k = k + temp,];]];
iteration = iteration + 1;
(*=== === === === === ===== === Or - Opt(p = 3 Insert) === === === === === === ==*)
For[i = 1, i ≤ n - 2, i++,For[j = 1, j ≤ n, j++,If[i == 1, temp = dist2[[seq[[j]], seq[[i]]]] + dist2[[seq[[i +
1]], seq[[j + 1]]]] - dist2[[seq[[j]], seq[[j + 1]]]] - dist2[[seq[[n]], seq[[i]]]] - dist2[[seq[[i + 1]], seq[[i +
2]]]] + dist2[[seq[[n]], seq[[i + 2]]]], temp = dist2[[seq[[j]], seq[[i]]]] + dist2[[seq[[i + 1]],seq[[j + 1]]]]
- dist2[[seq[[j]], seq[[j + 1]]]] - dist2[[seq[[i - 1]], seq[[i]]]] - dist2[[seq[[i + 1]], seq[[i + 2]]]] +
dist2[[seq[[i - 1]], seq[[i + 2]]]]];If[i == 1; j == n, temp = 0, temp = temp];If[j == i - 1, temp = 0,];
If[j == i, temp = 0,];If[j == i + 1, temp = 0,];If[temp < -0.000001, a = Take[seq, {i, i + 1}]; b =
Drop[seq, {i, i + 1}];If[j < i,For[t = 1, t ≤ 2, t++,b = Insert[b, a[[3 - t]], j + 1]], For[t = 1, t ≤ 2, t++,b =
Insert[b, a[[3 - t]], j - 1]]]; seq = b; seq[[n + 1]] = seq[[1]]; k = k + temp,];]];
iteration = iteration + 1;
 (*=== === === === = === === === === === Or - Opt(p = 2 Insert) === === === === =====*)
For[i = 1, i ≤ n - 1, i++,For[j = 1, j ≤ n, j++,If[i == 1, temp =dist2[[seq[[j]], seq[[i]]]] + dist2[[seq[[i]],
seq[[j + 1]]]] - dist2[[seq[[j]], seq[[j + 1]]]] - dist2[[seq[[n]], seq[[i]]]] - dist2[[seq[[i]], seq[[i + 1]]]] +
dist2[[seq[[n]], seq[[i + 1]]]], temp = dist2[[seq[[j]], seq[[i]]]] + dist2[[seq[[i]], seq[[j + 1]]]] -
dist2[[seq[[j]], seq[[j + 1]]]] - dist2[[seq[[i - 1]], seq[[i]]]] - dist2[[seq[[i]], seq[[i + 1]]]] + dist2[[seq[[i
- 1]], seq[[i + 1]]]]];If[i == 1; j == n, temp = 0,];If[j == i - 1, temp = 0,];
If[j == i, temp = 0,];If[temp < -0.000001, a = Take[seq, {i, i}]; b = Drop[seq, {i, i}];If[j < i,
b = Insert[b, a[[1]], j + 1],b = Insert[b, a[[1]], j]]; seq = b; seq[[n + 1]] = seq[[1]]; k = k + temp,];]];
iteration = iteration + 1;
(*=== === === === === === Or - Opt(p = 1 Insert) === === === === ===== ==*)
For[k = 0; i = 1, i < n, i++, k = k + dist[[seq[[i]], seq[[i + 1]]]];]; k = k + dist[[seq[[n]], seq[[1]]]];
For[z = -1, z < 0,For[i = 1, i ≤ n - 2, i++,For[j = i + 2, j ≤ n, j++,temp = dist[[seq[[i]], seq[[j]]]] +
dist[[seq[[i + 1]], seq[[j + 1]]]] - dist[[seq[[i]], seq[[i + 1]]]] - dist[[seq[[j]], seq[[j + 1]]]];
v4[i, j] = temp]];If[Min[impr4] < 0, z = Min[impr4];k = k + z; iteration = iteration + 1;p =
Position[impr4, Min[impr4]];l = p[[1, 1]];m = p[[1, 2]];a = Take[seq, {l + 1, m}];c = Length[a];
b = Drop[seq, {l + 1, m}];For[j = 1, j ≤ c, j++,b = Insert[b, a[[j]], l + 1];seq = b;, z = 1]];
(*=== === === === === ===== == = Two - Opt === === === === === ===== === ==*)
]; If[LocalOpt > k, LocalOpt = k; LocalSolution = seq, k = LocalOpt; seq = LocalSolution];];];
Print[Tim];Print[""];Print["Total iteration times is: ", iteration];Print["The local optimal solutin is: ",
LocalOpt];Print[" And the sequence is: ", LocalSolution]
```