

國立交通大學

工業工程與管理學系

博士論文

運用混合演算法於田口動態特性之參數設計最佳化

Optimizing Parameter Design for Taguchi's Dynamic
Characteristics with Hybrid Algorithms

研究生：詹曉苓

指導教授：梁馨科 博士

中華民國九十五年八月

運用混合演算法於田口動態特性之參數設計最佳化

Optimizing Parameter Design for Taguchi's Dynamic
Characteristics with Hybrid Algorithms

研究生：詹曉苓

Student : Hsiao-Ling Chan

指導教授：梁馨科 博士

Advisor : Dr. Shing-Ko Liang

國立交通大學

工業工程與管理學系

博士論文



Submitted to Department of Industrial Engineering and Management

College of Management

National Chiao Tung University

In partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

In

Industrial Engineering

August 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年八月

運用混合演算法於田口動態特性之參數設計最佳化

研究生：詹曉苓

指導教授：梁馨科 博士

國立交通大學

工業工程與管理學系博士班

摘要

田口玄一所提倡的參數設計，主要目的是要決定產品或製程的因子水準等設定值，使其對雜音變數的敏感性最小。但多數侷限於單一品質特性之靜態特性問題，對於連續型參數無法求取真正的最佳參數解。

本研究應用混合演算法的求解程序，來協助解決田口動態特性的多品質特性問題。首先，利用類神經網路模擬出控制因子與反應值的關係，再利用啟發式演算法求取最佳參數設定值。對於多重品質特性最佳化問題，利用望想函數或指數望想函數綜合衡量多個品質特性，以克服田口方法無法考量多重品質特性同時最佳化的缺點。室內無線區域網路傳輸品質評估系統的實際案例，說明方法之可行性。本研究之結果，將可協助區域網路工程師減少網路規劃所耗費的時間與判斷上的難度。

關鍵詞：參數設計、動態特性、類神經網路、混合演算法、啟發式演算法。

Optimizing Parameter Design for Taguchi's Dynamic Characteristics with Hybrid Algorithms

Student: Hsiao-Ling Chan

Advisor: Dr. Shing-Ko Liang

Department of Industrial Engineering and Management

National Chiao Tung University

ABSTRACT

Parameter design is critical to enhancing a system's robustness by identifying specific control factor and their levels that make the system less sensitive to noise. Most engineers applied Taguchi methods to optimize parameter design. However, Taguchi methods can only obtain the discrete optimal solution. They cannot identify the real optimum when the parameter values are continuous. The multi-response problem is too difficult to be considered by engineers with limited statistical knowledge.

This research proposes a hybrid approach for combining neural networks and meta-heuristic algorithm to optimize the continuous parameter design problem. First, neural networks are used to simulate the relationship between the control factor values and corresponding responses. Second, meta-heuristic algorithm is employed to obtain the optimal parameter settings. The desirability function (or exponential desirability function) is utilized to transform the multiple responses into a single response. The real case of the propagation system evaluation of indoor wireless LAN is presented to demonstrate the practicability of the proposed procedure.

Keywords: Parameter Design, Dynamic Characteristic, Neural Network, Hybrid Algorithm, Meta-Heuristic Algorithm.

誌 謝

本論文得以順利付梓，首先感謝指導教授梁馨科博士的悉心指導與啟迪鼓勵。承蒙恩師在學問上不厭其煩的耐心解惑及待人處世等多方面的引導，使學生在課業與生活中都獲益匪淺，師恩浩瀚永銘於心。論文口試期間，感謝陳義揚教授、徐世輝教授、廖又生教授及陳安斌教授不辭辛勞審閱論文，給予諸多精闢的建議並逐字斧正，使本論文更臻完備與嚴謹，謹此致上無限的謝意。

此外，特別感謝明新科技大學前電算中心主任張玉山教授，及中華大學系統網路中心研究助理江漢清先生，不吝提供該校建置校園無線網路的寶貴經驗。

感謝大華技術學院資訊管理系的長官及所有同仁，在我進修博士學位過程中的諸多配合與協助，讓我能兼顧學業與工作，順利完成論文。

謝謝產業管理研究室的學弟們，在行政事務處理上提供充分的協助。

感謝一直陪伴在身邊鼓勵我的先生，悉心地照顧我及我們即將出世的女兒，使我的生活充滿歡笑與活力。在研究這條寂寞而漫長的道路，有他的相伴讓我不覺孤單。

最後謝謝親愛的雙親與弟妹，由於他們長久以來毫不保留的支持、關懷與包容，使我得以順利完成學業。

僅以本文獻給摯愛的家人及所有關心我的朋友，願你們平安喜樂！

詹曉苓 謹誌

中華民國九十五年八月

於新竹交通大學

目 次

中文摘要.....	I
英文摘要.....	II
誌謝.....	III
目次.....	IV
表目次.....	VI
圖目次.....	VII
第一章 緒論.....	1
1.1 研究動機與背景.....	1
1.2 研究目的.....	1
1.3 論文架構.....	2
第二章 文獻探討.....	3
2.1 田口方法動態特性.....	3
2.2 類神經網路.....	6
2.3 分散式搜尋法.....	9
2.4 基因演算法.....	14
2.5 模擬退火法.....	15
2.6 望想函數.....	15
2.7 混合演算法.....	17
2.7.1 結合類神經網路與分散式搜尋法.....	18
2.7.2 結合類神經網路與基因演算法.....	18
2.7.3 結合類神經網路與模擬退火法.....	19
第三章 研究方法.....	20
3.1 整合田口方法與倒傳遞類神經網路演算法.....	20
3.2 整合倒傳遞類神經網路與基因演算法.....	21
3.3 整合倒傳遞類神經網路與分散式搜尋法.....	22

3.4 整合倒傳遞類神經網路與模擬退火法.....	25
第四章 室內無線區域網路傳輸品質評估系統之實證分析.....	28
4.1 問題描述.....	28
4.2 研究設計之架構.....	30
4.3 利用田口方法求解.....	33
4.4 利用混合演算法求解.....	36
4.5 確認實驗與比較.....	39
第五章 結論與建議.....	41
5.1 結論.....	41
5.2 建議.....	42
參考文獻.....	43
附錄.....	50



表 目 次

表 4-1 實驗的控制因子與水準.....	31
表 4-2 實驗結果.....	32
表 4-3 SN 比的回應值表.....	34
表 4-4 β 的回應值表.....	34
表 4-5 實驗的變異數分析表(SN).....	35
表 4-6 實驗的變異數分析表 (β).....	36
表 4-7 類神經網路訓練結果.....	37
表 4-8 基因演算法的執行結果 (指數望想函數).....	38
表 4-9 基因演算法的執行結果 (望想函數).....	38
表 4-10 參數設定值.....	38
表 4-11 四種方法求得各控制因子最佳值.....	39
表 4-12 確認實驗數據.....	39



圖 目 次

圖 2-1 田口問題之動態系統參數圖.....	3
圖 2-2 動態系統的理想函數圖.....	4
圖 2-3 動態特性的參數設計流程圖.....	6
圖 2-4 類神經網路架構：以倒傳遞類神經網路為例.....	8
圖 2-5 二維空間參考集合.....	11
圖 2-6 重劃路徑.....	11
圖 2-7 分散式搜尋法的流程圖.....	14
圖 3-1 整合倒傳遞類神經網路與分散式搜尋法流程圖.....	22
圖 3-2 整合倒傳遞類神經網路與基因演算法流程圖.....	24
圖 3-3 整合倒傳遞類神經網路與模擬退火法流程圖.....	26
圖 4-1 教學大樓平面圖.....	29
圖 4-2 建立目標函數.....	30
圖 4-3 SN 比的回應圖.....	34
圖 4-4 β 的回應圖.....	35



第一章 緒論

1.1 研究動機與背景

參數設計主要目的是要決定產品或製程的因子水準等設定值，使其對雜音變數的敏感性最小。由於田口方法(Taguchi methods)強調在設計階段即考慮品質觀念，依成本效益的觀念找出最佳的因子水準組合，且計算方法簡單、試驗次數少，故成為工業上最常被使用的參數設計工具之一。雖然線上工程人員時常運用田口方法來解決最佳化問題，但由於田口方法在實際應用上只能找出原先設定好的因子水準值，無法處理因子水準為連續型的問題。因此，工程人員在進行實驗時，通常是根據工程知識訂出大約的因子水準，而這些水準值並不一定是最佳值。針對田口方法無法處理因子水準為連續型問題，有兩位學者提出以反應曲面法(Response Surface Methodology; RSM)來解決這個問題(Vining 與 Mylers, 1990)。然而，也有學者提出，若實驗設計無法蒐集到正確的資料，則反應曲面依然無法正確有效的預估參數值(Montgomery, 2004)。

本研究基於品質工程中，以直交表配置之實驗數據轉換為訊號雜音比，來尋求最適組合之設計參數與水準的技法上，仍有藉助主觀決策的缺憾，故應用混合演算法(hybrid algorithms)的求解程序來解決連續型參數設計問題。首先，利用類神經網路(neural network)模擬出控制因子與反應值(responses)的關係，再利用啟發式演算法(meta-heuristic algorithm)求取最佳參數設定值。對於多重品質特性(multiple responses)最佳化問題，利用望想函數(desirability function)或指數望想函數(exponential desirability function)綜合衡量多個品質特性，以克服田口方法無法考量多重品質特性同時最佳化的缺點。最後，本研究以室內無線區域網路傳輸品質評估系統為實證案例來說明本研究所應用方法之可行性。

1.2 研究目的

本研究使用混合演算法的程序，考量在多重品質特性下，解決田口動態特性的參數設計最佳化問題。

1. 降低變異性(variability)與調整靈敏度(sensitivity)為田口動態特性主要的兩個最佳化程序，將變異性與靈敏度同時最佳化為本研究目的之一。
2. 本研究使用望想函數與指數望想函數將多重品質特性轉換為單一品質特

性(single response)。望想函數(或指數望想函數)的優點是容易了解，而且使用者可以根據反應值的重要性加以調整權重值。

3. 本研究利用混合演算法解決田口方法無法處理因子水準為連續型的問題。田口方法雖然由於計算簡單、試驗次數少等原因，使得它成為工業上最常被使用的參數設計工具之一。但是田口方法在實際應用上仍有一些不足，例如無法處理因子水準為連續型數值的問題。換言之，工程人員在進行實驗時，通常僅根據工程知識大約訂出幾個因子水準，但這幾個水準值並不一定是最佳的。因此，本研究利用混合演算法，以倒類神經網路建立目標函數，再結合啟發式演算法最佳化其連續型的參數設計問題。
4. 實例研究是以台灣某技術學院之室內無線區域網路傳輸品質評估系統為例。經由實證研究證實本研究方法的可行性，本研究也比較實驗中各種混合演算法的有效性與優缺點。

1.3 論文架構

本研究共分為五章，第一章為緒論，說明研究動機與背景、研究目的、論文架構等。第二章介紹田口方法動態特性相關文獻，及論文中所使用的啟發式演算法。包含類神經網路、分散式搜尋法、基因演算法、模擬退火法、望想函數及混合演算法之文獻探討。第三章利用四個小節說明本研究所應用之四種混合演算法如何執行。第四章則提出「室內無線區域網路傳輸品質評估系統案例」之實證分析。第五章為本論文之結論與建議。

第二章 文獻探討

本研究使用混合演算法來解決田口動態特性案例的問題。本節介紹田口方法動態特性相關文獻，與論文中所使用的啟發式演算法，包含類神經網路、分散式搜尋法、基因演算法和模擬退火法，以及各種啟發式演算法與類神經網路所結合的混合演算法相關文獻。

2.1 田口方法動態特性

田口玄一博士在1949年發展一透過實驗進行系統參數最佳化設計的方法。田口方法的主要目標為提昇品質使其能達到目標值並減少變異，該法強調在產品與製程設計時就應考慮品質問題。田口方法發展初期以探討靜態系統方面的文獻較多(Logothetis 與 Haigh,1988; 蕭綱衡, 民79)，然而針對工業界的實際需求，田口方法在動態系統方面的應用也日益增多(Taguchi, 1990)。國內學者提出利用多項迴歸模式解決田口方法動態系統的問題(陳明宏, 民83; 洪饒峰, 民85)。也有學者提出以柔性演算法為基 (soft computing-based)的求解程序來解決田口方法動態特性問題，並利用模擬實驗結果證明方法論的有效性(張旭華, 民88)。針對血糖測試片的製程改善，學者也針對田口方法動態特性的問題，成功的解決製程不良率偏高的現象，並建議適當之製程參數 (Su 等, 2005; Chan 與 Liang, 2005)。

所謂靜態系統是指品質特性的目標值只有一個。動態系統中除了控制因子與雜音因子外，還較靜態系統多了信號因子 (signal factor)，品質特性的目標值會隨著信號因子水準的不同而有所改變(Phadke, 1989; Fowlkes 與 Creveling, 1995)。田口問題之動態系統參數圖如下(蘇朝墩, 民91)：

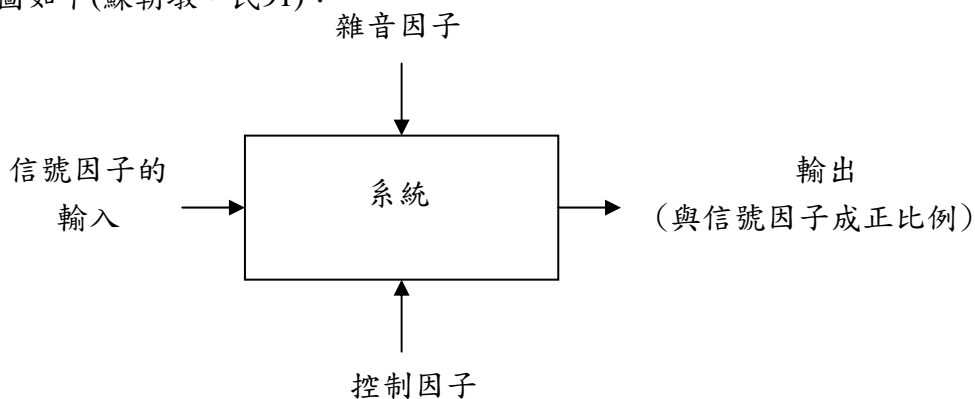


圖 2-1 田口問題之動態系統參數圖

信號因子水準可由產品設計者或使用者自行設定以達到所需的產品功能，亦即可任意調整信號因子而降低雜音因子的影響以使得品質變異最小。田口動態系統在理想狀態下，可將品質特性與信號因子的關係視為線性，如公式(2-1)所示：

$$Y_{ijk} = \beta_i M_j + e_{ijk} \quad (2-1)$$

其中 Y_{ijk} 為品質特性，假設有 i 個控制因子， j 個信號因子與 k 個雜音因子； M 為信號因子； β_i 為斜率且 $\sigma_i^2 = \text{Var}(e_{ijk})$ ，代表系統的靈敏度； e_{ijk} 為隨機誤差項。圖2-2為零點比例式的動態系統理想函數圖(蘇朝墩，民91)。

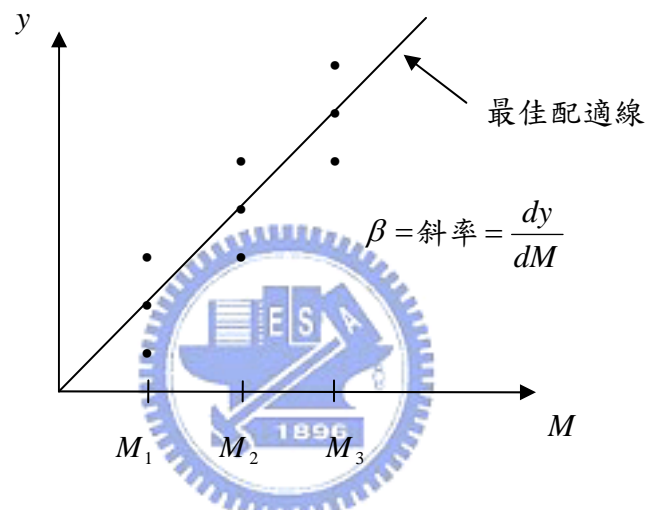


圖2-2 動態系統的理想函數圖

在動態系統中，評量系統績效所使用的特徵有以下三種(Fowlkes 與 Creveling, 1995)：

1. 靈敏度：當信號因子改變一單位時所造成品質特性的變化量稱為靈敏度或敏感度，亦即為公式(2-1)中之斜率 β 。靈敏度的理想值是因個別的動態系統要求而異，例如一般工業製造系統大多數希望靈敏度愈大愈好；量測系統方面則認為靈敏度的理想值為1；而對於某些輸入信號值較難更動的射出成型製造系統而言，則可利用調整靈敏度的方式來改變輸出值。
2. 線性關係 (linearity)：在動態田口方法中，信號因子與品質特性間呈線性關係是非常重要的，其目的是希望若品質特性與信號因子間存在著線性關係，除了可簡化分析過程外，並能藉由調整信號輸入更容易達到所要求之品質輸出目標。
3. 變異性：在動態系統中，我們希望在每個信號因子水準下，品質特性都盡可能地

接近目標值，並希望降低在每一信號因子水準下品質特性對於其目標值的變異。

為了能量化以上的這三種評估準則，動態系統中用信號雜音比（signal to noise ratio, SN 比）與靈敏度作為衡量品質變異與敏感度的指標，此二指標之表示方法如下 (Fowlkes 與 Creveling, 1995)：

$$SN_i = 10 \cdot \log_{10} \left(\frac{\beta_i^2}{\sigma_i^2} \right) \quad (2-2)$$

$$S_i = 10 \cdot \log_{10} \beta_i^2 \quad (2-3)$$

其中 σ_i^2 通常以均方誤(mean square error, MSE)來估計， β_i^2 則是以最小平方法估計而得。

由於動態系統有信號因子用以調整輸出值，所以根據以上兩個評估指標，田口博士發展出一個含有信號因子的兩階段最適化法(Taguchi, 1990)：

- 1.減少伴隨著線性函數關係的變異：找出影響SN比的顯著因子，再找出這些因子的最佳水準設定值，以達成一個穩定的機能。
- 2.調整信號因子以使輸出值能調整到目標值：利用調整因子來調整線性關係的斜率，以達到所要求的靈敏度。

執行動態特性的參數設計實驗必須先將問題定義清楚。辨認理想的函數之後再發展出信號因子與雜音因子。與相關工程人員討論後確定實驗中的控制因子，並設定其水準值。選定適用的直交表後，依表指派直交表中的控制因子。實驗規劃完成後即開始執行實驗並收集數據。根據實驗所收集的數據來分析資料，首先計算直交表中的SN比及 β 值，並畫出其回應圖。接著執行上述的兩階段最適畫法，評估最佳水準組合下的SN比及 β 值。代入最佳實驗結果執行確認實驗後，判斷是否可行。若確認實驗結果不可行，則從規劃實驗開始檢示每一步驟的缺失；若接受確認實驗的結果，則執行最佳參數設計值的結果。圖2-3為動態特性的參數設計流程圖。

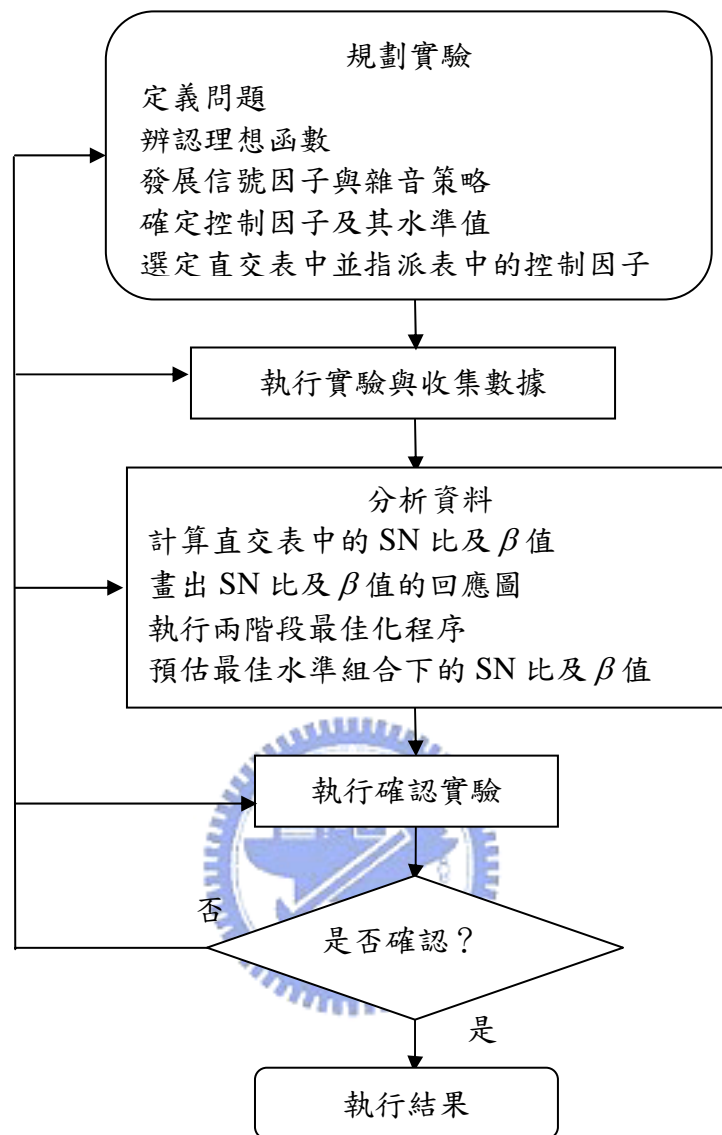


圖 2-3 動態特性的參數設計流程圖

2.2 類神經網路

類神經網路是近年來熱門的話題，由於使用類神經網路時並不需要複雜的計算或太多的假設，只需有一些輸出、輸入的例子就可以利用網路學習建立模式(model)，因此被應用在許多的領域中，如工業、醫療、股市預測、手寫辨識、商業等都有很大的成效，近幾年來常被做為預測、醫療、分類的重要工具(鄒文杰，民 91)。

類神經網路最早由兩位數學家所提出(McCulloch 與 Pitts, 1943)，到了 1950 年代感知機(Perceptron)利用簡單的網路模式和簡單的學習法則，可以解決一些簡單的分類問題，造成感知機曾經被廣泛被使用。1969 年時被其他學者證實感知機只能解決線

性分類的問題，1980 年代這樣的情形有重大改變，有許多學者著手開始解決感知機的問題，其中較重要的突破網路有霍普菲爾(Hopfield neural network；HNN)與倒傳遞網路(back-propagation network；BPN)，這兩個網路藉由隱藏單元(hidden unit)使用，使得類神經網路可以解決更複雜的問題(Behzad 等，1990)。

神經生物學家和神經解剖專家在許多實驗後，發現人類腦細胞是由幾百億的神經元細胞連結而成，於是許多的學者就模擬生物神經元網路而發明了類神經網路(Kohonen, 1989)。類神經網路使用人工神經元簡單模擬生物神經元，利用神經元之間的連結給於輸出輸入，利用學習法則來學習，這樣可以完成模式建立(Funahashi, 1989)。

類神經網路主要有兩種分類，分別為監督式學習網路(supervised learning network)和非監督式學習網路(unsupervised learning network)：

(1)監督式學習網路：在網路的訓練過程中，一組輸入值(input)必須有對應的輸出值(output)，利用和目標輸出值的差異進行網路學習，建立輸出值和輸入值的規則模式，一般常被使用到的監督式學習網路有感知機、倒傳遞網路、學習向量量化網路(learning vector quantization；LVQ)、放射基準機能網路(radial basis function；RBF)、反傳遞網路(counter-propagation network；CPN)等網路。

(2)非監督式學習網路：在網路的訓練過程中，只需要有輸入值而不需要輸出值，這類的神經網路會自己學習找出聚類規則，自己產生對應的輸出，一般常見的網路有自組織映射圖網路(self-organizing map；SOM)、自適應共振理論(adaptive resonance theory；ART)等網路。

類神經網路在運作前必須對網路結構和各階層的節點屬性作定義，然後輸入要訓練的樣本，之後網路對輸入的樣本進行訓練。訓練結果後可以得到一輸出值，此時和目標值作比較，調整網路連結的加權值來修正兩個值的誤差，經過重複多次的學習訓練可以得到一個網路模式，最後會輸入一組測試樣本，這些樣本是用來評估所訓練網路的精確度。

我們要決定一個類神經網路的好壞，常使用一個指標誤差均方根(root mean square error，RMSE)，RMSE 越小越好。在選擇網路時先看訓練樣本的 RMSE 值，如果訓練樣本(training)沒有辦法比較出優劣，這時要把測試(testing)樣本的 RMSE 加入考慮，希望兩者 RMSE 同時小且趨於一致，也就訓練樣本和測試樣本這兩者的 RMSE 要越接近越好。

倒傳遞類神經網路模式最早出現在 1974 年，當時被命名為動態回饋演算法 (dynamic feedback algorithm)，直到 1988 年才正式確定命名倒傳遞類神經網路，解決了感知機只能解線性分類的問題，使得類神經網路領域有所突破(Nitta, 1994)。

倒傳遞類神經網路屬於監督式學習網路，由於它有學習精度高、回想(recall)速度快、理論簡明、應用普遍，造成它最近幾年最常被使用的類神經網路。基本原理是利用最陡坡降法(the gradient steepest descent method)的觀念，使得誤差函數越來越小來提昇網路準確率。它最大的特點在於增加隱藏層，可以將輸入神經元之間的交互作用表現出來，它所使用的轉換函數也和之前感知機時代有所不同，採用平滑可微分的轉換函數，這兩個特點使得類神經網路在理論和模式開發上有重大的突破(Funahashi, 1989)。

倒傳遞類神經網路的架構如圖 2-4 所示，包含三個層級：輸入層、隱藏層、輸出層。輸入層為讀取外界資訊用，而輸出層將網路訓練學習的結果讀出，這兩層的個數依問題而定，最重要的是隱藏層的功用，負責將資料分類或是結果間關係作過濾的工作。隱藏層的層級和各層級單元的個數有一些學者提出證明和建議，隱藏層層級一般一至二層就可以解決大部分的問題，至於隱藏層各層級單元的個數要在 2^{n+1} 以下，其中 n 為輸入層個數，也有學者提出第二層隱藏層單元的個數要為第一層的一半，但一般都使用經驗法則加上試誤法。倒傳遞類神經網路最常使用的激發函數為雙彎曲函數 (sigmoid function)，其定義為 $f(x) = \frac{1}{1+e^{-x}}$ ，當輸入值大於 3 時函數值會趨近於 1，輸入值小於 -3 時其函數值會趨近於 0，這個激發函數的值域是界於 0 和 1 之間。

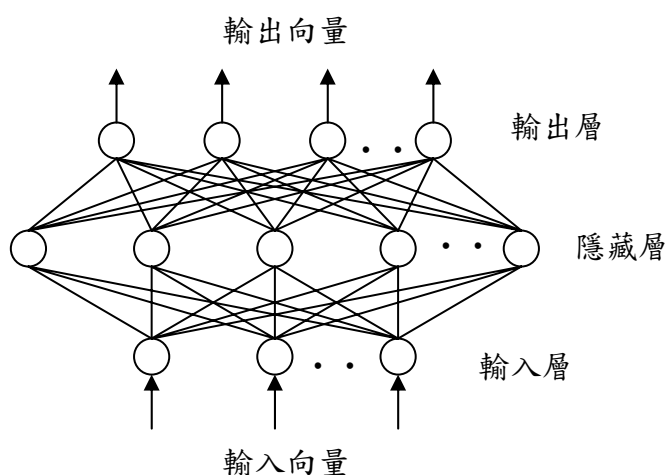


圖 2-4 類神經網路架構：以倒傳遞類神經網路為例

倒傳遞類神經網路的計算方式分為向前(forward)和向後(backward)兩方面。首先，我們將資料從輸入層輸入後，網路會將這些資料直接送至隱藏層，隱藏層這時對這些資料作加權計算，在利用事先定義的激發函數將加權後的值轉換成激發值傳送至輸出層，輸出層也會和隱藏層做一樣的動作，將加權計算後的值給激發函數作計算，最後把計算後的值輸出，因為倒傳遞類神經網路是監督式的學習網路，所以會有輸出的目標值，這時網路會比較輸出值和目標值的差異，利用學習法則對權重作調整，一直重複進行直到滿足停止條件，一般停止條件為訓練次數或是誤差值小於設定的值。倒傳遞類神經網路的訓練過程如下：

- (1) 設定網路架構，隱藏層層數及各層級神經元個數。
- (2) 設定起始權重，一般由電腦自動隨機產生。
- (3) 選擇網路訓練的激發函數。
- (4) 設定學習率 (learning rate)、動量 (momentum)和學習次數。
- (5) 利用輸入值和起始的權重，在隱藏層中計算各隱藏層神經元的輸出值。
- (6) 利用隱藏層的輸入值和起始的權重，在輸出層中計算各輸出層神經元的輸出值。
- (7) 求出輸出值和目標值的差異。
- (8) 計算隱藏層神經元和輸出層神經元的修正差異量。
- (9) 修正輸出層至隱藏層和隱藏層至輸出層的權重。
- (10) 重複(5)~(9)步驟直到到達停止條件為止。

由於倒傳遞類神經網路在建立函數有很精確的效果，使用上方便快捷，所以在本研究中我們使用倒傳遞類神經網路來適配(fit)輸入層和輸出層的函數關係。

2.3 分散式搜尋法

分散式搜尋法(Scatter Search; SS)最原始的概念與原則是學者在1977年所提出，當時被應用在整數規劃問題上(Glover, 1986)。1994年Glover提出一個改良過的分散搜尋法，結合了禁忌搜尋法(Tabu Search)裡的一些技巧。這些技巧主要被用來控制新解進入參考解集合的許可和從參考解集合中選出某些參考解來繁衍下一代，且將兩個候選解配對後進行直線搜尋，再配合加權合成(weighted combination)機制選出新的試驗解。權重值不是隨機產生的，而是事先設定好幾個特定值，此版本的分散式搜尋法應用在解決非線性最佳化問題、二元和排序問題上(Fleurent等,1996; Cung等,1997; Laguna等,2000)。

分散式搜尋法主要的想法起源於結合決定法則(combination decision rule)和代理者限制(surrogate constraints)觀念(Glover, 1965)，利用這兩個想法而產生了重劃路徑(path relinking)的觀念，最後發展出具體的演算法。有別於一般演進搜尋法(evolutionary search)皆由隨機方式開始，分散式搜尋法一開始就使用具體化的方式產生初始解，來增加求解過程的效率和品質，在求解過程中和禁忌演算法一樣利用記憶的方式，對搜尋過程中作紀錄，幫助更廣泛的尋找最佳的解和避免陷入局部最佳解中。分散式搜尋法的許多想法都是很簡單而且方法具體化，現在很多學者把分散式搜尋法的想法方法放入基因演算法中，當成加強型策略。由於分散式搜尋法在每個步驟都可以隨著問題的不同由使用者自己設計的特色，近幾年來被許多學者當成求取最佳解的一個重要工具(Glover, 1999 與 Glover 等人, 2000)。目前運用在半導體製程(鄒文杰, 民 91; 周柏宏, 民 92)、備用零件庫存管理問題(陳士偉, 民 93)、車輛途程問題、二次指派問題和求取函數最佳解這些領域中都相當成功(劉吉峰, 民 93)。

由於分散式搜尋法主要的好處之一是使用者可對問題的不同，自行採用適合的方法並不是固定的，在使用上有很大的好處，所以使用 SS 除了必須注意參數設定和執行步驟外，對於它的基本想法也必須了解。在分散式搜尋法的起始想法中於結合決定法則是很重要的一個，其觀念是把目前的解以特定的結合規則產生新的解。首先使用者可以自行定義規則，從目前所有解中找出有興趣值得觀察的解，將這些解成為一個新的解集合，利用權重(weights)的方式將集合中的解結合而產生新的解，這樣的方式讓解集合成為一空間，將單一的解組合後相當於在空間中搜尋，可以得到比一般產生新解方式得到更好的效果。

分散式搜尋法的另一個重要的想法是代理者限制觀念，和結合決定法則不同的是利用非負權重(nonnegative weights)的方式產生新的不等限制式，首先將原本的限制利用規則分成不同的集合，將這些集合用非負權重方式結合成新限制式，重新對問題求解，這樣的方式被發現隊線性規劃和非線性規劃的求解有很大的幫助。結合決定法則和代理者限制主要的功能是提供評估的規則，可以幫助產生較好的新解或對目前的解做修正。

分散式搜尋法最重要的觀念在於參考集合和重劃路徑：

(1)參考集合

分散式搜尋法利用前面的想法將搜尋的方式成為空間上的搜尋，先選取搜尋結果好或有興趣的幾個解成為一個集合，稱為參考集合(reference set)，利用這個集合產生

新的解，這樣的方式好處可以減少搜尋的個數，因為一般來說參考集合只要小於 20 個就夠用了，像基因演算法每一代的個數最常使用是 100 個，相對於分散式搜尋法就小多了，而且基因演算法在交配的階段是以隨機的方式由兩個染色體組合而成新的解，而分散式搜尋法是用系統化的方式產生新的解，基因演算法利用大量的染色體維持求解的多樣性，而分散式搜尋法利用系統化的方式維持，所以分散式搜尋法可以詳細紀錄求解過程，利用和禁忌搜尋法類似的長期搜尋策略來降低發生局部最佳解機率。

我們使用圖 2-5 說明參考集合，一開始有三個元素 A、B、C 成為一個參考集合，利用不同權重組合產生 1、2、3、4、5 五個新元素。

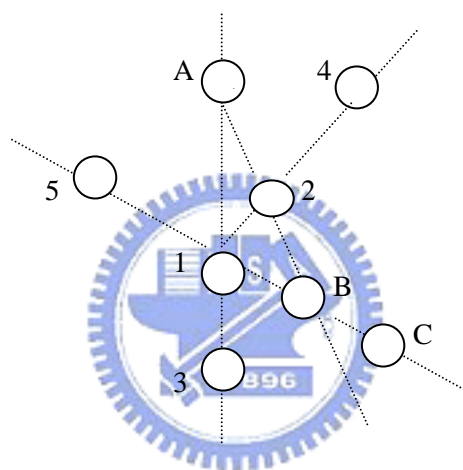


圖 2-5 二維空間參考集合

(2)重要路徑

重劃路徑之觀念是在參考集合產生新的路徑時，利用一些引導的準則幫助產生較好的移動方向，其目的在於減少求解的步驟可以快速找到解，利用這種加強型的策略增加求解的品質，圖 2-6 中實線為原來路徑，虛線為重劃路徑後的路徑。

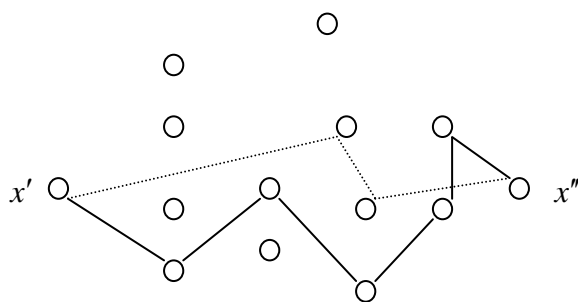


圖 2-6 重劃路徑

在重劃路徑中縮短路徑時並不一定要在可行解(feasible)當中，可以繞過不可行解的範圍裡，在參考集中可能會產生不可行的解，藉由這樣的方式可以處理，同時也希望從不可行解中獲得一些資訊。重劃路徑的想法中使用許多這樣的機制，目的都在增加求解的可能性加強求解的速度，而分散式搜尋法主要的精神也在這裡(鄒文杰，民 91)。

分散式搜尋法有五個基本的執行步驟或稱為基本元素，分別為：(1)多樣化解產生方法 (diversification generator method; DGM)；(2)改進方法(improving method; IM)；(3)參考解結合更新方法(reference set update method; RSUM)；(4)子集合產生方法(subset generation method; SGM)；(5)新解合成方法(solution combination method; SCM)。分述如下：

(1)多樣化解產生方法

「多樣化解產生方法」的功能，是製造一組多樣化解以提供初始化和重組解集合時被挑選使用。在整個搜尋過程中，「多樣化解產生方法」也是最早被用來製造且存放在集合 P 裡的 Psize 個初始多樣化解。Psize 值一般設定為 $\max(100, 5*b)$ ，而 b 則代表參考解集合的大小。這個方法著重於解的多樣性(diversificaiton)而不是品質，換言之，在製造解的過程裡將不考慮問題的目標函數。「多樣化解產生方法」強調利用有系統或受控制的隨機化(controlled randomization)的方式來製造解，而不是採用完全隨機化的方式。原因是利用前者方式所產生的初始多樣化參數解將兼具解的品質與多樣性，若採用後者方式則所產生的初始多樣化參數解難大量偏向解的多樣性。

(2)改進方法

這個步驟是利用一些簡單改進方法，將上一步驟中產生的初始解改進成品質較好的解。雖然一般簡單改進方法產生的解並不是很好，但目的只在於將解做初步改善，可以避免運算的複雜增加求解速度。改進方法可隨問題而定，但必須要有能力處理可行的初始解和不可行初始解，例如 Glover(1994)對沒有限制連且續函數求取極值時採用禁忌演算法當改進方法，利用以前有效的方法當成分散式搜尋法的一部分，再加上分散式搜尋法的基本想法可以改善過去的方法缺點，所以有許多人將分散式搜尋法和舊有的啟發式演算法作結合。

(3)參考解結合更新方法

在分散式搜尋法的想法中參考集合是很重要的一部分，在前面我們知道參考集合如何產生新解和使用的好處，在這部份介紹如何選取參考集合。首先將參考集合分成

兩部分，第一部分 b_1 選取方式是針對加強型策略，使用選取想興趣值得觀察的解加入第一部分，一般方式是選取目前解中將好的，第二部分的參考集合 b_1 選取方式是針對分散策略，選取和第一部分 b_1 距離最近或最遠的解，可以幫助求解的廣泛性，Glover 以實際的問題研究發現，當 b_1 個數等於 b_2 個數時效果最好。至於如何參考解結合更新方法和下面兩個元素有很大的關係，利用下面元素產生品質比 b_1 更好解就加入 b_1 ，而 b_2 的更新方式和前面相同，差別只重複執行前兩步驟產生新解，而不是利用原先的初始解。

(4)子集合產生方法

「子集合產生方法」將參考解集合裡的 b 個優良解分成一組組的子集合，以提供新解合成時取用。這個步驟 Glover 提出四種方式：

- (1) 兩兩相配而成，每一組子集合擁有兩個優良解。
- (2) 由一個擁有兩個優良解的子集合加上參考解集合剩下的解中最好的一個所組成，每一組子集合擁有三個優良解。
- (3) 由一個擁有三個優良解的子集合加上參考解集合裡剩下的解中最好的一個所組成，因此每一組子集合擁有四個優良解。
- (4) 每一組子集合擁有 i 個優良解，其中 $i=5\sim b$ 。

選取的個數多或寡，只是在求解的廣度和廣度上作互補，經過實際驗證發現第一種方式是較佳的。

(5)新解合成方法

這個步驟是利用結合決定法則想法，將上一步集中的解作權重組合，再利用特定選取規則產生新解。其規則由問題而定，一般採取可以多選取 b_1 中品質較好的解，較少選取 b_2 的想法定立規則，再重複執行改進方法與參考解結合更新方法，經過一定的次數後得到最佳解。

一般使用的停止條件有兩種，一種是參考集合 b_1 的解趨於一致，也就是解幾乎都一樣，另一種是參考集合 b_2 更新的次數。

分散式搜尋法由這五個基本元素組成，在介紹過程中不難發現分散式搜尋法使用許多加強型策略和分散型策略，利用這些系統化方法來解決問題，可以清楚追蹤求解的過程，不像其他演算法無法對解的來源作解釋。分散式搜尋法的流程圖如 2-7 所示(葉謀銓，民 92)：

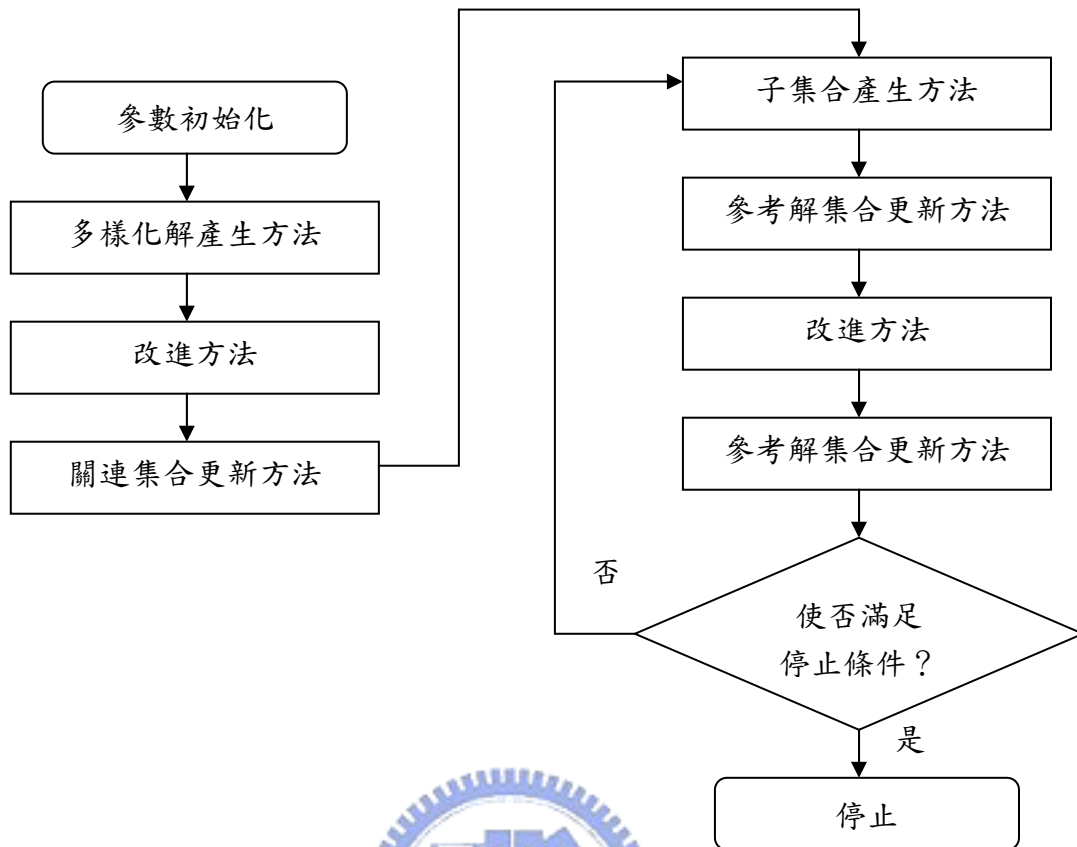


圖 2-7 分散式搜尋法的流程圖

2.4 基因演算法

達爾文在經過多年的研究提出了「物競天擇」的理論，認為在自然界的生物是以「適者生存、不適者淘汰」的規則演進，適合目前生存條件的物種擁有較大的機率存活下來，這樣的想法被Holland (1975)應用到人工智慧的領域中，而提出基因演算法 (Genetic Algorithm; GA)。隨後有學者陸續深入探討並加以推廣(De Jong, 1987)。1989年確定了基因演算法的三種基本演算因子—複製、交配、突變。經由複製、交配及突變的方式產生下一代染色體，其平均表現會比上一代染色體好，如此重複可以求得最佳解。

遺傳演算法結合交配、篩選、突變三個機制，讓隨機選取的動作，具備了強大的搜尋能力，能夠解決許多非線性的問題，目前已經廣為應用於許多領域，如財務決策和投資決策問題(Leinweber與Arnott, 1990; Bauer,1994)，Goldberg(1989)則利用遺傳演算法作為機器學習辨識系統的處理機制，而經濟學家更利用遺傳演算法來做時間序列的預測。

使用遺傳演算法搜尋最佳解，同時執行多點搜尋而非單點搜尋，比較不容易陷入局部最佳解，且在基因演算法的程序中是對編碼後的字串來做運算，只要對問題作合適的編碼都可以使用，因此基因演算法可以適用於不同領域的最佳化問題。近年來有學者將此方法運用於室內定位系統最佳化、及解決無線通訊頻道指定等問題皆有良好的成效(章俊彥，民93；李善誠，民94)。

2.5 模擬退火法

模擬退火演算法(Simulated Anneal; SA)是模擬物質結晶的退火(annealing)程序之演算法。Metropolis 等人首次在 1953 年提出 SA 的演算法，1983 年時 Kirkpatrick 等運用模 SA 來解決組合最佳化的問題(盧炳勳等，民 86)，之後 Cerny (1985)利用 SA 來解決旅行推銷員工作排程問及與 VLSI 設計問題。1986 年，Hinton 等人從原有的 SA 架構中發展出波茲曼機(Boltzmann mechanism)(Kirkpatrick 等，1983)，以及 Miller 等人以模擬退火法為基礎發展出退火神經網路(Cerny, 1985)。

假設我們要極小化目標函數 $f(x)$ ，SA 一開始先隨機產生一個可行解 x ，再由我們所定義好的鄰域中產生新解 x' 。接下來計算 $\Delta E = f(x') - f(x)$ ，如果 $\Delta E < 0$ 就將新解取代舊解，如果 $\Delta E \geq 0$ 時計算一個機率指標 $e^{\Delta E/T}$ ，其中 T 是一控制參數稱為溫度，由這個指標決定是否接受新解。如此在同一個溫度 T 下重複 N 次， N 次後將溫度 T 下降，如此重複搜尋至設定好的最小溫度後停止。SA 利用 $e^{\Delta E/T}$ 的指標可以接受較差的解，因此有避免陷入局部最佳解的能力。

假設我們求一個方程式 $F(x)$ 的最大值， x 的可解區域為 X 。SA 是單點搜尋，首先要由電腦隨機產生一個可行解 $x \in X$ ，然後要定義鄰近解的區域，一般鄰近解設定方式隨問題而有所不同。

2.6 望想函數

望想函數(Desirability Function)是由 Derring 與 Suich(1980)所提出，代表任一反應值(response value)在可接受區域(region of acceptance)內與目標值的接近程度。其目的是將多品質特性問題轉換為單一變數的評估指標，以供決策方便。Young 等學者於 1991 年應用田口方法中損失函數的概念，推導出望想函數的期望損失，並將其用來解決超大型積體電路製造的問題；Chang 與 Shivuri(1995)則結合望想函數與多目標決

策(multiple-objective decision-making)方法，發展出一多品質特性問題的分析模式；Goik 等學者(1995)則結合望想函數與反應曲面法，發展出一解決產品設計問題的分析模式；Chapman 結合望想函數與非線性規劃(nonlinear programming)方法，解決光化學(photochemistry)製程的問題。

就實務上而言，品質特性與控制因子間的關係通常是未知的。經過實驗後，便可利用實驗所得的數據來求得兩者間的關係。一旦得到品質特性與控制因子間的函數關係後，即可對品質特性做預測，接下來便可導入望想函數的應用(洪饒峰，民 85)。

假設將任一品質特性與多個控制因子間的關係以下列函數表示：

$$Y_i = f_i(X_1, X_2, \dots, X_p) + \varepsilon_i \quad (2-4)$$

其中， Y 為品質特性， X 表控制因子， ε 為隨機誤差。通常以下式估計兩者間的關係：

$$\hat{Y}_i = f_i(X_1, X_2, \dots, X_p) \quad (2-5)$$

因此可根據不同的控制因子設定來得到不同的品質特性預測值。

當品質特性屬於望大時，公式如下：

$$d_i = \begin{cases} 0 & \hat{Y}_i \leq Y_{\min} \\ \left[\frac{\hat{Y}_i - Y_{\min}}{\hat{Y}_{\max} - Y_{\min}} \right]^r & Y_{\min} < \hat{Y}_i < \hat{Y}_{\max} \\ 1 & Y_{\max} \leq \hat{Y}_i \end{cases} \quad 0 \leq d_i \leq 1 \quad (2-6)$$

當品質特性屬於望小時，其表示如下：

$$d_i = \begin{cases} 0 & \hat{Y}_i \geq Y_{\max} \\ \left[\frac{\hat{Y}_i - Y_{\max}}{\hat{Y}_{\min} - Y_{\max}} \right]^r & Y_{\max} < \hat{Y}_i < \hat{Y}_{\min} \\ 1 & \hat{Y}_i \leq Y_{\min} \end{cases} \quad 0 \leq d_i \leq 1 \quad (2-7)$$

當品質特性有一最佳的目標值，即望目的問題時，公式如下：

$$d_i = \begin{cases} \left[\frac{\hat{Y}_i - Y_{\min}}{c_i - Y_{\min}} \right]^s & Y_{\min} \leq \hat{Y}_i \leq c_i \\ \left[\frac{Y_{\max} - \hat{Y}_i}{Y_{\max} - c_i} \right]^t & c_i < \hat{Y}_i \leq Y_{\max} \\ 0 & \hat{Y}_i < Y_{\min} \text{ or } \hat{Y}_i > Y_{\max} \end{cases} \quad 0 \leq d_i \leq 1 \quad (2-8)$$

其中：

\hat{Y}_i ：為第 i 組因子組合(Run)經轉換後的品質特性預測值。

\hat{Y}_{\max} ：為預測所得到品質特性的最大值。

\hat{Y}_{\min} ：為預測所得到品質特性的最小值。

Y_{\max} ：為已做實驗結果實際最大值。

Y_{\min} ：為已做實驗結果實際最小值。

c_i 為品質特性的目標值； r 、 s 、 t 為表示重要程度的參數，值愈小表示其相對重要性愈大。

實際應用時，必須先將品質特性與控制因子間的關係找到後，得到品質特性的預測值後再代入 d_i 的公式，以求其值，而 d_i 具有愈大愈好的特性。經過對同一因子水準組合所產生的每一個品質特性轉換為 d_i 之後，將所有的 d_i 相乘並取其幾何平均數 (geometric mean)，可得到該組因子水準組合的綜合判斷指標 D ，其公式如下所示：

$$D = (d_1 * d_2 * \dots * d_k)^{\frac{1}{k}} \quad 0 \leq D \leq 1 \quad (2-9)$$

D 為多品質特性問題中，任一因子組合所產生的結果接近理想狀態的程度。 D 與 d_i 相同，也具有愈大愈好的特性，且最大值為 1，即愈接近 1 愈能符合使用者的理想。最後可以 D 作為決定最佳因子水準組合的依據。

2.7 混合演算法

由於各種啟發式演算法因其本身的特性，有其不同的優缺點，故許多學者致力於實驗各種啟發式演算法以解決組合最佳化問題，諸如：基因演算法、禁忌搜尋法、模擬退火法等。有部分學者結合不同的演算法來作為求解基礎模式，期望藉由良好的結合方式，讓演算法之間產生互補的效果；而許多組合問題的研究結果亦顯示：混合演

算法無論在求解的品質或效率上常常比單一演算法為佳(Yamada 與 Nakano, 1996; Balas 與 Vazacopoulos, 1998; Kolonko, 1999; Pezzella 與 Merelli, 2000; Chang 與 Lo, 2001; Wang 與 Zheng, 2001; Murovec 與 Suhel, 2004)。

2.7.1 結合類神經網路與分散式搜尋法

學者運用類神經網與分散式搜尋法的技術來求取參數的最佳設計。亦即，使用類神經網路學習田口實驗的結果，模擬系統輸入輸出之間的關係，以突破田口方法在非線性上的限制；再使用分散式搜尋法求取參數的最佳設計，並本提供一個二階段的求解程序並以積體電路焊線製程為例進行最佳化參數設計 (鄒文杰, 民 91)。

2.7.2 結合類神經網路與基因演算法

Kumar 與 Smuda(1994)將傳遞類神經網路模式中隱藏層的連結關係以基因演算法的 Simple GA 表示之，並將 0 與 1 代表各層有無連結，目的在求取連結架構的最佳值，以新的判斷值衡量其所呈現出的誤差程度。Ichimura、Oeda 和 Yoshida(2001)，提到以基因演算法與類神經網路的結合方式。假設整個類神經網路的架構，是由基因演算法搜尋倒傳遞類神經網路模式架構的每個世代(generation)而來。即把類神經的每個參數，以基因演算法搜尋可能的最佳解再去架構倒傳遞類神經網路模式，則連結的權重與學習速率則被倒傳遞類神經模式所限定，其方法可以很穩健的架構傳遞類神經網路模式。但對傳遞類神經模式的參數因子，需定其區域值以找最佳解。Chelouch 與 Siarry(2003)也發展出一個兩階段的混合演算法來求解全域最佳解。周立德(民 83)結合類神經網路及基因演算法應用於非同步傳送模式網路之研究。先利用類神經網路預估每一可能之控制參數組未來的服務品質，再由基因演算法選取最適當的一組參數用以動態地調整控制真實網路。詹緒林(民 86)也結合類神經網路與基因演算法，利用具自我學習能力之類神經網路使電腦自動的編寫音樂的和弦，再利用基因演算法選取適當的和聲基礎，完成電腦輔助和聲編寫系統。姜台林(民 90)提出三種整合式智慧型方法，整合類神經網路、基因演算法及模糊理論，以解決田口方法靜態特性只能獲取間斷型參數的問題，並以台灣三家半導體廠商的實際操作結果說明方法的可行性。涂育璋(民 91)基於品質工程中，以直交表配置之實驗數據轉換為訊號雜音比，來尋求最適組合之設計參數與水準的技法上，仍有藉助主觀決策的缺憾，於是提出結合類神經網路與基因演算法則，於最適產品特徵之參數水準值的求取。最後，並將所求得之參數水準值，導入實際產品特徵，建構一結合類神經網路與基因演算法則之最適化

搜尋平台。張家瑋(民 91)嘗試結合類神經網路與基因演算法，藉由基因演算法預先搜尋類神經網路訓練的初始權重值，再由類神經網路訓練以求取系統的估測反應，並根據所定義的誤差指數，透過基因演算法進行反覆的選擇與複製、交配、突變等演算過程，直到搜索出最符合系統的初始權重值。

2.7.3 結合類神經網路與模擬退火法


林正鄰(民 82)提出以類神經網路作為製程輸入的關係經驗模式，而以模擬退火為實驗設計法，規劃建立模式所需之資訊的蒐集。這種最佳化的經驗模式建立程序，應用於具有多個局部最佳解的函數時，可以成功地求得整體最小值；應用於函數在陡峭峽谷的區域，能夠如梯度法般很快地找到谷底，而在目標函數為平坦之谷底區，又能比梯度法更有效率地找到最佳值。因此可利用類神經網路建立最適化變數，設計出具有穩健性的控制參數，證明以模擬退火與類神經網路結合之實驗設計法在製程最佳化的應用極具潛力。江欽源(民 83)改進林正鄰(民 82)提出來的實驗設計法，用統計上 F-test 的方法來決定遞回式類神經網路結構中隱藏層神經元的數目，找到一個適當的神經元數目，可避免學習時訓練不足或是過度訓練(over-training)的現象，改進以往使用過多神經元的缺點，故加快了學習的速度。改進後的實驗設計法，具有廣泛搜尋的能力，作為搜尋實驗點的方法，可以避免傳統用最陡梯度搜尋法被陷在局部極值的缺點。由於倒傳遞類神經網路面臨不同的問題時，所需的參數與架構也有所不同，常要讓使用者進行類似「試誤」的動作，林欣志(民 95)提出使用模擬退火法尋找倒傳遞類神經網路中的架構、參數並進行屬性篩選，挑選出對於預測正確率有助益之屬性。證實過去的研究只考慮參數與架構調整的做法，若同時考慮屬性篩選時，在大部分的資料集測試正確率皆提高，除了可以有效找到良好的網路架構與參數，並可以找出有幫助的屬性，提高分類正確率。

第三章 研究方法

混合演算法無論在求解的品質或效率上常常比單一演算法為佳，故本研究採用混合演算法來解決多重品質特性下參數設計於田口動態特性的問題，期望藉由良好的結合方式，讓演算法之間產生互補的效果。四種混合演算法包含：(1)整合田口方法與倒傳遞類神經網路演算法，(2)整合倒傳遞類神經網路與分散式搜尋法，(3)整合倒傳遞類神經網路與基因演算法，(4)整合倒傳遞類神經網路與模擬退火法，四種方法皆利用望想函數及指數望想函數來解決多重品質特性問題。

3.1 整合田口方法與倒傳遞類神經網路演算法

本研究使用此方法分為兩個階段。第一階段採用最佳化兩段步驟分析資料，第二階段使用倒傳遞類神經網路得到輸入參數值與輸出反應值之間的關係。藉由訓練出來的網路可以精確的預估各種可能的參數組合，故我們可藉由訓練輸入的參數值來獲得與反應值之間的關係。最後利用望想函數及指數望想函數將多重品質特性轉換為單一品質特性。綜合執行步驟如下：

- 
- 步驟一：定義輸入參數與相關的反應值。
 - 步驟二：規劃並執行田口實驗，以取得實驗數據資料。
 - 步驟三：發展倒傳遞類神經網路獲知輸入參數值與輸出反應值之間的關係。
 - 步驟四：透過田口方法及變異數分析法，確認重要的控制因子及其水準。
 - 步驟五：利用倒傳遞訓練參數值與反應值之間的關係。
 - 步驟六：經由輸入目標反應值至訓練完成的類神經網路獲得最佳參數設定。
 - 步驟七：執行確認實驗並產生實驗結果。
 - 步驟八：將結果代入實際案例中執行。

3.2 整合倒傳遞類神經網路與分散式搜尋法

本研究使用此方法分為兩個階段。第一階段使用倒傳遞類神經網路得到輸入參數值與輸出反應值之間的關係，藉由訓練出來的網路產生各種可能的參數組合。最後利用望想函數及指數望想函數將多重品質特性轉換為單一品質特性。第二階段利用分散式搜尋法獲得滿意度(degree of satisfaction; λ)的最佳值。綜合執行步驟如下，執行流程如圖 3-1 所示：

- 步驟一：定義輸入參數與相關的反應值。
- 步驟二：規劃並執行田口實驗，以取得實驗數據資料。
- 步驟三：發展倒傳遞類神經網路獲知輸入參數值與輸出反應值之間的關係。
- 步驟四：利用望想函數及指數望想函數將多重品質特性轉換為單一品質特性。
- 步驟五：設定分散式搜尋法的起始參數。
- 步驟六：多樣化解產生器從初始解開始製造儲存在 P 集合裡的 Psize 個初始多樣化解，這個步驟使用了分散式搜尋法的第一個特定方法。
- 步驟七：將步驟六所產生出來的每一個初始多樣化解交由「改進方法」處理，以鄰近搜尋機制產生品質較佳的解。
- 步驟八：利用「參考解集合更新方法」對 Psize 個品質被提升後的解進行進入參考解集合的資格檢定，從中選出 b_1 個高品質解和 b_2 個高相異解來成為參考解集合的成員。
- 步驟九：將參考解集合中的解以「子集合產生方法」組成子集合，以提供「新解合成方法」步驟的取用。
- 步驟十：利用「新解合成方法」將步驟九所組成的每一個子集合轉換成一個或多個新解。
- 步驟十一：將「新解合成方法」所產生的新解交由「改進方法」處理之後，才進行檢定，考慮是否放入參考解集合。
- 步驟十二：利用「參考解集合更新方法」將具有資格放入參考解集合的新解，替換原有參考解集合裡相對最差的解。
- 步驟十三：重複執行步驟九至步驟十二，直到符合迴圈停止條件。最常被選用的迴圈停止條件是反覆次數達一特定值或當最佳解收斂時。

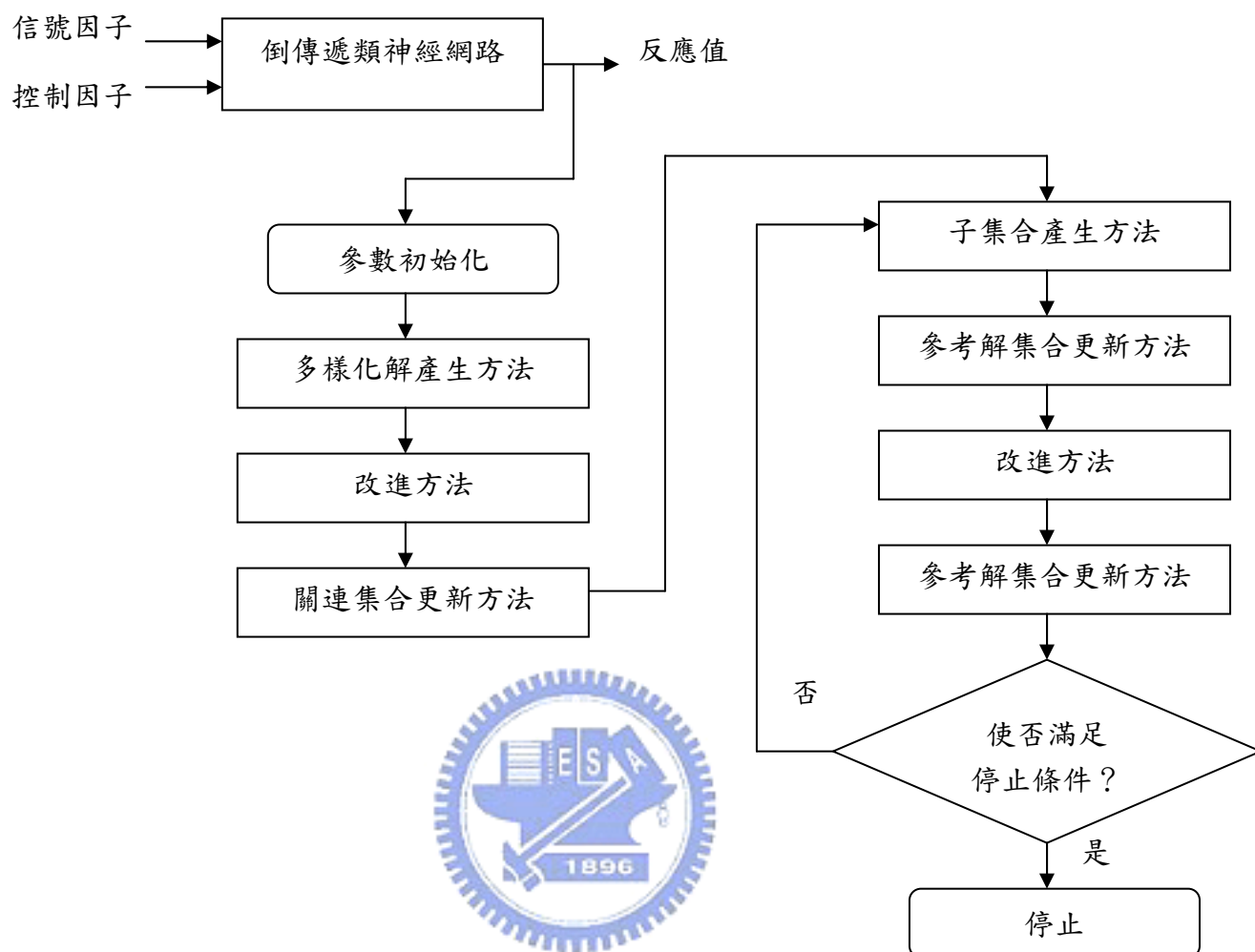


圖 3-1 整合倒傳遞類神經網路與分散式搜尋法流程圖

3.3 整合倒傳遞類神經網路與基因演算法

本研究使用此方法分為兩個階段。第一階段採用倒傳遞類神經網路來建立田口方法中品質特性和實驗因子之間函數的關係，訓練出來的網路產生可能的參數組合。利用望想函數及指數望想函數將多重品質特性轉換為單一品質特性，第二階段利用基因演算法獲得滿意度(degree of satisfaction; λ)的最佳值。染色體代表可能的各種解，染色體中的每個基因代表輸入的參數值。執行步驟綜合如下，執行流程如圖 3-2 所示：

步驟一：蒐集輸入參數值與各反應值數據。

步驟二：發展倒傳遞類神經網路獲知輸入參數值與輸出反應值之間的關係。

步驟三：利用望想函數及指數望想函數將多重品質特性轉換為單一品質特性。

步驟四：設定基因演算法的執行條件，例如族群(population)個數、基因個數、參數個數、交配率及突變率等。

步驟五：從輸入參數值中隨機選擇初始族群。

步驟六：不斷重複步驟七至步驟十一直到符合停止條件為止。

步驟七：由輸入參數值至配適函數(fitness function)計算出配適值。

步驟八：根據計算出的反應值選擇參數值。

步驟九：進行配適參數值的交配。首先計算交配的個數，其值是由群體個數乘以交配率而得，決定個數後從母代中隨機選取成對的染色體出來交配。

步驟十：突變參數值以改進下一代基因。首先設定突變率參數，計算母代要進行突變的個數，其值由族群個數乘以突變率而得，然後從母體中隨機產生要突變的染色體。

步驟十一：經由交配與突變步驟產生的新染色體取代母體的染色體，獲得新一代的染色體，即最佳參數值條件。

步驟十二：獲得最佳參數值設定。



為了結束基因演算法的演化循環，必須事先設定終止規則，當基因演算過程中滿足所設定的終止規則時，能夠停止演化循環。常見的終止條件包括 (黃寶賢，民 91)：

一、設定誤差範圍：即到達使用者可以容許與接受的誤差範圍內而終止。

二、根據統計資料：如果適應函數的分布已達某統計特性即終止，如達到高斯分布即終止。

三、設定最大運算時間：當達到所設定之最大運算時間及宣告終止。

四、設定最大世代：利用程式計算合理的觀察出最大世代數，達到所設定的最大世代

代數則宣告終止。

本研究是採用設定最大演化世代數的方式來做為測試收斂性的依據。

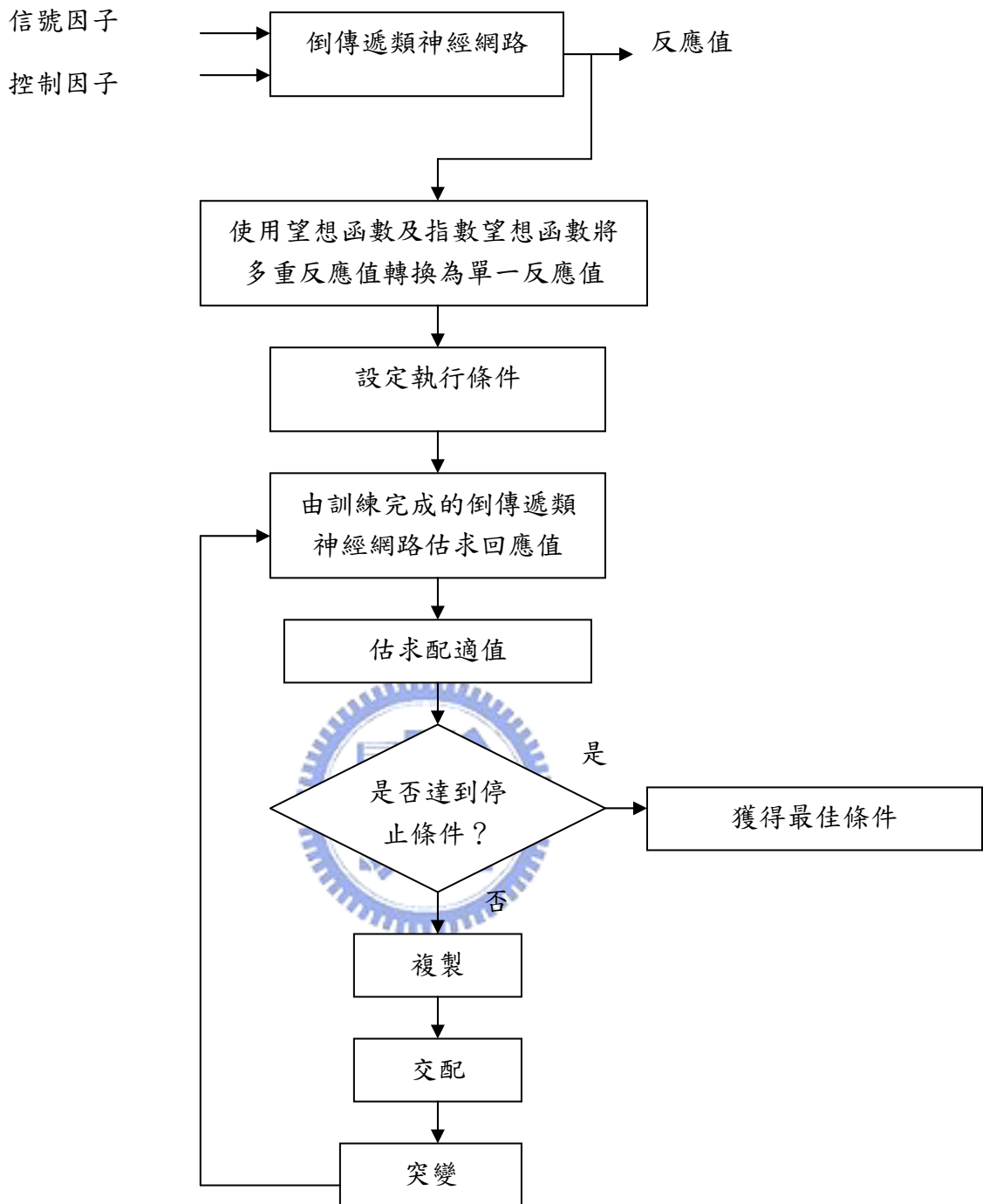


圖 3-2 整合倒傳遞類神經網路與基因演算法流程圖

3.4 整合倒傳遞類神經網路與模擬退火法

本研究使用此方法分為兩個階段。第一階段採用到傳遞類神經網路來建立田口方法中品質特性和實驗因子之間函數的關係，訓練出來的網路可以精確的預測出可能的參數組合，故我們可藉由訓練輸入的參數值來獲得與反應值之間的關係。利用望想函數及指數望想函數將多重品質特性轉換為單一品質特性後，第二階段利用模擬退火法獲得最佳解。執行步驟與參數設定如下，執行流程如圖 3-3 所示：

步驟一：隨機由直交表中選擇資料以獲得倒傳遞的訓練及測試組合值。

步驟二：發展倒傳遞類神經網路預測控制因子、信號因子及反應值之間的關係。

步驟三：將能階函數(energy function)公式化。

步驟四：根據問題的品質特性將關係函數轉換成能階函數。

步驟五：隨機選取候選值 x 。

步驟六：設定起始條件，首先要決定起始溫度 T 和終止溫度 T^* ，其中 $T > T^*$ ，一般起始溫度設定隨問題而變，不過一般都設較高一點可以比溫度設低更容易找到最佳解，Kirkpatrick 等建議 T 的設定最好可以使初期鄰近解被接受機率 0.8 以上，而 T^* 一般都設接近 0。

步驟七：先定義每個溫度搜尋次數 M ，一般都設固定值也有學者採用不同溫度不同 M ，Kirkpatrick 建議為決策變數的倍數，不過實際上還是必須看問題大小而定。然後設定降溫梯度 α ，一般 α 越大越好最好在 0.5 以上，如果設太小容易陷入區域最佳中，建議值為 0.8~0.99。

步驟八：重複步驟九到步驟十四直到搜尋溫度 T 小於停止溫度 T^* 。

步驟九：重複步驟十到步驟十三直到在溫度 T 下搜尋次數大於 M 。

步驟十：由電腦在鄰近解中產生一個新解 x' 。

步驟十一：計算 $\Delta E = f(x) - f(x')$

步驟十二：由電腦隨機產生一個均勻分布於 0~1 的值 r 。

步驟十三：如果 $\Delta E < 0$ 表示新解比就解好以新解取代舊解，如果 $\Delta E > 0$ 如果 $r < e^{\Delta E/T}$ ，我們以新解取代舊解。

步驟十四：降低搜尋溫度 T ， $T = \alpha T$ 。

步驟十五：求得最佳解 x^* 。

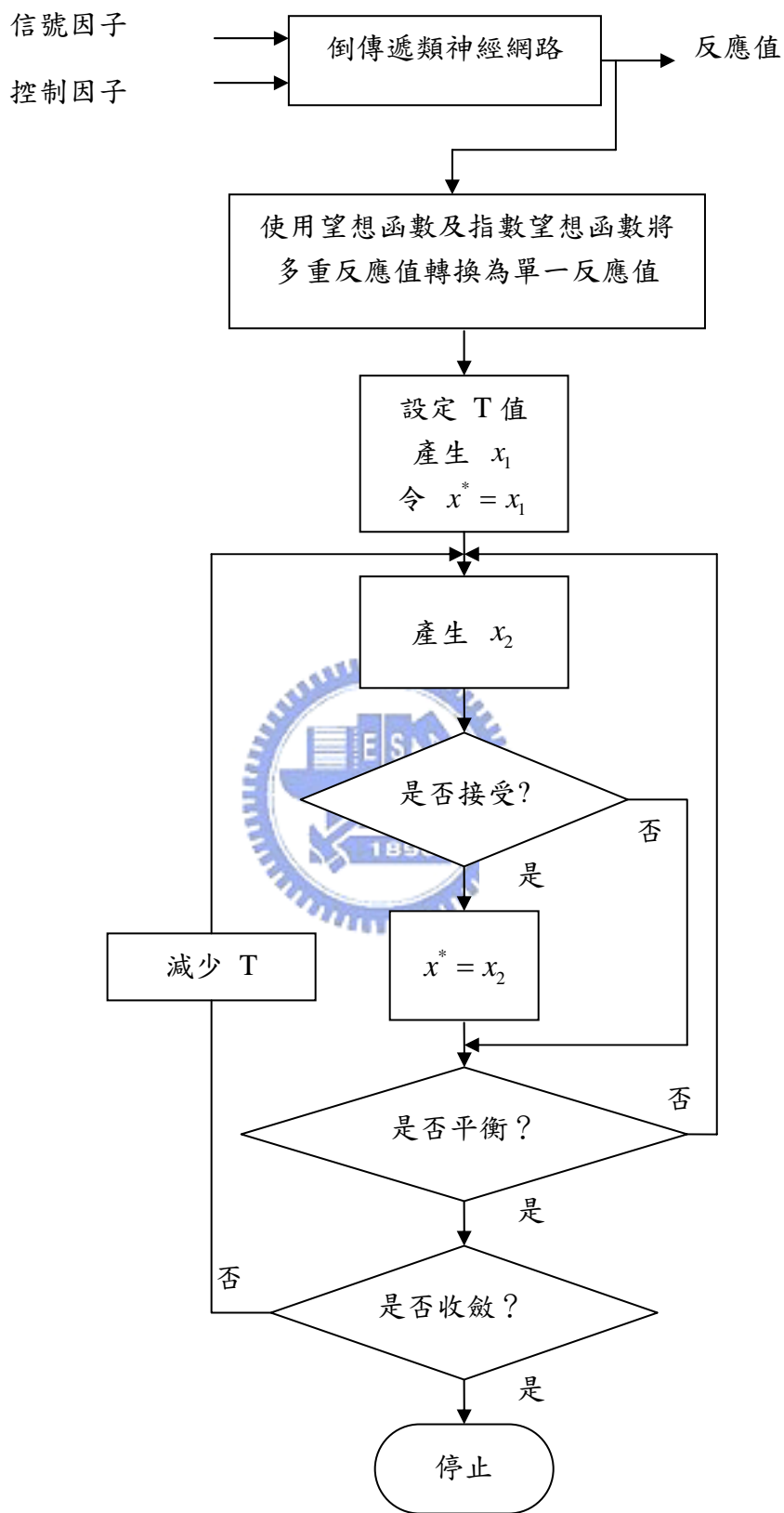


圖 3-3 整合倒傳遞類神經網路與模擬退火法流程圖

停止準則為用來判定是否結束模擬退火演算法的最主要依據。Johnson(1989)提出，計數值在鄰近解沒有取代現行解時應加 1，當現行解被取代時，計數值應歸零重新計算，模擬退火法在計數值達到預定上限時應停止。Van Larrhoven(1992)等人認為，模擬退火法應在溫度下降到預設的最後溫度時停止。常見的停止準則有(盧研伯，民 92)：

- 一、當溫度已達到每一溫度 T_k 時，則停止此演算法。
- 二、當連續降溫 t 次，始終無法達到較佳解時，停止演算法。
- 三、當所求之解已達到下限值之某個比例，則停止演算法。

在本文中所採用的停止準則，考慮上述的前兩項，以增加演算法的效率。若單純考慮停止溫度時，當系統達到最佳解後或許已經失去向上爬升的能力，限制降溫的次數，將有助於演算法的求解效率。



第四章 室內無線區域網路傳輸品質評估系統之實證分析

為了驗證第三章所建構模式的可行性，本章節提出「室內無線區域網路傳輸品質評估系統」案例。根據本文所敘述的方法論規劃實驗，進而從實際的案例中蒐集數據並作分析，以證實本研究方法論之可行性與有效性。

4.1 問題描述

隨著資訊時代的全面來臨及網際網路普及，無線通信、寬頻網路及數位傳播的啟用等，使資訊的傳遞更加迅速，人們使用資訊的分式將更多元化(潘泰吉，民 88)。對於許多人來說，網路不只是消遣般的休閒娛樂活動，更是每日工作與生活不可或缺的重要部份，在工作上常免不了要做資訊交換的動作。透過電腦與電腦、電腦與電腦週邊設備的連結，來處理資料的存取與列印、或檔案交換等(陳文江，民 88)。然而有線的連結方式，不管是透過電話線、電纜線或是光纖，人們始終受到纜線的禁錮，所有活動都必須在網路連接埠附近進行，大大的侷限了活動的空間範圍。另外對於一些大型展覽或是大型會議等臨時性需求的通訊傳輸、具有歷史紀念價值的古蹟建築、或有地形地物障礙無法佈線的地方，也有眾多的限制與不便！同時，網路線的鋪設與維護花費的成本，也是一筆不小的開銷。

由於無線區域網路(wireless local area network; WLAN)標準 IEEE 802.11 的制定，不僅無線區域網路系統的發展隨之蓬勃，且依據行政院挑戰 2008 國家發展重點計畫，科技顧問組規劃 e-Taiwan 計畫中之「寬頻到家」項下，規劃「推廣無線寬頻網路計畫」以強化台灣無線寬頻網路之建設與應用發展。目前全台灣無線上網據點已超過千個，地方政府如台北市、台中市也積極佈建全市涵蓋的無線網路，打造無線城(經濟部工業局，民 94)。其中台北市政府更榮獲美國世界電信協會(WTA)所屬「智慧社區論壇」(ICF)頒發 2006 年年度智慧城市首獎(彭佳芸，民 95)。透過無線區域網路的推廣，不僅可解決有線網路不便的問題，更可享受「無線」所帶來隨時隨地的上網或進行資料存取傳輸的「無限」便利性與移動性！

雖然無線區域網路具有許多傳統有線網路所沒有的優點，但當我們在規劃室內無線區域網路時，卻發現一些尚待解決的問題。由於室內無線電的傳播環境相當複雜，不論是建築物的隔間材質、室內的擺設等都會造成不同的傳播損失，因此在傳統上會在一層樓層中放置一個以上的存取點(access point; AP)，以期能增加無線電波的

覆蓋範圍。

傳統上，AP 的數量及位置的選擇經常是由工程師依據經驗採用試誤法(trial and error)來做決定，不同的工程師常會做出不同的判斷結果。一般判斷擷取位置與數目的標準，傳統上是以某一特定訊號強度的涵蓋範圍作為指標(Cheung, 1998; Chiu, 1996; Panjwani, 1996)，各種指標雖然彼此之間有一定程度的相關性，但使用者卻可以在需求不同時，選擇不同的參考指標作為無線區域網路傳輸品質評估的標準，才能讓當時的無線電波傳輸環境品質評估，能夠以最適當的方式表達出來。

本研究提供一個二階段的求解程序並以無線區域網路估測系統為例，以某技術學院之教學大樓來建構此估測系統的模型，利用類神經網路與分散式搜尋法、基因演算法和模擬退火法進行最佳化參數設計，未來即使要估測不同隔間、擺設的大樓，皆能以此估測系統為基礎。只要量測的數據準確且數目足夠，再經由輸入變數的決定，與類神經網路中各層神經元的調整，及不同網路架構、演算法的修正，本系統即可對其他室內無線網路的通訊品質做出正確的估測，圖 4-1 為該技術學院教學大樓的平面圖。

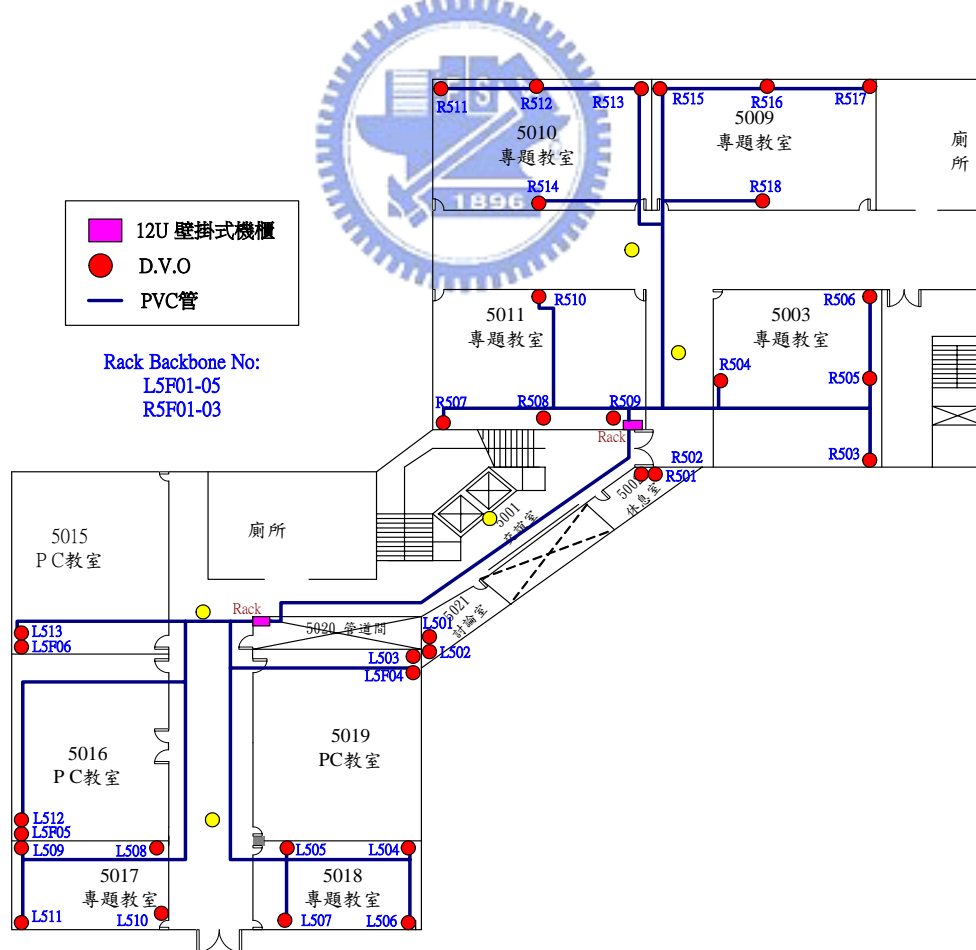


圖 4-1 教學大樓平面圖

4.2 研究設計之架構

在田口的參數設計法中使用直交表來配置實驗，利用 SN 比與 β 值來分析各因子影響程度，以達到工程品質目標。本研究方法分為兩階段：

第一階段為「建立模式階段」，共分為兩個步驟：步驟一為設計並執行「田口實驗」，對於一個品質問題，考慮信號因子、雜音因子與控制因子水準組合，利用直交表進行田口實驗，取得實驗數據及 SN 比、 β 值等基本資料。步驟二則「利用倒類神經網路建立模式」，進行田口實驗時，使用 SN 比的好處在於簡單計算就可以釐清各個控制因子的影響，而且同時考慮品質特性的平均和變異效果。本論文利用倒傳遞類神經網路建立直交表中各因子水準組合和 SN 比及 β 值的函數關係，將這個網路模式當成目標函數，如圖 4-2 所示。

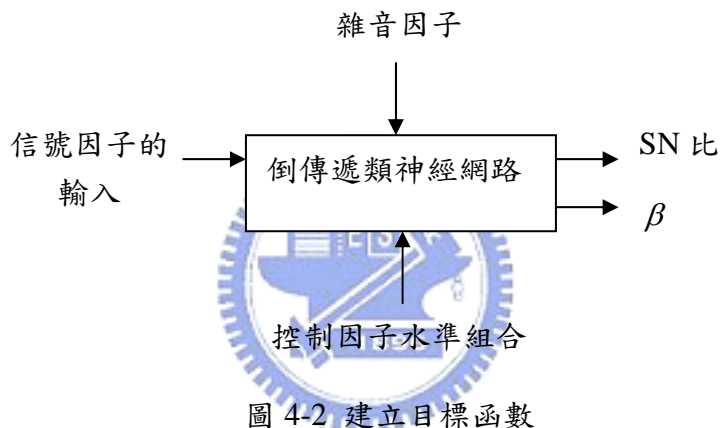


圖 4-2 建立目標函數

第二階段「利用啟發式演算法，最佳化階段一所建立的目標函數」。本實驗採取的啟發式演算法有三個，分別為分散式搜尋法、基因演算法和模擬退火法，其中模擬退火法所採取方式和 Su 與 Chang (2000) 之差異在於鄰域選取方式及停止準則。執行模擬退化法時我們產生新解的方式有兩種，一種是由現在解的位置，以隨機增加方式向解的最大值方向增加，另一種是向解的極小值方向減少，由電腦進行隨機選擇採取哪一方式。而停止條件考慮限制降溫的次數，以增加演算法的效率。而基因演算法所採取方式和 Su 與 Chiang (2002) 的差別，在於採用設定最大演化世代數的方式來做為測試收斂性的依據參數，且執行條件設定上亦不相同。

針對室內無線區域傳輸品質問題，我們進行田口實驗。首先，先選取品質特性 y ，會影響傳輸品質的品質因素有很多，我們選取最重要的接收訊號強度(Received Signal Strength Index; RSSI)為品質特性。

室內無線區域傳輸品質問題的理想函數為：

$$y = \beta M$$

其中 M 為信號因子，即模擬器測出的收訊強度值； y 輸出值為接收訊號強度，即品質特性； β 為靈敏度，即輸入與輸出之間直線關係的直線斜率。在實驗室中以模擬器測出的收訊強度值代替實際的接收訊號強度， β 的理想值為 1，因為模擬器測出的收訊強度值與實際的接收訊號強度最好是一致的。信號因子的水準設定如下：

$$M_1 = 10, M_2 = 50, M_3 = 100$$

我們的目的是希望接收訊號強度在模擬器測出的收訊強度範圍內有好的線性關係。

經過相關研究人員針對製程可能的問題提出討論後，製作特性要因圖，並從中選定八個控制因子及其水準如表 4-1 所示。在本研究的案例中使用八個控制因子，每個控制因子有三個水準。我們選取 $L_{27}(3^{13})$ 直交表(orthogonal table)進行實驗。總共有 27 個水準組合，在每個組合下依網路卡型號的不同，共取得六個實驗數據，計算出 SN 比及 β 值，實驗結果如表 4-2 所示。

表 4-1 控制因子及其水準

控制因子	水準	水準	水準
A：無線電波直射路徑穿過建築物中庭上方的中空範圍/平方公尺	10	15	20
B：傳輸的距離/公尺	5	10	15
C：傳輸的高度/公尺	3	6	9
D：OA 隔間牆個數	5	10	15
E：磚牆個數	2	4	6
F：木板牆個數	3	6	9
G：室內面積大小/平方公尺	25	50	75
H：附近微波爐個數	1	2	3

表 4-2 實驗結果

Run									$M_1=10$		$M_2=50$		$M_3=100$		$y_1 = SN$	$y_2 = \beta$
	A	B	C	D	E	F	G	H	N_1	N_2	N_1	N_2	N_1	N_2		
1	10	5	3	5	2	3	25	1	11	13	55	62	99	100	-14.657	1.031
2	10	5	3	5	4	6	50	2	9	8	45	42	97	98	-10.568	0.953
3	10	5	3	5	6	9	75	3	8	8	42	42	98	95	-11.911	0.939
4	10	10	6	10	2	3	25	2	13	12	56	53	100	99	-10.089	1.016
5	10	10	6	10	4	6	50	3	6	7	43	43	90	91	-7.968	0.892
6	10	10	6	10	6	9	75	1	10	11	40	41	81	81	-5.840	0.810
7	10	15	9	15	2	3	25	3	13	14	54	53	100	100	-8.847	1.015
8	10	15	9	15	4	6	50	1	9	9	42	43	92	90	-6.749	0.897
9	10	15	9	15	6	9	75	2	10	9	37	36	90	88	-14.751	0.858
10	15	5	6	15	2	6	75	1	9	10	56	55	97	98	-11.819	1.001
11	15	5	6	15	4	9	25	2	14	15	77	76	100	100	-22.600	1.107
12	15	5	6	15	6	3	50	3	9	8	45	45	100	100	-10.020	0.978
13	15	10	9	5	2	6	75	2	11	11	52	52	98	98	-5.548	0.992
14	15	10	9	5	4	9	25	3	12	13	48	46	98	95	-7.526	0.962
15	15	10	9	5	6	3	50	1	8	8	52	49	100	100	-4.145	1.000
16	15	15	3	10	2	6	75	3	12	12	45	47	95	97	-7.112	0.955
17	15	15	3	10	4	9	25	1	8	9	62	60	98	98	-16.578	1.027
18	15	15	3	10	6	3	50	2	10	10	45	42	100	100	-11.827	0.974
19	20	5	9	10	2	9	50	1	11	12	50	52	85	89	-14.438	0.902
20	20	5	9	10	4	3	75	2	7	7	52	50	86	84	-15.116	0.883
21	20	5	9	10	6	6	25	3	11	13	55	62	100	100	-14.403	1.035
22	20	10	3	15	2	9	50	2	8	9	55	62	99	100	-14.658	1.029
23	20	10	3	15	4	3	75	3	10	10	55	62	100	98	-14.732	1.026
24	20	10	3	15	6	6	25	1	11	12	62	60	98	98	-16.496	1.029
25	20	15	6	5	2	9	50	3	7	7	45	42	100	100	-12.657	0.972
26	20	15	6	5	4	3	75	1	10	9	50	52	85	89	-13.969	0.900
27	20	15	6	5	6	6	25	2	8	8	52	50	86	84	-14.921	0.883
Average: $\overline{SN} = -11.850$ $\overline{\beta} = 0.965$																

4.3 利用田口方法求解

實驗結果及計算後的 SN 比的回應表及回應圖如表 4-3 與圖 4-3 所示， β 的回應表及回應圖如表 4-4 與圖 4-4 所示。SN 比的變異數分析結果如表 5 所示，較為顯著的因子為 A (21.35%), B (16.765%) 和 G (13.315%)。 β 的變異數分析結果如表 6 所示，較為顯著的因子為 G (24.259%) 與 A (13.633%)。

接著，執行兩階段最佳化程序：

步驟一：最大化 SN 比

此時可暫時設定最佳條件為 $A_1 B_3 C_3 G_2$ ，在此條件下 SN 的估計值為：

$$\begin{aligned}\hat{SN} &= \overline{SN} + (\overline{A_1} - \overline{SN}) + (\overline{B_3} - \overline{SN}) + (\overline{C_3} - \overline{SN}) + (\overline{G_2} - \overline{SN}) \\ &= -11.850 + (-10.15 + 11.850) + (-9.185 + 11.850) + (-10.34 + 11.850) + (-10.17 + 11.850) \\ &= -4.295\end{aligned}$$

β 的估計值為：

$$\begin{aligned}\hat{\beta} &= \overline{\beta} + (\overline{A_1} - \overline{\beta}) + (\overline{B_3} - \overline{\beta}) + (\overline{C_3} - \overline{\beta}) + (\overline{G_2} - \overline{\beta}) \\ &= 0.965 + (0.935 - 0.965) + (0.936 - 0.965) + (0.949 - 0.965) + (0.955 - 0.965) \\ &= 0.965 + (-0.03) + (-0.029) + (-0.016) + (-0.01) \\ &= 0.88\end{aligned}$$

觀察在此最佳組合下的 β 值，若其靠近 1 則不必調整，否則必須進行調整。

由於 β 的估計值為 0.88，此值遠小於 1，所以必須進行調整。

步驟二：調整 β

我們可看出 G 因子對 SN 比的效果不如對 β 的效果顯著，因此，可利用 G 因子來調整 β 值至靠近 1 (理想的量測系統 $\beta=1$)。

當， $G_1=25$ 則 $\hat{SN}=-4.775$ ， $\hat{\beta}=0.937$ 。因此，本實驗可將 G 因子設為 25。經過兩



階段最佳化的調整之後，實驗的最佳條件設為， $A=10, B=15, C=9, D=10, E=2, F=6, G=25, H=3$ 。

表 4-3 SN 比的回應值表

SN	A	B	C	D	E	F	G	H	平均
水準 1	-10.15	-13.95	-13.17	-10.66	-11.09	-11.49	-14.01	-11.63	-12.019
水準 2	-10.8	-9.667	-12.21	-11.49	-12.87	-10.62	-10.34	-13.34	-11.416
水準 3	-14.6	-9.185	-10.17	-13.41	-11.59	-13.44	-11.2	-10.58	-11.771
效果	4.446	4.763	3.002	2.752	1.775	2.819	3.676	2.767	3.250

表 4-4 β 的回應值表

β	A	B	C	D	E	F	G	H	平均
水準 1	0.935	0.981	0.996	0.959	0.990	0.980	1.012	0.955	0.976
水準 2	1.000	0.973	0.951	0.944	0.961	0.960	0.955	0.966	0.964
水準 3	0.962	0.936	0.949	0.993	0.945	0.956	0.929	0.975	0.956
效果	0.065	0.045	0.046	0.050	0.045	0.024	0.082	0.020	0.047

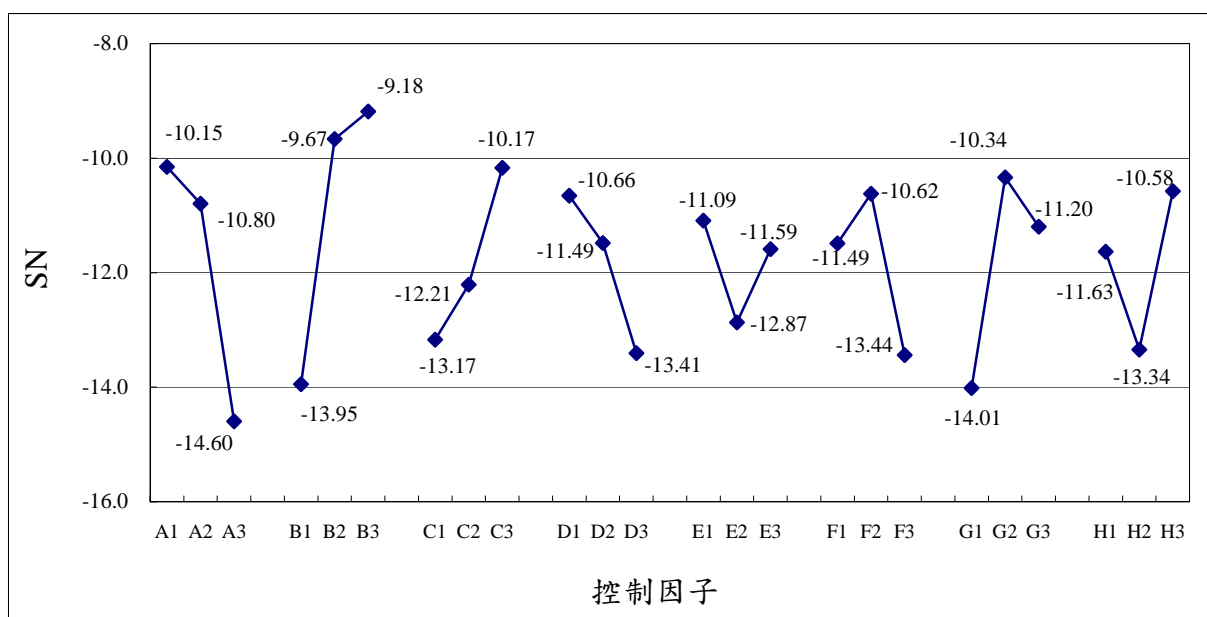


圖 4-3 SN 比的回應圖

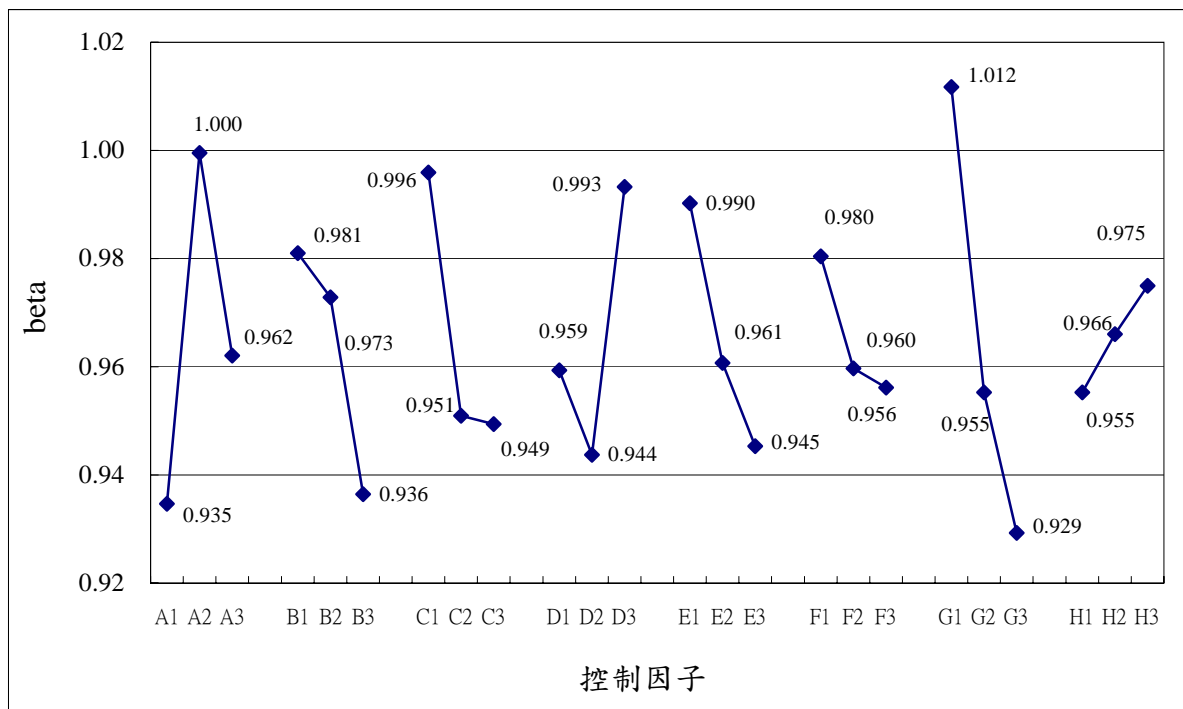


圖 4-4 β 的回應圖

表 4-5 實驗的變異數分析表 (SN)

變源	平方和	自由度	均方和	F	P-值	F-危險域		$\alpha=0.1$		$\alpha=0.05$		貢獻度 (%)
						0.1	0.05	F-test	P-test	F-test	P-test	
A	103.89	2	51.946	11.260	0.002	2.924	4.102	*	*	*	*	21.350
B	82.57	2	41.286	8.949	0.005	2.924	4.102	*	*	*	*	16.765
C	42.28	2	21.142	4.583	0.038	2.924	4.102	*	*	*	*	8.101
D**	35.87	2	17.939	3.888	0.056	2.924	4.102	*	*	-	-	6.723
E**	15.09	2	7.545	1.635	0.242	2.924	4.102	-	-	-	-	2.253
F	37.527	2	18.762	4.067	0.050	2.924	4.102	*	*	-	-	7.078
G	66.52	2	33.264	7.210	0.011	2.924	4.102	*	*	*	*	13.315
H**	35.08	2	17.544	3.803	0.059	2.924	4.102	*	*	-	-	6.553
誤差	46.132	10	4.613									-
(合併誤差)	(86.06)	(6)	(43.029)									(15.531)
總合	464.99	26	17.884									82.142

*代表顯著，**代表合併誤差項

表 4-6 實驗的變異數分析表 (β)

變源	平方和	自由度	均方和	F	P-值	F-危險域		$\alpha=0.1$		$\alpha=0.05$		貢獻度 (%)
						0.1	0.05	F-test	P-test	F-test	P-test	
A	0.019	2	0.009	3.840	0.057	2.924	4.102	*	*	-	-	13.633
B	0.007	2	0.003	1.500	0.269	2.924	4.102	-	-	-	-	4.083
C	0.012	2	0.006	2.525	0.129	2.924	4.102	-	-	-	-	8.2679
D	0.011	2	0.005	2.323	0.148	2.924	4.102	-	-	-	-	7.443
E	0.009	2	0.004	1.890	0.201	2.924	4.102	-	-	-	-	5.677
F**	0.003	2	0.001	0.621	0.556	2.924	4.102	-	-	-	-	0.497
G	0.031	2	0.015	6.443	0.015	2.924	4.102	*	*	*	*	24.259
H**	0.001	2	0.001	0.354	0.710	2.924	4.102	-	-	-	-	0.5956
誤差	0.024	10	0.002									-
(合併誤差)	0.004											1.093
總合	0.121	26	0.004									64.458

*代表顯著，**代表合併誤差項



4.4 利用倒傳遞類神經網路結合啟發式演算法求解

接下來使用倒傳遞類神經網路來建立模式，將八個控制因子的水準值當成輸入值，將其對應的 SN 比當成輸出值，因此我們可以得到 27 組樣本。在這 27 組樣本中，隨機選取三個當成測試 (testing) 樣本，剩下的 24 個當成訓練 (training) 樣本。本研究利用 RMSE 來決定一個類神經網路的好壞，以 RMSE 越小越好。在選擇網路時，我們希望訓練樣本與測試樣本的 RMSE 值，兩者同時小且趨於一致，也就訓練樣本和測試樣本這兩者的 RMSE 要越接近越好。表 4-7 是以倒傳遞網路在不同網路架構及參數設定下之訓練結果，在這些網路架構中，我們選取網路架構 8-5-2 作為本案例的目標函數模式。

表 4-7 類神經網路訓練結果

網路架構	Training RMSE (15000 epochs)	Testing RMSE (10000 epochs)	學習率 (learning rate)	慣性係數 (momentum)
8-2-2	0.0556	0.0598	0.25	0.9
8-3-2	0.0504	0.0498	0.25	0.9
8-4-2	0.0383	0.0419	0.25	0.9
8-5-2	0.0335	0.0352	0.25	0.9
8-6-2	0.0421	0.0489	0.25	0.9
8-7-2	0.0331	0.0380	0.25	0.9
8-8-2	0.0519	0.0649	0.25	0.9
8-9-2	0.0438	0.0615	0.25	0.9

本步驟將前面所獲得的 8-5-2 網路當成分散式搜尋法、基因演算法與模擬退火法的目標函數，藉以求取其最佳值。

在本案例中，回應值 y_1 為 SN， y_2 為 β 值，為多重品質特性。因此，使用指數望想函數將多重品質特性轉換為單一品質特性，利用公式：

$$\lambda = \min(d_1, d_2) \quad (4-1)$$

由於評估 y_1 與 y_2 的重要性同等重要，因此將其皆設為 $t=1$ 。所以，將滿意度(degree of satisfaction; λ) 的值代入基因演算法中的配適函數，而每一個案例中的參數都會透過運算成為介於 0 與 1 之間的標準值。例如，表 4-1 中的輸入參數值($x_1, x_2, x_3, \dots, x_8$) 將轉換為染色體中的可能解，即染色體中的每個基因代表輸入的參數值。接著從輸入參數值中隨機選擇初始族群，交配率為 0.5，突變率為 0.01，一代 100 個染色體，重複次數 2 萬次。

將以上的參數代入演算法重複執行 30 次，表 4-8 為執行結果。其中， λ 的最大值代表有較佳的滿意度，最佳的 λ 值為 0.931 且其最適的染色體組合為(16, 11, 6.4, 9, 2, 5, 46.2, 1)。同樣地，使用望想函數解決多重品質特性問題，執行的結果如表 4-9 所示， λ 的最大值代表有較佳的滿意度，最佳的 λ 值為 0.962 且其最適的染色體組合為(15.4, 8.9, 5.6, 8, 2, 5, 42.6, 1)，這八個參數值即為基因演算法執行結果之最佳製程參數值，表 4-10 為三種演算法的參數設定值。

表 4-8 基因演算法的執行結果 (指數望想函數)

項目	數據資料
執行 30 次之最大 λ 值	0.931
執行 30 次之最小 λ 值	0.862
平均 λ 值	0.911
標準差	0.025

表 4-9 基因演算法的執行結果 (望想函數)

項目	數據資料
執行 30 次之最大 λ 值	0.962
執行 30 次之最小 λ 值	0.924
平均 λ 值	0.946
標準差	0.013

表 4-10 參數設定值

模擬退火法	初始溫度 100, 停止溫度 1, 一個溫度搜尋 500 次, 冷卻因子 $\alpha=0.98$ 。
基因演算法	交配率 0.5, 突變率 0.01, 一代 100 個染色體, 重複次數 2 萬次。
分散式搜尋法	參考集合一和二個數皆為 10, 更新次數 100, 起始解 100 個。

表 4-11 四種方法求得各控制因子最佳值

	A	B	C	D	E	F	G	H
田口方法	10	15	9	10	2	6	25	3
模擬退火法(望想函數)	18	6.53	5.8	8	2.3	4.6	50.8	2
模擬退火法(指數望想函數)	21.5	7.2	6	8	2	6	53.5	2
基因演算法(望想函數)	15.4	8.9	5.6	8	2	5	42.6	1
基因演算法(指數望想函數)	16	11	6.4	9	2	5	46.2	1
分散式搜尋法(望想函數)	10.1	8	4.6	6	2	5	40.3	1
分散式搜尋法(指數望想函數)	12.7	9	6	8	2	4	45.8	1

4.5 確認實驗與比較

田口方法與三種啟發式演算法所求得各控制因子最佳值如表 4-11。將所得到的最佳值實際在教學大樓內做確認實驗，每一個方法重複作二十次實驗，將這二十次實驗所得接收訊號強度之最大值、最小值，平均值及變異數列於表 4-12。由表 4-12 可以明顯的看出四個方法變異數彼此差距不大，但在品質特性上，傳統田口方法比其他三個演算法顯著較差，其確認實驗最大值幾乎是其他三種方法的最小值。確認實驗顯示本研究的方法確實比傳統田口方法更為有效。

表 4-12 確認實驗數據

	最小值	最大值	平均值	變異數
田口方法	80	88	84.67	4.23
模擬退火法(望想函數)	85	94	91.76	4.86
模擬退火法(指數望想函數)	82	94	91.02	5.01
基因演算法(望想函數)	88	96	93.54	4.64
基因演算法(指數望想函數)	86	94	92.69	4.66
分散式搜尋法(望想函數)	89	96	93.59	4.49
分散式搜尋法(指數望想函數)	89	96	92.76	4.51

本實驗針對室內無線區域網路傳輸品質之估測系統，結合類神經網路的架構與啟

發式演算法的技巧，利用演算法對於所量測的數據不斷地學習、訓練與測試之後，比較四種方法確認實驗的接收訊號強度，證明本研究所提的方法，並不需要複雜計算和嚴謹的假設，對於實務應用上有很大的實用性。未來即使要估測不同隔間、擺設的大樓，皆能以此估測系統為基礎，只要量測的數據準確且數目足夠，再經由輸入變數的決定，與類神經網路中各層神經元的調整，及不同網路架構、演算法的修正，本系統即可對其他室內無線網路的通訊品質做出正確的估測。

雖然過去有些學者利用各種訊號產生器的模擬方式來評估無線網路系統，但在實際應用上，很難克服符合統計基本假設和實驗設備限制這兩個問題。本研究所應用的混合演算法利用兩個階段，在第一階段利用類神經網路建立控制因子和 SN 比的函數模式，可以將控制因子之間的交互作用完整的表現出來；在第二階段利用啟發式搜尋法來降低區域網路工程師判斷上的難度，不但可以找出比田口方法更具效果的解，並可減少網路規劃所耗費的時間。



第五章 結論與建議

5.1 結論

本研究應用混合演算法來處理多重品質特性的田口動態離散性問題，由實際案例可以看出其成效。經本研究比較之後，三種啟發式演算法中以模擬退火法成效較差，但模擬退火法在其使用上比其他兩者更簡單，使用時僅需注意鄰域的選法。且由於模擬退火法可以接受比目前解更差的新解，藉此避免區域最佳解情況，故有學者認為只要溫度搜尋次數 M 和起始溫度 T 夠大，模擬退火法有能力找到全區域最佳解。

基因演算法和分散式搜尋法效用是差不多的，但在使用的意義有顯著不同。基因演算法是屬於用固定步驟方式處理任何問題，使用基因演算法搜尋最佳解時，比較不容易陷入局部最佳解，對於使用者而言較方便。另外，在使用基因演算法的程序中，由於是對編碼後的字串來做運算，只要對問題做合適的編碼都可以使用，因此基因演算法可以適用於不同領域的最佳化問題。其缺點是使用者無法將對問題所了解的工程知識，和基因演算法作結合。分散式搜尋法屬於問題導向，在每一個步驟都可以加入使用者自己想法，幫助求解過程的品質更好。利用分散式搜尋法求解時，使用者定義的方式可以和工程知識作結合。但相對上，使用分散式搜尋法比其他兩者要複雜。

本研究應用之混合演算法具有以下優點：

- 一、整合田口方法與倒傳遞類神經網路的方法，利用製程資料進行模擬，不但可以找出重要的製程參數，更可以依據不同的客戶需求決定出個別的製程參數值。
- 二、本研究所應用的混合演算法，求得的最佳解皆屬於連續型數值，比田口方法只能獲取間斷型參數值為佳。
- 三、本研究所應用的方法都能夠有效的處理多重品質特性問題，求出重要的製程參數以及獲得最佳化的參數組合。
- 四、對於只具備一般統計知識訓練的工程師而言，本研究所應用的方法不僅容易應用，對於使用的數據也無需任何的假設條件，具有高度的可行性。

由於方法論在想法上有本質的差異，所以使用者可以依自己對問題及方法了解程度，從中選一。但不論是哪一種演算法，都可以比傳統田口方法得到更好的答案。本研究運用倒傳遞類神經網路架構與啟發式演算法兩個階段結合的混合演算法，並不需要複雜計算和嚴謹的假設，對於實務應用上有很大的實用性。加上使用者只須具備田口方法知識、會使用類神經網路及套裝軟體，即可進行分析，對於採用上有很大的方

便性。

5.2 建議

本研究所應用之混合演算法結合倒傳遞類神經網路架構結合啟發式演算法，未來建議可針對啟發式演算法做修正，以符合不同品質特性的問題。

在案例中，本研究以「接收訊號強度」做為品質特性，未來根據不同的需求可考慮以連線品質(link quality)、訊框錯誤率(frame error rate; FER)或傳輸速率(packet per second or throughput)等不同的傳輸品質指標，取代本實驗中的「接收訊號強度」做為品質特性。另外在評估系統的輸入變數方面，可考慮再加入不同的筆記型電腦或是 PDA、WAP 手機等同時進行預估。利用倒傳遞類神經網路易建立系統模型之能力，整合所有的傳輸品質之參考指數，可更完整的整合估測系統，為網路規劃者帶來更多的參考數值，設計出更佳的無線區域網路系統。



參考文獻

一、中文部份

- [1] 李善誠，「以混合式螞蟻族群最佳化解決無線通訊頻道指定問題」，國立暨南國際大學資訊管理學系，碩士論文，民國 94 年。
- [2] 林正鄰，「模擬退火與類神經網路在製程最適化之應用」，國立清華大學化學工程研究所，博士論文，民國 82 年。
- [3] 林欣志，「模擬退火法應用於倒傳遞類神經網路之參數調整與屬性篩選」，華梵大學資訊管理學系，碩士論文，民國 95 年。
- [4] 洪饒峰，「田口方法動態系統多品質特性之應用研究」，中原大學工業工程研究所，碩士論文，民國 85 年。
- [5] 周立德，「類神經網路與基因演算法應用於非同步傳送模式網路之研究」，國立台灣科技大學工程技術研究所，博士論文，民國 83 年。
- [6] 周柏宏，「多目標超模式法解參數設計問題之研究—以半導體封裝印字製程為例」，國立成功大學製造工程研究所，碩士論文，民國 92 年。
- [7] 姜台林，「整合智慧型最佳化參數設計之研究」，國立交通大學工業工程與管理學系，博士論文，民國 90 年。
- [8] 涂育璋，「應用類神經網路模式與基因演算法則於品質設計之研究」，國立成功大學工業設計學系，碩士論文，民國 91 年。
- [9] 陳士偉，「備用零件存貨管理之研究」，國立台北科技大學商業自動化與管理研究所，碩士論文，民國 93 年。
- [10] 陳文江，「高速無線區域網路的發展與標準制定的近況」，電腦與通訊，第七十九期，3~8 頁，民國 88 年。
- [11] 陳明宏，「多種品質特性在動態系統中之研究」，中原大學工業工程研究所，碩士論文，民國 83 年。
- [12] 章俊彥，「階層式室內定位系統與其服務」，國立中央大學資訊工程研究所，碩士論文，民國 93 年。
- [13] 彭佳芸，「智慧城市 台北未來『無線』」，台視全球資訊網，民國 95 年。
(<http://www.ttv.com.tw/news/html/095/06/0950629/095062990606018.htm>)

- [14] 黃國凌，「螞蟻/塔布混合演算法求解零工型排程問題」，國立台灣科技大學工業管理系，碩士論文，民國 92 年。
- [15] 黃寶賢，「應用物件導向技術及智慧型計算建構半導體製程之最佳化品質控制系統—以化學機械研磨為例」，大葉大學工業工程學系，碩士論文，民國 91 年。
- [16] 張旭華，「運用柔性演算法求解最佳參數設計」，國立交通大學工業工程與管理學系，博士論文，民國 88 年。
- [17] 張家瑄，「結合類神經網路與基因演算法於系統識別」，朝陽科技大學營建工程系，碩士論文，民國 91 年。
- [18] 葉謀銓，「應用啟發式解法求解多項式普羅比模式之最大概似估計值」，暨南國際大學土木工程學系，碩士論文，民國 92 年。
- [19] 詹緒林，「以基因演算法及類神經網路為基礎的電腦輔助和聲編寫系統」，中華大學電機工程研究所，碩士論文，民國 86 年。
- [20] 潘泰吉，「無線區域網路系列-1」，無線電界，第八十卷第六期，73~77 頁，民國 88 年。
- [21] 鄒文杰，「應用類神經網路與共通式演算法於參數設計最佳化—以積體電路焊線製程為例」，國立交通大學工業工程與管理學系，碩士論文，民國 91 年。
- [22] 劉吉峰，「分散搜尋法於配水管網最佳化設計之應用」，國立中興大學環境工程學系，碩士論文，民國 93 年。
- [23] 盧炳勳，「組織因素對資訊系統規劃之影響探討」，國立中央大學資訊管理學系，碩士論文，民國 96 年。
- [24] 盧研伯，「混合式模擬退火法應用於具迴流特性流程工廠之研究」，國立台灣科技大學工業管理系，碩士論文，民國 92 年。
- [25] 蕭綱衡，「田口式參數設計在鐵礦燒結之應用研究」，中國統計學報，第二十八卷第二期，253~275 頁，民國 79 年。
- [26] 蘇朝墩，品質工程，初版，台北，中華民國品質學會，民國九十一年。
- [27] 魏忠堅，「經濟部技術處通過投入鎂合金低壓成形整合系統技術開發等 5 項計畫」，產業資訊服務電子報，第 169 期，民國 95 年。
(http://cdnet.stpi.org.tw/techroom/policy/policy_06_058.htm)

二、英文部份

- [28] Balas, E. and Vazacopoulos, A. (1998) Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, **44**, 262-275.
- [29] Bauer Jr., R. J. (1994) *Genetic Algorithms and Investment Strategies*. John Wiley & Sons. Behzad, K. P. and Kamgar-Parsi, B. (1990) On problem solving with Hopfield Neural Networks. *Biological Cybernetics*, **62**, 415-423.
- [30] Cerny, V. (1985) Thermo dynamical Approach to the Traveling Salesman Problem: An efficient simulation algorithm. *Journal of Optimization Theory and Application*, **45**, 41-51.
- [31] Chan, H. L. and Liang, S. K. (2005) Yield improvement for Blood Glucose Strip by Taguchi Dynamic Approach. *International Journal of Information Systems for Logistics and Management*, **1**(1), 9-16.
- [32] Chang, P. T. and Lo, Y. T. (2001) Modeling of job shop scheduling with multiple quantitative and qualitative objectives and a GA/TS mixture approach. *International Journal of Computer Integrated Manufacturing*, **14**, 367-384.
- [33] Chang, S. I. and Shivpuri, R. (1995) A Multiple-Objective Decision Making Approach for Assessing Simultaneous Improvement in Die Life Casting Quality in A Die Casting Process. *Quality Engineering*, **7**(2), 371-383.
- [34] Chapman, R. E. (1996) Photochemistry Multiple Response Co-Optimization.. *Quality Engineering*, **8**(1), 31-45.
- [35] Cheung, K. W. (1998) A New Empirical Model for Indoor Propagation Prediction. *IEEE Transactions on Vehicular technology*, **47**(8), 996-1001.
- [36] Chiu, C. C. (1996) Coverage Prediction in Indoor Wireless Communication. *IEICE Transactions on Communications*, **79**(9), 1346-1350.
- [37] Chryssolouris, G. and Guillot, M. A. (1990) Comparison of Statistical and AI Approaches to the Selection of Process Parameters in Intelligent Machining. *American Society of Mechanical Engineers Journal of Engineering*, **112**(2), 122-131.
- [38] Cung, V. D., Mautor, T., Michelon, P. and Tavares, A. (1997). A Scatter Search Based Approach for the Quadratic Assignment Problem. *IEEE International Conference on Evolutionary Computation*, 165-169.

- [39] De Jong, K. A. (1987) On Using Genetic Algorithms to Search Program Space. *Proceedings of the Second International Conference on Genetic Algorithm and Their Application*, 210-216.
- [40] Derringer, G. and Suich, R. (1980) Simultaneous Optimization of Several Response Variables. *Journal of Quality Technology*, **12**(4), 214-219.
- [41] Fleurent, C., Glover, F., Michelon, P. and Valli, Z.(1996) A Scatter Search Approach for Unconstrained Continuous Optimization. *Proceedings of IEEE International Conference on Evolutionary Computation*, 643 -648.
- [42] Fowlkes, W. Y. and Creveling, C. M. (1995) *Engineering Methods for Robust Product Design*. Addison-Wesley.
- [43] Funahashi, K. (1989) On the Approximate Realization of Continuous Mapping by Neural Network. *Neural Networks*, **2**, 183-192.
- [44] Glover, F. (1965) A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem. *Operations Research*, **13**(6), 879.
- [45] Glover, F. (1986) Future Path for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, **13**, 533-549.
- [46] Glover, F. (1994) Tabu Search for Nonlinear Parametric Optimization (with links to Genetic Algorithm. *Discrete Applied Mathematics*, **49**, 231-55.
- [47] Glover, F. (1999) *Scatter search and path relinking*. In: Corne D, Dorigo M, Glover F(eds), *New Ideas in Optimization*, McGraw-Hill, London.
- [48] Glover, F. (2000) Laguna M, Marti R, *Fundamentals of Scatter Search and Path Relinking*. *Control and Cybernetics*, **39**(3), 653-684.
- [49] Goik, P., Liddy , J. W. and Taam, W. (1995) Use of Desirability Functions to Determine Operating Windows for New Product Designs. *Quality Engineering*, **7**(2), 267-276.
- [50] Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.
- [51] Holland, J. (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press.
- [52] Ichimura, T., Oeda, S. and Yoshida, K. (2001) An adaptive evolutionary learning method using genetic search and extraction of rules from trained networks. *Journal of Evolutionary Computation*, **2**, 1343-1350.
- [53] Johnson, D. S., Aragon, C. R., McGroch, L. A. and Schevon , C. (1989) Optimization by simulated annealing : An experiment evaluation. *Operations*

Research, **37**, 865-893.

- [54] Kohonen, T. (1989) *Self-Organization and Associative Memory*. Springer-Verlag, Berlin.
- [55] Kolonko, M. (1999) Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, **113**, 123-136.
- [56] Kirkpatrick, S., Gelatt, C. D, and Vecchi, M. P. (1983) Optimization by Simulated Annealing. *Science*, **220**,671-680.
- [57] Kumar, K.K. and Smuda, E. (1994) Robust controll using ga-optimized neural networks. *Journal of Control Applications*, **3**, 1573-1578.
- [58] Laguna, M., Lino, P., Pérez, A., Quintanilla, S. and Valls, V. (2000) Minimizing Weighted Tardiness of Jobs with Stochastic Interruptions in Parallel Machines. *European Journal of Operational Research*, **127**(2), 444-457.
- [59]Leinweber, D. J. and Arnott, R. D. (1990) Quantitative and Computational Innovation in Investment Management. *Journal of Portfolio Management*, **21**(2) , 8-15.
- [60] Li, K.L., Huang, H.S., Lin, J.D. ,Huang, B.Y. ,Huang M.J. and Wang P.W.(1994) Comparing Self-Monitoring Blood Glucose Devices. *Laboratory Medicine*, **25**, 585-591.
- [61] Logothetis, N. and Haigh, A. (1988) Charaterizing and Optimizing Multi-Response Process by the Taguchi Method. *Quality and Reliability Engineering International*, **4**(2),159-169.
- [62] McCulloch, W. S. and Pitts, W. (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115-133.
- [63] Metropolis, N., Rosenbluth. , A., Teller, A. and Teller. E. (1953) Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, **21**, 1087-1092.
- [64] Montgomery, D. C. (2004) *Design and Analysis of Experiments*. Wiley, New York.
- [65] Murovec, B. and Suhel, P. (2004) A repairing technique for the local search of the job-shop problem. *European Journal of Operational Research*, **153**, 220-238.
- [66] Nitta, T. (1994) An analysis on decision boundaries in the complex back-propagation network. *IEEE International Conference on Neural Networks*, **2**, 934-939.
- [67] Packard, N. H. (1987) A Genetic Learning Algorithm for the Analysis of Complex Data. *Complex Systems*, **4**, 543-572.

- [68] Panjwani, M. A. (1996) Interactive Computation of Coverage Regions for Wireless Communication in Multifloored Indoor Environments. *IEEE Journal on Selected Areas in Communications*, **14**(3), 420-429.
- [69] Pezzella, F. and Merelli, E. (2000) A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, **120**, 297-310.
- [70] Phadke, M. S. (1989) *Quality Engineering Using Robust Design*. Prentice Hall, Englewood Cliffs, New Jersey.
- [71] Su, C. T., Chen, M. C. and Chan, H. L. (2005) Applying Neural Network and Scatter Search to Optimize Parameter Design with Dynamic Characteristics, *Journal of the Operational Research Society*, **56**(10), 1132-1140.
- [72] Su, C. T. and Chang, H. H. (2000) Optimization of Parameter Design: An Intelligent Approach Using Neural Network and Simulated Annealing. *International Journal of Systems Science*, **13**(12), 1543-1549.
- [73] Su, C. T. and Chiang, T. L. (2002) Optimal Design for a Ball Grid Array Wire Bonding Process Using a Neural-Genetic Approach. *IEEE Transactions on Electronics Packaging Manufacturing*, **25**(1), 1-7.
- [74] Taguchi G. (1990) *Introduction to Quality Engineering*. Asian Productivity Organization, Tokyo.
- [75] Van Laarhoven, P. J. M., Aarts, E. H. L. and Lenstra, J. K. (1992) Job shop scheduling by simulated annealing. *Operations Research*, **40**, 113-125.
- [76] Vining, G. S. and Mylers, R. H. (1990) Combining Taguchi and response surface philosophies: a dual response approach. *JQT*, **22** (1), 38-45.
- [77] Wang, L. and Zheng, D. Z. (2001). An effective optimization strategy for job shop scheduling problems. *Computers & Operations Research*, **28**, 585-596.
- [78] William, L.C., Daniel, C., Linda, A., William, C. and Stephen, L. (1987) Evaluating Clinical Accuracy of Systems for Self-Monitoring of Blood Glucose. *Diabetes Care*, **10**, 622-628.
- [79] Yamada, T. and Nakano, R. (1996) *Job-shop scheduling by simulated annealing combined with deterministic local search. Meta-heuristics: Theory and Applications*. Kluwer Academic Publishers, Hingham, MA.
- [80] Young, D. L., Teplik, J., Weed, H. D., Tracht, N. T. and Alvarez, A. R. (1991) Application of Statistical Design and Response Surface Methods to Computer-Aided VLSI Device Design II : Desirability Functions and Taguchi Methods. *IEEE*



附 錄

分散式搜尋法之 C 語言程式碼

1. SS_main

```
#include "ss.h"

double evaluate(SS *prob,double *x)
{
    double value;
    int nvar;

    nvar=prob->n_var;

    value = 100*pow(x[2]-pow(x[2],2),2)+pow(1-x[1],2)+90*pow(x[4]-pow(x[3],2),2)+
pow(1-x[3],2)+10.1*pow(x[2]-1,2)+10.1*pow(x[4]-1,2)+19.8*(x[2]-1)*(x[4]-1);

    return value;
}

void main(void)
{
    int MaxIter,Iter,i,nvar,b1,b2,PSize,LocalSearch;
    double *best,best_value,LowerBound,UpperBound;
    SS *prob;

    /* Parameters Defined by the User */
    MaxIter    = 2; /* Number of global iterations */
    nvar       = 4; /* Number of variables (size of x) */
    LowerBound = -10; /* Lower Bound of all variables */
    UpperBound = 10; /* Upper Bound of all variables */
}
```

```

b1      = 10; /* Size of reference set 1 (quality) */
b2      = 10; /* Size of reference set 2 (diversity) */
PSize   = 100; /* Size of the diversification generator population */
LocalSearch= 1; /* =1 Nelder and Mead Simplex Method activated,=0 OFF */

```

```

prob=DataStructures_init(nvar,b1,b2,PSize,LocalSearch);

```

```

/* Assign Variable Bounds */

```

```

for(i=1;i<=nvar;i++)
{
    prob->low[i] = LowerBound;
    prob->high[i] = UpperBound;
}

```

```

/* Build Reference Set */

```

```

Initiate_RefSet(prob);

```

```

/* Perform Search */

```

```

for(Iter=1;Iter<=MaxIter;Iter++)
{
    if(prob->new_elements)
        Combine_RefSet(prob);
    else
    {
        Update_RefSet2(prob);
        Combine_RefSet(prob);
    }
}

```



```

/**** Best Solution ****/

```

```

best_value = prob->value1[prob->order1[1]];

```

```

best = SAllocate_double_array(nvar);

```

```

for(i=1;i<=nvar;i++) best[i]=prob->RefSet1[prob->order1[1]][i];

```

```

printf("\n\nBest Solution Found. Value = %f\n\n",best_value);
for(i=1;i<=nvar;i++) printf("solution[%d] = %f\n",i,best[i]);
free(best);

```

```

Free_DataStructures(prob);
}

```

2. simplex

```

/* Nelder and Mead Simplex Method for Nonlinear unconstrained optimization problems
*/

```

```

#include "ss.h"

```

```

#define NMAX 5000
#define ALPHA 1.0
#define BETA 0.5
#define GAMMA 2.0

```



```

#define GET_PSUM for (j=1;j<=ndim;j++) { for (i=1, sum=0.0;i<=mpts;i++) sum +=
p[i][j]; psum[j]=sum; }

```

```

int amoeba(double **p, double *y, int ndim, double ftol, double (*funk)(), int *nfunk, SS
*prob)

```

```

{
    int i, j, ilo, ihi, inhi, mpts=ndim+1;
    double ytry, ysave, sum, rtol, *psum;

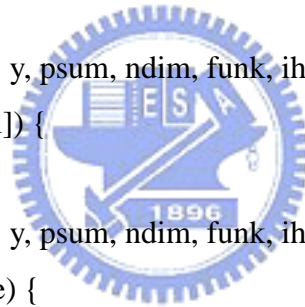
    psum = SSallocate_double_array(ndim+1);
    *nfunk = 0;
    GET_PSUM
    for (;;) {
        ilo = 1;

```

```

ihi = y[1] > y[2] ? (inhi = 2, 1) : (inhi = 1, 2);
for (i = 1; i <= mpts; i++) {
    if (y[i] < y[ilo]) ilo = i;
    if (y[i] > y[ihi]) {
        inhi = ihi;
        ihi = i;
    } else if (y[i] > y[inhi])
        if (i != ihi) inhi = i;
}
rtol = 2.0*fabs(y[ihi]-y[ilo])/(fabs(y[ihi])+fabs(y[ilo]));
if (rtol < ftol) break;
if (*nfunk >= NMAX)
    return -1;
ytry = amotry(p, y, psum, ndim, funk, ihi, nfunk, -ALPHA,prob);
if (ytry <= y[ilo])
    ytry = amotry(p, y, psum, ndim, funk, ihi, nfunk, GAMMA,prob);
else if (ytry >= y[inhi]) {
    ysave = y[ihi];
    ytry = amotry(p, y, psum, ndim, funk, ihi, nfunk, BETA,prob);
    if (ytry >= ysave) {
        for (i = 1; i <= mpts; i++) {
            if (i != ilo) {
                for (j = 1; j <= ndim; j++) {
                    psum[j] = 0.5*(p[i][j]+p[ilo][j]);
                    p[i][j] = psum[j];
                }
                y[i] = (*funk)(prob,psum);
            }
        }
        *nfunk += ndim;
        GET_PSUM
    }
}
}
}

```



```

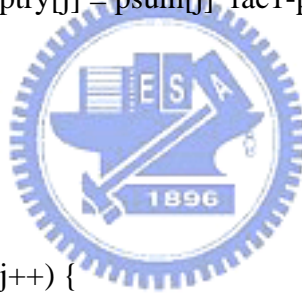
    free(psum);
    return 0;
}

double amotry(double **p, double *y, double *psum, int ndim, double (*funk)(), int ihi, int
*nfunk, double fac,SS *prob)

{
    int j;
    double fac1, fac2, ytry, *ptry;

    ptry = SSallocate_double_array(ndim+1);
    fac1 = (1.0-fac)/(double)ndim;
    fac2 = fac1 - fac;
    for (j = 1; j <= ndim; j++) ptry[j] = psum[j]*fac1-p[ihi][j]*fac2;
    ytry = (*funk)(prob,ptry);
    ++(*nfunk);
    if (ytry < y[ihi]) {
        y[ihi] = ytry;
        for (j = 1; j <= ndim; j++) {
            psum[j] += ptry[j]-p[ihi][j];
            p[ihi][j] = ptry[j];
        }
    }
    free(ptry);
    return ytry;
}

```



3. SS_Memory

```
#include "ss.h"
```

```
SS *DataStructures_init(int nvar,int b1,int b2,int PSize,int LocalSearch)
```

```
{
```

```

SS *problem;

problem = (SS*) calloc(1,sizeof(SS));
if(!problem) SSabort("Memory allocation problem");

problem->n_var = nvar;
problem->b1 = b1;
problem->b2 = b2;
problem->PSize = PSize;
problem->LS = LocalSearch;
problem->digits = 0;

problem->last_combine = 0;
problem->iter=0;

problem->high = SSallocate_double_array(nvar);
if(!problem->high) SSabort("Memory allocation problem");

problem->low = SSallocate_double_array(nvar);
if(!problem->low) SSabort("Memory allocation problem");

problem->ranges = SSallocate_int_matrix(nvar,5);
if(!problem->ranges) SSabort("Memory allocation problem");

problem->value1 = SSallocate_double_array(b1);
if(!problem->value1) SSabort("Memory allocation problem");

problem->RefSet1 = SSallocate_double_matrix(b1,nvar);
if(!problem->RefSet1) SSabort("Memory allocation problem");

problem->order1 = SSallocate_int_array(b1);
if(!problem->order1) SSabort("Memory allocation problem");

problem->iter1 = SSallocate_int_array(b1);

```



```

if(!problem->iter1) SSabort("Memory allocation problem");

problem->value2 = SSallocate_double_array(b2);
if(!problem->value2) SSabort("Memory allocation problem");

problem->RefSet2 = SSallocate_double_matrix(b2,nvar);
if(!problem->RefSet1) SSabort("Memory allocation problem");

problem->order2 = SSallocate_int_array(b2);
if(!problem->order1) SSabort("Memory allocation problem");

problem->iter2 = SSallocate_int_array(b2);
if(!problem->iter2) SSabort("Memory allocation problem");

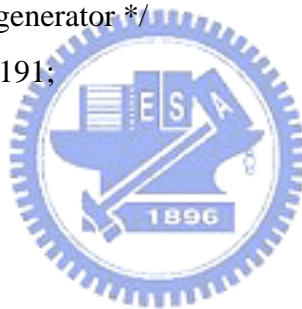
/* Parameters for random generator */
problem->idum = 13171191;
problem->seed_reset = 1;
problem->iff = 0;

return problem;
}

void Free_DataStructures(SS *prob)
{
    SSfree_double_matrix(prob->RefSet1,prob->b1);
    free(prob->value1);
    free(prob->order1);
    free(prob->iter1);

    SSfree_double_matrix(prob->RefSet2,prob->b2);
    free(prob->value2);
    free(prob->order2);
    free(prob->iter2);
}

```



```

SSfree_int_matrix(prob->ranges,prob->n_var);

free(prob->high);
free(prob->low);
free(prob);
}

int **SSallocate_int_matrix(int rows,int columns)
{
    int **aux;
    int i;

    aux=(int**)calloc(rows+1,sizeof(int*));
    if(!aux) SSabort("Memory allocation problem");

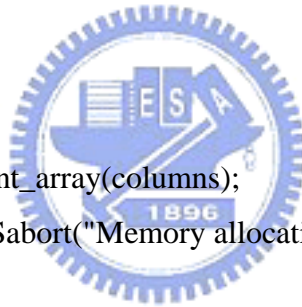
    for(i=1;i<=rows;i++)
    {
        aux[i] = SSallocate_int_array(columns);
        if(aux[i]==NULL) SSabort("Memory allocation problem");
    }
    return aux;
}

double **SSallocate_double_matrix(int rows,int columns)
{
    double **aux;
    int i;

    aux=(double**)calloc(rows+1,sizeof(double*));
    if(!aux) SSabort("Memory allocation problem");

    for(i=0;i<=rows;i++)
    {
        aux[i] = SSallocate_double_array(columns);

```



```

        if(!aux[i]) SSabort("Memory allocation problem");
    }
    return aux;
}

```

```

double *SSallocate_double_array(int size)

```

```

{
    double *aux;

    aux=(double*)calloc(size+1,sizeof(double));
    if(!aux) SSabort("Memory allocation problem");
    return aux;
}

```

```

int *SSallocate_int_array(int size)

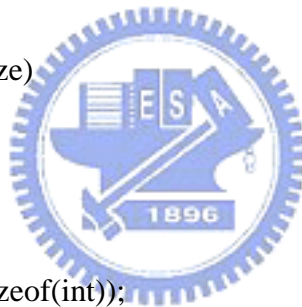
```

```

{
    int *aux;

    aux=(int*)calloc(size+1,sizeof(int));
    if(!aux) SSabort("Memory allocation problem");
    return aux;
}

```



```

void SSfree_double_matrix(double **matrix,int rows)

```

```

{
    int i;

    for(i=0;i<=rows;i++)
        free(matrix[i]);
    free(matrix);
}

```

```

void SSfree_int_matrix(int **matrix,int rows)

```

```

{
    int i;

    for(i=1;i<=rows;i++)
        free(matrix[i]);
    free(matrix);
}

```

4. SS_RefSet

```
#include "ss.h"
```

```
void Initiate_RefSet(SS *prob)
```

```

{
    double *current,*min_dist,*value,**solutions;
    int j,i,i1,a,*index,*index2,cont=0;
    double d,dmax,current_value;

    prob->iter=1;
    prob->new_elements = 1;

    current = SSallocate_double_array(prob->n_var);
    if(!current) SSabort("Memory allocation problem");

    min_dist = SSallocate_double_array(prob->PSize);
    if(!min_dist) SSabort("Memory allocation problem");

    value = SSallocate_double_array(prob->PSize);
    if(!min_dist) SSabort("Memory allocation problem");

    solutions = SSallocate_double_matrix(prob->PSize,prob->n_var);
    if(!solutions) SSabort("Memory allocation problem");

    for(i=1;i<=prob->PSize;i++)

```



```

{
    /* Generate new solution */
    for(j=1;j<=prob->n_var;j++)
        current[j] = SSGenerate_value(prob,j);

    /* Evaluate Solution */
    current_value=evaluate(prob,current);

    if(prob->LS) SSimprove_solution(prob,current,&current_value);

    if(is_new(prob,solutions,i-1,current))
    {
        /* Store solution in matrix "solutions" */
        for(j=1;j<=prob->n_var;j++)
            solutions[i][j] = current[j];

        value[i]=current_value;
    }
    else
    {
        i--;
        cont++;
    }

    if(cont>prob->PSize/2)
    {
        prob->digits++;
        cont=0;
    }
}

index = orden_indices(value,prob->PSize,-1);

/* Add the best b1 to RefSet1 */

```



```

for(i=1;i<=prob->b1;i++)
{
    for(j=1;j<=prob->n_var;j++)
        prob->RefSet1[i][j] = solutions[index[i]][j];

    prob->value1[i] = value[index[i]];
    prob->order1[i] = i;
    prob->iter1[i] = 1;
}

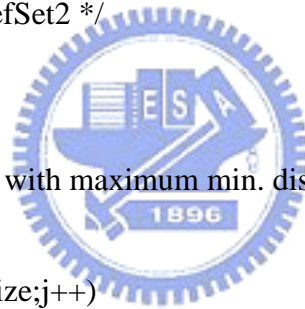
/* Compute minimum distances */
for(i=1;i<=prob->PSize;i++)
    min_dist[i]=distance_to_RefSet1(prob,solutions[i]);

/*Add the second b2 to RefSet2 */
for(i=1;i<=prob->b2;i++)
{
    /* Select the solution with maximum min. dist */
    dmax = -1;
    for(j=1;j<=prob->PSize;j++)
        if(min_dist[j]>dmax)
        {
            dmax=min_dist[j];
            a=j;
        }

    for(j=1;j<=prob->n_var;j++)
        prob->RefSet2[i][j] = solutions[a][j];
    prob->value2[i] = min_dist[a];

    /* Update minimum distances */
    for(i1=1;i1<=prob->PSize;i1++)
    {
        d=0;

```




```

        for(j=1;j<=prob->n_var;j++)
            d += pow(solutions[i1][j]-solutions[a][j],2);
        if(d<min_dist[i1])
            min_dist[i1]=d;
    }
}

/* Update minimum distances in RefSet2 */

for(i=1;i<=prob->b2;i++)
{
    for(a=1;a<=prob->b2 && a!=i ;a++)
    {
        d=0;
        for(j=1;j<=prob->n_var;j++)
            d += pow(prob->RefSet2[i][j]-prob->RefSet2[a][j],2);
        if(prob->value2[i] > d)
            prob->value2[i]=d;
    }
}

index2 = orden_indices(prob->value2,prob->b2,1);

for(i=1;i<=prob->b2;i++)
{
    prob->order2[i] = index2[i];
    prob->iter2[i] = 1;
}

free(index);free(index2);free(current);free(min_dist);
free(value);SSfree_double_matrix(solutions,prob->PSize);
}

```

```

void Update_RefSet2(SS *prob)
{
    double *current,*min_dist,*value,**solutions;
    int j,i,i1,a,*index2,cont=0;
    double d,dmax;

    prob->iter++;
    prob->digits++;

    current = SSallocate_double_array(prob->n_var);
    if(!current) SSabort("Memory allocation problem");

    min_dist = SSallocate_double_array(prob->PSize);
    if(!min_dist) SSabort("Memory allocation problem");

    value = SSallocate_double_array(prob->PSize);
    if(!min_dist) SSabort("Memory allocation problem");

    solutions = SSallocate_double_matrix(prob->PSize,prob->n_var);
    if(!solutions) SSabort("Memory allocation problem");

    for(i=1;i<=prob->PSize;i++)
    {
        /* Generate new solution */
        for(j=1;j<=prob->n_var;j++)
            current[j] = SSGenerate_value(prob,j);

        if(is_new(prob,solutions,i-1,current))
        {
            /* Store solution in matrix "solutions" */
            for(j=1;j<=prob->n_var;j++)
                solutions[i][j] = current[j];
        }
        else

```

```

    {
        i--;
        cont++;
    }

    if(cont>prob->PSize/2)
    {
        prob->digits++;
        cont=0;
    }
}

/* Compute minimum distances */
for(i=1;i<=prob->PSize;i++)
    min_dist[i]=distance_to_RefSet1(prob,solutions[i]);

/*Add to RefSet */
for(i=1;i<=prob->b2;i++)
{
    /* Select the solution with maximum min. dist */
    dmax = -1;
    for(j=1;j<=prob->PSize;j++)
        if(min_dist[j]>dmax)
        {
            dmax=min_dist[j];
            a=j;
        }

    for(j=1;j<=prob->n_var;j++)
        prob->RefSet2[i][j] = solutions[a][j];

    /* Update minimum distances */
    for(i1=1;i1<=prob->PSize;i1++)
    {

```



```

        d=0;
        for(j=1;j<=prob->n_var;j++)
            d += pow(solutions[i1][j]-solutions[a][j],2);
        if(d<min_dist[i1])
            min_dist[i1]=d;
    }
}

/* Update minimum distances in RefSet2 */

for(i=1;i<=prob->b2;i++)
{
    for(a=1;a<=prob->b2 && a!=i ;a++)
    {
        d=0;
        for(j=1;j<=prob->n_var;j++)
            d += pow(prob->RefSet2[i][j]-prob->RefSet2[a][j],2);
        if(min_dist[i] > d)
            min_dist[i]=d;
    }
    prob->value2[i] = min_dist[i];
}

index2 = orden_indices(prob->value2,prob->b2,1);

for(i=1;i<=prob->b2;i++)
{
    prob->order2[i] = index2[i];
    prob->iter2[i] = prob->iter;
}

prob->new_elements = 1;

free(index2);

```

```

    free(current);
    free(min_dist);
    free(value);
    SSfree_double_matrix(solutions,prob->PSize);
}

void Combine_RefSet(SS *prob)
{
    int i,j,a,s,pull_size,total_size;
    double **offsprings,**pull;

    prob->new_elements=0;
    offsprings = SSallocate_double_matrix(4,prob->n_var);

    /* New solutions are temporarily stored in a pull */
    pull_size=0;
    total_size=(4*prob->b1*prob->b1)/2+(3*prob->b1*prob->b2)+
        (2*prob->b2*prob->b2)/2;
    pull = SSallocate_double_matrix(total_size,prob->n_var);

    /* Combine elements in RefSet1 */
    for(i=1;i<prob->b1;i++)
    for(j=i+1;j<=prob->b1;j++)
    {
        /* Combine solutions not combined in the past */
        if(prob->iter1[i]>prob->last_combine ||
            prob->iter1[j]>prob->last_combine )
        {
            SScombine(prob,prob->RefSet1[i],prob->RefSet1[j],offsprings,4);

            for(a=1;a<=4;a++)
            {
                pull_size++;

```

```

        for(s=1;s<=prob->n_var;s++)
            pull[pull_size][s]=offsprings[a][s];
    }
}
}

```

```

/* Combine RefSet1 with RefSet2*/

```

```

for(i=1;i<= prob->b1;i++)
for(j=1;j<= prob->b2;j++)
{
    if(prob->iter1[i]>prob->last_combine ||
        prob->iter2[j]>prob->last_combine    )
    {
        SScombine(prob,prob->RefSet1[i],prob->RefSet2[j],offsprings,3);

        for(a=1;a<=3;a++)
        {
            pull_size++;
            for(s=1;s<=prob->n_var;s++)
                pull[pull_size][s]=offsprings[a][s];
        }
    }
}

```

```

/* Combine elements in Refset2 */

```

```

for(i=1;i<prob->b2;i++)
for(j=i+1;j<=prob->b2;j++)
{
    if(prob->iter2[i]>prob->last_combine ||
        prob->iter2[j]>prob->last_combine    )
    {
        SScombine(prob,prob->RefSet2[i],prob->RefSet2[j],offsprings,2);
    }
}

```



```

        for(a=1;a<=2;a++)
        {
            pull_size++;
            for(s=1;s<=prob->n_var;s++)
                pull[pull_size][s]=offsprings[a][s];
        }
    }
}

```

/* Update, if necessary, Reference Set */

```

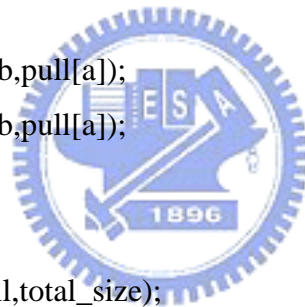
prob->last_combine=prob->iter;
prob->iter++;
for(a=1;a<=pull_size;a++)

```

```

{
    try_add_RefSet1(prob,pull[a]);
    try_add_RefSet2(prob,pull[a]);
}

```



```

SSfree_double_matrix(pull,total_size);
SSfree_double_matrix(offsprings,4);

```

}

double SSGenerate_value(SS *prob,int a)

```

{
    int i,j;
    int *rfrec; /* reverse frec to penalize high frecs. */
    double r,value,low,range;
    int *frec;

    frec = prob->ranges[a];
    low  = prob->low[a];

```

```

range= prob->high[a]-prob->low[a];

rfrec = SSallocate_int_array(5);
if(!rfrec) SSabort("Problems allocating memory");

for(i=1;i<=4;i++)
{
    rfrec[i] = frec[0] - frec[i];
    rfrec[0] += rfrec[i];
}

if(rfrec[0]==0)
    i = SSgetrandom(prob,1,4);
else
{
    /* Select a subrange (from 1 to 4) according to rfrec */
    j = SSgetrandom(prob,1,rfrec[0]);
    i=1;
    while(j>rfrec[i])
        j -= rfrec[i++];
}
if(i>4) SSabort("Problems generating values");

/* i is the selected subrange */
frec[i]++;
frec[0]++;
free(rfrec);

/* Randomly select an element in subrange i */
r = SSrandNum(prob);
value=low+(i-1)*(range/4) + (r*range/4);
return value;
}

```



```

void SSImprove_solution(SS *prob, double *sol, double *value)
{
    double **p, *y, range, perturb;
    int i, j, a, nfunk, best_sol;

    p = SSallocate_double_matrix(prob->n_var+1, prob->n_var);
    y = SSallocate_double_array(prob->n_var+1);

    for(i=1; i<=prob->n_var; i++)
        p[1][i]=sol[i];

    y[1]=*value;

    for(j=1; j<=prob->n_var; j++)
    {
        range = prob->high[j]-prob->low[j];
        perturb = 0.1*range;
        sol[j] += perturb;

        if(sol[j]>prob->high[j]) sol[j]=prob->high[j];
        if(sol[j]<prob->low[j]) sol[j]=prob->low[j];

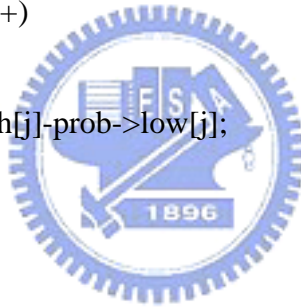
        for(i=1; i<=prob->n_var; i++)
            p[j+1][i]=sol[i];

        y[j+1] = evaluate(prob, sol);
        sol[j] -= perturb;
    }

    /* Call Nelder and Mead's Simplex method */
    a=amoeba(p, y, prob->n_var, 0.1, evaluate, &nfunk, prob);

    best_sol=0;

```



```

for(i=1;i<=prob->n_var+1;i++)
{
    if( *value>y[i])
    {
        *value=y[i];
        best_sol=i;
    }
}

if(best_sol>0)
{
    for(i=1;i<=prob->n_var;i++)
    {
        sol[i]=p[best_sol][i];

        if(sol[i]<prob->low[i])
            sol[i]=prob->low[i];
        if(sol[i]>prob->high[i])
            sol[i]=prob->high[i];
    }

    *value = evaluate(prob,sol);
}

free(y);
SSfree_double_matrix(p,prob->n_var+1);
}

```

5. SS_tools

```
#include "SS.h"
```

```
#define M 714025
```

```
#define IA 1366
```

```
#define IC 150889
```

```
float SSrandNum(SS *p)
```

```
{
```

```
    int j;
```

```
    if(p->seed_reset==1)
```

```
    {
```

```
        p->seed_reset=0;
```

```
        p->iff=0;
```

```
    }
```

```
    if (p->idum < 0 || p->iff == 0) {
```

```
        p->iff=1;
```

```
        if ((p->idum=(IC-(p->idum)) % M) < 0) p->idum = -(p->idum);
```

```
        for (j=1;j<=97;j++) {
```

```
            p->idum=(IA*(p->idum)+IC) % M;
```

```
            p->ir[j]=(p->idum);
```

```
        }
```

```
        p->idum=(IA*(p->idum)+IC) % M;
```

```
        p->iy=(p->idum);
```

```
    }
```

```
    j=(int)(1 + 97.0*(p->iy)/M);
```

```
    if (j > 97 || j < 1)
```

```
        SSabort("Failure in random number generator");
```

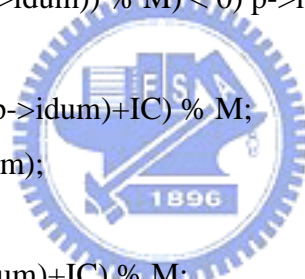
```
    p->iy=p->ir[j];
```

```
    p->idum=(IA*(p->idum)+IC) % M;
```

```
    p->ir[j]=(p->idum);
```

```
    return (float) (p->iy)/M;
```

```
}
```



```

#undef M
#undef IA
#undef IC

void try_add_RefSet1(SS *prob,double *sol)
{
    int i,j,worst_index;
    double value,worst_value;

    value=evaluate(prob,sol);
    if(prob->LS) SSimprove_solution(prob,sol,&value);

    worst_index=prob->order1[prob->b1];
    worst_value=prob->value1[worst_index];

    if(is_new(prob,prob->RefSet1,prob->b1,sol) && value<worst_value)
    {
        i=prob->b1;
        while(i>=1 && value<prob->value1[prob->order1[i]])
            i--;
        i++;

        /* Replace solution */
        for(j=1;j<=prob->n_var;j++)
            prob->RefSet1[worst_index][j]=sol[j];
        prob->value1[worst_index]=value;
        prob->iter1[worst_index]=prob->iter;

        /* Update Order */
        for(j=prob->b1;j>i;j--)
            prob->order1[j]=prob->order1[j-1];

        prob->order1[i]=worst_index;
        prob->new_elements=1;
    }
}

```




```

    }
}

void try_add_RefSet2(SS *prob,double *sol)
{
    int i,j,worst_index;
    double value,worst_value;

    /* It should be noted that solutions are not improved
    to increase the diversity in RefSet2 */

    value=distance_to_RefSet(prob,sol);

    worst_index=prob->order2[prob->b2];
    worst_value=prob->value2[worst_index];

    if(value>worst_value)
    {
        i=prob->b2;
        while(i>=1 && value>prob->value2[prob->order2[i]])
            i--;
        i++;

        /* Replace solution */
        for(j=1;j<=prob->n_var;j++)
            prob->RefSet2[prob->order2[prob->b2]][j]=sol[j];
        prob->value2[worst_index]=value;
        prob->iter2[worst_index]=prob->iter;

        /* Update Order */
        for(j=prob->b2;j>i;j--)
            prob->order2[j]=prob->order2[j-1];

        prob->order2[i]=worst_index;
    }
}

```



```

        prob->new_elements=1;
    }
}

void SScombine(SS *prob,double *x,double *y,double **offsprings,int number)
{
    int j;
    double a,*d,r;

    d=SSallocate_double_array(prob->n_var);
    if(!d) SSabort("Memory allocation problem");

    r = SSrandNum(prob);
    for(j=1;j<=prob->n_var;j++)
        d[j] = (y[j] - x[j]) / 2;

    /* Generate C2 */
    for(j=1;j<=prob->n_var;j++)
    {
        offsprings[1][j] = x[j] + r*d[j];
        if(offsprings[1][j]>prob->high[j]) offsprings[1][j]=prob->high[j];
        if(offsprings[1][j]<prob->low[j]) offsprings[1][j]=prob->low[j];
    }

    if(number>=2) /* Generate C1 or C3 */
    {
        a =SSrandNum(prob);
        for(j=1;j<=prob->n_var;j++)
        {
            if(a<=0.5) offsprings[2][j] = x[j] - r*d[j];
            else offsprings[2][j] = y[j] + r*d[j];
            if(offsprings[2][j]>prob->high[j]) offsprings[2][j]=prob->high[j];
            if(offsprings[2][j]<prob->low[j]) offsprings[2][j]=prob->low[j];
        }
    }
}

```



```

}

if(number>=3) /* Generate the other one (C1 or C3) */
{
    a =SSrandNum(prob);
    for(j=1;j<=prob->n_var;j++)
    {
        if(a>0.5) offsprings[3][j] = x[j] - r*d[j];
        else     offsprings[3][j] = y[j] + r*d[j];
        if(offsprings[3][j]>prob->high[j]) offsprings[3][j]=prob->high[j];
        if(offsprings[3][j]<prob->low[j])  offsprings[3][j]=prob->low[j];
    }
}

if(number==4) /* Generate another C2 */
{
    r = SSrandNum(prob);
    for(j=1;j<=prob->n_var;j++)
    {
        offsprings[4][j] = x[j] + r*d[j];
        if(offsprings[4][j]>prob->high[j]) offsprings[4][j]=prob->high[j];
        if(offsprings[4][j]<prob->low[j])  offsprings[4][j]=prob->low[j];
    }
}

free(d);
}

int is_new(SS *prob,double **solutions,int dim,double *sol)
{
    int i,j,is_new;
    double precision=0;

```



```

precision=1/pow(10,prob->digits);

for(i=1;i<=dim;i++)
{
    is_new=0;
    for(j=1;j<=prob->n_var;j++)
        if(fabs(solutions[i][j] - sol[j]) >= precision)
            is_new=1;
    if(is_new==0) return 0;
}
return 1;
}

```

```

double distance_to_RefSet1(SS *prob,double *sol)
{
    double d,min_dist=DBL_MAX;
    int a,j;

    for(a=1;a<=prob->b1;a++)
    {
        d=0;
        for(j=1;j<=prob->n_var;j++)
            d += pow(sol[j]-prob->RefSet1[a][j],2);
        if(min_dist> d)
            min_dist=d;
    }

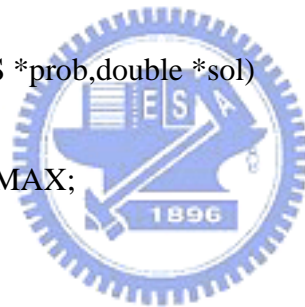
    return min_dist;
}

```

```

double distance_to_RefSet(SS *prob,double *sol)
{

```



```

double d,min_dist=DBL_MAX;
int a,j;

for(a=1;a<=prob->b1;a++)
{
    d=0;
    for(j=1;j<=prob->n_var;j++)
        d += pow(sol[j]-prob->RefSet1[a][j],2);
    if(min_dist> d)
        min_dist=d;
}

for(a=1;a<=prob->b2;a++)
{
    d=0;
    for(j=1;j<=prob->n_var;j++)
        d += pow(sol[j]-prob->RefSet2[a][j],2);
    if(min_dist> d)
        min_dist=d;
}

return min_dist;
}

int *orden_indices(double *pesos,int num,int tipo)
{
    int *indices,b,t,j,i,tempi;
    double temp,*coste;

    coste=SSallocate_double_array(num);
    indices=SSallocate_int_array(num);

    for(i=1;i<=num;i++)
    {

```



```

    coste[i]=pesos[i];
    indices[i]=i;
}

b=num;
while(b!=0)
{
    t=0;
    for(j=1;j<=b-1;j++)
    {
        if( (tipo==1  && coste[j]<coste[j+1]) ||
            (tipo==-1 && coste[j]>coste[j+1])    )
        {
            temp=coste[j+1];
            coste[j+1]=coste[j];
            coste[j]=temp;

            tempi=indices[j+1];
            indices[j+1]=indices[j];
            indices[j]=tempi;

            t=j;
        }
    }
    b=t;
}
free(coste);
return indices;
}

```

```

void SSabort(char *texto)
{
    printf("%s",texto);
    exit(6);
}

```

```
}
```

6. SS

```
#include "stdio.h"
```

```
#include "stdlib.h"
```

```
#include "malloc.h"
```

```
#include "math.h"
```

```
#include "string.h"
```

```
#include "time.h"
```

```
typedef struct SS {
```

```
    int n_var;
```

```
    double digits;
```

```
    double *high;
```

```
    double *low;
```

```
    int **ranges; /* Diversification Generator */
```

```
    int PSize;
```

```
    int LS;          /* =1 LocalSearch ON, 0 OFF */
```

```
    int iter;
```

```
    int b1;
```

```
    double **RefSet1; // Solutions
```

```
    double *value1; // Objective value
```

```
    int *order1; // Order of solutions
```

```
    int *iter1; // Number of iter of each sol.
```

```
    int b2;
```

```
    double **RefSet2;
```

```
    double *value2; // Dissim value
```

```
    int *order2;
```

```
    int *iter2;
```

```
    int last_combine; //Number of iter of last solution combination
```




```

int new_elements; //True if new elem. added since last combine

/* Random number parameters */
long idum;
int seed_reset;
int iff;
long ir[98];
long iy;

} SS;

#define SSgetrandom(p,min,max) (int)( (SSrandNum(p))*(max+1-min)) + (min))
#define DBL_MAX 1.7976931348623158e+308 /* max value */

/* File SS_main.c */
double evaluate(SS *prob,double *x);

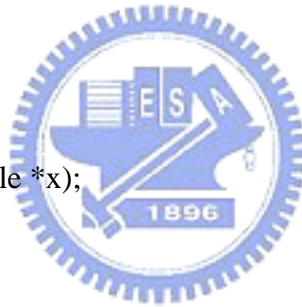
/* File SS_RefSet.c */

void Initiate_RefSet(SS *prob);
double SSGenerate_value(SS *prob,int a);
void SSimprove_solution(SS *prob, double *sol,double *value);
void Update_RefSet2(SS *prob);
void Combine_RefSet(SS *prob);

/* File SS_Memory.c */

SS *DataStructures_init(int nvar,int b1,int b2,int PSize,int LocalSearch);
void Free_DataStructures(SS *prob);
int **SSallocate_int_matrix(int rows,int columns);
double **SSallocate_double_matrix(int rows,int columns);
double *SSallocate_double_array(int size);

```



```

int *SSallocate_int_array(int size);
void SSfree_double_matrix(double **matrix,int rows);
void SSfree_int_matrix(int **matrix,int rows);

/* File SS_tools.c */

float SSrandNum(SS *p);
int *orden_indices(double *pesos,int num,int tipo);
void SSabort(char *texto);
double distance_to_RefSet(SS *prob,double *sol);
double distance_to_RefSet1(SS *prob,double *sol);
int is_new(SS *prob,double **solutions,int dim,double *sol);
void SScombine(SS *prob,double *x,double *y,double **offsprings,int number);
void try_add_RefSet1(SS *prob,double *sol);
void try_add_RefSet2(SS *prob,double *sol);

/* File nmsimplex.c */

int amoeba(double **p, double *y, int ndim, double ftol, double (*funkt)(), int
*nfunk,SS *prob);
double amotry(double **p, double *y, double *psum, int ndim, double (*funkt)(), int ihi, int
*nfunk, double fac,SS *prob);

```

