

東 海 大 學
工業工程與經營資訊研究所

碩士論文

運用遺傳演算法求解

可維修串並聯系統參數之研究

**A Genetic Algorithms Approach to Determining the
Parameters of Repairable Series and Parallel Systems**

研 究 生：劉子麟

指導老師：林水順 博士

蔡禎騰 博士

中華民國九十二年六月

A Genetic Algorithms Approach to Determining the Parameters of Repairable Series and Parallel Systems

By

Tzu-Lin Liu

Advisor : Dr. Shui-Shun Lin

Dr. Jen-Teng Tsai

A Thesis

Submitted to the Institute of Industrial Engineering and

Enterprise Information at Tunghai University

in Partial Fulfillment of the Requirements

for the Degree of Master of Science

in

Industrial Engineering and Enterprise Information

June 2003

Taichung, Taiwan, Republic of China

運用遺傳演算法求解 可維修串並聯系統參數之研究

學生：劉子麟

指導教授：林水順 博士

蔡禎騰 博士

東海大學工業工程與經營資訊研究所

摘 要

系統可用度關係著可靠度及系統建置成本，在工業系統設計領域中，是一個重要課題。隨著系統結構日趨複雜化，當系統中有一小零件發生失效時，導致系統功能無法發揮的弊害是相當嚴重的。但在提升系統可靠度的同時，成本往往也相對增加，因此，決定可用度參數，是一種複雜的尋優問題。

串並聯系統的複置配置問題(Redundancy Allocation Problem ; RAP)，大多仰賴系統設計人員的經驗。本研究的目的，在於可修復串並聯系統設計之初，即運用最佳化技術，輔助系統設計人員進行系統元件參數擬定之決策。具體而言，即在既定的系統架構下，決策出各組件的平均失效時間 (Mean Time Between Failure ; MTBF) 與平均修護時間 (Mean Time To Repair ; MTTR) 的最經濟策略。本研究提出一種兩階段的方法。第一階段，是先列出系統可用度估算式，並提出系統的總成本計算式，第二階段，以系統可用度與與系統總成本形成目標函數，並運用遺傳演算法求解模式。

在遺傳演算法平行搜尋與世代演化的特性下，可快速得到符合成本效益下之系統可用度最佳化參數，供系統設計人員輔助制定組件之選用策略與修復策略，而對於以往所難以傳承的經驗，也將可經由所

建立的設計知識，達到知識累積的目的。

關鍵詞：平均失效間隔時間、平均修護時間、可用度、遺傳演算法、最佳化

A Study of Using Genetic Algorithm to Determine the Parameters of Repairable Series Parallel System

Student : Tzu-Lin Liu

Advisor : Dr. Shui-Shun Lin

Dr. Jen-Teng Tsai

Department of Industrial Engineering and Enterprise Information

Tunghai University

Abstract

As the system structure becomes more and more complicated, failure of system function will always result in huge damage. It goes along with increasing cost while trying to improve system reliability or availability. Therefore, how to make an adequate decision to fit the real functional requirement is of importance.

The main purpose of this study is to conduct a decision support method for repairable system designers. This method attempts to substitute the traditional experience-based means. Two important parameters of system parts are to be determined in the phase of parameter design for a series and parallel system: Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR). This study proposes a two-step, genetic algorithms based, method to determine the parameters. The step one is to list the approximate expressions of the system availability and summarize the whole cost. The step two is to take availability/TC as the objective function and thereafter use Genetic Algorithms to solve the optimization problem.

The results show that using this method is capable of choosing a proper system parameters and producing reasonable repair policies.

Key Words: *MTBF; MTTR; Availability; Genetic Algorithms*

誌 謝

本論文能夠順利的完成，要感謝的人有很多。首先要先感謝指導老師林水順博士、蔡禎騰博士、彭泉博士、與姚銘忠博士，除了在專業知識以及研究方法的傳授之外，更以身教的方式導引我建立更開闊的人生觀，使我受益良多。感謝老師們對我論文的詳盡指導與寶貴的意見。另外要感謝李慶恩博士、邱文志博士與鄭順林博士於論文口試期間所給予的意見與指正，淑惠學姊與創鈞學長對我的關心及論文的指正，也都使本論文得以更加完善。

研究室諸位同學們與學弟妹們對我的鼓勵與支持也深深讓我感到懷念，謝謝政憲、穎威、志宏、崇民、勝翔、宣彥、茂洲，以及佩勳、曉玲、晏妃、榮華、修貝、玉鈴、岳庭、瓊瑩。在這些日子裡，你們帶給我許多美好的回憶。

最後，要特別感謝系上的助教素卿及俊良對我的種種協助與照顧，使我能夠順利的完成我的論文。更要感謝的是我的爸爸媽媽還有哥哥們，在我這幾年的求學生涯中，給予我最多的支持與鼓勵。在此謹將這份小小的成果與所有關心我的人一同分享。

劉子麟 謹誌於

東海大學工業工程與經營資訊研究所

中華民國九十二年六月

目 錄

頁次

| | |
|-----------------------------|-----------|
| 摘 要 | i |
| Abstract..... | iii |
| 誌 謝 | iv |
| 目 錄 | v |
| 表目錄 | vii |
| 圖目錄 | viii |
| 第一章 緒論 | 1 |
| 1.1 研究背景 | 1 |
| 1.2 研究動機與目的 | 3 |
| 1.3 研究方法與步驟 | 5 |
| 1.4 研究範圍與限制 | 8 |
| 1.5 研究架構 | 8 |
| 第二章 文獻探討 | 10 |
| 2.1 串並聯系統可靠度求解之相關研究 | 10 |
| 2.2 系統可用度或其他相關研究 | 14 |
| 第三章 模式建構 | 16 |
| 3.1 可修復系統的元件製造成本與修復成本 | 16 |
| 3.1.1 製造成本..... | 16 |
| 3.1.2 修復成本..... | 17 |
| 3.2 串並聯系統的系統可用度估算 | 18 |
| 3.3 遺傳演算法 | 25 |
| 3.3.1 遺傳演算法簡介 | 26 |
| 3.3.2 遺傳演算法運作方式 | 28 |
| 3.3.3 遺傳演算法參數設定 | 34 |
| 3.4 模式的建立 | 36 |

| | |
|----------------------------|-----------|
| 3.4.1 符號與假設..... | 36 |
| 3.4.2 問題形態..... | 37 |
| 3.4.3 求解步驟..... | 38 |
| 第四章 範例說明與結果分析 | 40 |
| 4.1 問題描述 | 40 |
| 4.2 問題分析 | 43 |
| 4.3 系統參數估計 | 45 |
| 第五章 結論與建議 | 52 |
| 5.1 結論 | 52 |
| 5.2 後續研究與建議 | 52 |
| 參考文獻 | 54 |
| 附錄一：演化的過程 | 61 |
| 附錄二：案例 GA 程式碼 | 64 |

表目錄

| | | |
|-------|-------------------------|----|
| 表 1-1 | 可靠性的尺度..... | 3 |
| 表 3-1 | 串並聯系統基本架構之可用度估算表..... | 19 |
| 表 3-2 | 文獻中 GA 求解參數設定..... | 35 |
| 表 4-1 | 各元件的 MTBF 範圍與製造成本..... | 41 |
| 表 4-2 | 各元件的 MTTR 範圍與修復成本..... | 42 |
| 表 4-3 | 各參數組電腦執行時間與最佳目標函數值..... | 50 |
| 表 4-4 | 電腦所得輔助決策之最佳參數..... | 51 |

圖目錄

| | | |
|--------|------------------------|----|
| 圖 1-1 | 研究步驟流程圖 | 7 |
| 圖 1-2 | 論文架構圖 | 9 |
| 圖 2-1 | 串並聯系統架構圖 | 10 |
| 圖 3-1 | MTBF 與成本間之關係 | 17 |
| 圖 3-2 | MTTR 與成本間之關係 | 18 |
| 圖 3-3 | 串並聯系統架構之例一 | 20 |
| 圖 3-4 | 架構簡化 1 | 20 |
| 圖 3-5 | 架構簡化 2 | 20 |
| 圖 3-6 | 架構簡化 3 | 21 |
| 圖 3-7 | 使用 Relex 軟體-RBD 圖 | 22 |
| 圖 3-8 | 使用 Relex 軟體計算系統可用度結果 1 | 23 |
| 圖 3-9 | 串並聯系統架構之例二 | 23 |
| 圖 3-10 | 使用 Relex 軟體-RBD 圖之二 | 24 |
| 圖 3-11 | 使用 Relex 軟體計算系統可用度結果 2 | 25 |
| 圖 3-12 | 遺傳演算法的三個運算子 | 26 |
| 圖 3-13 | 遺傳演算法組成圖 | 29 |
| 圖 3-14 | 遺傳演算法運作流程圖 | 30 |
| 圖 3-15 | 交配運算示意圖 | 32 |
| 圖 3-16 | 突變運算示意圖 | 33 |
| 圖 4-1 | 可修復系統的結構圖 | 40 |
| 圖 4-2 | 範例系統架構圖 | 45 |
| 圖 4-3 | 範例架構簡化 1 | 46 |
| 圖 4-4 | 範例架構簡化 2 | 46 |
| 圖 4-5 | GA 求解結果 | 50 |

第一章 緒論

1.1 研究背景

可靠度最泛用的定義，是 1952 年美國國防部所成立的電子裝備可靠度顧問小組（Advisory Group on the Reliability of Electronic Equipment；AGREE）所提出：可靠度為產品於既定的時間內，在特定的使用（環境）之下，執行特定性能或功能，成功達成工作目標的機率[54]。而關於 JIS(日本工業規格協會)所訂出的可靠性尺度則見表 1-1[56]。

在產品生產製造過程中，一定會牽扯到品管的問題，但可靠度問題卻容易為人所忽略。一般而言，高品質並不一定是高可靠度的同義字，例如，應用在產品中的某個元件，即使已通過了所有必須的品管程序，符合了所有的規格需求，但是，在產品中與其他組件一起使用時仍可能發生問題。其原因可能是，在零件之間發生電子或機械介面相容性的問題，這類就是可靠度設計的問題。

可靠度通常在產品被使用，並發生損害或問題浮現之前，不會被人們所重視，投資於可靠度上，也無法立即看到它的效果，且當可靠度的效果出來時，產品就更加不易故障，也更加不會為人所注意。但是隨著科技的快速發展，產品的系統結構日趨複雜化，這時萬一有一小零件發生失效時，導致的產品功能無法發揮，弊害有時是相當嚴重的。舉國防武器為例，每一套武器系統的結構，皆是由數個子系統、組裝件、組件和零件交織而成型，進而構成一錯綜複雜的軍事武器。在這樣複雜的系統之內，假使有任何一個組件，甚至是小到某一個單一零件發生失效或故障時，就可能立即影響到對武器系統的操控性

及性能的發揮，嚴重的話，更會危害到國家安全。

所謂的可用度係指衡量可修復系統可靠度的尺度。可修復系統，指的是當發生失效時可以經由修理而回復運作的系統，如電腦網路，製造系統，發電廠，或是防火系統等等都屬之。可用度(Availability)是由「可靠度」與「經由維修使不可靠恢復的部分」所構成，為「能修理的系統、機器或零件等在某特定的瞬間維持機能之機率」[10]，多以下式表示：

$$\text{可用度} = \frac{\text{(能動作時間)}}{\text{(能動作時間)} + \text{(不能動作時間)}}。$$

近年來，可靠度與可用度的相關應用已漸漸在各種行業及領域擴大了影響力，更是組織體系及產品製造上，不可或缺的一種品質。

表 1-1 可靠性的尺度

| | |
|--------------------|---|
| 可靠性的尺度 | 依據 JIS(Z 8115)的定義 |
| 可靠度(R) | 系統、機器、零件等在規定的條件下在期望的時間中，達成規定機能之機率 |
| 故障率() | 至某時點為止一直動作之系統、機器或零件等，在後續的單位時間內發生故障的比率 |
| 平均失效間隔時間 (MTBF) | 修理後可使用之系統、機器、零件等，相隔故障間之動作時間的平均值 |
| 至故障為止之平均動作時間(MTTF) | 不能修理之系統、機器、零件等，至故障為止之動作時間的平均值 |
| 平均修護時間 (MTTR) | 故障發生以後維護的不能動作時間之平均 |
| 可用度 | 能修理之系統、機器或零件等，在某特定之瞬間維持機能之機率，多以下式來求 $\text{可用度} = \frac{\text{(能動作時間)}}{\text{(能動作時間)} + \text{(不能動作時間)}}$ |

1.2 研究動機與目的

要讓結構複雜的系統，可靠度維持在一定的水準之上時，最通常的做法是對系統的結構設計進行改良，或是改採用較高可靠度的系統元件，也可同時針對這兩方面著手進行[21]。以飛機的引擎系統為例，飛機絕不能因為引擎的失效或損壞而立即墜落，飛行的可靠性與安全性是相當重要的，這時它採用的是複置的方式，只要一般飛機所具備的四具引擎，其中任何兩具引擎功能正常，飛機即可正常飛行[54]。

實際的系統結構設計時，往往會面臨重量，體積或一些技術上的

限制，使得可靠度無法再向上突破，這時，就必須更換較高可靠度的組件，才可能提高系統可靠度。而較高可靠度的組件，卻也因為製作較為精良而導致成本會較高，所以，在對系統可靠度提昇的同時，伴隨著是成本也會相對增加。這時，如何在兩者衝突之下，根據實務上的考量，作出符合需求的決策，是最重要的議題。

串並聯系統的複置配置問題(Redundancy Allocation Problem ; RAP)被指出是困難的 NP-hard 問題[4]，以往仰賴的都是系統設計人員的經驗[5]。而經驗法則有其優缺點，優點是：(1)仰賴多年來的經驗和資歷，依主觀意識來進行決策，既省時又方便。(2)在資訊不充分的情況下，憑藉經驗所作出的決策，仍有某種程度上的參考價值。缺點則為：(1)此乃依個人的主觀意識所作出的決策，其決策的背後並無科學的理論或證據來作為支撐，不易讓人信服。(2)依個人經驗所作出的決策值為一粗略值，無法提供一精確之數值，可能會造成多餘的成本支出和浪費。採用經驗法則有相當大的風險存在，實不宜貿然採用[55]。經驗的累積與傳承不易，如果能有一系統化的方法，善用電腦，將它轉換為知識進行累積與管理，將可提供相當大的幫助。Jeang[25][24]指出可使用電腦模擬軟體來輔助做系統之設計或產品的參數決策。

可修復的串並聯系統架構下，有各種的方法可用來決策出組件的最佳參數值，方法有動態規劃、整數規劃，非線性整數規劃，以及一些啟發式(Heuristic)或超啟發式(Metaheuristic)解法。遺傳演算法(Genetic Algorithms ; GA)是屬於超啟發式解法的一員，它已經被證明對於各種困難的問題，雖不能保證得到最佳解，但卻能以很快的速度逼近最佳解。本研究的目的，即是希望在可修護系統的設計階段，在既定的系統架構下，以遺傳演算法幫助決定各組件的平均失

效間隔時間 (Mean Time Between Failure ; MTBF) 及平均修復時間 (Mean Time To Repair ; MTTR)。

本研究提出的方法，是在列出可修復串並聯系統的可用度估算式後，將系統可用度估算式，亦即可靠度除以系統成本做為欲求解的目標函數，再運用遺傳演算法對之進行目標函數最大化的運算。期在設定的可接受時間範圍內，求得符合成本效益下系統可用度最佳化的參數，該參數可以幫助系統設計人員用來制定組件的最佳選用策略與修復策略，取代以往單靠“經驗法則”決策的方式。

總而言之，本研究的目的有以下二點：

1. 發展串並聯系統可用度優化模式，並分析模式行為。
2. 列出可修復串並聯系統可用度估算式後，運用遺傳演算法 (Genetic Algorithms) 求解最符合經濟效益之系統組件最佳化參數。讓系統設計人員在作參數決策時，能有一科學的依據。

1.3 研究方法與步驟

本研究欲提出一方法，輔助系統設計人員決策系統元件參數，使用的方法為遺傳演算法。整個研究的步驟如下：

1. 收集問題的相關文獻，如串並聯系統可靠度、可靠度最佳化之求解方法、系統可用度等相關主題。
2. 第一階段先列出系統可用度估算式並計算系統的整體成本。
3. 第二階段訂定出本研究目標函數，並以遺傳演算法 (Genetic Algorithms) 進行求解。
4. 以一範例說明演算過程：使用 Microsoft Visual C++ 進行 GA

程式之撰寫，在 Pentium 4 賽揚 1.7G 執行並記錄結果。

本研究的步驟流程如圖 1-1 所示。

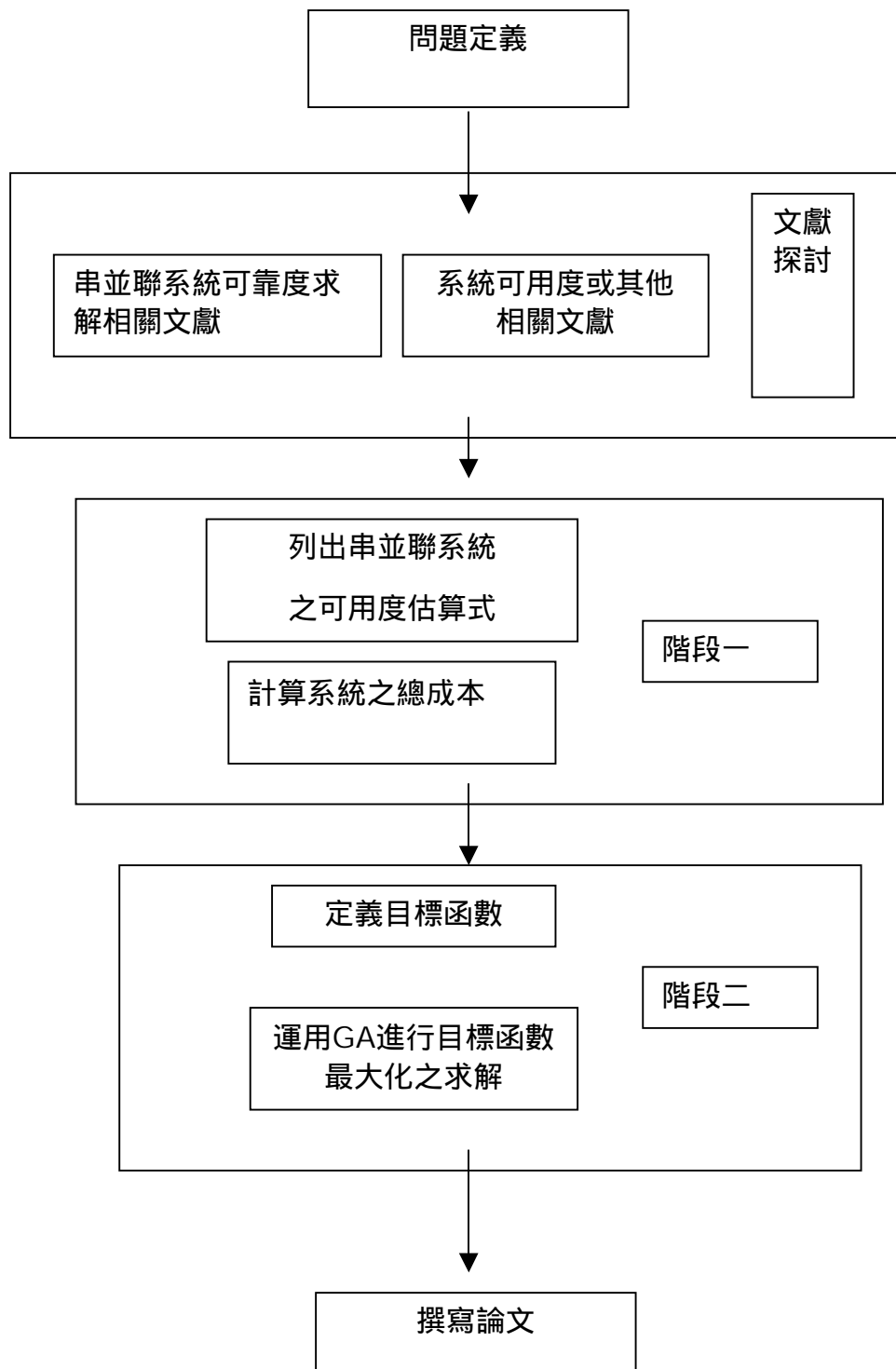


圖 1-1 研究步驟流程圖

1.4 研究範圍與限制

本研究所探討的問題範圍為串並聯系統，並假設系統組件的失效率與修復率 μ 皆為常數，故障時間與修理時間服從指數分配。由於要對系統可用度列出估算式，故須滿足 $MTBF \gg MTTR$ ，即 $\mu \gg \lambda$ ，並假設元件間的維修互相獨立。元件 MTBF 或 MTTR 與其成本間的函數關係可以明確定義。

1.5 研究架構

本研究之架構依章節排列，如圖 1-2 所示，內容共分為五章。第一章敘述本研究之背景與動機、目的、範圍與限制、研究方法與步驟。第二章對串並聯可靠度、可用度求解方法等相關文獻進行探討。第三章則對本研究之可用度估算式之擬定、遺傳演算法等議題做一說明，並列出模式。第四章進行案例說明，明確列出方法後，用一範例實際推演整個研究的求解過程，並對結果進行分析探討。第五章做一整體歸納，做出結論與建議。

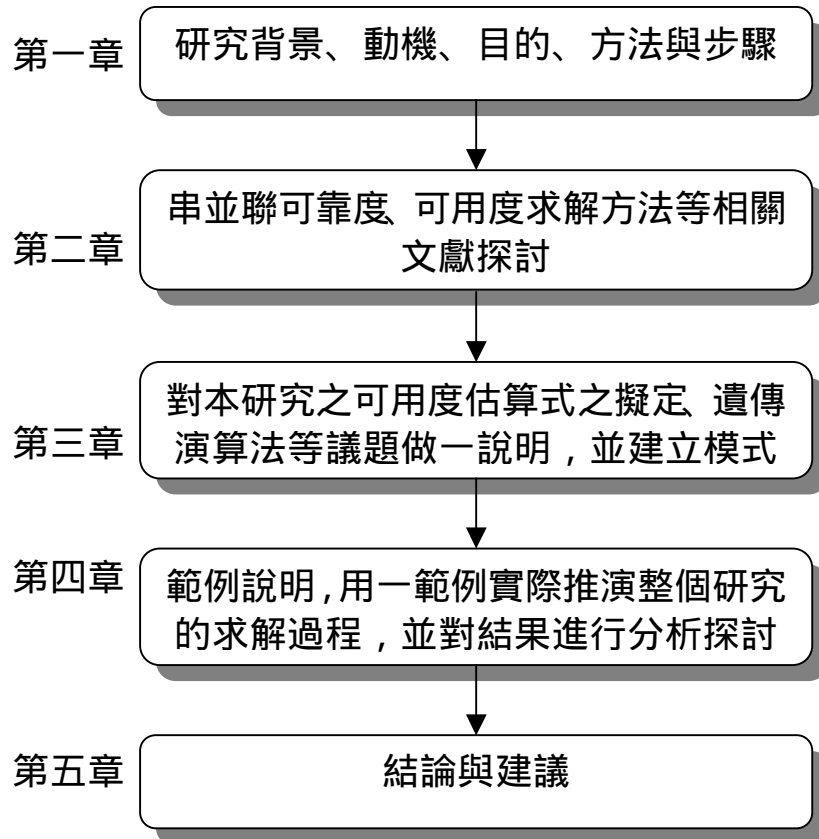


圖 1-2 論文架構圖

第二章 文獻探討

2.1 串並聯系統可靠度求解之相關研究

所謂串並聯系統，是指由數個元件並聯成幾個子系統，再將這些子系統進行串聯，或是數個元件串聯成幾個子系統，再將這些子系統進行並聯所形成的系統，架構圖如圖 2-1。

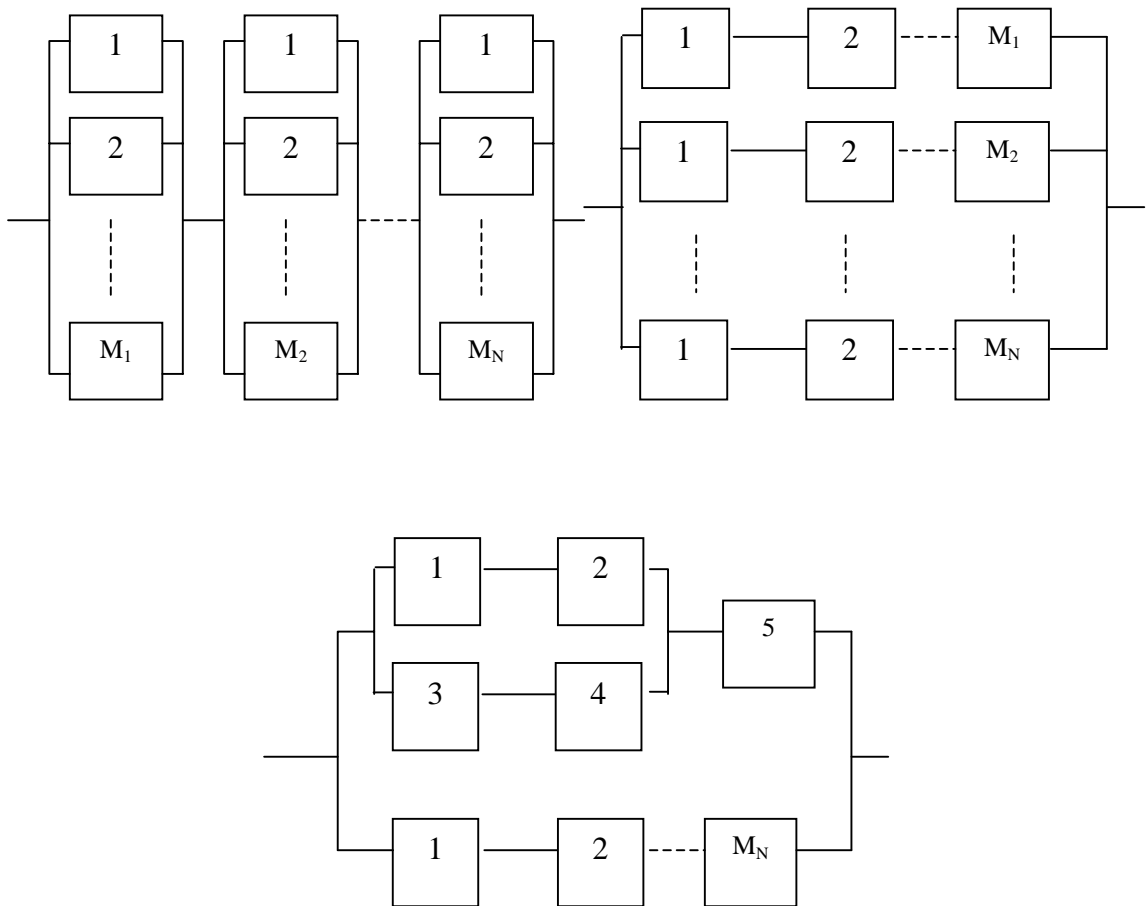


圖 2-1 串並聯系統架構圖

在串並聯系統的設計上，對系統可靠度的提昇，可以有四種方式：1.使用更可靠的元件；2.增加並聯狀態的複置元件；3.同時採行#1 和#2；4.對整個系統架構重新配置[31]。而串並聯系統的架構，在多重限制條件之下，要找到一個最佳解，往往是相當困難的[4]，

也因此發展出一些方法用來解這些問題。

(1)動態規劃、整數規劃之類的方法

一些的研究,是將具有多重限制式的串並聯系統可靠度最佳化問題,以動態規劃進行求解,或是視為非線性整數規劃問題求解。

Bellman 和 Dreyfus[1]使用動態規劃求解系統單一成本限制式下,決定每個子系統的最佳複置個數的可靠度最大化問題。Fyfee 等人[12]使用動態規劃的方法解決更為困難的設計問題,考慮的系統具有 14 個子系統,限制條件有成本和重量,每個子系統有三到四個元件(可靠度、成本與重量皆不同)可供選擇,為了考慮到多重的限制式,他們在目標函數中使用了 Lagrange 乘數,並以人為限制解的搜尋空間。

接著有一段時間,使用 Lagrange 乘數的動態規劃(DP-LM) 被認為是用來求解某些兩個限制式的可靠度問題之最佳方法,但 Nakagawa 和 Miyazaki[38]指出使用 DP-LM 常常是沒什效率的,並提出 N&M 演算法。N&M 演算法將兩個限制式組合成為一個替代的限制式來看,然後再以動態規劃進行求解,提供了比 DP-LM 更為優越的效率。

對於某些非線性離散限制式的可靠度最佳化問題,可將之轉換成整數規劃問題處理,但它必須以人為去限制搜尋空間,並且不能在子系統中混合不同的元件。Bulfin 和 Liu[3]是將可靠度最佳化問題視為背包問題,並使用學者 Fisher[11]提出的方法,成功求解了 Fyfee 等人 [12] 及 Nakagawa[38]的串並聯系統複置元件問題。

將問題視為整數規劃進行求解的還有, Misra 和 Sharma[35]提出

Misra 演算法，它是一個以搜尋可行解區域界限附近的演算法，該方法使求解變得簡單且有效率，可以被用來求解許多可靠度最佳化的問題。Prasad 和 Kuo[42]卻用例子指出 Misra 演算法有時並不能產生最佳解，並提出一個搜尋可靠度目標函數上界的方法。Gen 等人[13][14]的研究，也是使用整數規劃方法進行求解。

當問題還不大的情況下，這類方法相當有用的，但隨著問題變得越來越複雜以後，這類方法將由於進行大量計算，需耗用巨額的記憶體，故問題較大時，研究者多轉而發展利用啟發式方法來求解。

(2)啟發式演算法求解

Gopal 等人[19]、Kuo 等人[30]、Nakagawa 與 Nakashima [39]、Sharma 與 Venkateswaran [44]所做的研究，都是在非線性條件限制之下，求解系統配置問題的啟發式解法。Nakagawa 與 Miyazaki[40]使用了廣泛數字例子，對上述這些方法的演算時間與平均錯誤率，做出了列表式的比較。

Kuo 等人[32]則合併 Lagrange 乘數與 branch-and-bound 兩種技術後，提出了一種啟發式解法 LMBB。它是先以 Lagrange 乘數快速求得接近最佳解的實數解，而後再用 branch-and-bound 求取整數最佳解

李[53]同時考慮系統可靠度最大及總成本最小，並將之視為多目標規劃問題與模糊多目標規劃問題分別進行求解，之後對這兩種方式進行比較。

Jianping[26]發展出一種稱為界限啟發式解法的方法。學者 Dinghua [9]、Kohda 與 Inoue [29]、Kim 與 Yum[28]做的也都是以啟發式解法求解串並聯系統可靠度問題。

(3)其他方法

其他還有，像是 Li 和 Haimes[33]對線性資源限制的大型系統，提出一個三層式的分解方法求解系統可靠度最佳化。Mohan 和 Shanker[37]則對複雜系統，根據成本上的限制，採隨機搜尋法來選擇系統組成元件。

而最佳化方法隨著時間的發展，已有許多被運用來求解可靠度問題。最近的幾年，遺傳演算法(Genetic Algorithms ; GA)亦被成功用來求解串並聯系統可靠度最佳化問題。GA 嘗試模仿生物演化的過程，它可以有效率的求解複雜的組合問題。

Painton 和 Campbell[41]解決了個人電腦設計相關的可靠度最佳化問題。他們將整台個人電腦視為一個 12 個元件的串並聯系統，其中每個元件都有 3 種選擇方案，故共有 3^{12} 種組合，該研究使用了 GA 求算在預算限制之下的最佳解。

Ida 等人[50]運用 GA，成功求解了系統並聯元件具有數種失效模式的串並聯系統可靠度最佳化問題，這類問題之前被用列舉式方法求解過[45]。

Coit 與 Smith[8][7]使用 GA 求解滿足成本與重量限制式的串並聯系統可靠度最佳化問題。其求解結果證明使用 GA 所得到的解，比 Bulfin 和 Liu 提出的方法[3]，及 Nakagawa 和 Miyazaki 的 N&M 演算法[38]好，所花費的時間也更短。

Yokota 等人[51]將 GA 用來求解非線性混合整數規劃問題，並舉可靠度應用為例。其他像 Yang 等人[49]及 Gen 和 Cheng[15]也都是運用 GA 求解可靠度問題。

2.2 系統可用度或其他相關研究

學者 Wang[48]對可用度的點估計值，提出方法估算其區間的上下界（Interval Estimation），他提出的方法有二，第一種是當 MTBF 與 MTTR 的分配皆服從指數分配時可用，第二種方法是當兩者不服從指數分配時，提出估算可用度區間的方式。最後，兩種方法以蒙第卡羅模擬法進行結果驗證比較。學者 Gordan[20]使用統計及圖解的技術來檢定資料，配合 SAS 統計軟體進行可靠度與可用度之資料分析。

在系統日益複雜及多樣化的情境下，由於一般的估算法有較為嚴苛之假設條件，故有些學者偏向於模擬方法與模擬工具的使用，來評估複雜系統之可靠度或可用度。學者 Lieber 等人[34]對於評估水力發電系統的可靠度，提出運用重點抽樣（Importance Sampling）的技術，可增加傳統 Monte Carlo Simulation 的效率。學者 M S Chodos[6]對電話通訊網路系統之可用度進行討論，文中指出大型網路複雜，可使用不同的模擬工具（使用 Pascal, 與 C，或是用模擬語言 GPSS 進行程式之撰寫，或直接使用特定領域的套裝軟體如 BONEs 等），進行系統可用度最佳化之模擬。學者 Chisman[5]對非串並聯的液壓系統，提出失效模式法（failure-mode method）與流程法（flow method）兩種方式，使用 GPSS 的系統模擬軟體來模擬出兩者系統的 TBF 與 TTR 的分配，並得出結論建議採用流程法於大型系統的分析上。學者 Jeang[25][24]使用電腦模擬軟體 VSA-3D/Pro 來輔助做系統之設計，再以統計迴歸的方式求最佳解，做為產品的參數決策之依據。學者 Mitchell 和 Murry[36]使用可靠度區塊（RBD）之方法來模擬系統的結構，預測簡單並聯系統的系統可用度，文中指出使用備用元件可提升系統可用度。

其他一些研究同時考量了成本因素,像是學者 Propst 與 Doan[43] 在考量投資成本下,提供了一些系統架構及評估的試算表,對電力系統的可靠度與可用度進行評估,以用來改善系統設計。而在追求可靠度最大與成本最小的兩者衝突之下,學者 Huang[23]利用模糊(Fuzzy)多目標最佳化的方法,將決策中較為含糊的資訊,運用模糊歸屬函數,求得符合成本與可靠度限制下的最佳參數值。學者郭[55]以 BBD(Box-Behnken Design)的實驗矩陣,將不同水準下之實驗組合,以 Monte Carlo Simulation 進行系統之模擬實驗,以單位成本下之系統可用度作為反應值,運用反應曲面法(Response Surface Methodology ; RSM)建構的迴歸模式,決策出可修復串並聯系統的最佳參數值。

綜觀這些文獻,發現系統可用度的求算有估算式法與蒙第卡羅模擬法,估算式法的優點是求算快速但限制較多,模擬法則是較為耗時。遺傳演算法目前多運用在系統可靠度相關問題上之求解,在系統可用度方面的運用並不多。而對於串並聯系統可用度參數決策方面,要在參數的限制範圍內,求得一最適參數並不容易,此時,遺傳演算法世代演化與平行搜尋的特性,可以在有限時間內逼近最佳解,因此,本研究即欲提出一方法,以可用度估算式法搭配 GA 進行求解,用以輔助系統設計人員在可修復串並聯系統元件參數之擬定。

第三章 模式建構

本章針對本研究所用的方法及相關基礎理論進行說明，並建立整個方法的模式。第一小節對系統元件參數 MTBF 與 MTTR 進行說明，第二小節介紹可修復系統可用度的點估計估算方法，第三小節介紹遺傳演算法運作的方式，並於最後一小節建立本研究求解問題的方法模式。

3.1 可修復系統的元件製造成本與修復成本

本小節對可修復系統元件的兩個參數值 MTBF 與 MTTR 與其成本間的關係做一介紹，並述及本研究對它的假設。

3.1.1 製造成本

各類產品的形式及規格不盡相同，製造的成本也會有所差異。以電子組件來說，當製造的組件使用壽命（MTBF）越長，代表該組件的失效率（ λ ）越低且不易損壞，即意味著組件具有高品質與高可靠度之特點。在製造上，當組件的失效率低到某一程度以後，若想再突破其瓶頸是相當不容易的，會因為產品本身會受到某些條件上的限制，如受限於材質、技術或設備等，這個時候，就算投入再多的研發成本，可靠度上升的幅度依舊相當有限。原則上，越精良的組件越難製作，製造成本的開銷上也會相對高出許多，且失效率（ λ ）越小，越會導致製造成本的急遽增加[53]。因此，本研究對各元件之 MTBF 與其製造成本間的關係假設為[47]

$$C(MTBF) = \alpha * (MTBF)^\beta + \gamma ,$$

其中 $C(MTBF)$ ：代表該 MTBF 的製造成本

α, β, γ : 為常數，代表該元件的物理特性，且 $\gamma > 1$ ，

關係圖如圖 3-1

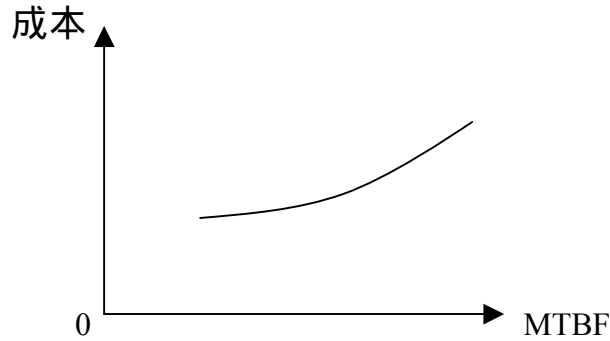


圖 3-1 MTBF 與成本間之關係

3.1.2 修復成本

就可修復系統而言，當系統結構中的某個組件發生失效時，有可能只會造成部份功能喪失而已，但嚴重的話卻會危害到整個系統的功效。此時為避免後者的情況發生，就必須對系統內已故障的組件進行修復之工作。且當系統發生失效時，總是希望能在一定時間內盡快修復完成，在受限於時間緊迫之下，為了提早在完成修復工作，會考慮以趕工的方式，要求修復單位採用資深的員工，採行加班制或使用高科技設備來進行修理。在此條件之下，雖然組件的修復時間能縮短，但也因投入大量的人力與設備成本，而使得修復成本變高。本研究各元件之 MTTR 與其修復成本間的關係假設為線性，且由於平均修復時間 (MTTR) 越小，修復成本越高，關係圖如圖 3-2。

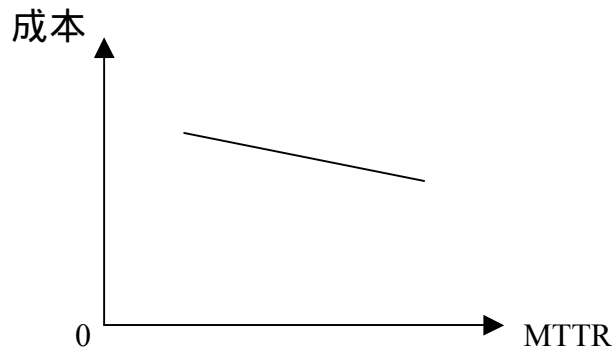


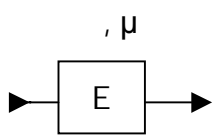
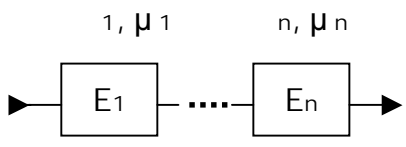
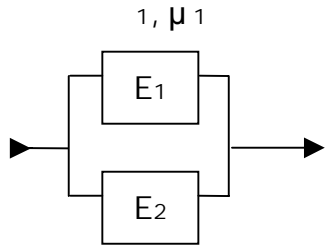
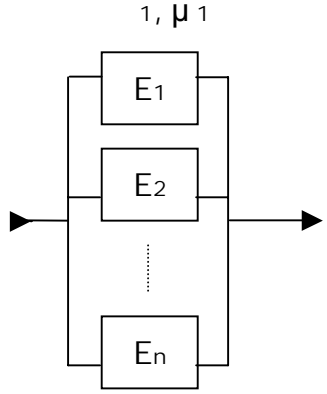
圖 3-2 MTTR 與成本間之關係

3.2 串並聯系統的系統可用度估算

對於可靠度或可用度的計算，在串並聯系統架構下，使用模擬法是相當耗時的，也因此，當在時間成本的考量下，若能使用逼近式來估算也就有其用途。在假設串並聯中的每個元件的失效率 $\lambda_i (=1/MTBF_i)$ 與修復率 $\mu_i (=1/MTTR_i)$ 皆為常數，故障時間與修理時間服從指數分配，且 $\lambda_i \ll \mu_i$ 。則其系統可靠度或可用度可由下列方法列出估算式[2]：

當在可靠度區塊(RBD)中的每個元件，元件和元件間的運作都是彼此獨立由一組維修團隊負責，則串並聯的架構可以被合併視為單一物件的架構。表 3-1 整理了一些基本串並聯架構的參數估算，依據此表進行依序簡化，一直向上推至整個系統的層級，即可列出串並聯系統的可用度估算式。

表 3-1 串並聯系統基本架構之可用度估算表

| | |
|--|--|
|  <p style="text-align: center;">λ, μ</p> | $\lambda_S = \lambda \quad \mu_S = \mu$ $PA_S = \frac{1}{1 + \lambda_S / \mu_S} \approx 1 - \frac{\lambda_S}{\mu_S}$ |
|  <p style="text-align: center;">$1, \mu_1 \quad n, \mu_n$</p> | $PA_S = PA_1 \Lambda PA_n \approx 1 - \left(\frac{\lambda_1}{\mu_1} + \Lambda + \frac{\lambda_n}{\mu_n} \right)$ $\lambda_S = \lambda_1 + \Lambda + \lambda_n$ $\mu_S \approx \frac{\lambda_1 + \Lambda + \lambda_n}{\lambda_1 / \mu_1 + \Lambda + \lambda_n / \mu_n}$ |
|  <p style="text-align: center;">$1, \mu_1$ $2, \mu_2$</p> | $PA_S = PA_1 + PA_2 - PA_1 PA_2 \approx 1 - \frac{\lambda_1 \lambda_2}{\mu_1 \mu_2}$ $\lambda_S \approx \frac{\lambda_1 \lambda_2 (\mu_1 + \mu_2)}{\mu_1 \mu_2}$ $\mu_S \approx \mu_1 + \mu_2$ |
|  <p style="text-align: center;">$1, \mu_1$ n, μ_n</p> | $PA_S \approx 1 - \frac{\lambda_1 \lambda_2 \Lambda \lambda_n}{\mu_1 \mu_2 \Lambda \mu_n}$ $\lambda_S \approx \frac{\lambda_1 \lambda_2 \Lambda \lambda_n (\mu_1 + \mu_2 + \Lambda + \mu_n)}{\mu_1 \mu_2 \Lambda \mu_n}$ $\mu_S \approx \mu_1 + \mu_2 + \Lambda + \mu_n$ |

* 本表假設元件跟元件間彼此獨立，失效率為常數 λ_i ，維修率為常數 μ_i ， $\lambda_i \ll \mu_i$ ，且每個元件都有一組維修團隊負責

茲舉一例，若系統架構如圖 3-3

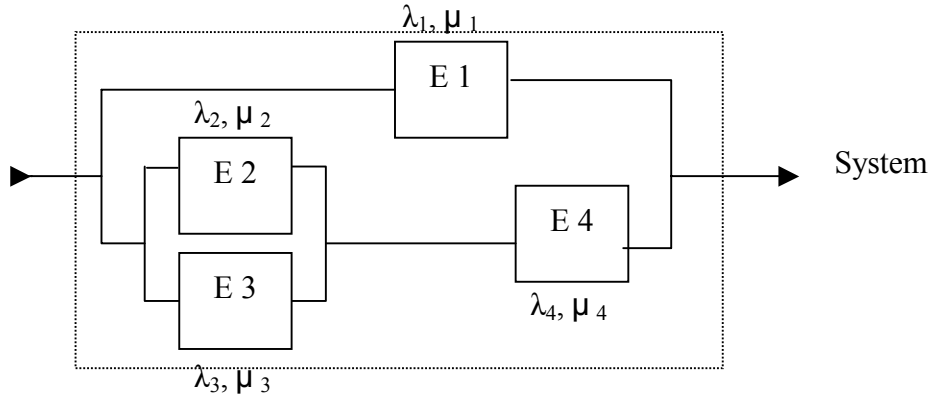


圖 3-3 串並聯系統架構之例一

則系統可用度的估算式求法，是使用表 3-1，一一簡化系統架構，步驟如下

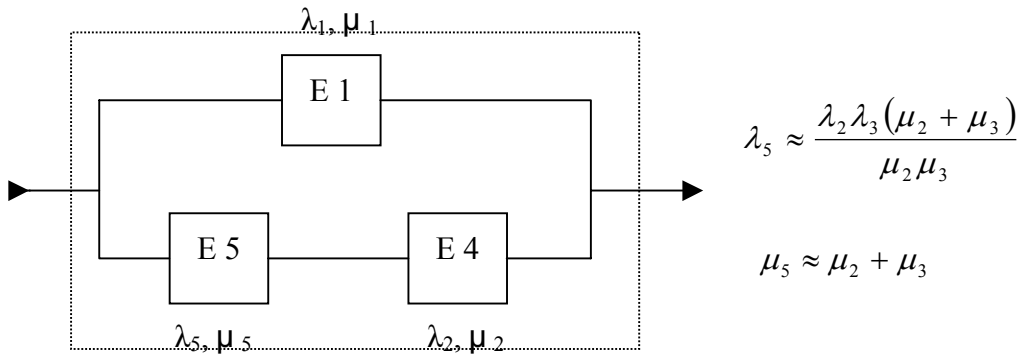


圖 3-4 架構簡化 1

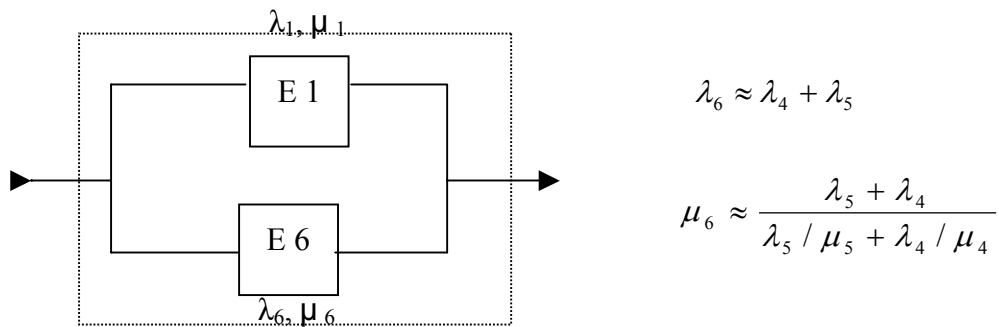


圖 3-5 架構簡化 2

代入求算得到系統可用度為

$$PA_s = 1 - \frac{80}{2100} \left[\frac{50}{2300} \cdot \frac{65}{1800} + \frac{130}{3600} \right] = 0.998594432$$

執行可靠度模擬軟體 Relx，拉出系統可靠度區塊圖(RBD)，依序輸入各 Entity 的 MTBF 值與 MTTR 值，如圖 3-7。

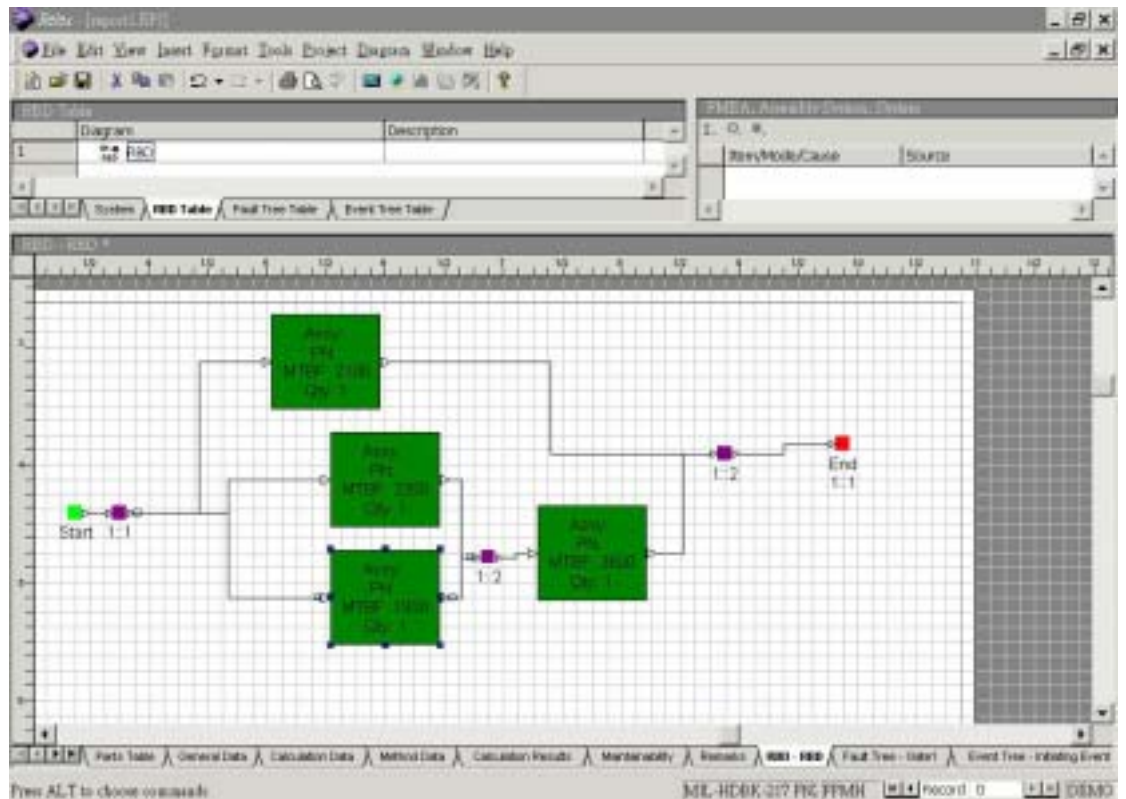


圖 3-7 使用 Relx 軟體-RBD 圖

得到結果如圖 3-8。

RBD Calculation Results

| | | |
|-------------------------------------|--------------|--|
| MTBF (simulated): | 39332.681478 | Results at time 1000.00: |
| MTTF: | 2883.256643 | Reliability: R 0.864985 |
| Steady-State Availability: A_{ss} | 0.998695 | Availability: A 0.998695 |
| Operational Availability: A_o | 0.998695 | Unreliability: F 0.135015 |
| Cost of Configuration: C_{rb} | NA | Unavailability: Q 0.001305 |
| Cost of Spares: C_{sp} | NA | Failure Rate: λ_{sys} 277.260950 |
| Achieved Availability: A_a | NA | |
| Confidence Level (Reliability): | NA | |
| Confidence Level (Availability): | NA | |
| Calculation Method: | Analytical | |

| Time | Availability | Reliability | Failure Rate | Unavailability |
|---------|--------------|-------------|--------------|----------------|
| 0 | 1.000000 | 1.000000 | 0.000000 | 0.000000 |
| 100.00 | 0.999476 | 0.998622 | 26.056388 | 0.000524 |
| 200.00 | 0.999034 | 0.994338 | 58.133119 | 0.000966 |
| 300.00 | 0.998836 | 0.987057 | 88.887798 | 0.001164 |
| 400.00 | 0.998754 | 0.976825 | 119.421133 | 0.001246 |
| 500.00 | 0.998720 | 0.963767 | 149.147523 | 0.001280 |
| 600.00 | 0.998706 | 0.948154 | 177.700745 | 0.001294 |
| 700.00 | 0.998700 | 0.930179 | 204.866386 | 0.001300 |
| 800.00 | 0.998697 | 0.910136 | 230.533975 | 0.001303 |
| 900.00 | 0.998696 | 0.888310 | 254.663317 | 0.001304 |
| 1000.00 | 0.998695 | 0.864985 | 277.260950 | 0.001305 |

OK

圖 3-8 使用 Relex 軟體計算系統可用度結果 1

兩者間的誤差率為

$$(0.998594432-0.998695)/0.998695= - 0.0100699 \%$$

架構二：若系統架構如圖 3-9。

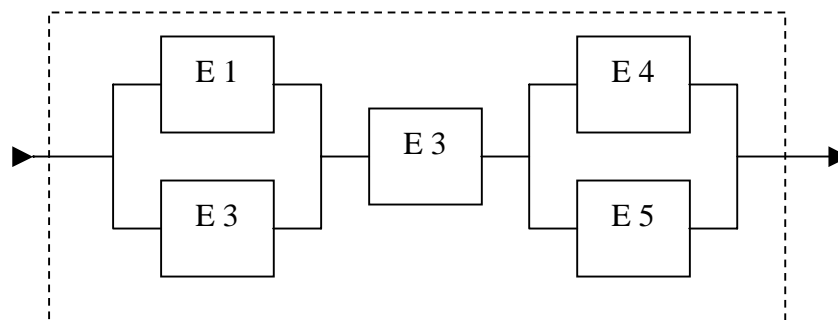


圖 3-9 串並聯系統架構之例二

且各系統元件的元件參數分別為

$MTBF_1=2100$, $MTTR_1=20$, $MTBF_2=2300$, $MTTR_2=16$

$MTBF_3=1800$, $MTTR_3=27$, $MTBF_4=1900$, $MTTR_4=35$

$MTBF_5=3600$, $MTTR_5=28$

則根據本節的方法可列出系統可用度估算式為

$$PA_S = 1 - \frac{MTTR_1}{MTBF_1} \cdot \frac{MTTR_2}{MTBF_2} - \frac{MTTR_3}{MTBF_3} + \frac{MTTR_4}{MTBF_4} \cdot \frac{MTTR_5}{MTBF_5}$$

代入求算得到系統可用度為

$$PA_S = 1 - \frac{20}{2100} \cdot \frac{16}{2300} - \frac{27}{1800} - \frac{35}{1900} \cdot \frac{28}{3600} = 0.984790472$$

執行可靠度模擬軟體 Relex , 拉出系統可靠度區塊圖(RBD) , 依序輸入各 Entity 的 MTBF 值與 MTTR 值 , 如圖 3-10。

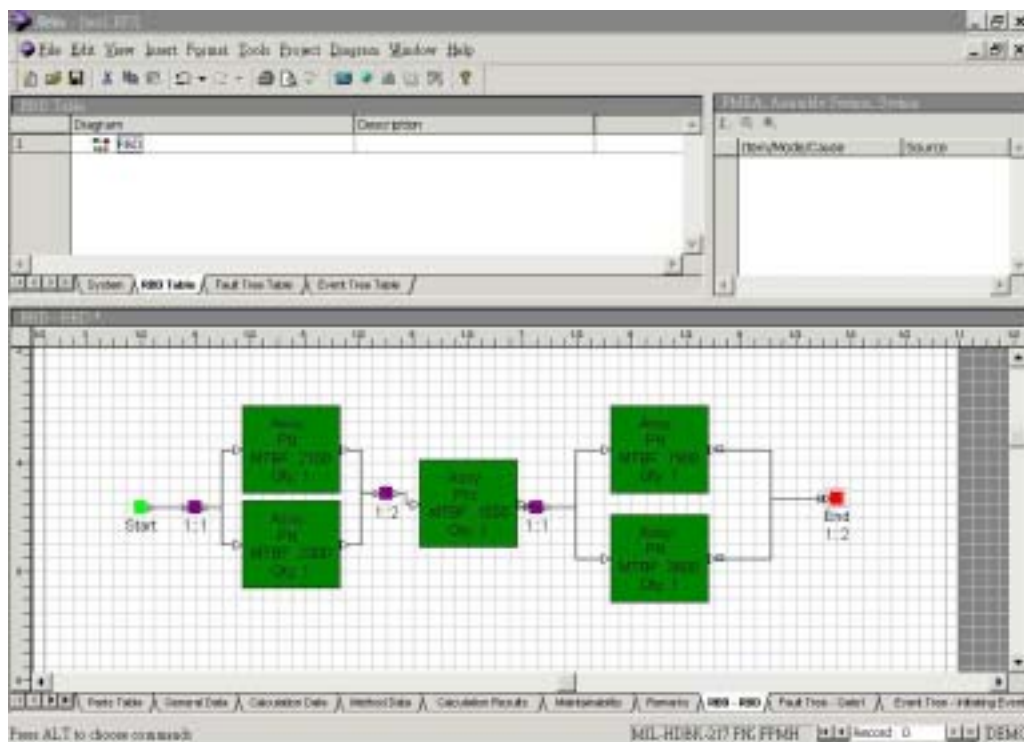


圖 3-10 使用 Relex 軟體-RBD 圖之二

得到結果如圖 3-11。

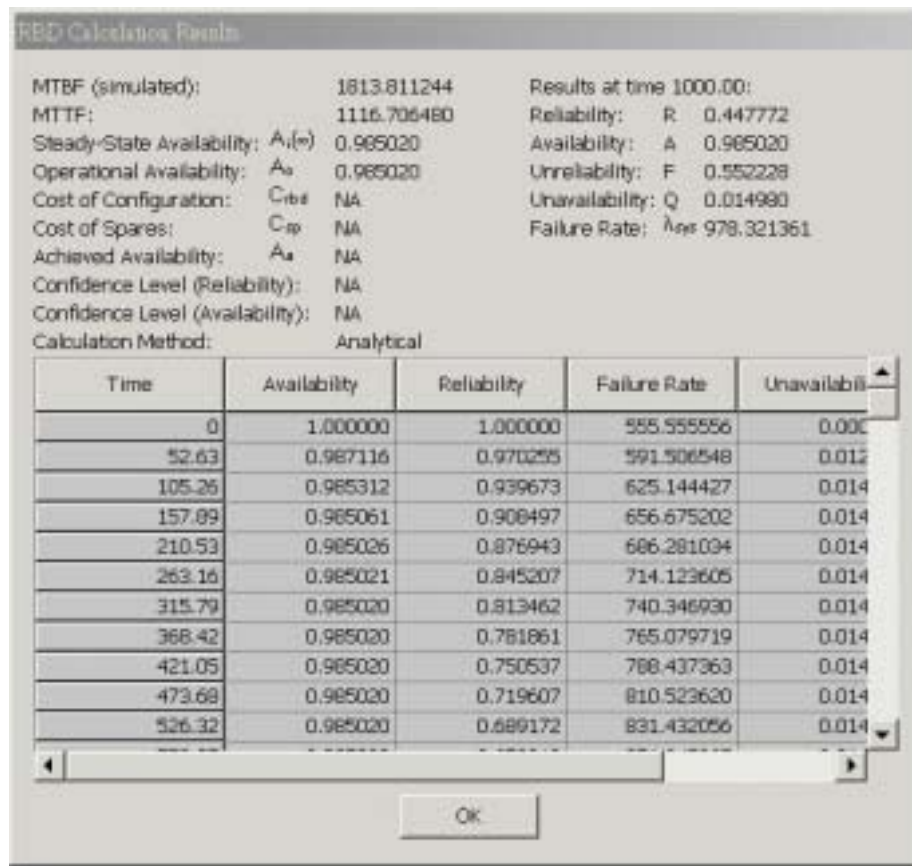


圖 3-11 使用 Relex 軟體計算系統可用度結果 2

兩者間的誤差率為

$$(0.984790472 - 0.985020) / 0.985020 = -0.02330186\%$$

由此可看出運用本小節所列的估算法所產生的誤差與模擬所得差異不大。

3.3 遺傳演算法

遺傳演算法(GA)是求解最佳化問題的一種機率搜尋法，它藉由模仿生物體一代一代演化的現象，透過複製、交配、突變這三個機制來產生優於上一代的子代。它是從 1970 年代開始發展，到了 80 年代以後，使得越來越普及，應用也越來越多。GA 適合可行解空間廣大

且具有非線性或不連續特性的問題，也因此，它可以有效率的求解複雜的串並聯可靠度最佳化問題。

3.3.1 遺傳演算法簡介

達爾文（Darwin）經過多年的研究觀察，於 1859 年出書「物種原始」，倡言演化思想，提出「物競天擇」理論，即自然界的演化規則乃是以「適者生存，不適者淘汰」做為基本原則。適合目前生活環境的物種擁有較大的機會生存下來，而不適合此一環境的物種則會被自然所淘汰，這就是「天擇」。經過天擇後的生物會繼續繁衍，使得子代遺傳母代更適應環境的特徵，以期在環境中得以生存。如此代代演化下去，整個物種就朝著更適應生存環境的方向走下去。

「適者生存，不適者淘汰」，此觀念最早是由達爾文所提出的，而把此種觀念運用到人工智慧系統的重要開創者則是 John J. Holland。1960 年代，Holland 與他在密西根大學的同事及學生們一起研究電腦系統開發時，參考了自然界的這個適應現象，提出了遺傳演算法[22]。在這個啟發下，David E. Goldberg 在 1989 年於他的著作[18]中系統化的研究遺傳演算法的機制，確定三種基本演算子（operator），如圖 3-12 所示。



圖 3-12 遺傳演算法的三個運算子

遺傳演算法是利用生物學的生存法則—「物競天擇，適者生存，不適者淘汰」的原理，運用電腦運算模擬所發展出來的一種搜尋法

則。遵循物種適者生存的原理，由電腦程式決定一個初始族群，模擬物種的演化與篩選作用，並經由複製（reproduction）、交配（crossover）、突變（mutation）三個運算子交互作用之後，產生較母代優良之個體，如此反覆下去，最後留下最適合在此環境中生存的個體。

近年來，隨著遺傳演算法理論基礎的日益成熟，已經有許多學者將遺傳演算法應用在各種不同的領域中。當所要搜尋的解答空間很大、有雜訊、非線性且複雜，並且對於可能解答一無所知時，正是應用遺傳演算法的最佳時機。潘順興[57]等人提到，遺傳演算法具備了下列的優點：

1. 使用隨機的方法進行每一個世代之間的轉換，而非採用固定的模式。
2. 只需面對參數在編碼後的字串，而不用再考慮參數本身特殊的限制。
3. 能夠同時以多點搜尋，而不是以單獨一點來搜尋。
4. 適合度函數（fitness function）是唯一的資訊來源，不需要其他特殊的條件來進行搜尋。

以往的演算法則大都是隨著某一特定的問題模式，有著相當多的限制條件，而遺傳演算法完全以適合度函數作為評估的考量，也因此，它具有一些彈性是其他方法所無法達到的。

同時遺傳演算法也有著平行處理的能力，這種平行處理的能力是巧妙的隱藏在演算法之中，靠著在空間中散佈的各點同時搜索不同的區域。如此靠著每一個世代不斷的演化，在加上有效的利用前一代的

資訊來搜尋，可加快獲得最佳解的速度。而突變的機制，使它有更多的機會能夠跳脫空間中局部最佳解的限制，能夠往整體最佳解的方向收斂。

3.3.2 遺傳演算法運作方式

遺傳演算法為一隨機性的搜尋法，與傳統上需要初始設計值的搜尋法不同，遺傳演算法的初始設計值為一群由電腦隨機產生的設計值組，稱為母體（population），而母體中每一組設計值則稱為個體（individual），每一個體均有代表其特性的染色體（chromosome），而染色體則由遺傳因子，即所謂的基因（gene）串連而成（如圖 3-13 所示）。一般多利用二進位字串（binary bit）來表示染色體，而經過解碼後所呈現出來的便是一組組的設計參數了。

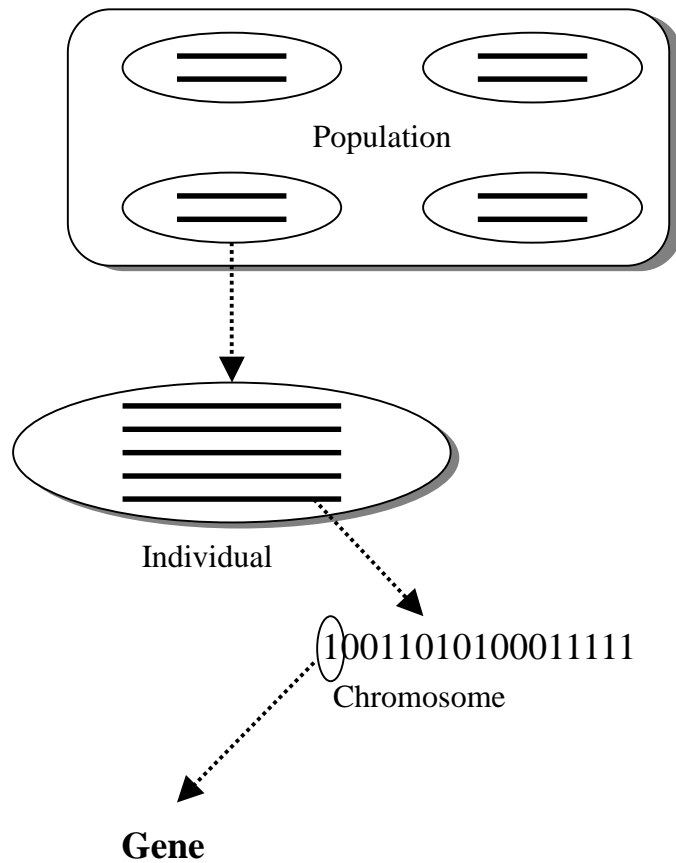


圖 3-13 遺傳演算法組成圖

十進位參數之染色體編碼與解碼方式如下，假設欲求解的變數共 k 個，且每個變數 x_i 的範圍介於實數 a_i 與 b_i 間，變數的精確度要求到小數點下 m 位，則：

1. 染色體長度的決定方式，是找出最小整數 li ，使

$(b_i - a_i) \cdot 10^m < 2^{li} - 1$ ， li 即為 x_i 二進位的編碼長度，故整體求解變數的染色體長度即為 $l = \sum_{i=1}^k li$ 。

5. 解碼：將整體編碼後的染色體，依順序裁切回 k 組二進位字串後，並依據下式將它轉換回十進位。

$$x_i = a_i + decimal(1001K\ 001_2) \cdot \frac{b_i - a_i}{2^{li-1}}$$

遺傳演算法的基本機制乃是遵循自然界中有性生殖的法則所推演出來的，故其中包含了染色體中基因的架構方式、選擇、複製、交配、基因突變、產生新的個體等方式。其運作之基本流程如圖 3-14 所示。

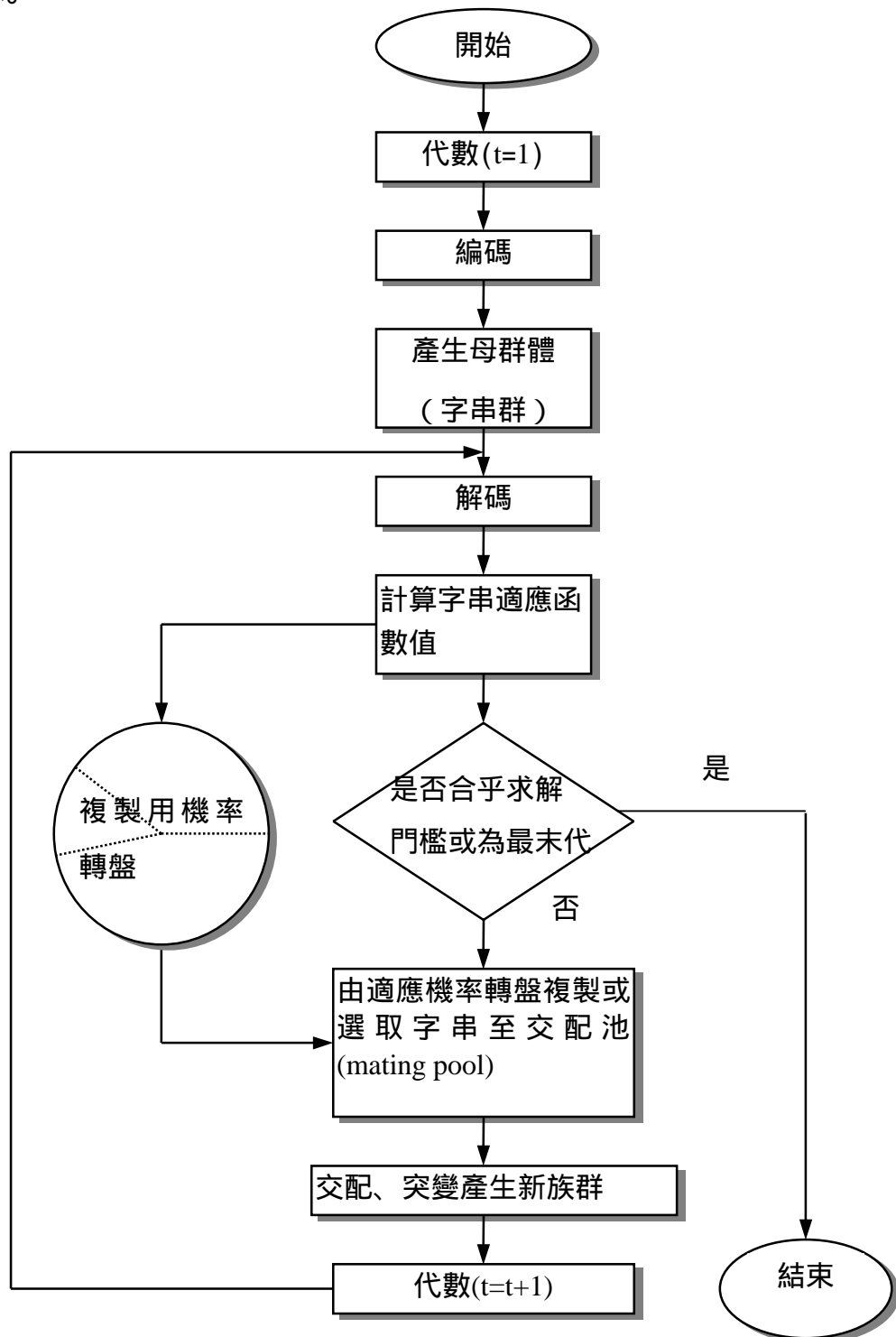


圖 3-14 遺傳演算法運作流程圖

在本研究中,使用遺傳演算法求解單位成本下之系統可用度最佳化,是故目標函數定義為單位成本下之系統可用度,求解問題的模式見第 4 小節。以下分別對遺傳演算法中複製 (reproduction)、交配 (crossover) 以及突變 (mutation) 這三種運算法則進行說明。

1. 複製 (reproduction)

複製運算主要是在產生新的母體時,將原母體的染色體原封不動地複製到新的母體中。在複製的運算中,本研究使用的是輪盤法做為複製的依據,當染色體的適合度愈高時,則複製到新的母體的機率則愈高,也就是,當某組參數達到較佳的目標函數值的話,就有比較高的機率複製到新的母體。

一般在複製運算中,輪盤法是最常被運用到的複製法則,其簡述如下:

在輪盤法中,將一個輪盤分為許多扇型區域,各區域面積大小代表每個個體被選出來的機率,轉動輪盤,假想射出一支飛鏢,射中的區域,代表被選出來進行複製的動作,而區域面積的大小,則和染色體的適合度有關,當適合度越大,代表所佔有的區域面積越大,也愈容易被選中,每一個個體被選中的機率,可由下列期望值的數學式表示:

$$ex(j) = \frac{f_j}{f}$$

$$\bar{f} = \frac{\sum_{j=1}^N f_j}{N}$$

f_j : 為第 j 個染色體的適應值

\bar{f} : 為族群的平均適應值

N : 為族群中所含有的個體數目

2. 交配 (crossover)

將複製的兩組染色體，模擬自然界生物體有性生殖的交配現象，交換彼此部分的基因，以產生新的個體，這個機制的主要作用是希望透過這樣的一個程序，產生適應度較佳的染色體。基本上，交配的方式有三種，分別是單點交配、雙點交配及均勻交配。本研究所採用的為雙點交配，其交配的方式為圖 3-15 所示。在雙點交配運算中，交配點的選擇，為隨機產生的，並且後代染色體中的每個基因都有可能來自於不同的祖先，也就是說，在選擇後代染色體的每個基因時，有特定的機率來自不同的祖先染色體。而這個特定的機率就稱為交配率 (crossover rate)。

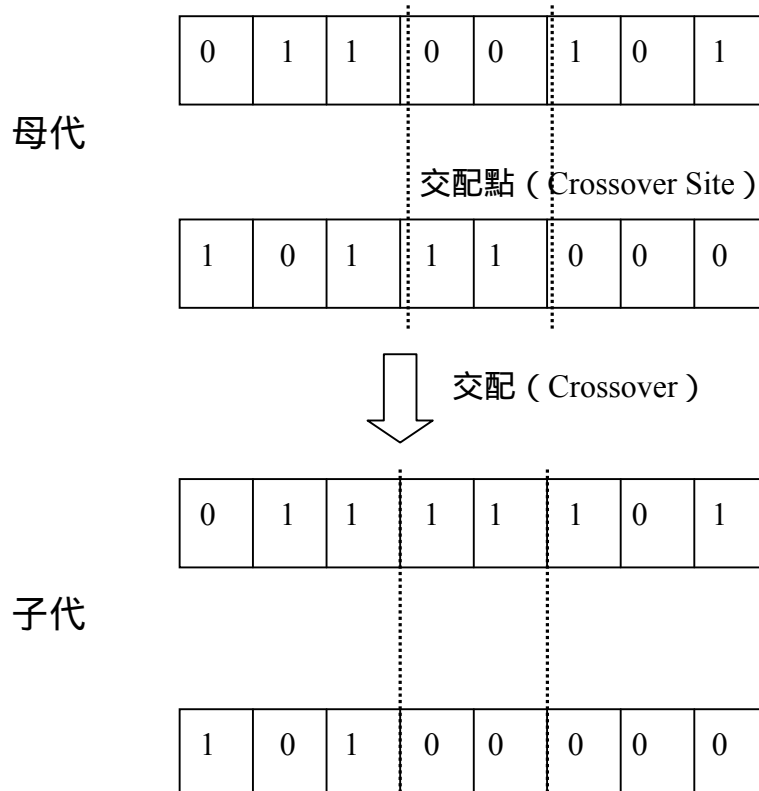


圖 3-15 交配運算示意圖

3. 突變 (mutation)

基本上經由複製及交配的運算步驟，已經使得問題的解朝向最佳化邁進，但是為了避免陷入區域最佳解 (local optimal)，突變即是尋求另一種參數組合 (染色體) 的方式。突變是染色體中的某些基因偶然改變，染色體中的每一個基因都有相等的機會突變，由突變機率 (mutation rate) 來決定其選擇，通常突變機率都非常小，這種作法是為了避免將優良的染色體大量遺失，染色體之基因並不是全部都要突變，通常給定一個突變機率來決定其是否突變。

突變過程如圖 3-16 所示，假設突變點為染色體中的第三個基因，則原本的值就會由 0 變 1，1 變 0。

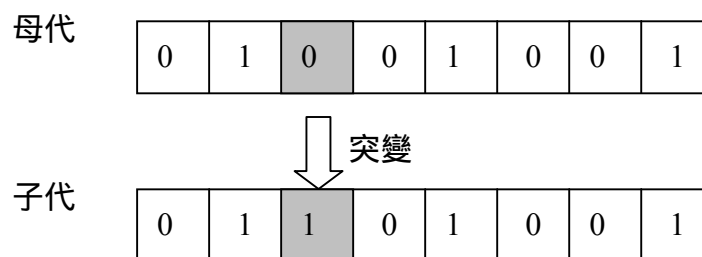


圖 3-16 突變運算示意圖

在經過複製、交配及突變三個運算過程後，接下來便是要判斷何時停止演化，常用的簡單方式就是直接設定演化幾個世代以後停止。至於幾代才合適則需看問題的複雜程度而定。另一種是設定時間，例如一小時以後停止，缺點皆是在於停止時並無法確定是否已經找到最佳解。此外，亦有以每代進步的幅度大小來決定是否停止，亦有兩種方式，第一種是以進步情形為依據，分為以每代族群的適合度為平均值、及以目前搜尋所得之最適適合度在演化過程中已經無法進步或進

步有限時，表示已經收斂，第二種方式是以適合度平均值與收尋過程中所得最佳適合度的誤差值百分比為依據，達到某種程度來代表已經收斂。

本研究是以由設計者直接設定世代數的方式，讓程式在允許的時間內執行，或容許直接中斷程式並顯示所得之最佳解，供系統設計人員作為輔助決策之依據。

3.3.3 遺傳演算法參數設定

前兩小節介紹了遺傳演算法在本研究中的運用方式，但對於本研究中求解時所用到的演算法參數值，則尚未加以說明，這些參數值選擇的最佳化，它們影響的是求解時的效率。

對本研究而言，第四章所舉的例子中 GA 求解參數的設定，是根據運用 GA 於可靠度設計的文獻[49][15][51]，這些文獻中所使用的參數如下表 3-2。

表 3-2 文獻中 GA 求解參數設定

| 文獻 | 使用的 GA 參數 | |
|------|-----------|------|
| [49] | 母體大小 | 100 |
| | 交配率 | 0.9 |
| | 突變率 | 0.01 |
| | | |
| [15] | 母體大小 | 40 |
| | 交配率 | 0.4 |
| | 突變率 | 0.4 |
| | | |
| [51] | 母體大小 | 14 |
| | 交配率 | 0.4 |
| | 突變率 | 0.1 |

3.4 模式的建立

3.4.1 符號與假設

| | |
|-------------------------------|-------------------------------------|
| $Avai_s$ | 系統可用度 |
| $Tcost_s$ | 系統總成本 |
| $MTBF_i$ | 元件 i 的 MTBF |
| $CMTBF_i$ | 元件 i 的製造成本 |
| Lb_MTBF_i | 元件 i 的 MTBF 下限 |
| Lb_CMTBF_i | 元件 i MTBF 下限的製造成本 |
| Ub_MTBF_i | 元件 i 的 MTBF 上限 |
| Ub_CMTBF_i | 元件 i MTBF 上限的製造成本 |
| $MTTR_i$ | 元件 i 的 MTTR |
| $CMTTR_i$ | 元件 i 的維修成本 |
| Lb_MTTR_i | 元件 i 的 MTTR 下限 |
| Lb_CMTTR_i | 元件 i MTTR 下限的維修成本 |
| Ub_MTTR_i | 元件 i 的 MTTR 上限 |
| Ub_CMTTR_i | 元件 i MTTR 上限的維修成本 |
| $\alpha_i, \beta_i, \gamma_i$ | 皆為正的常數, 代表每一元件製造成本與其 MTBF 間函數關係的係數值 |
| a_i, b_i | 皆每一元件維修成本與其 MTTR 間函數關係的係數值 |

假設系統組件的失效率 $(=1/MTBF)$ 與修復率 $\mu (=1/MTTR)$ 皆為常數，故障時間與修理時間服從指數分配，且 $MTBF \gg MTTR$ ，元件間的維修彼此獨立。元件 MTBF 或 MTTR 與其成本間的函數關係可以明確列出。

3.4.2 問題形態

可修復的串並聯系統共有 k 個元件，每個元件 i 各有兩個待決策參數 $MTBF_i$ 與 $MTTR_i$ ，且其與成本間的關係為

$$CMTBF_i = \alpha_i \cdot (MTBF_i)^{\beta_i} + \gamma_i \quad i=1,2,\dots,k$$

$$CMTTR_i = a_i - b_i \cdot MTTR_i \quad i=1,2,\dots,k$$

欲求得最符合經濟效益之參數即

$$\text{Max} \quad Avai_s / Tcost_s$$

$$= f(MTBF_1, MTBF_2, \dots, MTBF_k, MTTR_1, MTTR_2, \dots, MTTR_k)$$

$$\text{Subject to} \quad Lb_MTBF_i \leq MTBF_i \leq Ub_MTBF_i$$

$$Lb_MTTR_i \leq MTTR_i \leq Ub_MTTR_i$$

$$\text{For } i=1,2,\dots,k$$

當各系統元件參數設定為 MTBF 的上限與 MTTR 的下限時，會出現系統可用度的極大值，但此時所對應的系統總成本也是最大的。對此兩難的抉擇，我們定義目標函數為以系統可用度除以系統總成本，如此一來即意味著，決策目標是要找出一最符合於經濟效益的參數決策點。

3.4.3 求解步驟

求解步驟分兩階段進行 -

階段一：

Step 1：依據 3.2 節所述的方法列出系統可靠度估算式。

$$Avai_s = f(MTBF_{1,\Lambda}, MTBF_k, MTTR_{1,\Lambda}, MTTR_k)$$

系統設計人員並將該系統架構分析完的可用度估算式予以確實紀錄。往後遇到同樣的產品設計架構時，將可快速由以往所累積的知識庫中直接取用，避免浪費心力於求算相同估算式。

Step 2：列出系統成本加總計算式

$$Tcost_s = \sum_{i=1}^k (\alpha_i \cdot (MTBF_i)^{\beta_i} + \gamma_i) + \sum_{i=1}^k (a_i - b_i \cdot MTTR_i)$$

至於每一種類系統元件的 MTBF，或是 MTTR 與成本間的函數關係，可以先建立估算列表，供使用時查詢。

階段二：

Step 1：列出目標函數

$$Obj_fun f(MTBF_{1,\Lambda}, MTBF_k, MTTR_{1,\Lambda}, MTTR_k) = Avai_s / Tcost_s$$

Step 2：進行 GA 求解

程式的部分，如果每次進行系統設計時，都需進行大量程式撰寫，將會對系統設計人員造成極大負擔，亦不符合知識累積的精神。而無論串並聯系統的架構為何，其運用遺傳演算法交配複製突變的機制並無不同，是故，當要設計新系統時，只要針對舊有程式庫中的程

式主體，進行目標函數的變更、參數設定、entity 增減等部分微幅修改即可，於程式撰寫部分將可快速完成。往後若有程式庫中同樣個數 entity 要進行最佳參數求解時，只要直接從舊有程式庫中叫出該系統結構所屬程式碼，設定新的 GA 參數，即可執行程式求算結果。

本方法 GA 求解時所設定的參數，如 3.3.3 節所述。其中的染色體長度可依據所要求的精確度，參考 3.3.2 小節對變數進行 GA 編碼，換算成染色體所需長度。執行代數的設定上，可自行設定。並且程式設計成在執行階段，每當目標函數有所突破時，即予以輸出到文字檔，並可隨時因應需求而終止程式，如此一來可以直接由該文字檔得知程式執行停止時所獲得的最佳參數值。

依據本小節所提出的方法，不論串並聯系統的架構為何，都可估算其系統可用度，也因此可列出目標函數並運用 GA 進行求解，輔助做出參數之決策。

第四章 範例說明與結果分析

4.1 問題描述

以一可修復系統為例，該系統由 8 個元件所構成，其整體系統架構如圖 4-1 所示，欲在可用度與成本因素的考量下，能找出使整個系統的總體效益最大。

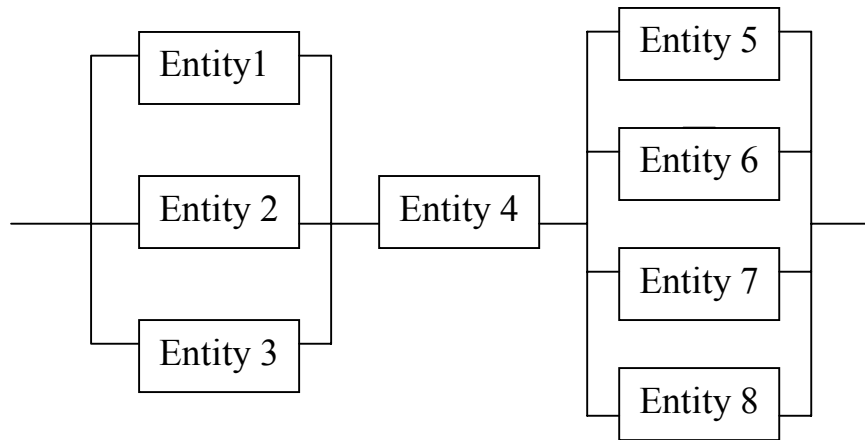


圖 4-1 可修復系統的結構圖

已知各元件的 MTBF 的變動範圍、製造成本以及其函數關係如表 4-1，MTTR 的變動範圍、修復成本及其線性函數關係如表 4-2。

表 4-1 各元件的 MTBF 範圍與製造成本

| 組件 I | MTBF | |
|---------|---------------------------|---------------------------|
| | MTBF 下限 | MTBF 上限 |
| Entity1 | 1400Hrs (\$ 1,012,452) | 1600Hrs (\$ 1,029,622) |
| Entity2 | 1250Hrs (\$ 903,874) | 1450Hrs (\$ 917,857) |
| Entity3 | 1300Hrs (\$ 908,976) | 1500Hrs (\$ 924,782) |
| Entity4 | 1850Hrs (\$ 1,353,896) | 2150Hrs (\$ 1,373,164) |
| Entity5 | 1150Hrs (\$ 983,849) | 1250Hrs (\$ 990,354) |
| Entity6 | 1000Hrs (\$ 852,421) | 1100Hrs (\$ 860,265) |
| Entity7 | 700Hrs (\$ 677,592) | 800Hrs (\$ 685,559) |
| Entity8 | 850Hrs (\$ 837,584) | 950Hrs (\$ 846,704) |

各 $MTBF_i$ 與其成本間之函數關係為

$$CMTBF_1 = 973286.144487441 + 0.000106632269624698 * MTBF_1^{2.7224}$$

$$CMTBF_2 = 875253.236168306 + 0.000143229775819721 * MTBF_2^{2.6803}$$

$$CMTBF_3 = 875728.687792822 + 0.000114385218051994 * MTBF_3^{2.7179}$$

$$CMTBF_4 = 1313531.84768831 + 0.000132374487580487 * MTBF_4^{2.5968}$$

$$CMTBF_5 = 958553.747741483 + 0.000100540641606103 * MTBF_5^{2.7447}$$

$$CMTBF_6 = 826532.937373873 + 0.000120892553407300 * MTBF_6^{2.7769}$$

$$CMTBF_7 = 660434.372677493 + 0.000128234150368538 * MTBF_7^{2.8563}$$

$$CMTBF_8 = 812979.143765579 + 0.000122179122897449 * MTBF_8^{2.8347}$$

表 4-2 各元件的 MTTR 範圍與修復成本

| 組件 i | MTTR 下限 | MTTR 上限 |
|---------|-----------------------|------------------------|
| Entity1 | 80Hrs (\$ 350,102) | 100Hrs (\$ 335,105) |
| Entity2 | 65Hrs (\$ 341,271) | 85Hrs (\$ 330,272) |
| Entity3 | 70Hrs (\$ 334,936) | 90Hrs (\$ 322,938) |
| Entity4 | 2Hrs (\$ 405,736) | 30Hrs (\$ 400,749) |
| Entity5 | 60Hrs (\$ 296,436) | 70Hrs (\$ 291,438) |
| Entity6 | 50Hrs (\$ 291,972) | 60Hrs (\$ 286,964) |
| Entity7 | 35Hrs (\$ 286,103) | 45Hrs (\$ 281,305) |
| Entity8 | 40Hrs (\$ 290,038) | 50Hrs (\$ 285,437) |

各 $MTTR_i$ 與其成本間之函數關係為

$$CMTR_1 = 350102 - 749.85 * (MTTR_1 - 80)$$

$$CMTR_2 = 341271 - 549.95 * (MTTR_2 - 65)$$

$$CMTR_3 = 334936 - 599.9 * (MTTR_3 - 70)$$

$$CMTR_4 = 405736 - 178.107142857143 * (MTTR_4 - 2)$$

$$CMTR_5 = 296436 - 499.8 * (MTTR_5 - 60)$$

$$CMTR_6 = 291972 - 500.8 * (MTTR_6 - 50)$$

$$CMTR_7 = 286103 - 479.8 * (MTTR_7 - 35)$$

$$CMTR_8 = 290038 - 460.1 * (MTTR_8 - 40)$$

4.2 問題分析

此系統的可用度最大值將會是 8 個系統元件的參數皆選取 MTBF 上限與 MTTR 下限。

系統可用度最大值

$$= 1 - \frac{MTTR_1 \cdot MTTR_2 \cdot MTTR_3}{MTBF_1 \cdot MTBF_2 \cdot MTBF_3} - \frac{MTTR_4}{MTBF_4} - \frac{MTTR_5 \cdot MTTR_6 \cdot MTTR_7 \cdot MTTR_8}{MTBF_5 \cdot MTBF_6 \cdot MTBF_7 \cdot MTBF_8}$$

(本式的由來見 4.3 節)

$$= 1 - \frac{80 \cdot 65 \cdot 70}{1600 \cdot 1450 \cdot 1500} - \frac{2}{2150} - \frac{60 \cdot 50 \cdot 35 \cdot 40}{1250 \cdot 1100 \cdot 800 \cdot 950}$$

$$= 0.998961$$

此時對應的系統的總成本也是最大，總成本

= MTBF 成本 + MTTR 成本

$$= (CMTBF_1 + CMTBF_2 + CMTBF_3 + CMTBF_4 + CMTBF_5 + CMTBF_6 + CMTBF_7 + CMTBF_8) + (CMTRR_1 + CMTRR_2 + CMTBF_3 + CMTBF_4 + CMTBF_5 + CMTBF_6 + CMTBF_7 + CMTBF_8)$$

$$= (1029622 + 917857 + 924782 + 1373164 + 990354 + 860265 + 685559 +$$

$$846704) + (350102 + 341271 + 334936 + 405736 + 296436 + 291972 +$$

$$286103 + 290038)$$

$$= 10224901$$

目標函數最佳值(可用度/總成本) 為 9.76988e-008

而這個系統可用度的最小值將會是當 8 個系統元件的參數皆選

取選取 MTBF 下限與 MTTR 上限時。

這個系統可用度最小值

$$= 1 - \frac{MTTR_1 \cdot MTTR_2 \cdot MTTR_3}{MTBF_1 \cdot MTBF_2 \cdot MTBF_3} - \frac{MTTR_4}{MTBF_4} - \frac{MTTR_5 \cdot MTTR_6 \cdot MTTR_7 \cdot MTTR_8}{MTBF_5 \cdot MTBF_6 \cdot MTBF_7 \cdot MTBF_8}$$

$$= 1 - \frac{100 \cdot 85 \cdot 90}{1400 \cdot 1250 \cdot 1300} - \frac{30}{1850} - \frac{70 \cdot 60 \cdot 45 \cdot 50}{1150 \cdot 1000 \cdot 700 \cdot 850}$$

$$= 0.983434$$

此時對應的系統的總成本會最小，總成本

= MTBF 成本 + MTTR 成本

$$= (CMTBF_1 + CMTBF_2 + CMTBF_3 + CMTBF_4 + CMTBF_5 + CMTBF_6 + CMTBF_7 + CMTBF_8) + (CMTTR_1 + CMTTR_2 + CMTBF_3 + CMTBF_4 + CMTBF_5 + CMTBF_6 + CMTBF_7 + CMTBF_8)$$

$$= (1012452 + 903874 + 908976 + 1353896 + 983849 + 852421 + 677592 +$$

$$837584) + (335105 + 330272 + 322938 + 400749 + 291438 + 286964 +$$

$$281305 + 285437)$$

$$= 10064852$$

此時目標函數最佳值(可用度/總成本) 為 $9.77097e-008$

亦即當有這些元件參數可供決策時，在該系統架構下，系統可用度的範圍將介於

$$0.983434 < \text{系統可用度} < 0.998961$$

對應之成本範圍為

$$10064852 < \text{系統總成本} < 10224901$$

面對這樣的問題，欲求得一最符合經濟效益之參數決策，單憑藉經驗法是困難的，因此運用本研究所提出的方法，以輔助做出較佳的決策。

4.3 系統參數估計

本研究所提出的方法，參數估算的求解分兩階段。

階段一：先查詢歷史知識庫，看是否過去有相同架構之系統可用度估算式，若無，則根據 3-2 節的方法，依下列步驟查表 3-1 列出這個系統的可用度估算式。

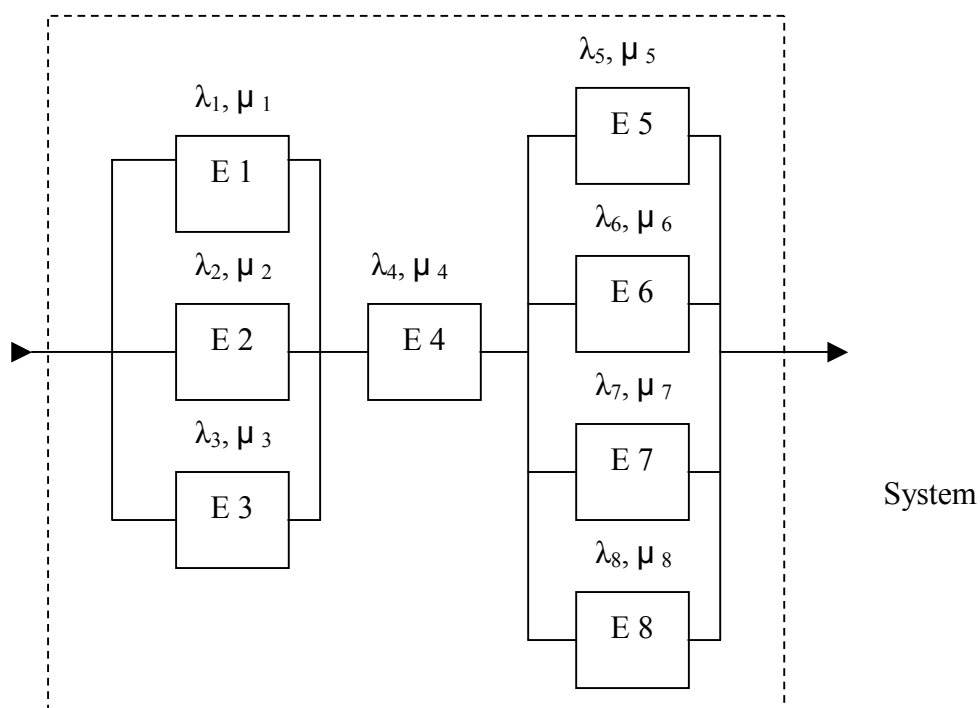


圖 4-2 範例系統架構圖

簡化成

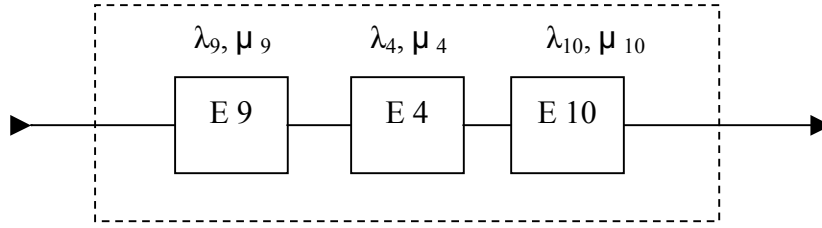
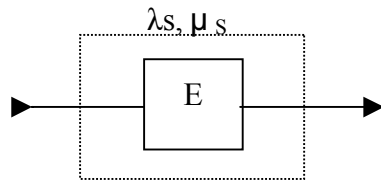


圖 4-3 範例架構簡化 1

$$\lambda_9 \approx \frac{\lambda_1 \lambda_2 \lambda_3 (\mu_1 + \mu_2 + \mu_3)}{\mu_1 \mu_2 \mu_3} \quad \lambda_{10} \approx \frac{\lambda_5 \lambda_6 \lambda_7 \lambda_8 (\mu_5 + \mu_6 + \mu_7 + \mu_8)}{\mu_5 \mu_6 \mu_7 \mu_8}$$

$$\mu_9 \approx \mu_1 + \mu_2 + \mu_3$$

$$\mu_{10} \approx \mu_5 + \mu_6 + \mu_7 + \mu_8$$



$$\lambda_s \approx \lambda_9 + \lambda_4 + \lambda_{10}$$

$$\mu_s \approx \frac{\lambda_9 + \lambda_4 + \lambda_{10}}{\lambda_9 / \mu_9 + \lambda_4 / \mu_4 + \lambda_{10} / \mu_{10}}$$

圖 4-4 範例架構簡化 2

所以

$$PA_s = 1 - \frac{\lambda_s}{\mu_s} = 1 - \frac{(\lambda_9 + \lambda_4 + \lambda_{10})}{\left(\frac{\lambda_9 + \lambda_4 + \lambda_{10}}{\lambda_9 / \mu_9 + \lambda_4 / \mu_4 + \lambda_{10} / \mu_{10}} \right)} = 1 - \frac{\lambda_9}{\mu_9} - \frac{\lambda_4}{\mu_4} - \frac{\lambda_{10}}{\mu_{10}}$$

$$= 1 - \frac{\lambda_1 \lambda_2 \lambda_3}{\mu_1 \mu_2 \mu_3} - \frac{\lambda_4}{\mu_4} - \frac{\lambda_5 \lambda_6 \lambda_7 \lambda_8}{\mu_5 \mu_6 \mu_7 \mu_8}$$

$$= 1 - \frac{MTTR_1 \cdot MTTR_2 \cdot MTTR_3}{MTBF_1 \cdot MTBF_2 \cdot MTBF_3} - \frac{MTTR_4}{MTBF_4} - \frac{MTTR_5 \cdot MTTR_6 \cdot MTTR_7 \cdot MTTR_8}{MTBF_5 \cdot MTBF_6 \cdot MTBF_7 \cdot MTBF_8}$$

系統的總成本

$$= \text{MTBF 成本} + \text{MTTR 成本}$$

$$\begin{aligned}
&= (CMTBF_1 + CMTBF_2 + CMTBF_3 + CMTBF_4 + CMTBF_5 + CMTBF_6 + \\
&CMTBF_7 + CMTBF_8) + (CMTTR_1 + CMTTR_2 + CMTBF_3 + CMTBF_4 + \\
&CMTBF_5 + CMTBF_6 + CMTBF_7 + CMTBF_8) \\
&= 973286.144487441 + 0.000106632269624698 * MTBF_1^{2.7224} + \\
&875253.236168306 + 0.000143229775819721 * MTBF_2^{2.6803} + \\
&875728.687792822 + 0.000114385218051994 * MTBF_3^{2.7179} + \\
&1313531.84768831 + 0.000132374487580487 * MTBF_4^{2.5968} + \\
&958553.747741483 + 0.000100540641606103 * MTBF_5^{2.7447} + \\
&826532.937373873 + 0.000120892553407300 * MTBF_6^{2.7769} + \\
&660434.372677493 + 0.000128234150368538 * MTBF_7^{2.8563} + \\
&812979.143765579 + 0.000122179122897449 * MTBF_8^{2.8347} + \\
&+ 350102 - 749.85 * (MTTR_1 - 80) + 341271 - 549.95 * (MTTR_2 - 65) \\
&+ 334936 - 599.9 * (MTTR_3 - 70) + 405736 - 178.107142857143 * (MTTR_4 - 2) \\
&+ 296436 - 499.8 * (MTTR_5 - 60) + 291972 - 500.8 * (MTTR_6 - 50) \\
&+ 286103 - 479.8 * (MTTR_7 - 35) + 290038 - 460.1 * (MTTR_8 - 40)
\end{aligned}$$

階段二：

定義目標函數為 $f(x) = \frac{\text{系統可用度}}{\text{系統總成本}}$ ，並運用 GA 作為運算核心進行

求解。

首先依據精度的需求，決定染色體長度，這部分可直接以一個小程序完成，該程式並可不需修改重複使用，假設我們只要求精確到整數部分，程式執行如下。

執行計算染色體程式

請問共有幾個變數? 16

要求精確到小數點下幾位? 0

請輸入 MTBF1 的下限?1400 請輸入 MTBF1 的上限?1600

請輸入 MTBF2 的下限?1250 請輸入 MTBF2 的上限?1450

請輸入 MTBF3 的下限?1300 請輸入 MTBF3 的上限?1500

請輸入 MTBF4 的下限?1850 請輸入 MTBF4 的上限?2150

請輸入 MTBF5 的下限?1150 請輸入 MTBF5 的上限?1250

請輸入 MTBF6 的下限?1000 請輸入 MTBF6 的上限?1100

請輸入 MTBF7 的下限?700 請輸入 MTBF7 的上限?800

請輸入 MTBF8 的下限?850 請輸入 MTBF8 的上限?950

請輸入 MTTR1 的下限?80 請輸入 MTTR1 的上限?100

請輸入 MTTR2 的下限?65 請輸入 MTTR2 的上限?85

請輸入 MTTR3 的下限?70 請輸入 MTTR3 的上限?90

請輸入 MTTR4 的下限?2 請輸入 MTTR4 的上限?30

請輸入 MTTR5 的下限?60 請輸入 MTTR5 的上限?70

請輸入 MTTR6 的下限?50 請輸入 MTTR6 的上限?60

請輸入 MTTR7 的下限?35 請輸入 MTTR7 的上限?45

請輸入 MTTR8 的下限?40 請輸入 MTTR8 的上限?50

得到結果如下

代表 MTBF1 的染色體長度應為 8
代表 MTBF2 的染色體長度應為 8
代表 MTBF3 的染色體長度應為 8
代表 MTBF4 的染色體長度應為 9
代表 MTBF5 的染色體長度應為 7
代表 MTBF6 的染色體長度應為 7
代表 MTBF7 的染色體長度應為 7
代表 MTBF8 的染色體長度應為 7
代表 MTTR1 的染色體長度應為 5
代表 MTTR2 的染色體長度應為 5
代表 MTTR3 的染色體長度應為 5
代表 MTTR4 的染色體長度應為 5
代表 MTTR5 的染色體長度應為 4
代表 MTTR6 的染色體長度應為 4
代表 MTTR7 的染色體長度應為 4
代表 MTTR8 的染色體長度應為 4
染色體的長度應設定為 97

至於 GA 求解的程式部分，我們可針對知識庫裡已有的程式主體，進行目標函數的變更及參數重新設定。

這裡執行的程式是直接以 C++ 進行 GA 程式之撰寫。求解時所設定的 GA 參數，如 3.3.3 節所述，嘗試了表 3-2 中的幾組 GA 參數的設定，執行代數設定 4000 代，分別在 Pentium 4 賽揚 1.7G 執行程式 20 次，得到的結果如圖 4-5。

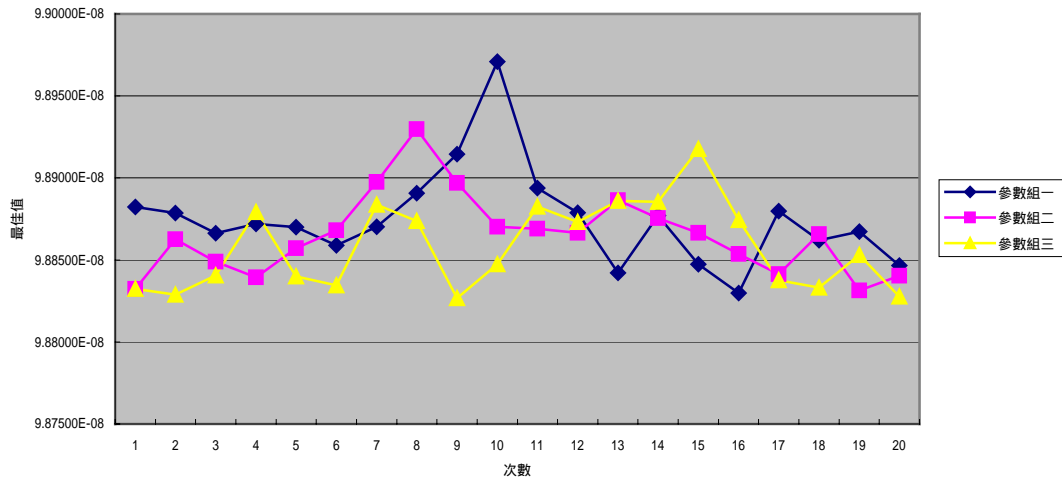


圖 4-5 GA 求解結果

各參數的電腦執行時間與所得最佳目標函數值如表 4-3。

表 4-3 各參數組電腦執行時間與最佳目標函數值

| 使用的 GA 參數 | | 平均執行時間(s) | 最佳目標函數值 |
|-----------|------|-----------|-------------|
| 參數組一 | | | |
| 母體大小 | 100 | 459.55 | 9.89709E-08 |
| 交配率 | 0.9 | | |
| 突變率 | 0.01 | | |
| 參數組二 | | | |
| 母體大小 | 40 | 208.85 | 9.89298E-08 |
| 交配率 | 0.4 | | |
| 突變率 | 0.4 | | |
| 參數組三 | | | |
| 母體大小 | 14 | 83.85 | 9.89177E-08 |
| 交配率 | 0.4 | | |
| 突變率 | 0.1 | | |

電腦所得輔助決策之最佳參數為如表 4-4，其演化的過程可參見附錄一，GA 程式碼見附錄二。

表 4-4 電腦所得輔助決策之最佳參數

| | |
|----------------|----------------|
| MTBF1= 1403.14 | MTTR1= 97.8824 |
| MTBF2= 1270.39 | MTTR2= 81.8689 |
| MTBF3= 1314.17 | MTTR3= 87.9528 |
| MTBF4= 1880.71 | MTTR4= 2.22047 |
| MTBF5= 1153.23 | MTTR5= 67.4194 |
| MTBF6= 1025.81 | MTTR6= 59.3548 |
| MTBF7= 706.667 | MTTR7= 37 |
| MTBF8= 870 | MTTR8= 48.6667 |

目標函數最佳值(可用度/總成本)

9.89709e-008

這時即可以此資訊選取符合需求的系統組件，並選定相對應之元件修復策略。

第五章 結論與建議

5.1 結論

在知識經濟的時代，對於可修復串並聯系統的設計上，若是依舊倚靠經驗法則來進行系統參數之決策，不僅老手經驗的傳承不易，也往往容易造成成本上的額外浪費。

而在科學昌明的時代，電腦的運算能力日新月異，如果能夠善用它強大的運算功能，將所面臨的問題與予模式化，並使用電腦事先進行結果評估，將可提供參數決策不少幫助。

本研究提出一套方法，先以查表法列出系統可用度估算式，再以單位成本下系統可用度為目標函數，運用遺傳演算法進行目標函數最大化之求解。對於程式的撰寫部分，猶如經驗法則之傳承，無須整個重新撰寫，只要取出過去系統設計知識庫中的類似程式，對之進行微幅修改即可。

在面臨系統設計參數的決策上，本研究提出一最佳化模式並運用遺傳演算法求得最佳解，提供系統參數決策時一有利且更容易使人信服的方法。

5.2 後續研究與建議

在本研究中，對於系統可用度的估算，是以查表的方式列出系統的可用度估算式，再將訂定出來的目標函數（系統可用度/系統總成本）輸入程式中進行求解。在實務的運用上，有些複雜系統的可用度並不易列出估算式，此時，可以運用 Monte Carlo Simulation 進行系統可用度的模擬，如結合商用套裝軟體或是撰寫模擬的程式，於程式

中運算執行。不過結合模擬估算系統可靠度或可用度的情形下，程式的執行時間也將會相對變得冗長。

系統可用度的提高是我們的目標，但在可用度提高的同時，也伴隨著系統成本的提高，這兩者目標間的衝突如何取捨是一大關鍵。本研究使用的方法，是以單位成本的系統可用度作為最佳化的目標，後續研究者可針對此一議題，對本方法的應用進行更深入的研究。例如使用多目標最佳化的方式，在把各個目標的尺度予以正規化後，根據重視的程度分別賦予權重，將多目標結合成為單一目標函數，並使用遺傳演算法或其他方法進行求解。或將各個目標間的模糊（Fuzzy）程度，運用模糊理論中模糊歸屬函數，以更切合實際決策之情境。

此外，本研究使用遺傳演算法對問題進行求解，但未針對求解的效率予以分析，後續研究者可以此作為進一步研究的標的，分析其運算效率。

參考文獻

1. Bellman R. E., and E. Dreyfus, “Dynamic Programming and Reliability of Multicomponent Devices,” *Operations Research*, Vol. 6, pp. 200-206, 1958.
2. Birolini A., *Reliability Engineering Theory and Practice*, Springer, 1999.
3. Bulfin R. L., and C. Y. Liu, “Optimal Allocation of Redundant Components for Large Systems,” *IEEE Trans. Reliability*, Vol. R-34, pp. 241-247, 1985.
4. Chern, M. S., “On the Computational Complexity of Reliability Redundancy Allocation in a Series System,” *Operations Research Letters*, Vol. 11, pp. 309-315, 1992.
5. Chisman James A., “Using Discrete Simulation Modeling to Study Large-Scale System Reliability/Availability,” *Computer Ops. Res.*, Vol. 25, No. 3, pp. 169-174, 1998.
6. Chodos M S, “Availability and Maintenance Considerations in Telecommunication Network Design and the Use of Simulation,” *B HAMERSMA, Member, IEEE*, pp. 267-270, 1992.
7. Coit David W. and Alice E. Smith, “Solving the Redundancy Allocation Problem Using a Combined Neural Network/Genetic Algorithm Approach,” *Computers and Operations Research*, Vol. 23, No. 6, pp.515-526, 1996.
8. Coit, David W. and Alice E. Smith, “Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm,” *IEEE Trans. Reliability*, Vol. 45, No. 2, pp. 254-260, 1996.

9. Dinghua S., "A New Heuristic Algorithm for Constrained Redundancy-Optimization in Complex Systems," *IEEE Trans. Reliability*, Vol. R-36, No. 5, pp. 621–623, 1987.
10. Ebeling Charles E., *An Introduction to Reliability and Maintainability Engineering*, McGraw-Hill, 1997.
11. Fisher M., "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science*, Vol. 27, pp.1-18, 1981.
12. Fyfee D.E., W. W. Hines, and N. K. Lee, "System Reliability Allocation and a Computational Algorithm," *IEEE Trans. Reliability*, Vol. R-17, pp. 64-69, 1968.
13. Gen M., K. Ida, and J. U. Lee, "A Computational Algorithm for Solving 0-1 Goal Programming with GUB Structures and Its Application for Optimization Problems in System Reliability," *Electronics and Communications in Japan*, part 3, Vol. 73, pp. 88-96, 1990.
14. Gen M., K. Ida, Y. Tsujimura, and C. E. Kim, "Large-scale 0-1 Fuzzy Goal Programming and Its Application to Reliability Optimization Problem," *Computers and Industrial Engineering*, Vol. 24, pp. 539-549, 1993.
15. Gen Mitsuo and Runwei Cheng, "Optimal Design of System Reliability Using Interval Programming and Genetic Algorithms," *Computers Ind. Engng*, Vol. 31, No. 1/2, pp.237-240, 1996.
16. Gen Mitsuo and Runwei Cheng, *Genetic Algorithms and Engineering Optimization*, Wiley-Interscience, 2000.
17. Ghare P. M., and R. E. Taylor, "Optimal Redundancy for Reliability in

- Series System,” *Operations Research*, Vol. 17, pp. 838-847, 1969.
18. Goldberg D. E., *Genetic Algorithm in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
 19. Gopal K., K. K. Aggarwal, and J. S. Gupta, “An Improved Algorithm for Reliability Optimization,” *IEEE Trans. Reliability*, Vol. 27, No. 5, pp. 325–328, 1978.
 20. Gordan John, “Computational Methods for Reliability Data Analysis,” *Annual Reliability and Maintainability Symposium*, pp. 287-290, 1996.
 21. Henley Ernest j. and Hiromitsu Kumamoto, *Design for Reliability and Safety Control*, Prentice-Hall, Englewood Cliffs, N. J., 1985.
 22. Holland J. H., *Adaptation in Natural and Artificial Systems*, Ann Arbor: The University of Michigan Press, 1975.
 23. Huang H. Z., “Fuzzy Multi-Objective Optimization Decision-Making of Reliability of Series System,” *Microelectron. Reliab.*, Vol. 37, No. 3, pp. 447-449, 1997.
 24. Jeang, A., “Computer-Aided Tolerance Synthesis with Statistical Method and Optimization Techniques,” *Quality and Reliability Engineering International*, No. 17, pp. 131-139, 2001.
 25. Jeang, A., “Optimal Process Parameter Determination for Computer-Aided Manufacturing,” *Quality and Reliability Engineering International*, Vol. 15, pp. 3-16, 1999.
 26. Jianping L., “A Bound Heuristic Algorithm for Solving Reliability Redundancy Optimization,” *Microelectronics and Reliability*, Vol. 3, No. 5, pp. 335–339, 1996.

27. Kales Paul, *Reliability for Technology, Engineering and Management*, Prentice Hall, 1998.
28. Kim J. H. and B. J. Yum, "A Heuristic Method for Solving Reliability Redundancy Optimization Problems in Complex Systems," *IEEE Trans. Reliability*, Vol. 42, No. 4, pp. 572-578, 1993.
29. Kohda T. and K. Inoue, "A Reliability Optimization Method for Complex Systems with the Criterion of Local Optimality," *IEEE Trans. Reliability*, Vol. R-31, No. 1, pp. 109-111, 1982.
30. Kuo W., C. L. Hwang, and F. A. Tillman, "A Note on Heuristic Methods in Optimal System Reliability," *IEEE Trans. Reliability*, Vol. 27, No. 5, pp. 320–324, 1978.
31. Kuo, W. and V. R. Prasad, "An Annotated Overview of System Reliability Optimization," *IEEE Trans. Reliability*, Vol. 49, No. 2, pp. 176-187, 2000.
32. Kuo, W., H. Lin, Z. Xu and W. Zhang, "Reliability Optimization with the Lagrange Multiplier and Branch-and-Bound Technique," *IEEE Trans. Reliability*, Vol. 36, pp. 624-630, 1987.
33. Li D. and Y. Y. Haimes, "A Decomposition Method for Optimization of Large-System Reliability," *IEEE Trans. Reliability*, Vol. 41, pp. 183-188, 1992.
34. Lieber Dmitrii, Arkadii Nemirovskii, and Reuven Y. Rubinstein, "A Fast Monte Carlo Method for Evaluating Reliability Indexes," *IEEE Trans. Reliability*, Vol. 48, No. 3, pp. 256-261, 1999.
35. Misra K. B. and U. Sharma, "An Efficient Algorithm to Solve Integer Programming Problems Arising in System Reliability Design," *IEEE Trans. Reliability*, Vol. 40 No. 1, pp. 81-91, 1991.

36. Mitchell Billy F. and Robert J. Murry, "Predicting Operational Availability for Systems with Redundant, Repairable Components and Multiple Sparing Levels," *Annual Reliability and Maintainability Symposium*, pp. 301-305, 1996.
37. Mohan C. and K. Shanker, "Reliability Optimization of Complex Systems Using Random Search Technique," *Microelectronics and Reliability*, Vol. 28, No. 4, pp.513-518, 1998.
38. Nakagawa Y. and K. Miyazaki, "Surrogate Constraints Algorithm for Reliability Optimization Problem with Two Constraints," *IEEE Trans. Reliability*, Vol. 30, pp. 175-180, June 1981.
39. Nakagawa Y. and K. Nakashima, "A Heuristic Method for Determining Optimal Reliability Allocation," *IEEE Trans. Reliability*, Vol. 26, No. 3, pp. 156–161, 1977.
40. Nakagawa Y. and S. Miyazaki, "An Experimental Comparison of the Heuristic Methods for Solving Reliability Optimization Problems," *IEEE Trans. Reliability*, Vol. 30, pp. 181–184, June 1981.
41. Painton L. and J. Campbell, "Genetic Algorithms in Optimization of System Reliability," *IEEE Trans. Reliability*, Vol. 44, pp. 172–178, 1995.
42. Prasad V. R. and W. Kuo, "Reliability Optimization of Coherent Systems," *IEEE Trans. Reliability*, Vol. 49, pp. 323-330, Sept 2000.
43. Propst John E. and Daniel R. Doan, "Improvements in Modeling and Evaluation of Electrical Power System Reliability," *IEEE Transactions on Industry Applications*, Vol. 37, No. 5, pp. 1413-1422, 2001.
44. Sharma J. and K. V. Venkateswaran, "A Direct Method for

- Maximizing the System Reliability,” *IEEE Trans. Reliability*, Vol. 20, No. 1, pp. 256–259, 1971.
45. Tillman F. A., “Optimization by Integer Programming of Constrained Reliability Problems with Several Modes of failure,” *IEEE Trans. Reliability*, Vol. R-18, No. 2, pp. 47-53, 1969.
46. Tillman F. A., C. L. Hwang, and W. Kuo, “Determining Component Reliability and Redundancy for Optimum System Reliability,” *IEEE Trans. Reliability*, Vol. R-26, No. 3, pp. 162-165, 1977.
47. Tillman Frank A., Ching-Lai Hwang and Way Kuo, *Optimization of Systems Reliability*, Marcel Dekker, 1980.
48. Wang Wendai, “Confidence Limits on the Inherent Availability of Equipment,” *Annual Reliability and Maintainability Symposium*, pp. 162-168, 2000.
49. Yang Joon-Eon, Mee-Jung Hwang, Tae-Yong Sung, and Youngho Jin, ”Application of Genetic Algorithm for Reliability Allocation in Nuclear Power Plants,” *Reliability Engineering and System Safety*, No. 65, pp. 229-238, 1999.
50. Yokota T., M. Gen, and K. Ida, “System Reliability of Optimization Problems with Several Failure Modes by Genetic Algorithm,” *Japanese J. Fuzzy Theory and Systems*, Vol. 7, No. 1, pp. 117-135, 1995.
51. Yokota Takao, Mitsuo Gen and Yin-Xiu Li, “Genetic Algorithm for Non-linear Mixed Integer Programming Problems and Its Applications,” *Computers Ind. Engng*, Vol. 30, No. 4, pp.905-917, 1996.
52. 王宗華編著，可靠度工程技術的理論與實用，中華民國品質管制

- 學會，民國 81 年，五版。
- 53.李正龍，「目標規劃求解串並聯系統之可靠度配置問題」，逢甲大學碩士論文，民國 90 年。
- 54.柯輝耀編著，可靠度保證 - 工程與管理技術之應用，中華民國品質學會，民國 89 年，二版二刷。
- 55.郭昇源，「在成本的考量下來決定系統組件之選用及修復策略的應用以達到系統可用度的最佳化」，逢甲大學碩士論文，民國 91 年。
- 56.陳耀茂編著，可靠性方法與應用，雙葉書廊，民國 87 年，一版一刷。
- 57.潘順興、陳振明、陳稼興，「遺傳演算法於配送點選擇之應用」，資訊管理研究，第二卷，第一期，民國 86 年 7 月。
- 58.蘇木春、張孝德編著，機器學習：類神經網路、模糊系統以及基因演算法則，全華科技圖書股份有限公司，民國 89 年，二版三刷。

附錄一：演化的過程

第 1 代母體

突破了!

=1456.47,89.6471,1355.88,76.9374,1498.43,86.063,1875.98,3.32283,1
150,62.9032,1041.94,59.6774,786.667,38.3333,856.667,42.6667.....目
標函數值=9.84874e-008

第 1 代母體

突破了!

=1507.45,91.2157,1373.92,75.0978,1325.2,79.9213,2083.86,2.22047,1
162.9,68.3871,1006.45,55.1613,713.333,39.6667,930,44.....目標函數值
=9.85019e-008

第 1 代母體

突變突破了!

=1439.22,93.4118,1280.59,69.5793,1487.4,82.7559,1991.73,2.22047,1
211.29,66.129,1009.68,55.1613,800,44.3333,863.333,44.....
目標函數值=9.85197e-008

第 2 代母體

突變突破了!

=1520,98.7451,1257.06,81.5558,1441.73,71.5748,1961.02,4.4252,1159
.68,70,1012.9,56.129,726.667,39,863.333,42.....
目標函數值=9.85306e-008

第 3 代母體

交配突破了!

=1504.31,98.902,1412.35,81.7123,1301.57,89.3701,2010.63,3.10236,1
169.35,68.3871,1064.52,57.4194,726.667,35,930,46.....
目標函數值=9.85771e-008

第 3 代母體

突變突破了!

=1504.31,98.902,1412.35,81.7123,1301.57,89.3701,2010.63,3.10236,1
169.35,68.3871,1012.9,57.4194,726.667,35,930,46.....

目標函數值=9.86158e-008

第 4 代母體

交配突破了!

=1460.39,91.2941,1330.84,2564,1378.74,88.1102,1878.35,3.10236,116
9.35,68.3871,1064.52,57.4194,726.667,35,936.667,44.....

目標函數值=9.86201e-008

第 10 代母體

交配突破了!

=1429.02,89.8039,1330.78,82.0646,1317.32,81.9685,1883.07,2.44094,
1224.19,65.8065,1070.97,58.0645,746.667,43,903.333,40.6667.....

目標函數值=9.86516e-008

第 43 代母體

突變突破了!

=1516.08,91.6863,1275.1,84.726,1326.77,89.0551,1852.36,3.32283,12
46.77,61.9355,1006.45,57.0968,720,35.6667,883.333,48.....

目標函數值=9.86921e-008

第 124 代母體

交配突破了!

=1428.24,94.5098,1311.18,76.1937,1333.07,72.2047,1951.57,3.10236,1
175.81,66.129,1003.23,59.3548,740,43.6667,850,46.....

目標函數值=9.87014e-008

第 142 代母體

交配突破了!

=1407.06,99.7647,1304.12,82.3386,1366.14,81.9685,2112.2,2,1156.45,
64.5161,1064.52,59.0323,713.333,43.6667,890,44.....

目標函數值=9.8713e-008

第 143 代母體

交配突破了!

=1407.06,99.7647,1304.12,82.3386,1366.14,82.126,1885.43,2,1156.45,
64.5161,1064.52,59.0323,713.333,43.6667,890,44.....

目標函數值=9.88447e-008

第 1047 代母體

交配突破了!

=1403.14,97.8824,1270.39,81.8689,1314.17,87.9528,2031.89,2.22047,
1153.23,67.4194,1025.81,59.3548,706.667,37,870,48.6667.....

目標函數值=9.88878e-008

第 1047 代母體

突變突破了!

=1403.14,97.8824,1270.39,81.8689,1314.17,87.9528,1880.71,2.22047,
1153.23,67.4194,1025.81,59.3548,706.667,37,870,48.6667.....

目標函數值=9.89709e-008

共花了 464 秒..

最佳解為 9.89709e-008

附錄二：案例 GA 程式碼

```
// 8 個 Entity 的程式;
//含入檔的部分;
#include "stdafx.h"
#include <math.h>
#include <time.h>
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
// 函數原型的宣告;
// 參數的定義;
#define PchromN 40 //母體大小,偶數較佳;
#define PchromW 97 //染色體總長度;
#define generation 10 //代數;
#define pcrossover 0.4 //交配的機率;
#define pmutation 0.4 //突變的機率;
#define NVAR 16 //變數個數;
double Vara1_low=1400; //變數 a1 下界;
double Vara1_up=1600; //變數 a1 上界;
double Varb1_low=80; //變數 b1 下界;
double Varb1_up=100; //變數 b1 上界;
double Vara2_low=1250; //變數 a2 下界;
double Vara2_up=1450; //變數 a2 上界;
double Varb2_low=65; //變數 b2 下界;
double Varb2_up=85; //變數 b2 上界;
double Vara3_low=1300; //變數 a3 下界;
double Vara3_up=1500; //變數 a3 上界;
double Varb3_low=70; //變數 b3 下界;
double Varb3_up=90; //變數 b3 上界;
double Vara4_low=1850; //變數 a4 下界;
double Vara4_up=2150; //變數 a4 上界;
double Varb4_low=2; //變數 b4 下界;
double Varb4_up=30; //變數 b4 上界;
double Vara5_low=1150; //變數 a5 下界;
double Vara5_up=1250; //變數 a5 上界;
double Varb5_low=60; //變數 b5 下界;
double Varb5_up=70; //變數 b5 上界;
```

```

double Vara6_low=1000; //變數 a6 下界;
double Vara6_up=1100; //變數 a6 上界;
double Varb6_low=50; //變數 b6 下界;
double Varb6_up=60; //變數 b6 上界;
double Vara7_low=700; //變數 a7 下界;
double Vara7_up=800; //變數 a7 上界;
double Varb7_low=35; //變數 b7 下界;
double Varb7_up=45; //變數 a7 上界;
double Vara8_low=850; //變數 a8 下界;
double Vara8_up=950; //變數 a8 上界;
double Varb8_low=40; //變數 b8 下界;
double Varb8_up=50; //變數 b8 上界;
#define m1 8 //變數 a1 染色體的長度;
#define m2 8 //變數 b1 染色體的長度;
#define m3 8 //變數 a2 染色體的長度;
#define m4 9 //變數 b2 染色體的長度;
#define m5 7 //變數 a3 染色體的長度;
#define m6 7 //變數 b3 染色體的長度;
#define m7 7 //變數 a4 染色體的長度;
#define m8 7 //變數 b4 染色體的長度;
#define m9 5 //變數 a5 染色體的長度;
#define m10 5 //變數 b5 染色體的長度;
#define m11 5 //變數 a6 染色體的長度;
#define m12 5 //變數 b6 染色體的長度;
#define m13 4 //變數 a7 染色體的長度;
#define m14 4 //變數 b7 染色體的長度;
#define m15 4 //變數 a8 染色體的長度;
#define m16 4 //變數 b8 染色體的長度;
// 定義目標函數;
double Object_fun(double a1,double b1,double a2,double b2,double a3,double b3,double a4,double b4,double
a5,double b5,double a6,double b6,double a7,double b7,double a8,double b8);
// 主程式;
void main(void)
{
    ofstream Table;
    Table.open("c:\\8entity97b.txt");
    time_t start,end;
    double Maxsolute=0;
    double Bestone[NVAR];

```

```

int beststring[PchromW];
start=time(NULL);
srand((unsigned) time(NULL)); //確保亂數不一樣;
//第 n 代的母體;
Table<<"...8entity....\n";
cout<<"...8entity....\n";
for (int i=1;i<=generation;i++)
{
cout<<"\n"<<"第"<<i<<"代"<<"母體\n\n";
double Sumfitness=0; //整個輪盤的大小;
double rollPb[PchromN]; //初始母體中的各個輪盤機率值(複製用)
double T_rollPb[PchromN]; //初始母體中的各個累計加總輪盤機率值(複製用)
double Rn_select[PchromN]; //初始母體中的輪盤旋轉(複製用)
int chromosome[PchromN][PchromW]; //初始母體第 i 個個體的第 j 個基因
int rpchromosome[PchromN][PchromW]; //複製後母體第 i 個個體的第 j 個基因
int mpchromosome[PchromN][PchromW];
int crossNumber=0,PcrossNumber=0;
double Rcrossover[PchromN];
int Placecross[PchromW];
int Cross_pos1,Cross_pos2; //交配區間
int R_crossNumber;
//染色體的顯示及計算;
if (i==1)
{
for (int j=1;j<=PchromN;j++)
{
double fitvalue[PchromN];
double value10[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
double value_flow[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
cout<<"染色體"<<j<<": ";
for (int k=1;k<=PchromW;k++)
{
chromosome[j-1][k-1]=rand()%2;
cout<<chromosome[j-1][k-1];
}
//增加 Entity 時改這裡.....
for (int cut0=1;cut0<=m1;cut0++)
{
if (chromosome[j-1][cut0-1]==1)

```

```

        {
        value10[0]=value10[0]+pow(2,(m1-cut0));
        }
    }

for (int cut1=1;cut1<=m2;cut1++)
    {
        if (chromosome[j-1][m1+cut1-1]==1)
            {
                value10[1]=value10[1]+pow(2,(m2-cut1));
            }
    }

for (int cut2=1;cut2<=m3;cut2++)
    {
        if (chromosome[j-1][m1+m2+cut2-1]==1)
            {
                value10[2]=value10[2]+pow(2,(m3-cut2));
            }
    }

for (int cut3=1;cut3<=m4;cut3++)
    {
        if (chromosome[j-1][m1+m2+m3+cut3-1]==1)
            {
                value10[3]=value10[3]+pow(2,(m4-cut3));
            }
    }

for (int cut4=1;cut4<=m5;cut4++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+cut4-1]==1)
            {
                value10[4]=value10[4]+pow(2,(m5-cut4));
            }
    }

for (int cut5=1;cut5<=m6;cut5++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+m5+cut5-1]==1)
            {
                value10[5]=value10[5]+pow(2,(m6-cut5));
            }
    }

```

```

    }
    for (int cut6=1;cut6<=m7;cut6++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+m5+m6+cut6-1]==1)
        {
            value10[6]=value10[6]+pow(2,(m7-cut6));
        }
    }
    for (int cut7=1;cut7<=m8;cut7++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+cut7-1]==1)
        {
            value10[7]=value10[7]+pow(2,(m8-cut7));
        }
    }
    for (int cut8=1;cut8<=m9;cut8++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+cut8-1]==1)
        {
            value10[8]=value10[8]+pow(2,(m9-cut8));
        }
    }
    for (int cut9=1;cut9<=m10;cut9++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+cut9-1]==1)
        {
            value10[9]=value10[9]+pow(2,(m10-cut9));
        }
    }
    for (int cut10=1;cut10<=m11;cut10++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+cut10-1]==1)
        {
            value10[10]=value10[10]+pow(2,(m11-cut10));
        }
    }
    for (int cut11=1;cut11<=m12;cut11++)
    {
        if (chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+cut11-1]==1)

```

```

        {
            value10[11]=value10[11]+pow(2,(m12-cut11));
        }
    }
    for (int cut12=1;cut12<=m13;cut12++)
    {
        if
(chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+cut12-1]==1)
        {
            value10[12]=value10[12]+pow(2,(m13-cut12));
        }
    }
    for (int cut13=1;cut13<=m14;cut13++)
    {
        if
(chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+cut13-1]==1)
        {
            value10[13]=value10[13]+pow(2,(m14-cut13));
        }
    }
    for (int cut14=1;cut14<=m15;cut14++)
    {
        if
(chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+cut14-1]==1)
        {
            value10[14]=value10[14]+pow(2,(m15-cut14));
        }
    }
    for (int cut15=1;cut15<=m16;cut15++)
    {
        if
(chromosome[j-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+m15+cut15-1]==1)
        {
            value10[15]=value10[15]+pow(2,(m16-cut15));
        }
    }
    value_flow[0]=Vara1_low+value10[0]*(Vara1_up-Vara1_low)/(pow(2,m1)-1);
    value_flow[1]=Varb1_low+value10[1]*(Varb1_up-Varb1_low)/(pow(2,m2)-1);
    value_flow[2]=Vara2_low+value10[2]*(Vara2_up-Vara2_low)/(pow(2,m3)-1);

```

```

value_flow[3]=Varb2_low+value10[3]*(Varb2_up-Varb2_low)/(pow(2,m4)-1);
value_flow[4]=Vara3_low+value10[4]*(Vara3_up-Vara3_low)/(pow(2,m5)-1);
value_flow[5]=Varb3_low+value10[5]*(Varb3_up-Varb3_low)/(pow(2,m6)-1);
value_flow[6]=Vara4_low+value10[6]*(Vara4_up-Vara4_low)/(pow(2,m7)-1);
value_flow[7]=Varb4_low+value10[7]*(Varb4_up-Varb4_low)/(pow(2,m8)-1);
value_flow[8]=Vara5_low+value10[8]*(Vara5_up-Vara5_low)/(pow(2,m9)-1);
value_flow[9]=Varb5_low+value10[9]*(Varb5_up-Varb5_low)/(pow(2,m10)-1);
value_flow[10]=Vara6_low+value10[10]*(Vara6_up-Vara6_low)/(pow(2,m11)-1);
value_flow[11]=Varb6_low+value10[11]*(Varb6_up-Varb6_low)/(pow(2,m12)-1);
value_flow[12]=Vara7_low+value10[12]*(Vara7_up-Vara7_low)/(pow(2,m13)-1);
value_flow[13]=Varb7_low+value10[13]*(Varb7_up-Varb7_low)/(pow(2,m14)-1);
value_flow[14]=Vara8_low+value10[14]*(Vara8_up-Vara8_low)/(pow(2,m15)-1);
value_flow[15]=Varb8_low+value10[15]*(Varb8_up-Varb8_low)/(pow(2,m16)-1);

//到這裡.....

fitvalue[j-1]=Object_fun(value_flow[0],value_flow[1],value_flow[2],value_flow[3],value_flow[4],value_flow[5],value
_flow[6],value_flow[7],value_flow[8],value_flow[9],value_flow[10],value_flow[11],value_flow[12],value_flow[13],v
alue_flow[14],value_flow[15]);

Sumfitness=Sumfitness+fitvalue[j-1];
if (fitvalue[j-1]>Maxsolute) //取最大值;
{
Maxsolute=fitvalue[j-1];
for (int besta=1;besta<=PchromW;besta++)
{
beststring[besta-1]=chromosome[j-1][besta-1];
}
Table<<"n"<<"第"<<j<<"代"<<"母體\n\n";
Table<<"突破了!";
cout<<"突破了!";
for (int getbest=1;getbest<=NVAR;getbest++)
{
Bestone[getbest-1]=value_flow[getbest-1];
}

Table<<"=="<<value_flow[0]<<","<<value_flow[1]<<","<<value_flow[2]<<","<<value_flow[3]<<","<<value
_flow[4]<<","<<value_flow[5]<<","<<value_flow[6]<<","<<value_flow[7]<<","<<value_flow[8]<<","<<value_flow[
9]<<","<<value_flow[10]<<","<<value_flow[11]<<","<<value_flow[12]<<","<<value_flow[13]<<","<<value_flow[14
]<<","<<value_flow[15]<<".....目標函數值="<<fitvalue[j-1]<<"\n";
}

cout<<"=="<<value_flow[0]<<","<<value_flow[1]<<","<<value_flow[2]<<","<<value_flow[3]<<","<<value_f
low[4]<<","<<value_flow[5]<<","<<value_flow[6]<<","<<value_flow[7]<<","<<value_flow[8]<<","<<value_flow[9]

```

```

<<"<<value_flow[10]<<","<<value_flow[11]<<","<<value_flow[12]<<","<<value_flow[13]<<","<<value_flow[14]<
<","<<value_flow[15]<<.....目標函數值="<<fitvalue[j-1]<<"\n";
        if (j==PchromN)
        {
                cout<<"\n"<<"第"<<i<<"代整個輪盤大小為:"<<Sumfitness<<"\n\n";

                for (int l=1;l<=PchromN;l++)
                {
                        rollPb[l-1]=fitvalue[l-1]/Sumfitness;
                        cout<<"染色體"<<l<<"占整個輪盤的機率大小:"<<rollPb[l-1]<<"\n";
                }

                Sumfitness=0;
                cout<<"\n"<<"轉輪盤複製得到:"<<"\n\n";
        }
}
else
{
        for (int j2=1;j2<=PchromN;j2++)
        {
                double fitvalue2[PchromN];
                double value102[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
                double value_flow2[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
                cout<<"染色體"<<j2<<":   ";

                for (int k2=1;k2<=PchromW;k2++)
                {
                        chromosome[j2-1][k2-1]=mpchromosome[j2-1][k2-1];
                        cout<<chromosome[j2-1][k2-1];
                }

                //增加 Entity 時改這裡.....
                for (int cut02=1;cut02<=m1;cut02++)
                {
                        if (chromosome[j2-1][cut02-1]==1)
                        {
                                {
                                        value102[0]=value102[0]+pow(2,(m1-cut02));
                                }
                        }
                }

                for (int cut12=1;cut12<=m2;cut12++)
                {

```



```

        if (chromosome[j2-1][m1+cut12-1]==1)
        {
            value102[1]=value102[1]+pow(2,(m2-cut12));
        }
    }
for (int cut22=1;cut22<=m3;cut22++)
{
    if (chromosome[j2-1][m1+m2+cut22-1]==1)
    {
        value102[2]=value102[2]+pow(2,(m3-cut22));
    }
}
for (int cut32=1;cut32<=m4;cut32++)
{
    if (chromosome[j2-1][m1+m2+m3+cut32-1]==1)
    {
        value102[3]=value102[3]+pow(2,(m4-cut32));
    }
}
for (int cut42=1;cut42<=m5;cut42++)
{
    if (chromosome[j2-1][m1+m2+m3+m4+cut42-1]==1)
    {
        value102[4]=value102[4]+pow(2,(m5-cut42));
    }
}
for (int cut52=1;cut52<=m6;cut52++)
{
    if (chromosome[j2-1][m1+m2+m3+m4+m5+cut52-1]==1)
    {
        value102[5]=value102[5]+pow(2,(m6-cut52));
    }
}
for (int cut62=1;cut62<=m7;cut62++)
{
    if (chromosome[j2-1][m1+m2+m3+m4+m5+m6+cut62-1]==1)
    {
        value102[6]=value102[6]+pow(2,(m7-cut62));
    }
}

```

```

    }

for (int cut72=1;cut72<=m8;cut72++)
{
    if (chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+cut72-1]==1)
    {
        value102[7]=value102[7]+pow(2,(m8-cut72));
    }
}

for (int cut82=1;cut82<=m9;cut82++)
{
    if (chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+cut82-1]==1)
    {
        value102[8]=value102[8]+pow(2,(m9-cut82));
    }
}

for (int cut92=1;cut92<=m10;cut92++)
{
    if (chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+cut92-1]==1)
    {
        value102[9]=value102[9]+pow(2,(m10-cut92));
    }
}

for (int cut102=1;cut102<=m11;cut102++)
{
    if (chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+cut102-1]==1)
    {
        value102[10]=value102[10]+pow(2,(m11-cut102));
    }
}

for (int cut112=1;cut112<=m12;cut112++)
{
    if
(chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+cut112-1]==1)
    {
        value102[11]=value102[11]+pow(2,(m12-cut112));
    }
}

```

```

for (int cut122=1;cut122<=m13;cut122++)
{
    if
(chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+cut122-1]==1)
    {
        value102[12]=value102[12]+pow(2,(m13-cut122));
    }
}
for (int cut132=1;cut132<=m14;cut132++)
{
    if
(chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+cut132-1]==1)
    {
        value102[13]=value102[13]+pow(2,(m14-cut132));
    }
}
for (int cut142=1;cut142<=m15;cut142++)
{
    if
(chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+cut142-1]==1)
    {
        value102[14]=value102[14]+pow(2,(m15-cut142));
    }
}
for (int cut152=1;cut152<=m16;cut152++)
{
    if
(chromosome[j2-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+m15+cut152-1]==1)
    {
        value102[15]=value102[15]+pow(2,(m16-cut152));
    }
}
value_flow2[0]=Vara1_low+value102[0]*(Vara1_up-Vara1_low)/(pow(2,m1)-1);
value_flow2[1]=Varb1_low+value102[1]*(Varb1_up-Varb1_low)/(pow(2,m2)-1);
value_flow2[2]=Vara2_low+value102[2]*(Vara2_up-Vara2_low)/(pow(2,m3)-1);
value_flow2[3]=Varb2_low+value102[3]*(Varb2_up-Varb2_low)/(pow(2,m4)-1);
value_flow2[4]=Vara3_low+value102[4]*(Vara3_up-Vara3_low)/(pow(2,m5)-1);
value_flow2[5]=Varb3_low+value102[5]*(Varb3_up-Varb3_low)/(pow(2,m6)-1);
value_flow2[6]=Vara4_low+value102[6]*(Vara4_up-Vara4_low)/(pow(2,m7)-1);

```

```

value_flow2[7]=Varb4_low+value102[7]*(Varb4_up-Varb4_low)/(pow(2,m8)-1);
value_flow2[8]=Vara5_low+value102[8]*(Vara5_up-Vara5_low)/(pow(2,m9)-1);
value_flow2[9]=Varb5_low+value102[9]*(Varb5_up-Varb5_low)/(pow(2,m10)-1);
value_flow2[10]=Vara6_low+value102[10]*(Vara6_up-Vara6_low)/(pow(2,m11)-1);
value_flow2[11]=Varb6_low+value102[11]*(Varb6_up-Varb6_low)/(pow(2,m12)-1);
value_flow2[12]=Vara7_low+value102[12]*(Vara7_up-Vara7_low)/(pow(2,m13)-1);
value_flow2[13]=Varb7_low+value102[13]*(Varb7_up-Varb7_low)/(pow(2,m14)-1);
value_flow2[14]=Vara8_low+value102[14]*(Vara8_up-Vara8_low)/(pow(2,m15)-1);
value_flow2[15]=Varb8_low+value102[15]*(Varb8_up-Varb8_low)/(pow(2,m16)-1);

//到這裡.....

fitvalue2[j2-1]=Object_fun(value_flow2[0],value_flow2[1],value_flow2[2],value_flow2[3],value_flow2[4],value_flow
2[5],value_flow2[6],value_flow2[7],value_flow2[8],value_flow2[9],value_flow2[10],value_flow2[11],value_flow2[12
],value_flow2[13],value_flow2[14],value_flow2[15]);

Sumfitness=Sumfitness+fitvalue2[j2-1];

if (fitvalue2[j2-1]>Maxsolute) //取最大值;
{
    Maxsolute=fitvalue2[j2-1];

for (int besta2=1;besta2<=PchromW;besta2++)
{
    beststring[besta2-1]=chromosome[j2-1][besta2-1];
}
Table<<"\n"<<"第"<<i<<"代"<<"母體\n\n";
Table<<"突破了!";
cout<<"突破了!";

for (int getbest2=1;getbest2<=NVAR;getbest2++)
{
    Bestone[getbest2-1]=value_flow2[getbest2-1];
}

Table<<"=="<<value_flow2[0]<<","<<value_flow2[1]<<","<<value_flow2[2]<<","<<value_flow2[3]<<","<<v
alue_flow2[4]<<","<<value_flow2[5]<<","<<value_flow2[6]<<","<<value_flow2[7]<<","<<value_flow2[8]<<","<<va
lue_flow2[9]<<","<<value_flow2[10]<<","<<value_flow2[11]<<","<<value_flow2[12]<<","<<value_flow2[13]<<","<
<value_flow2[14]<<","<<value_flow2[15]<<".....目標函數值="<<fitvalue2[j2-1]<<"\n";
}

cout<<"=="<<value_flow2[0]<<","<<value_flow2[1]<<","<<value_flow2[2]<<","<<value_flow2[3]<<","<<va
lue_flow2[4]<<","<<value_flow2[5]<<","<<value_flow2[6]<<","<<value_flow2[7]<<","<<value_flow2[8]<<","<<val
ue_flow2[9]<<","<<value_flow2[10]<<","<<value_flow2[11]<<","<<value_flow2[12]<<","<<value_flow2[13]<<","<
<value_flow2[14]<<","<<value_flow2[15]<<".....目標函數值="<<fitvalue2[j2-1]<<"\n";

if (j2==PchromN)

```

```

    {
        cout<<"n"<<"第"<<i<<"代整個輪盤大小為:"<<Sumfitness<<"n\n";
        for (int l2=1;l2<=PchromN;l2++)
        {
            rollPb[l2-1]=fitvalue2[l2-1]/Sumfitness;
            cout<<"染色體"<<l2<<"占整個輪盤的機率大小:"<<rollPb[l2-1]<<"n";
        }
        Sumfitness=0;
        cout<<"n"<<"轉輪盤複製得到:"<<"n\n";
    }
}
}
//輪盤的分割;
for (int m=1;m<=PchromN;m++)
{
    if (m==1)
    {
        T_rollPb[m-1]=rollPb[m-1];
    }
    else
    {
        T_rollPb[m-1]=T_rollPb[m-2]+rollPb[m-1];
    }
    Rn_select[m-1]=double(rand())/double(RAND_MAX);
}
//取代複製的動作;
for (int n=1;n<=PchromN;n++)
{
    if (Rn_select[n-1]<=T_rollPb[0])
    {
        for (int o=1;o<=PchromW;o++)
        {
            rpchromosome[n-1][o-1]=chromosome[0][o-1];
        }
    }
    else
    {
        int uoo=2;
        do{

```

```

    if(Rn_select[n-1]>T_rollPb[uoo-2]&&Rn_select[n-1]<T_rollPb[uoo-1])
    {
        for (int p=1;p<=PchromW;p++)
        {
            rpchromosome[n-1][p-1]=chromosome[uoo-1][p-1];
        }
        break;
    }
    uoo++;
    }while(uoo<=PchromN);
}
}
for (int jj=1;jj<=PchromN;jj++)
{
    for (int kk=1;kk<=PchromW;kk++)
    {
        cout<<rpchromosome[jj-1][kk-1];
    }
    cout<<"\n";
}
cout<<"\n"<<"交配後得到:"<<"\n\n";
//crossover 的部分;
for (int ll=1;ll<=PchromN;ll++) //決定要交換的個數;
{
    Rcrossover[ll-1]=double(rand())/double(RAND_MAX);
    if (Rcrossover[ll-1]<pcrossover)
        crossNumber++;
}
for (int mm=1;mm<=PchromN;mm++)
{
    if (Rcrossover[mm-1]<pcrossover)
    {
        Placecross[PcrossNumber]=mm-1; //染色體位置編號;
        PcrossNumber++;
    }
}
R_crossNumber=floor(crossNumber/2);
R_crossNumber=R_crossNumber*2;

```

```

for (int nn=1;nn<=R_crossNumber;nn+=2)
{
//確保位置一二不等;
    Cross_pos1=rand()%PchromW;
do{
    Cross_pos2=rand()%PchromW;
}while(Cross_pos1==Cross_pos2);
//確保位置二大於位置一;
if(Cross_pos1>Cross_pos2)
{
    int temp_position1=Cross_pos1;
    Cross_pos1=Cross_pos2;
    Cross_pos2=temp_position1;
}
//進行交換的動作;
    int temp_position2;
for (int oo=Cross_pos1;oo<=Cross_pos2;oo++)
{
    temp_position2=rpchromosome[nn-1][oo];
    rpchromosome[nn-1][oo]=rpchromosome[nn][oo];
    rpchromosome[nn][oo]=temp_position2;
}
}
for (int jjj=1;jjj<=PchromN;jjj++)
{
    double fitvaluea[PchromN];
    double value10a[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
    double value_flowa[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
    cout<<"染色體"<<jjj<<":   ";
    for (int kkk=1;kkk<=PchromW;kkk++)
    {
        cout<<rpchromosome[jjj-1][kkk-1];
    }
    //增加 Entity 時改這裡.....
    for (int cut0_c=1;cut0_c<=m1;cut0_c++)
    {
        if (rpchromosome[jjj-1][cut0_c-1]==1)
        {
            value10a[0]=value10a[0]+pow(2,(m1-cut0_c));

```

```

    }
}
for (int cut1_c=1;cut1_c<=m2;cut1_c++)
{
    if (rpchromosome[jjj-1][m1+cut1_c-1]==1)
    {
        value10a[1]=value10a[1]+pow(2,(m2-cut1_c));
    }
}
for (int cut2_c=1;cut2_c<=m3;cut2_c++)
{
    if (rpchromosome[jjj-1][m1+m2+cut2_c-1]==1)
    {
        value10a[2]=value10a[2]+pow(2,(m3-cut2_c));
    }
}
for (int cut3_c=1;cut3_c<=m4;cut3_c++)
{
    if (rpchromosome[jjj-1][m1+m2+m3+cut3_c-1]==1)
    {
        value10a[3]=value10a[3]+pow(2,(m4-cut3_c));
    }
}
for (int cut4_c=1;cut4_c<=m5;cut4_c++)
{
    if (rpchromosome[jjj-1][m1+m2+m3+m4+cut4_c-1]==1)
    {
        value10a[4]=value10a[4]+pow(2,(m5-cut4_c));
    }
}
for (int cut5_c=1;cut5_c<=m6;cut5_c++)
{
    if (rpchromosome[jjj-1][m1+m2+m3+m4+m5+cut5_c-1]==1)
    {
        value10a[5]=value10a[5]+pow(2,(m6-cut5_c));
    }
}
for (int cut6_c=1;cut6_c<=m7;cut6_c++)
{

```



```

        if (rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+cut6_c-1]==1)
        {
value10a[6]=value10a[6]+pow(2,(m7-cut6_c));
        }
    }
    for (int cut7_c=1;cut7_c<=m8;cut7_c++)
    {
        if (rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+cut7_c-1]==1)
        {
value10a[7]=value10a[7]+pow(2,(m8-cut7_c));
        }
    }
    for (int cut8_c=1;cut8_c<=m9;cut8_c++)
    {
        if (rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+cut8_c-1]==1)
        {
value10a[8]=value10a[8]+pow(2,(m9-cut8_c));
        }
    }
    for (int cut9_c=1;cut9_c<=m10;cut9_c++)
    {
        if (rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+cut9_c-1]==1)
        {
value10a[9]=value10a[9]+pow(2,(m10-cut9_c));
        }
    }
    for (int cut10_c=1;cut10_c<=m11;cut10_c++)
    {
        if (rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+cut10_c-1]==1)
        {
value10a[10]=value10a[10]+pow(2,(m11-cut10_c));
        }
    }
    for (int cut11_c=1;cut11_c<=m12;cut11_c++)
    {
        if
(rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+cut11_c-1]==1)
        {
value10a[11]=value10a[11]+pow(2,(m12-cut11_c));

```

```

    }
}
for (int cut12_c=1;cut12_c<=m13;cut12_c++)
{
    if
(rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+cut12_c-1]==1)
    {
        value10a[12]=value10a[12]+pow(2,(m13-cut12_c));
    }
}
for (int cut13_c=1;cut13_c<=m14;cut13_c++)
{
    if
(rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+cut13_c-1]==1)
    {
        value10a[13]=value10a[13]+pow(2,(m14-cut13_c));
    }
}
for (int cut14_c=1;cut14_c<=m15;cut14_c++)
{
    if
(rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+cut14_c-1]==1)
    {
        value10a[14]=value10a[14]+pow(2,(m15-cut14_c));
    }
}
for (int cut15_c=1;cut15_c<=m16;cut15_c++)
{
    if
(rpchromosome[jjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+m15+cut15_c-1]==1)
    {
        value10a[15]=value10a[15]+pow(2,(m16-cut15_c));
    }
}
value_flowa[0]=Vara1_low+value10a[0]*(Vara1_up-Vara1_low)/(pow(2,m1)-1);
value_flowa[1]=Varb1_low+value10a[1]*(Varb1_up-Varb1_low)/(pow(2,m2)-1);
value_flowa[2]=Vara2_low+value10a[2]*(Vara2_up-Vara2_low)/(pow(2,m3)-1);
value_flowa[3]=Varb2_low+value10a[3]*(Varb2_up-Varb2_low)/(pow(2,m4)-1);
value_flowa[4]=Vara3_low+value10a[4]*(Vara3_up-Vara3_low)/(pow(2,m5)-1);

```

```

value_flowa[5]=Varb3_low+value10a[5]*(Varb3_up-Varb3_low)/(pow(2,m6)-1);
value_flowa[6]=Vara4_low+value10a[6]*(Vara4_up-Vara4_low)/(pow(2,m7)-1);
value_flowa[7]=Varb4_low+value10a[7]*(Varb4_up-Varb4_low)/(pow(2,m8)-1);
value_flowa[8]=Vara5_low+value10a[8]*(Vara5_up-Vara5_low)/(pow(2,m9)-1);
value_flowa[9]=Varb5_low+value10a[9]*(Varb5_up-Varb5_low)/(pow(2,m10)-1);
value_flowa[10]=Vara6_low+value10a[10]*(Vara6_up-Vara6_low)/(pow(2,m11)-1);
value_flowa[11]=Varb6_low+value10a[11]*(Varb6_up-Varb6_low)/(pow(2,m12)-1);
value_flowa[12]=Vara7_low+value10a[12]*(Vara7_up-Vara7_low)/(pow(2,m13)-1);
value_flowa[13]=Varb7_low+value10a[13]*(Varb7_up-Varb7_low)/(pow(2,m14)-1);
value_flowa[14]=Vara8_low+value10a[14]*(Vara8_up-Vara8_low)/(pow(2,m15)-1);
value_flowa[15]=Varb8_low+value10a[15]*(Varb8_up-Varb8_low)/(pow(2,m16)-1);
//到這裡.....

```

```

fitvaluea[jjj-1]=Object_fun(value_flowa[0],value_flowa[1],value_flowa[2],value_flowa[3],value_flowa[4],value_flowa
[5],value_flowa[6],value_flowa[7],value_flowa[8],value_flowa[9],value_flowa[10],value_flowa[11],value_flowa[12],v
alue_flowa[13],value_flowa[14],value_flowa[15]);

```

```

    if (fitvaluea[jjj-1]>Maxsolute)    //取最大值;
    {
        Maxsolute=fitvaluea[jjj-1];
        for (int bestb=1;bestb<=PchromW;bestb++)
        {
            beststring[bestb-1]=rpchromosome[jjj-1][bestb-1];
        }
        Table<<"\n"<<"第"<<i<<"代"<<"母體\n\n";
        Table<<"交配突破了!";
        cout<<"交配突破了!";
        for (int getbest_c=1;getbest_c<=NVAR;getbest_c++)
        {
            Bestone[getbest_c-1]=value_flowa[getbest_c-1];
        }
    }

```

```

    Table<<"=="<<value_flowa[0]<<","<<value_flowa[1]<<","<<value_flowa[2]<<","<<value_flowa[3]<<","<<v
alue_flowa[4]<<","<<value_flowa[5]<<","<<value_flowa[6]<<","<<value_flowa[7]<<","<<value_flowa[8]<<","<<val
ue_flowa[9]<<","<<value_flowa[10]<<","<<value_flowa[11]<<","<<value_flowa[12]<<","<<value_flowa[13]<<","<<
value_flowa[14]<<","<<value_flowa[15]<<".....目標函數值="<<fitvaluea[jjj-1]<<"\n";

```

```

    }

```

```

    cout<<"=="<<value_flowa[0]<<","<<value_flowa[1]<<","<<value_flowa[2]<<","<<value_flowa[3]<<","<<val
ue_flowa[4]<<","<<value_flowa[5]<<","<<value_flowa[6]<<","<<value_flowa[7]<<","<<value_flowa[8]<<","<<valu
e_flowa[9]<<","<<value_flowa[10]<<","<<value_flowa[11]<<","<<value_flowa[12]<<","<<value_flowa[13]<<","<<v
alue_flowa[14]<<","<<value_flowa[15]<<".....目標函數值="<<fitvaluea[jjj-1]<<"\n";

```

```

}

//mutation 的部分
for (int pp=1;pp<=PchromN;pp++)
{
    double Random_muta[PchromW];
    int Number_muta=0;
    for (int qq=1;qq<=PchromW;qq++)
    {
        Random_muta[qq-1]=double(rand())/double(RAND_MAX);
        if (Random_muta[qq-1]<pmutation)
        {
            if (rpchromosome[pp-1][qq-1]==0)
            {
                mpchromosome[pp-1][qq-1]=1;
            }
            else
            {
                mpchromosome[pp-1][qq-1]=0;
            }
            Number_muta++;
        }
        else
        {
            mpchromosome[pp-1][qq-1]=rpchromosome[pp-1][qq-1];
        }
    }
    if (Number_muta>0)
    {
        cout<<"第"<<pp<<"個染色體發生突變個數"<<Number_muta<<"個\n";
        Number_muta=0;
    }
}

for (int jjjj=1;jjjj<=PchromN;jjjj++)
{
    double fitvalueb[PchromN];
    double value10b[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
    double value_flowb[NVAR]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //增加 Entity 時改這裡
    cout<<"染色體"<<jjjj<<": ";
}

```

```

for (int kkkk=1;kkkk<=PchromW;kkkk++)
{
    cout<<mpchromosome[[]-1][kkkk-1];
}

//增加 Entity 時改這裡.....
for (int cut0_m=1;cut0_m<=m1;cut0_m++)
{
    if (mpchromosome[[]-1][cut0_m-1]==1)
    {
        value10b[0]=value10b[0]+pow(2,(m1-cut0_m));
    }
}

for (int cut1_m=1;cut1_m<=m2;cut1_m++)
{
    if (mpchromosome[[]-1][m1+cut1_m-1]==1)
    {
        value10b[1]=value10b[1]+pow(2,(m2-cut1_m));
    }
}

for (int cut2_m=1;cut2_m<=m3;cut2_m++)
{
    if (mpchromosome[[]-1][m1+m2+cut2_m-1]==1)
    {
        value10b[2]=value10b[2]+pow(2,(m3-cut2_m));
    }
}

for (int cut3_m=1;cut3_m<=m4;cut3_m++)
{
    if (mpchromosome[[]-1][m1+m2+m3+cut3_m-1]==1)
    {
        value10b[3]=value10b[3]+pow(2,(m4-cut3_m));
    }
}

for (int cut4_m=1;cut4_m<=m5;cut4_m++)
{
    if (mpchromosome[[]-1][m1+m2+m3+m4+cut4_m-1]==1)
    {
        value10b[4]=value10b[4]+pow(2,(m5-cut4_m));
    }
}

```

```

    }
}
for (int cut5_m=1;cut5_m<=m6;cut5_m++)
{
    if (mpchromosome[jjjj-1][m1+m2+m3+m4+m5+cut5_m-1]==1)
    {
        value10b[5]=value10b[5]+pow(2,(m6-cut5_m));
    }
}
for (int cut6_m=1;cut6_m<=m7;cut6_m++)
{
    if (mpchromosome[jjjj-1][m1+m2+m3+m4+m5+m6+cut6_m-1]==1)
    {
        value10b[6]=value10b[6]+pow(2,(m7-cut6_m));
    }
}
for (int cut7_m=1;cut7_m<=m8;cut7_m++)
{
    if (mpchromosome[jjjj-1][m1+m2+m3+m4+m5+m6+m7+cut7_m-1]==1)
    {
        value10b[7]=value10b[7]+pow(2,(m8-cut7_m));
    }
}
for (int cut8_m=1;cut8_m<=m9;cut8_m++)
{
    if (mpchromosome[jjjj-1][m1+m2+m3+m4+m5+m6+m7+m8+cut8_m-1]==1)
    {
        value10b[8]=value10b[8]+pow(2,(m9-cut8_m));
    }
}
for (int cut9_m=1;cut9_m<=m10;cut9_m++)
{
    if (mpchromosome[jjjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+cut9_m-1]==1)
    {
        value10b[9]=value10b[9]+pow(2,(m10-cut9_m));
    }
}

for (int cut10_m=1;cut10_m<=m11;cut10_m++)

```

```

        {
            if
(mpchromosome[jjj]-1)[m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+cut10_m-1]==1)
                {
                    value10b[10]=value10b[10]+pow(2,(m11-cut10_m));
                }
        }
        for (int cut11_m=1;cut11_m<=m12;cut11_m++)
        {
            if
(mpchromosome[jjj]-1)[m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+cut11_m-1]==1)
                {
                    value10b[11]=value10b[11]+pow(2,(m12-cut11_m));
                }
        }
        for (int cut12_m=1;cut12_m<=m13;cut12_m++)
        {
            if
(mpchromosome[jjj]-1)[m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+cut12_m-1]==1)
                {
                    value10b[12]=value10b[12]+pow(2,(m13-cut12_m));
                }
        }
        for (int cut13_m=1;cut13_m<=m14;cut13_m++)
        {
            if
(mpchromosome[jjj]-1)[m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+cut13_m-1]==1)
                {
                    value10b[13]=value10b[13]+pow(2,(m14-cut13_m));
                }
        }
        for (int cut14_m=1;cut14_m<=m15;cut14_m++)
        {
            if
(mpchromosome[jjj]-1)[m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+cut14_m-1]==1)
                {
                    value10b[14]=value10b[14]+pow(2,(m15-cut14_m));
                }
        }
    }

```

```

for (int cut15_m=1;cut15_m<=m16;cut15_m++)
{
    if
(mpchromosome[jjjj-1][m1+m2+m3+m4+m5+m6+m7+m8+m9+m10+m11+m12+m13+m14+m15+cut15_m-1]==1)
    {
        value10b[15]=value10b[15]+pow(2,(m16-cut15_m));
    }
}

value_flowb[0]=Vara1_low+value10b[0]*(Vara1_up-Vara1_low)/(pow(2,m1)-1);
value_flowb[1]=Varb1_low+value10b[1]*(Varb1_up-Varb1_low)/(pow(2,m2)-1);
value_flowb[2]=Vara2_low+value10b[2]*(Vara2_up-Vara2_low)/(pow(2,m3)-1);
value_flowb[3]=Varb2_low+value10b[3]*(Varb2_up-Varb2_low)/(pow(2,m4)-1);
value_flowb[4]=Vara3_low+value10b[4]*(Vara3_up-Vara3_low)/(pow(2,m5)-1);
value_flowb[5]=Varb3_low+value10b[5]*(Varb3_up-Varb3_low)/(pow(2,m6)-1);
value_flowb[6]=Vara4_low+value10b[6]*(Vara4_up-Vara4_low)/(pow(2,m7)-1);
value_flowb[7]=Varb4_low+value10b[7]*(Varb4_up-Varb4_low)/(pow(2,m8)-1);
value_flowb[8]=Vara5_low+value10b[8]*(Vara5_up-Vara5_low)/(pow(2,m9)-1);
value_flowb[9]=Varb5_low+value10b[9]*(Varb5_up-Varb5_low)/(pow(2,m10)-1);
value_flowb[10]=Vara6_low+value10b[10]*(Vara6_up-Vara6_low)/(pow(2,m11)-1);
value_flowb[11]=Varb6_low+value10b[11]*(Varb6_up-Varb6_low)/(pow(2,m12)-1);
value_flowb[12]=Vara7_low+value10b[12]*(Vara7_up-Vara7_low)/(pow(2,m13)-1);
value_flowb[13]=Varb7_low+value10b[13]*(Varb7_up-Varb7_low)/(pow(2,m14)-1);
value_flowb[14]=Vara8_low+value10b[14]*(Vara8_up-Vara8_low)/(pow(2,m15)-1);
value_flowb[15]=Varb8_low+value10b[15]*(Varb8_up-Varb8_low)/(pow(2,m16)-1);

//到這裡.....

fitvalueb[jjjj-1]=Object_fun(value_flowb[0],value_flowb[1],value_flowb[2],value_flowb[3],value_flowb[4],value_flowb[5],value_flowb[6],value_flowb[7],value_flowb[8],value_flowb[9],value_flowb[10],value_flowb[11],value_flowb[12],value_flowb[13],value_flowb[14],value_flowb[15]);

if (fitvalueb[jjjj-1]>Maxsolute)    //取最大值;
{
    Maxsolute=fitvalueb[jjjj-1];

    for (int bestc=1;bestc<=PchromW;bestc++)
    {
        beststring[bestc-1]=mpchromosome[jjjj-1][bestc-1];
    }

    Table<<"\n"<<"第"<<i<<"代"<<"母體\n\n";
    Table<<"突變突破了!";
    cout<<"突變突破了!";
}

```



```

        for (int getbest_m=1;getbest_m<=NVAR;getbest_m++)
        {
            Bestone[getbest_m-1]=value_flowb[getbest_m-1];
        }

        Table<<"="<<value_flowb[0]<<","<<value_flowb[1]<<","<<value_flowb[2]<<","<<value_flowb[3]<<","<<v
alue_flowb[4]<<","<<value_flowb[5]<<","<<value_flowb[6]<<","<<value_flowb[7]<<","<<value_flowb[8]<<","<<va
lue_flowb[9]<<","<<value_flowb[10]<<","<<value_flowb[11]<<","<<value_flowb[12]<<","<<value_flowb[13]<<","<
<value_flowb[14]<<","<<value_flowb[15]<<".....目標函數值="<<fitvalueb[jjjj-1]<<"\n";
        }

        cout<<"="<<value_flowb[0]<<","<<value_flowb[1]<<","<<value_flowb[2]<<","<<value_flowb[3]<<","<<va
lue_flowb[4]<<","<<value_flowb[5]<<","<<value_flowb[6]<<","<<value_flowb[7]<<","<<value_flowb[8]<<","<<va
lue_flowb[9]<<","<<value_flowb[10]<<","<<value_flowb[11]<<","<<value_flowb[12]<<","<<value_flowb[13]<<","<
<value_flowb[14]<<","<<value_flowb[15]<<".....目標函數值="<<fitvalueb[jjjj-1]<<"\n";
    }

    //結束 mutation
}

end=time(NULL);
Table<<"\n"<<"共花了"<<difftime(end,start)<<"秒.\n";
Table<<"\n"<<"最佳解為"<<Maxsolute<<"\n";
Table<<"\n"<<"最佳字串為.";
cout<<"\n"<<"共花了"<<difftime(end,start)<<"秒.\n";
cout<<"\n"<<"找到的最佳解為"<<Maxsolute<<"\n";
cout<<"\n"<<"找到的最佳字串為.";

for (int bestd=1;bestd<=PchromW;bestd++)
{
    Table<<beststring[bestd-1];
    cout<<beststring[bestd-1];
}

Table.close();
}

//目標函數
double Object_fun(double a1,double b1,double a2,double b2,double a3,double b3,double a4,double b4,double
a5,double b5,double a6,double b6,double a7,double b7,double a8,double b8)
{
    double Obj1,Obj2,Obj3,bb1,bb2,bb3,bb4,bb5,bb6,bb7,bb8,cc1,cc2,cc3,cc4,cc5,cc6,cc7,cc8;

    Obj1=1-(b1/a1*b2/a2*b3/a3+b4/a4+b5/a5*b6/a6*b7/a7*b8/a8);
    bb1=973286.144487441+0.000106632269624698/pow((1/a1),2.7224); //參數變時要改
    bb2=875253.236168306+0.000143229775819721/pow((1/a2),2.6803); //參數變時要改

```

```

bb3=875728.687792822+0.000114385218051994/pow((1/a3),2.7179); //參數變時要改
bb4=1313531.84768831+0.000132374487580487/pow((1/a4),2.5968); //參數變時要改
bb5=958553.747741483+0.000100540641606103/pow((1/a5),2.7447); //參數變時要改
bb6=826532.937373873+0.0001208925534073/pow((1/a6),2.7769); //參數變時要改
bb7=660434.372677493+0.000128234150368538/pow((1/a7),2.8563); //參數變時要改
bb8=812979.143765579+0.000122179122897449/pow((1/a8),2.8347); //參數變時要改

cc1=350102-749.85*(b1-80); //參數變時要改
cc2=341271-549.95*(b2-65); //參數變時要改
cc3=334936-599.9*(b3-70); //參數變時要改
cc4=405736-178.107142857143*(b4-2); //參數變時要改
cc5=296436-499.8*(b5-60); //參數變時要改
cc6=291972-500.8*(b6-50); //參數變時要改
cc7=286103-479.8*(b7-35); //參數變時要改
cc8=290038-460.1*(b8-40); //參數變時要改

Obj2=bb1+bb2+bb3+bb4+bb5+bb6+bb7+bb8+cc1+cc2+cc3+cc4+cc5+cc6+cc7+cc8;
Obj3=Obj1/Obj2;
return Obj3;
}

```