# Chapter 10

# 基因演算法與基因規劃法

演化 (**evolution**) 是一個改變生物族群基因組成的過程。**天擇** (**natural selection**) 是一個過程，在這過程中，擁有適應環境壓力特徵之生物的存活與繁衍數量會遠大於其他類似生物。因此，其後的子代擁有這些好的特徵的比例會升高。

**演化式演算** (**evolutionary computation**) 致力於找一些問題的近似解法，例如：利用天擇的演化過程來解最佳化問題。演化式演算涵蓋了四大領域：基因演算法 (genetic algorithm)、基因規劃法 (genetic programming)、演化式規劃 (evolutionary programming) 和演化策略 (evolutionary strategies)。前兩者會在此章節詳述，首先，我們簡略地回顧遺傳學，以便探討這些演算法。

## • 10.1 遺傳學回顧

此篇回顧是基於讀者已經閱讀過遺傳學教材為前提。對於遺傳學的介紹請參閱 *An Introduction to Genetic Analysis* (Griffiths et al.,2007) 或是 *Essential Genetics* (Hartl and Jones, 2006)。

**生物** (**organism**) 是指一個生命的個體，如植物或動物。**細胞** (**cell**) 是構築生物的基本架構。**染色體** (**chromosome**) 夾帶著遺傳特徵。**基因組** (**genome**) 是在生物中一組完整的染色體，人類的基因組有 23 個染色體。一個**單倍數細胞** (**haploid cell**) 包含了一基因組，也可稱其包含了一組染色體。所以一個人類的單倍數細胞包含了 23 個染色體。二

**倍數細胞** (**diploid cell**) 則是包含了兩個基因組 ; 也可稱其包含了兩組染色體。在二倍數細胞中每個染色體都會和另一基因組內的一個染色體形成配對，我們稱這樣的染色體配對為**同源配對** (**homologous pair**)，每個配對中的染色體稱為**同源基因** (**homolog**)。因此，一個人類二倍數細胞包含了 $2 \times 23 = 46$ 個染色體，且每個同源基因遺傳自父母代。

**體細胞** (**somatic cell**) 是有機體內眾多細胞之一。**單倍體生物** (**haploid organism**) 是擁有單倍數體細胞之生物 ; 而**二倍體生物** (**diploid organism**) 是擁有二倍數體細胞之生物。人類屬於二倍體生物。

**配子** (**gamete**) 是成熟的有性生殖細胞，並且會和其他配子結合成**受精卵** (**zygote**) 進而成長成一個新的個體。配子必定是單倍體細胞，雄性生物生成的配子稱為**精子** (**sperm**); 雌性生物則為**卵子** (**egg**)。**生殖細胞** (**germ cell**) 是配子的前體而且為二倍體細胞。

在二倍體生物中每個成熟個體會生成配子、兩個配子結合成受精卵，最後成長成一個新的個體。以上過程稱之為**有性生殖** (**sexual reproduction**)。單倍體單細胞生物 (Unicellular haploid organism) 通常藉由**二元分裂法** (**binary fission**) 進行無性生殖，過程中單一生物個體逕行分裂成兩個生物個體。因此每個生物個體擁有和原先個體一樣的基因組成。另外一些單倍體單細胞生物會以**細胞融合** (**fusion**) 的方式進行有性生殖，首先兩個細胞會結合成一個**過渡二倍數母細胞** (**transient diploid meiocyte**)，這個母細胞包含一對同源染色體，每個染色體分別來自雙親之一。每個子代可由雙親之一取得一條同源，因此子代的基因並不完全與雙親相同。例如 : 若一生物包含 3 個染色體，則生成的子代會有 $23 = 8$ 種不同的染色體組合排列。

染色體由去氧核醣核酸構成 (deoxyribonucleic acid, DNA)，而 DNA 再由 4 種**核苷酸** (**nucleotide**) 組成。每種核苷酸包含有五碳糖 (pentose sugar)( 去氧核醣 (deoxyribose))、磷酸 (phosphate group) 和嘌呤鹼基或嘧啶鹼基。兩種**嘌呤** (**purines**)，包含腺嘌呤 (adenine，簡稱 A) 和鳥嘌呤 (guanine，簡稱 G)，有相似的結構。兩種**嘧啶** (**pyrimidines**)，包含胞嘧啶 (cytosine，簡稱 C) 和胸腺嘧啶 (thymine，簡稱 T)，有相似的結構。DNA 是由兩個互補股 (complementary strands) 所構成的大分子，每個互補股再由一序列的核苷酸組成。兩股之間的核苷酸會由氫鍵鍵結成一對核苷酸，其中腺嘌呤必定和胸腺嘧啶鍵結而鳥嘌呤必定和胞嘧啶鍵結。每個核苷酸配對稱之為**鹼基對** (**canonical base pair**)，而腺嘌呤、鳥嘌呤、胞嘧啶和胸腺嘧啶稱為**鹼基** (**base**)。

圖 10.1 描繪了 DNA 片段，讀者可能會回想起生物課程上那兩股呈現右旋雙螺旋糾纏一起的樣子，但是對於計算方面上我們只需要將它們視為字串即可。


圖 10.1 DNA 的某一段

**基因** (**gene**) 是一個染色體的片段，通常由數以千計的鹼基對組成；不同基因其構成的鹼基對數目也會巨大的差異，基因同時負責掌控生物的構築 (structure) 和運行 (process)。一個生物的**基因型** (**genotype**) 是其基因的組成，然而**表現型** (**phenotype**) 是生物在環境和基因型的交互影響下所形成的外表。

**對偶基因** (**allele**) 是有多個不同基因源自於單一基因的統稱，通常由突變造成，對偶基因是造成遺傳變異的因素。

**• 範例 10.1** 已知控制人類眼睛顏色的基因為第 15 對染色體上的 bey2 基因。bey2 基因有兩種對偶基因，一種是讓我們有藍色眼睛稱為 BLUE，另一種是棕色眼睛稱為 BROWN。就如同其他基因的情形，一個人由父方母方各得到一個 bey2 對偶基因。BLUE 基因是**隱性基因** (**recessive**)；而 BROWN 是**顯性基因** (**dominant**)。也就是說一個人若遺傳到一個 BLUE 和 BROWN 的對偶基因，則那個人的眼睛顏色將會是棕色。若是想要有藍色眼睛的話會需要遺傳到兩個 BLUE 對偶基因，這樣眼睛顏色才會是藍色的。

人類的配子有 23 個染色體，每染色體可能源自任兩基因組之一，因此人類有 $2^{23} = 8{,}388{,}608$ 種不同的基因組合可以遺傳給下一代。但是實際上的數目會高於剛才的計算，因為在細胞進行**減數分裂** (**meiosis**) 時會染色體會自行複製並且和同源基因對齊，這些染色體的複製品稱為**染色分體** (**chromatids**)。通常同源染色分體之間會交換對應區塊的基因組成，這樣的動作我們稱為**互換** (**crossing-over**)，如圖 10.2 示。

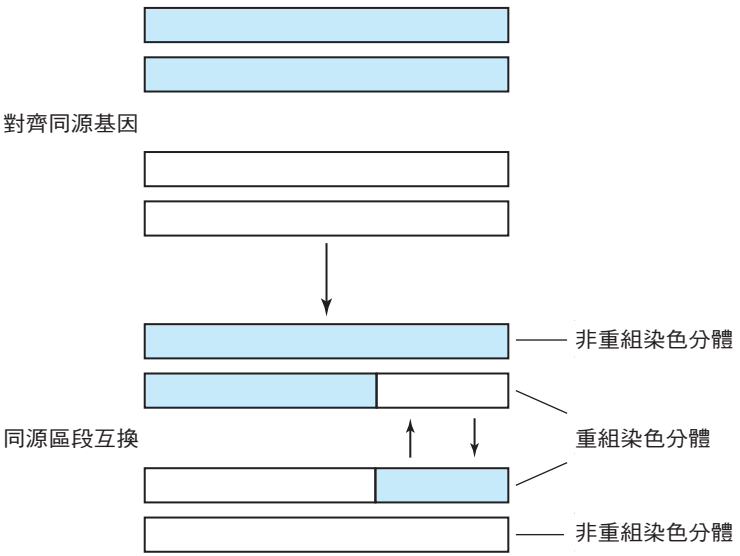


圖 10.2 解說互換過程

而有時當細胞分裂過程中錯誤可能會發生於 DNA 複製時，我們稱這樣的錯誤為**突變** (**mutation**)，突變可能發生在體細胞或生殖細胞。人們大多認為發生於生殖細胞的突變是造成演化上差異的緣由，而體細胞的突變卻只會影響到個體本身 ( 例如 : 癌症 )。

突變大致可分三類 : **取代** (**substitution**)、**插入** (**insertion**) 和**刪除** (**deletion**)。取代是把 DNA 序列某一核苷酸替換成其他的，插入是將 DNA 片段加入至一染色體中，而刪除是將 DNA 片段從一染色體中刪去。

**演化** (**evolution**) 是一個改變生物族群基因組成的過程，人們大多認同突變是致使生物基因組成改變的原因。**天擇** (**natural selection**) 讓擁有環境適應特徵 ( 如躲避掠食者、適應氣候變遷、食物或是交換的競爭 ) 之生物生存與繁衍其子代，進而使擁有適合特徵的生物得以增長，因此天擇可以讓適合環境的對偶基因出現的頻率增加。這種對偶基因出現頻率因機率而改變的現象我們稱之為**遺傳漂變** (**genetic drift**)，目前在科學界中天擇和遺傳漂變何者對於演化有更大的影響仍是無定論的 (Li, 1997)。

## 10.2 基因演算法

在此小節我們會先介紹基本的基因演算法，然後提供它的兩個應用。

### 10.2.1 演算法

從單倍體生物得到啟發，基因演算法利用細胞融合的概念建構演算法模型。每種問題的候選解由一個單倍體個體 (**individual**) 所代表。這邊的染色體構成不像生物那樣由四種不同 DNA (A, G, C 和 T) 組成，而是由組成解答的字母所構成。在每一世代中，只讓部份較符合最佳解答的個體繁衍新個體出來成為下一世代。對應到較佳解之個體被認為更加合適。由兩個較合適個體選出的染色體接著排好並透過互換交換基因內容 ( 問題解之子字串 )。此外，有可能發生突變。這影響了個體的下一代。以上過程會反覆執行直到達成特定條件。以下為上述基因演算法的虛擬程式碼：

```
void generate_populations ( )
{
    t=0;
    初始化族群 P0
    repeat
        評估族群 Pt 中每個個體 ;
        根據適應度 (fitness) 選擇個體 ;
        在選到的這些個體上執行交換與突變 ;
        t++;
    until 終止條件達到 ;
}
```

當根據適應度選擇個體時，我們不見得只會選最「適應」的個體。實際上，我們會同時採用「利用」(**exploitation**) 與「探索」(**exploration**) 兩種策略。一般來說，在評估要從搜尋空間中的各塊候選區域選哪塊走下去時，若採用「利用」的策略，表示我們透過集中心力於看起來較好的區域來利用已經獲得的知識。若採用「探索」的策略，表示我們不管區域目前看起來好不好，就是去探索新的區域。在選擇個體時，若要同時執行這兩種策略，我們可以設定 $\varepsilon$ 機率執行「探索」，也就是隨機選擇一個個體；我們可以設定剩下的 $1-\varepsilon$ 機率執行「利用」，也就是選擇適應度較高的個體。

### 10.2.2 範例演示

假設我們想找到讓下列函數值為極大值的 $x$ 值

$$f(x) = \sin\left(\frac{x\pi}{256}\right) \qquad \text{且} \qquad 0 \le x \le 255$$

其中 $x$ 為整數。當然我們從高中數學就學到了正弦函數在 $\pi/2$ 時會得極大值為 1，也就可以推得 $f(x)$ 的極大值發生於 $x = 128$。因此，現實上我們其實沒有必要用設計演算法來解上面的問題。我們只是為了解釋基因演算法的各種不同面向。我們會用以下步驟來開發基因演算法。

1. 選擇代表答案的字母。由於候選答案僅可能是在 0 至 255 區間的整數，因此我們以 8 個位元來表示每一個候選答案。例如 189 可以用 2 進位表示成：

   1 0 1 1 1 1 0 1

2. 決定一個族群裡面有多少個體，通常會包含有數以千計的個體。在這個範例我們決定只包含 8 個個體。

3. 決定如何初始化整個族群，通常會以隨機的方式產生。在此範例我們會隨機產生 8 個介於 0 至 255 間的數字。表 10.1 列出一些可能的初始值。

4. 決定如何評量個體的適應度 (fitness)，由於我們的目標是找到函數 $f(x) = \sin(x\pi/256)$ 的極大值，因此可直接採用 $f(x)$ 的數值做為 $x$ 的適應度。

5. 根據利用和探索兩個策略來決定哪些個體得以繁衍，適應度以所有個體的適應度總和來正規化，全部個體正規後的適應度總合為 1。之後我們利用正規化後的適應度來計算累計適應度。這些累計適應度就猶如輪盤上的楔子，例如表 10.1 中第二個體正規後的適應度是 .093 那麼它的楔子區間就介於 (0.144, 0.237] 且寬度是 .093。當我們隨機產生一個介於 (0,1] 之間的數字時，那個數字必定會落於某一個體的楔子區間，最後我們就選擇那個個體。我們會重複以上動作 8 次。假設我們選擇要進行繁衍的個體如表 10.2 所示。特別注意的是，同一個體可以被多次的選擇到，個體被選到的機率是取決於其適應度的。

• 表 10.1 初始之個體族群及每個個體的適應度

| 個體 | $x$ | $f$(x) | Normed $f(x)$ | Cumulative Normed $f(x)$ |
|---|---|---|---|---|
| 1 0 1 1 1 1 0 1 | 189 | .733 | .144 | .144 |
| 1 1 0 1 1 0 0 0 | 216 | .471 | .093 | .237 |
| 0 1 1 0 0 0 1 1 | 99 | .937 | .184 | .421 |
| 1 1 1 0 1 1 0 0 | 236 | .243 | .048 | .469 |
| 1 0 1 0 1 1 1 0 | 174 | .845 | .166 | .635 |
| 0 0 1 0 0 0 1 1 | 74 | .788 | .155 | .790 |
| 0 0 1 0 0 0 1 1 | 35 | .416 | .082 | .872 |
| 0 0 1 1 0 1 0 1 | 53 | .650 | .128 | 1.000 |

6. 決定如何進行交換和突變，首先我們隨機地將個體配對成 4 對，在每對個體中隨機選擇兩點，並且將兩點範圍內的位元互換。表 10.3 中列出幾種可能結果。如果第二點出現在第一點的左邊的話，我們會以個體頭尾纏繞一起的方式互換，表 10.3 第三配對展示了這樣的範例。根據表 10.1 和 10.3，在互換執行前的平均適應度為 0.635，而執行互換後提昇到了 0.792。同時互換後讓兩個體的適應度高於 0.99。

• 表 10.2　被選到可繁衍的個體

| 個體 |
|---|
| 0 1 1 0 0 0 1 1 |
| 0 0 1 1 0 1 0 1 |
| 1 1 0 1 1 0 0 0 |
| 1 0 1 0 1 1 1 0 |
| 0 1 0 0 1 0 1 0 |
| 1 0 1 0 1 1 1 0 |
| 0 1 1 0 0 0 1 1 |
| 1 0 1 1 1 1 0 1 |

• 表 10.3　由互換產生的親代與子代

| 親代 | 子代 | $x$ | $f(x)$ |
|---|---|---|---|
| $0\ 1\ 1^{1}\vert\mathbf{0\ 0\ 0}\vert\ ^{2}1\ 1$<br>$0\ 0\ 1\ \vert\mathbf{1\ 0\ 1}\vert\ 0\ \ 1$ | $0\ 1\ 1^{1}\vert\mathbf{1\ 0\ 1}\vert\ ^{2}1\ 1$<br>$0\ 0\ 1\ \vert\mathbf{0\ 0\ 0}\vert\ 0\ \ 1$ | 119<br>33 | .994<br>.394 |
| $1^{1}\vert\mathbf{1\ 0\ 1\ 1}\vert\ ^{2}0\ 0\ 0$<br>$1\ \vert\mathbf{0\ 1\ 0\ 1}\vert\ \ 1\ 1\ 0$ | $1^{1}\vert\mathbf{0\ 1\ 0\ 1}\vert\ ^{2}0\ 0\ 0$<br>$1\ \vert\mathbf{1\ 0\ 1\ 1}\vert\ \ 1\ 1\ 0$ | 168<br>222 | .882<br>.405 |
| $\mathbf{0\ 1}\vert^{2}0\ 0\ 1\ 0\ 1^{1}\vert\ 0$<br>$\mathbf{1\ 0}\vert 1\ \ 0\ 1\ 1\ 1\ \vert\ 0$ | $\mathbf{1\ 0}\vert^{2}0\ 0\ 1\ 0\ 1^{1}\vert\ \mathbf{0}$<br>$\mathbf{0\ 1}\vert\ 1\ 0\ 1\ 1\ 1\ \vert\ \mathbf{0}$ | 138<br>110 | .992<br>.976 |
| $0\ 1\ 1\ 0\ 0^{1}\vert\mathbf{0\ 1\ 1}\vert^{2}$<br>$1\ 0\ 1\ 1\ 1\ \vert\mathbf{1\ 0\ 1}\vert$ | $0\ 1\ 1\ 0\ 0^{1}\vert\mathbf{1\ 0\ 1}\vert^{2}$<br>$1\ 0\ 1\ 1\ 1\ \vert\mathbf{0\ 1\ 1}\vert$ | 101<br>187 | .946<br>.749 |

接下來要決定如何執行突變。我們會對於個體裡每個位元隨機的決定翻轉與否，這邊的翻轉示將 0 轉變成 1 或是將 1 轉變成 0。突變的機率通常介於 0.01 至 0.001 之間。

7. 決定終止演算法的條件，我們可以設定滿足複數或單一條件就讓演算法終止。例如超過一定世代數目、超出時間限制或是最大適應度到達門檻值。在這邊的範例我們可以指定當世代超過 10,000 代時或是適應度超過 0.999 時就終止演算法。

值得一提的是第 2、3、5 和 7 步驟所提及的策略對於很多問題都能適用。

### • 10.2.3 旅行業務員問題

旅行業務員問題 (Traveling Salesperson Problem, TSP) 是個廣為人知的 NP-hard 問題，**NP-hard 問題**是一類問題尚未有人能提出多項式時間的演算法，同時也沒人能夠證明這樣的演算法不無可能。

假設一名業務員計畫要到 $n$ 個城市出差，每個城市之間都有一些道路可以通往其他城市。為了減省旅行時間，我們希望找一條最短路線能夠從起始城市出發，途中經過每個城市恰好一次且最後再回到起始城市。要找到這樣最短的旅行路線就是旅行業務員問題，以下簡稱為 **TSP**。另外讀者要注意到起始城市和最後的最短旅行路線是不相關聯的。

我們可以將 TSP 以**加權有向圖** (**weighted directed graph**) 表示，其中**節點**表示城市，**邊**的權重表示道路的長度。一般來說 TSP 的圖不需要是個**完全圖** (**complete graph**)，也就是說每個城市之間不一定會有道路存在。而且若某兩城市間有道路往返，則往程與返程的權重也未必相同。除了運用在交通排程上之外，TSP 也可同時被用在電路板上鑿洞的排程和 DNA 定序。

接下來我們會展示三個套用基因演算法的 TSP 解法。

#### • 序列互換

我們第一個先介紹序列互換 (**order crossover**)，並且只列出和 10.2.2 小段不同的步驟。

• 表 10.4 序列互換

| 親代 | 另一個親代之模版 | 子代 |
|---|---|---|
| $p_1$ : 2 1 $3^1$ **9 5 4** $^2$8 7 6<br>$p_2$ : 5 3 2 **6 8 9** 7 1 4 | 6 8 7 1 3 2 from $p_2$<br>5 4 7 2 1 3 from $p_1$ | $c_1$ : 6 8 $7^1$ **9 5 4** $^2$1 3 2<br>$c_2$ : 5 4 7 **6 8 9** 2 1 3 |

1. 決定代表答案的字母。一個 TSP 解法的直觀表示法是將所有節點標註 1 到 $n$，然後依序列出要走訪節點的順序。例如 : 若圖存在 9 個節點，則 [2 1 3 9 5 4 8 7 6] 表示我們走訪節點 1 之後會到節點 2，…，走訪節點 6 之後會回到節點 2。

2. 這裡的適應度我們以旅行路線的長度來表示，代表長度越短的旅行路線越符合期待。

3. 決定如何執行互換和突變。如同之前介紹的一樣，我們隨機將個體配對並且隨機決定兩點，這邊我們將兩點間的區塊稱為 **pick**。我們必須確保互換後的旅行路線是合理的，也就是每個城市僅能出現在序列中一次，因此若僅是互換 pick 是不可行的。在**序列互換**中，孩子的 pick 和其中一個雙親的 pick 值是相同的。而非 pick 區域的值則由另一個雙親而來，由另一個雙親的 pick 開始，跳過已經出現的值，依序填入。這些值稱為另一個雙親的**模版 (template)**。表 10.4 展示了一個範例可供理解。請注意到孩子 $c_1$ 與其雙親 $p_1$ 均選定 [9 5 4] 為 pick，而雙親 $p_2$ 的 pick 為 [6 8 9]。雙親 $p_2$ 的樣板是從 6 開始尋找不包含 [9 5 4] 的數值，過程中經過序列末端則再由前端開始至 pick 開頭為止。因此得到最後樣板為 [6 8 7 1 3 2]，之後這些數值便依序被填入子代 $c_2$ 非 pick 區域中。

   若此有向圖並非完整圖，則我們必須檢查旅行路線是否真實存在於此圖中，若旅行路線不存在則否決互換的結果。

   至於突變的話我們也不能貿然改變序列中某一節點，因為那樣會造成存在兩個相同節點的狀況。然而我們可以藉由交換序列中兩個節點，或是顛倒某一小段子序列順序的方式來達成突變的結果。同樣的，若此圖並非完整的話，我們也必須要檢查旅行路線的存在與否。

### • 近鄰互換

**近鄰演算法 (Nearest Neighbor Algorithm, NNA)** 是解決 TSP 的一個貪婪演算法。NNA 首先隨機挑一個節點開始，並且反覆的加入近鄰中未被探索的節點直到完成整個旅行路線。和序列互換不同的是近鄰互換演算法假設此有向圖是完整的，否則 NNA 可能找不到一個旅行路線解法。其演算法如下式：

**▶ 演算法 10.1** **近鄰演算法用於 TSP 問題**

**問題：**為一個加權無向圖 (weighted undirected graph) 決定最佳旅行路線找到最佳旅行路線，其中每邊的權重均為非負數值。

**輸入：**加權無向圖 $G$，與其節點個數 $n$。

**輸出：**圖 $G$ 中節點的排序列表。

```
void generate_tour (int n, graph G, orderedlist& tour)
{
    tour = [v_i] 其中 v_i 為 G 中隨機選出的點 ;
    repeat
        新增一個離目前 tour 最後一個頂點最近的未拜訪頂點到 tour 中
    until tour 包含 n 個頂點
}
```

圖 10.3(a) 中展示了一個 TSP 的範例，其中無向邊代表其邊的兩端節點均有權重相等之有向邊指向對方。圖 10.3(b) 顯示了一個最短旅行路線，且圖 10.3(c) 展示了以 $v_4$ 當起使節點套用 NNA 得到的旅行路線，而圖 10.3(d) 則是以 $v_1$ 當起始點套用 NNA 得到的最佳旅行路線。

之後，我們將介紹**近鄰互換 (Nearest Neighbor Crossover, NNX)**，由 Süral 等人於 2010 年開發的。接著在下列介紹演算法中的各個步驟。

1. 選擇字母來表示 TSP 的解答，在這步驟中我們以和 10.2.3 小節中的序列互換方式一樣。
2. 決定族群由多少個體所組成，這裡我們分別決定 50 和 100。
3. 第三步驟有兩個技巧 : 一是在初始族群時採用完全隨機的方式，二是混和一半族群採用隨機，另一半採用 NNX 搭配貪婪邊演算法 ( 會於之後介紹 )。
4. 決定如何評量適應度，方法和序列互換一樣。
5. 決定哪些個體要選來繁衍子代。我們決定選擇適應度前 50% 的個體。明確的說是將這些個體每個複製四份組成 pool 再以不重覆抽樣的方式挑兩兩一對的個體繁衍子代。

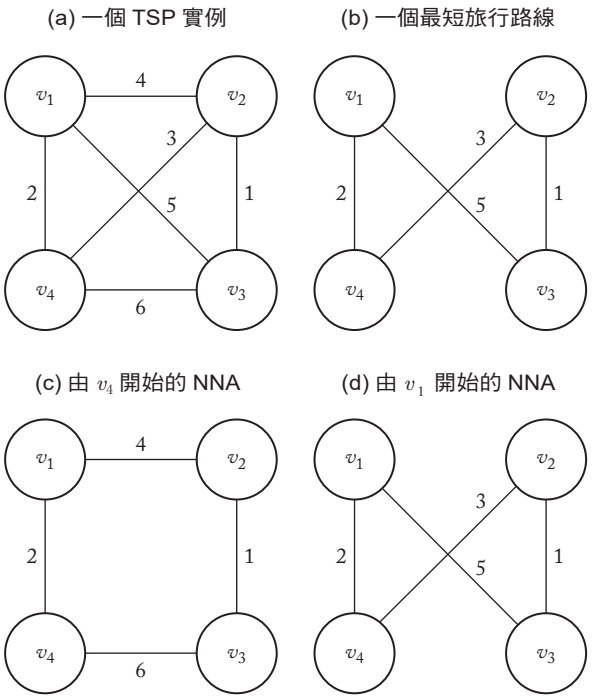


圖 10.3 以一個 TSP 的實例來說明 NNA

6. 決定如何執行交換與突變。在 NNX 中，一對親代只會產生一個子代，雖然和互換的定義有異但是我們還是認為這樣是互換的一種。首先將兩個親代的圖的邊聯集成一個新的圖，這個圖包含了兩個親代所有的邊，如圖 10.4。再來套用 NNA 到聯集的圖上，又如圖 10.4 這邊我們挑選 $v_6$ 當作起始節點，之後會得到比任一個親代更好的子代。現在我們也必須呈現若我們由 $v_1$ 開始，這樣的好結果就不會產生。若我們由 v3 開始，之後會到 $v_1$ 形成死結而沒有旅行路線產生。若選定的節點不能產生旅行路線的話，我們可以嘗試其他節點直到有旅行路線為止。若由任一節點開始，皆無法產生旅行路線，我們可以將其他在完整圖上的邊也加入考慮。

   值得注意的是，每位親代會拷貝四份，一對親代會產生一個子代，因此子代的數量會是可生育親代的兩倍。然由於只有一半的親代可生育，因此每一世代的族群大小是相等的。

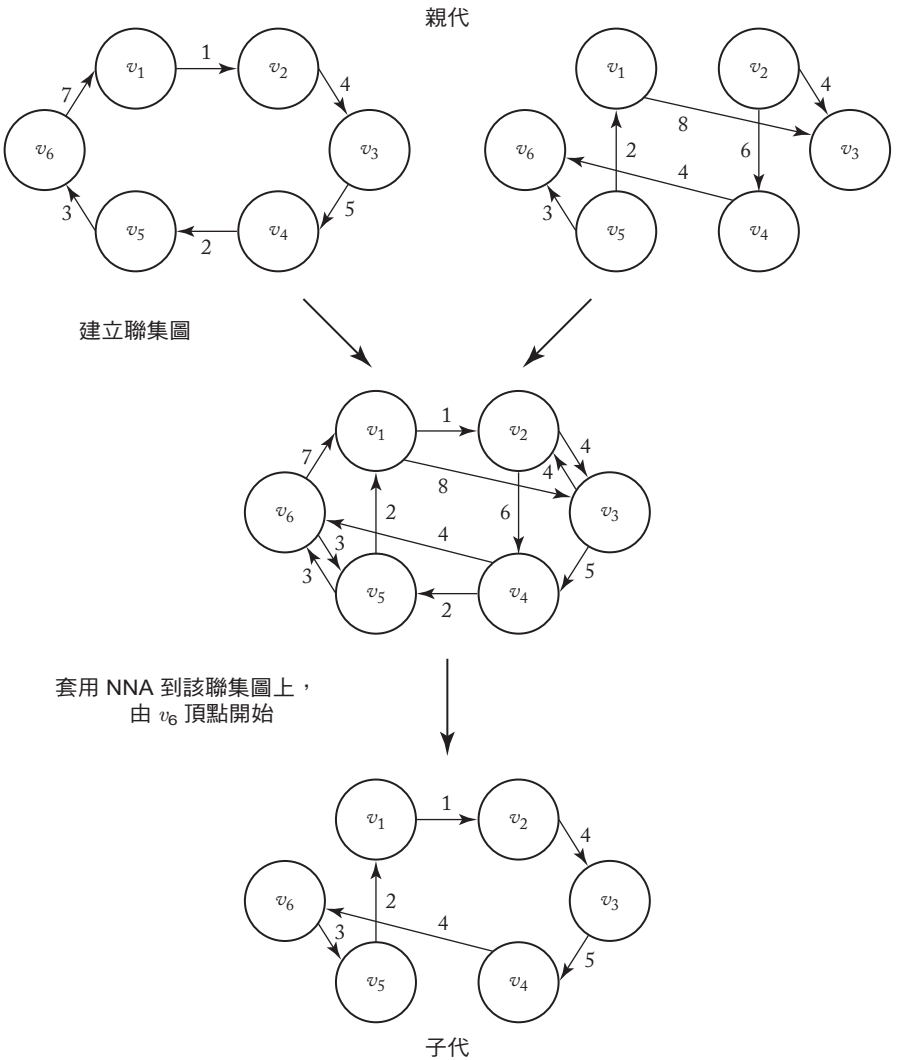


圖 10.4　建立聯集圖，接著將 NNA 套用到該聯集圖上，由 $v_6$ 頂點開始

突變則是藉由隨機選擇二節點，然後將包含這兩節點的子圖倒轉如圖 10.5，在上圖範例中我們選擇 $v_1$ 和 $v_7$。在進行突變時會有兩種選擇 : 其中 **M1** 只發生在最好的子代，而 **M2** 則會發生在所有子代。

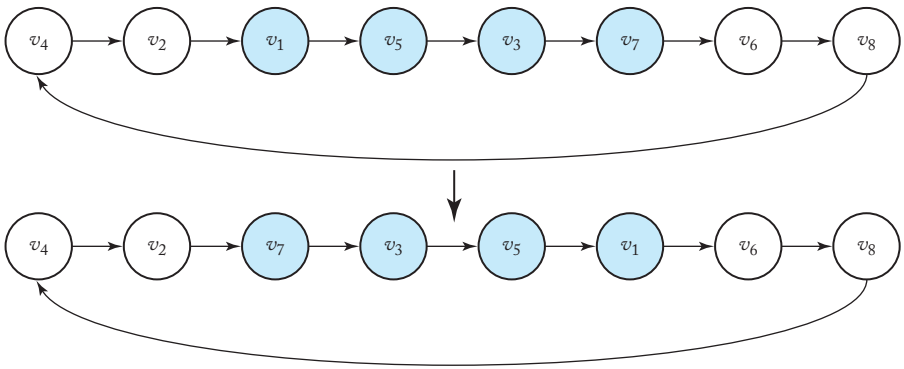


圖 10.5 將連接頂點 $v_1$ 到 $v_7$ 的子旅行路線倒轉所產生的一種突變

另外有一隨機版本的 NNX，會根據機率，由所有入射目前頂點的邊中挑選一個，而機率是以邊的長度成反比。藉由這樣的方法可以提升族群的多樣性，同時增加了找尋答案的搜索空間。然而 Süral 等人在 2010 年發現此方法比決定性的方法還要糟，因此他們並未將這個方法納入到最後的試驗中。

7. 最後當連續兩世代的平均適應度保持不變或是到達 500 世代時，便終止演算法。

NNA 和 NNX 非常類似於解決最短路徑問題的 Dijkstra 演算法，而且時間複雜度也是 $\theta(n^2)$，其中 $n$ 為節點個數。

### • 貪婪邊互換

在**貪婪邊演算法 (Greedy Edge Algorithm, GEA)** 中，我們首先將邊以非遞減順序排列，並且從第一個邊開始增加到旅行路線中，同時確保各個節點最多只和兩個邊相連，且不存在節點數小於 $n$ 的迴圈。貪婪邊演算法與近鄰演算法皆假設圖形是完全圖，否則可能找不到任何一種旅行路線解法。

**► 演算法 10.2** **貪婪邊演算法用於 TSP 問題**

**問題：**為一個加權無向圖決定最佳旅行路線找到最佳旅行路線，其中每邊的權重均為非負數值。

**輸入：**加權無向圖 $G$，與其節點個數 $n$。

**輸出：***tour* 變數，由最佳旅行路線中所經過之邊所構成的集合

```
void generate_tour (int n, graph G, setofedges& tour)
{
    將邊以非遞減順序排列 ;
    tour = 0;          //tour 由一個邊集合所代表
    repeat
        if 新增下一個邊到 tour 中不會造成任一個頂點有三個邊接觸到它
        and 不會造成任一個小於 n 的迴路
            將該邊加入 tour 中 ;
    until tour 中有 n 條邊 ;
}
```

圖 10.6 以和圖 10.3 一樣的例子展示了 GEA 演算法。同樣由 Süral 等人於 2010 年提出的貪婪邊互換演算法 (Greedy Edge Crossover Algorithm, GEX) 與 NNX 除了第六步驟之外其他步驟均相同，我們於以下指出相異之處。

1. 決定如何執行交換與突變。就如同 NNX 一樣，在 GEX 中親代也聯集的方式結合兩張圖，之後 GEA 便套用到聯集圖的邊。若無法產生任何旅程，一個旅程中其餘的邊便透過將 GEA 套用於完全圖上獲得。然而這樣方法的搜索空間狹窄，因此親代和子代邊的變化微小而且可能造成初期收斂至低品質的結果。為了提高探索範圍，我們可以由聯集圖中取出一半的旅程，再由完全圖取出另一半。在初步評估中，這樣的方法遠比前一個盡可能從聯集圖利用貪婪的方法還要來的好。下一節的評估也是基於這個版本所做的。

NNA 和 NNX 與最小生成樹之 Kruskal 演算法非常相似，它們的時間複雜度分別是 $\theta(n^2 \log n)$ 和 $\theta(m \log m)$，其中 $n$ 為節點個數，$m$ 為邊個數。

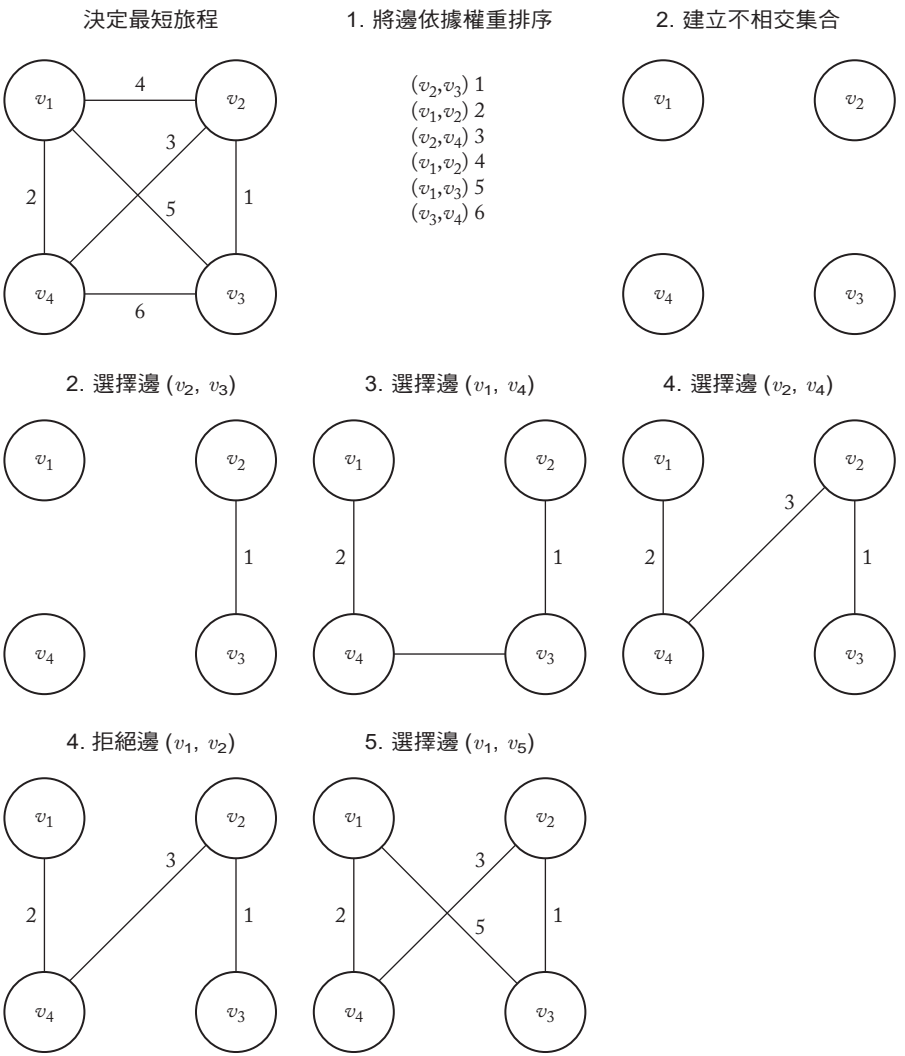


圖 10.6 我們以一個 TSP 實例來解釋貪婪邊演算法

## • 評估

就如其他眾多的啟發式演算法，基因演算法並沒有可供證明其正確性的性質。因此我們透過測試基因演算法在一些實際問題中的表現來評估。Süral 等人於 2010 年以下列方法測驗了 NNX 和 GEX。

首先，他們從 TSPLIB 取得了 10 個 TSP 的實例，這些實例都代表真實城市間的距離；這些距離是對稱的（雙向距離相同）。最大的實例包含 $n = 226$ 個節點。實驗過程中他

們套用了 NNX 和 GEX，且以突變於否分成三個變異實驗：無突變介入、M1 突變和 M2 突變，我們之前有介紹到 M1 和 M2，除此之外他們還嘗試以隨機 (R) 和混合 (H) 近鄰和貪婪邊的方式初始化個體。對每個突變與初始的組合，在每個實例中個別執行 30 次，最後總共取得 300 次的實驗結果。實驗中演算法以 ANSI C 實作，並且於 Pentium IV 1600 MHz 和有 256MB 的 RedHat Linux 8.0 系統上執行。

表 10.5 展示了所有 300 次實驗的平均結果，表中的標題的意義如下：

- Algorithm：所使用的演算法。
- Mutation：突變的方式，“No M” 表示沒有突變的介入。
- Init. Pop.：個體的初始化方式，(R) 表示隨機，(H) 表示混合的方式。
- Dev：最終結果與最佳解的差異百分比。
- #Gen：直到收斂的世代數量。
- Time：直到收斂的時間 ( 秒 )。

我們可以從表 10.5 中得知 NNX 比起 GEX 表現的要好，其中 M2 突變得到最好的結果而且混合初始並不會比隨機初始還要好。之後他們探討 NNX 搭配不同比例的 GEX 是否能提升結果。表中說 50%NNX 與 50%GEX，代表下世代的個體 50% 以 NNX 產生而另外 50% 以 GEX 產生。結果在 M2 突變下以高比例的 NNX 比純 NNX 的方式要有些微的提升。

根據以上結果，這些研究者們提出以下的結論：兩種演算法混合使用所增加的效益不敵額外的運算時間，因此之後的實驗僅探討 NNX 的結果。

NNX 演算法搭配隨機初始 100 個體的結果如表 10.6，注意到 M2 突變下的平均差異百分比是 0.35 且平均時間為 26.2 秒。再次回到表 10.5 中，我們看到同樣的實驗設定但初始 50 個體下的結果，其平均差異百分比是 0.55 且平均時間為 5.52 秒，這樣的正確率提升就值得這些多耗用的計算時間。

接下來，他們探討更大一點的實例，從 TSPLIB 取得了 $n$ 介於 318 至 1748 之間的範例。根據之前混合初始方法不值得多耗用時間之結論，他們對於每個實例均以隨機初始方式執行 NNX 10 次，並且設定族群大小為 100，表 10.7 展示了結果。令人好奇的是，在較大問題實例下，以 M2 突變相對於 M1 突變並沒有明顯的提升效能，但是執行時間卻大幅度的增加。再回到表 10.6 中，可看到在小規模的問題實例中，M2 比 M1 突變表現好很多，卻僅增加一點點計算成本。

• 表 10.5 當族群大小設定為 50 時，在 10 個小問題實例上執行 30 次實驗的平均結果

| Algorithm | Mutation | Init. Pop. | Dev | #Gen | Time |
|---|---|---|---|---|---|
| | No M | R | 3.10 | 45.39 | 0.38 |
| | | H | 4.82 | 33.52 | 2.95 |
| NNX | M1 | R | 1.67 | 40.21 | 0.62 |
| | | H | 1.57 | 36.09 | 3.55 |
| | M2 | R | 0.55 | 53.37 | 5.52 |
| | | H | 0.55 | 43.53 | 8.11 |
| | No M | R | 12.54 | 17.35 | 48.23 |
| | | H | 7.19 | 16.37 | 54.27 |
| GEX | M1 | R | 4.36 | 60.44 | 208.70 |
| | | H | 3.67 | 48.44 | 178.65 |
| | M2 | R | 3.30 | 26.30 | 82.79 |
| | | H | 3.01 | 25.83 | 90.58 |
| | No M | R | 8.15 | 42.50 | 73.25 |
| | | H | 5.53 | 38.47 | 75.67 |
| 50% NNX | M1 | R | 1.92 | 66.04 | 113.81 |
| 50 % GEX | | H | 1.68 | 61.21 | 112.77 |
| | M2 | R | 1.76 | 19.25 | 26.40 |
| | | H | 1.61 | 20.68 | 34.19 |
| | No M | R | 7.23 | 41.16 | 13.39 |
| | | H | 5.19 | 34.93 | 14.95 |
| 90% NNX | M1 | R | 1.84 | 55.60 | 19.14 |
| 10% GEX | | H | 1.67 | 46.93 | 20.16 |
| | M2 | R | 0.51 | 37.13 | 19.26 |
| | | H | 0.48 | 37.24 | 21.95 |
| | no M | R | 6,69 | 41.23 | 6.74 |
| | | H | 5.06 | 33.04 | 8.93 |
| 95% NNX | M1 | R | 1.77 | 52.62 | 10.03 |
| 5% GEX | | H | 1.41 | 44.33 | 11.30 |
| | M2 | R | 0.49 | 37.15 | 11.58 |
| | | H | 0.44 | 36.19 | 14.88 |

• 表 10.6 當族群大小設定為 100 且初始族群為隨機產生時，在 10 個小問題實例上執行 30 次實驗的平均結果

| Algorithm | Mutation | Dev | Time |
|---|---|---|---|
| NNX | No M | 5.40 | 18.4 |
| | M1 | 1.44 | 26.4 |
| | M2 | 0.35 | 26.2 |

• 表 10.7 當族群大小設定為 100 且初始族群為隨機產生時，在 15 個小問題實例上執行 30 次實驗的平均結果

| Algorithm | Mutation | Dev | Time |
|---|---|---|---|
| NNX | No M | 7.61 | 25.3 |
| | M1 | 4.94 | 65.0 |
| | M2 | 4.70 | 1063.0 |

• 表 10.8 在 10 個問題實例上進行 NNX 與另兩個 TSP 的啟發式演算法之比較結果

| Problem | NNX-Ml | | NNX-M2 | | Meta-RaPS | | ESOM3 | |
|---|---|---|---|---|---|---|---|---|
| | Dev | Time[1] | Dev | Time[1] | Dev | Time[2] | Dev | Time[3] |
| ei101 | 0.93 | 8.7 | 0.82 | 14.3 | NA | NA | 3.43 | NA |
| bier127 | 0.62 | 15.3 | 0.28 | 12.0 | 0.90 | 48 | 1.70 | NA |
| pr136 | 2.87 | 18.4 | 0.37 | 35.2 | 0.39 | 73 | 4.31 | NA |
| kroa200 | 1.78 | 87.3 | 0.32 | 98.6 | 1.07 | 190 | 2.91 | NA |
| pr226 | 0.79 | 93.0 | 0.01 | 21.6 | 0.23 | 357 | NA | NA |
| lin318 | 1.87 | 8.0 | 2.01 | 105 | NA | NA | 2.89 | NA |
| pr439 | 3.44 | 10. | 1.48 | 240 | 3.30 | 2265 | NA | NA |
| pcb442 | 4.75 | 15.0 | 3.18 | 270 | NA | NA | 7.43 | NA |
| pcb1173 | 3.00 | 97.0 | 8.01 | 1230 | NA | NA | 9.87 | 200 |
| vm1748 | 7.05 | 203 | 7.09 | 4215 | NA | NA | 7.27 | 475 |

[1]Pentium 4.16 GHz
[2]AMD Athlon 900 MHz
[3]SUN Ultra 5/270

單純的在少數實例中測試啟發式演算法並不能證明對比於其他方法真的比較好，我們必須要檢測他們和之前其他方法的優劣。Süral 等人於 2010 年比較了 NNX 和其他 TSP 的啟發式演算法，對象有：Meta-RaPS (DePuy et al., 2005) 和 ESOM (Leung et al., 2004) 測次於 10 個 TSP 指標範例上，最後結果列於表 10.8。在十個範例中 NNX-M1 或 NNM-M2 都比其他方法要來的好。

## • 10.3 基因規劃法

雖然在基因演算法中，「染色體」或「個體」表示問題的一種解法，但是在**基因規劃法** (**genetic programming**) 中，一個個體表示解決問題的一種規劃，而某個體的適應函數則是衡量該種規劃解決問題的能力。首先由一初始的規劃族群開始，讓適應的程式藉由互換來繁衍、和對於子代的族群實行突變，重複以上過程直到終止條件達成。此程序的演算法和基因演算法相同，並且我們將其展示於下列。

```
void generate_populations ( )
{
    t=0;
    初始族群 P0;
    repeat
        評估每個族群 Pt 中的適應度 ;
        基於適應度選擇可繁衍的個體 ;
        對選出可繁衍的個體實施交換與突變 ;
        t++;
    until 終止條件成立 ;
}
```

在基因規劃中的個體 ( 規劃 ) 會以樹狀結構來表示，樹中每個節點均代表著**終端符號** (**terminal symbol**) 或是**函數符號** (**function symbol**)。若一節點為函數符號，那麼它的參數為其子代。如下例指出：假設有一數學式 ( 規劃 ) 為

$$\frac{x+2}{5-3\times x} \tag{10.1}$$

其樹狀結構就如圖 10.7 表示。

### • 10.3.1 範例演示

一個能簡單示範基因規劃法的方法是藉由多個 $(x_i, y_i)$ 配對，學習得到函數 $y = f(x)$。例如我們擁有表 10.9 中的配對。

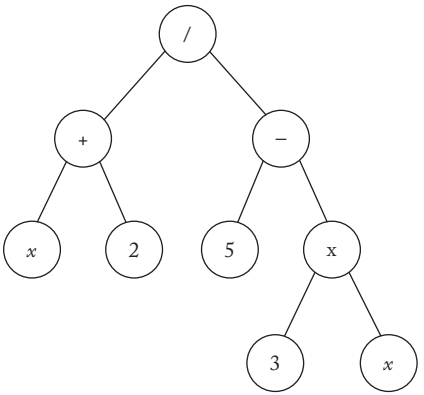


圖 10.7 公式 10.1 對應的樹

這些配對其實是用產生出來的。

$$y = x^2/2$$

假設我們並不知道這項事實，而嘗試去發掘該函數，設計基因規劃的步驟如下：

1. 決定終端集合 T，我們設定終端集合包含 $x$ 與整數範圍介於 -5 至 5。
2. 決定函數集合 F，我們設定函數集合包含 ＋、－、×、/。若需要的話我們也可以包含三角函數等其它函數到 F。
3. 決定族群由多少個體組成，我們設定由 600 個體組成族群。
4. 決定如何初始化族群，每個初始個體由稱為**生長樹 (growing the tree)** 的流程生成。首先，一個符號由 T ⋃ F 集合中隨機選出。若選中終端符號，則由該符號形成單一節點的樹狀結構；若選中函數符號，我們則隨機產生子代，並且繼續生成樹狀結構直到選中終止符號。如圖 10.7 所示，該樹狀結構是由以下符號序列產生：第一次先產生 / 符號並且產生其子代 ＋ 和 －，＋ 符號產生的子代為 $x$ 和 2 並且終止該分支的生成。之後對 – 產生子代 5 和 × 並且終止 5 的分支生成，最後對 × 產生子代 3 和 $x$。

• 表 10.9 我們希望能基於這 10 個點，學出描述 $x$ 跟 $y$ 之間關係的函數

| $x$ | $y$ |
|---|---|
| 0 | 0 |
| .1 | .005 |
| .2 | .020 |
| .3 | .045 |
| .4 | .080 |
| .5 | .125 |
| .6 | .180 |
| .7 | .245 |
| .8 | .320 |
| .9 | .405 |

5. 決定適應函數為平方誤差，定義如：

$$\sum_{i=1}^{10}(f(x_i)-y_i)^2$$

其中 $(x_i, y_i)$ 為表 10.9 中的各個資料點，誤差值越小表示函數越符合期望。

6. 決定以那些個體執行繁衍，我們透過 4 個體競爭的方式篩選。在**競爭選擇過程** (**tournament selection process**) 中，$n$ 個體被隨機選出，在我們這邊 $n = 4$。我們讓 $n$ 個體互相競爭，結果會有 $n/2$ 適應者勝出和 $n/2$ 較不適應者被淘汰。這些 $n/2$ 的贏家得以繁衍出 $n/2$ 個子代，再由這些子代取代被淘汰者，和贏家組成下世代的族群。在這範例中我們會有 2 個贏家產生，並且產生 2 個子代取代被淘汰者。

若競爭的團體太小則會造成低微的選擇過程，若太大則會提升個體生存壓力。

7. 決定如何執行互換和突變。兩個體間的互換是藉由隨機抽換子樹結構來達成，這樣的方法如圖 10.8 所示。突變是由隨機抽選一節點，並且將其節點底下的子樹藉由重生的方式達成。突變只會隨機的套用於 5% 的子代上。

8. 決定何時終止。當最適應的個體之平方誤差到達 0 或是經歷了 100 世代便會終止演算法。

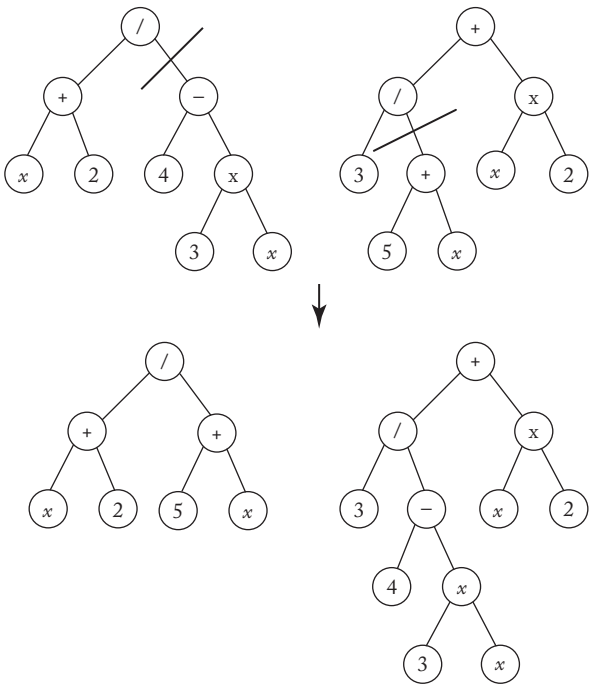


圖 10.8 由交換子樹達成的「交換」動作

Banzhaf 等人於 1998 年應用了上述方法到表 10.9 的資料上，其前 4 世代的結果如下列：

| 子代 | 最適應的個體 |
|---|---|
| 1 | $\frac{x}{3}$ |
| 2 | $\frac{x}{6-3x}$ |
| 3 | $\frac{x}{x(x-4)-1+\frac{4}{x}-\frac{\frac{9(x+1)}{5x}+x}{6-3x}}$ |
| 4 | $\frac{x^2}{2}$ |

其中適應度最高的個體到第 3 世代被擴張成很大的樹狀結構，但是後來到第 4 世代被縮減成為正確解答。

### • 10.3.2 人造螞蟻

試想現在要造一隻機械螞蟻，而它會沿著食物的蹤跡前行。圖 10.9 展示了一個可能的蹤跡，我們稱之為聖菲小道 (Santa Fe trail)。其中每個黑方型表示食物顆粒，圖中有包含 89 個黑方形。螞蟻會從標記有 start 的方形向右開始遊走，它的目的是以最少步驟走訪所有 89 個黑方形並且抵達標記有 89 的方形，注意到路徑之間可能存在著間隔。這樣的問題被視為是**規劃問題** (**planning problem**)。

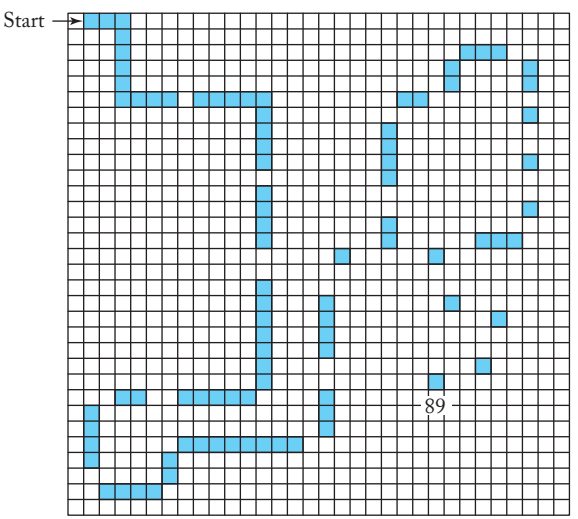


圖 10.9 聖菲小道。每個黑色的方格代表有食物

而且螞蟻有個感應器如下：

*food_ahead*（前方有食物）：感應器回傳 True 若螞蟻所面對的方形有食物，否則回傳 False。

螞蟻可以在任一時間下採取以下三個動作：

*right*（**右轉**）：螞蟻在不移動的情況下向右轉 90 度。

*left*（**左轉**）：螞蟻在不移動的情況下向左轉 90 度。

*move*（**移動**）：螞蟻向前移動至其所面向的方形，若方形包含食物則螞蟻吃掉食物並且將食物從方形上消除。

Koza 於 1992 年對於這個問題發展了以下基因規劃法。

1. 終端集合 T 由以下動作構成：

$$\mathrm{T} = \{right,\ left,\ move\}$$

2. 函數集合 F 如下列出：

   (a) *if_food_ahead* ( 指令 1, 指令 2)

   (b) *do*2 ( 指令 1, 指令 2)

   (c) *do*3 ( 指令 1, 指令 2, 指令 3)

   第一個函數執行指令 1 若**前方有食物**的感應器回傳 True，否則執行指令 2。第二個函數無差別的執行指令 1 和指令 2，同樣的第三個函數也無差別的執行指令 1、指令 2 和指令 3。例如：

$$do2\ (right,\ move)$$

   會讓螞蟻先右轉在向前行。

3. 決定族群以 500 個體構成。
4. 個體均由生長樹流程 ( 參閱段落 10.3.1) 產生。
5. 假設執行每個動作均需求一個時間單位，而且個體最多僅有 400 單位的時間。所有個體均由左上角朝東開始前進，我們則將適應度定義為在最大時間內個體所消耗的食物個數，因此最大適應度為 89。
6. 決定那些個體得以繁衍。
7. 互換和突變則猶如 10.3.1 段落之第 7 步驟所示。
8. 當個體適應度到達 89 或是經歷的世代數量達到了最高值時就停止演算法。

每次迭代中族群裡的每個個體 ( 程式 ) 均執行至 400 時間單位才終止，最後才評估個別適應度。Koza 於 1992 年在某次實驗得到了位於第 22 世代中取得了適應度為 89 的個體，如圖 10.10 示。其中第 0 世代的平均適應度為 3.5，而最佳個體之適應度為 32。

### 10.3.3 金融交易之應用

對於股市或其他金融市場的投資人最為重要的決定，不外乎是在一天內的買進、賣出或繼續持有。先前已經有人致力於研發這樣的自動決策系統，例如 Farnsworth 等人於 2004 年以基因規劃法發展了這樣的系統，他們的系統根據市場指標來做當天的投資決策。我們接下來將探討他們的系統。

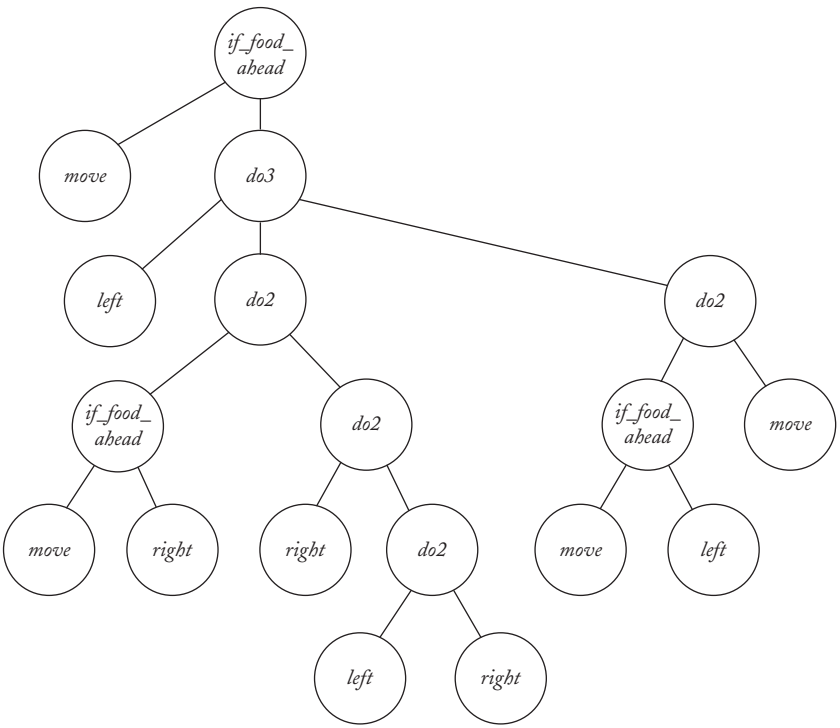


圖 10.10 第 22 代具有適應度 89 的某個個體

## • 發展交易系統

首先我們列出他們建置系統的 8 步驟：

1. 他們設定市場指標為終端符號，研究者們在決定最終符號前先行測試了不同指標，我們只討論他們用於最後系統上的指標：

   (a) S&P 500 是根據在美國經濟中主導產業的 500 大龍頭企業的市場指數，對於當天而言，S&P 500 的指標是以下列函式算得：

$$\frac{S\&P\ 500_{today} - S\&P\ 500_{avg}}{S\&P\ 500_{\sigma}}$$

   其中 $S\&P\ 500_{today}$ 為當日 $S\&P\ 500$ 數值，$S\&P\ 500_{avg}$ 為過去 200 日內 $S\&P\ 500$ 數值平均，而 $S\&P\ 500_{\sigma}$ 是過去 200 日內 $S\&P\ 500$ 的標準差。我們稱這項指標為 $SP\ 500$。

(b) 債券的 $k$-日 **指數移動平均** (**Exponential Moving Average**, **EMA**) 是一個債券於過往 $k$ 日內的加權平均。**移動平均收斂發散** (**Moving Average Convergence/Divergence**, **MACD**) 對於 $x$ 債券如下式定義：

$$MACD(x) = \text{12-day } EMA(x) - \text{26-day } EMA(x)$$

MACD 被視為動量指標，當它為正數時投資人稱之為有利動量 (upside momentum) 上升；反之當它為負數時則稱為不利動量 (downside momentum) 上升。第二個系統中的指標為 $MACD(S\&P\ 500)$，我們簡單稱之為 $MACD$。

(c) $MACD9$ 指標是 S&P 500 的 MACD 的 9 日指數移動平均。

(d) 設 $Diff$ 為上升和下降的債券數差值，則 **McClennan Oscillator** (**MCCL**) 如下式：

$$MCCL = \text{19-day } EMA(Diff) - \text{39-day } EMA(Diff)$$

投資人認為當 $MCCL$ 大於 100 時則表示市場超買，若小於 -100 則表示市場超售。我們稱此指標為 $MCCL$。

(e) $SP\,500lag$，$MACDlag$，$MACD9lag$ 和 $MCCLlag$ 這些指標代表前一天市場的指標。

(f) 其餘終端符號包含介於 $[-1,1]$ 間的實數。所有指標均被正規化到 $[-1,1]$ 區間中。

2. 函數符號包含 $+$、$-$ 和 $\times$，還有下列控制結構：

(a) 若 $x > 0$ 則回傳 $y$，反之回傳 $z$。這樣的結構可以樹狀結構表示成 IF 符號有三個子代節點 $x$, $y$ 和 $z$。

(b) 若 $x > w$ 則回傳 $y$，反之回傳 $z$。這樣的結構可以樹狀結構表示成 IFGT 符號有四個子代結點 $x$, $w$, $y$ 和 $z$。

3. 嘗試多個族群大小，若族群小於 500 個體則顯得不足，若介於 2500 左右則可以得到最佳的結果。

4. 個體初始化的方法於 10.3.1 提及之生長樹的流程一樣，其中最大層數量設定為 4 且最大結點個數設定為 24，這樣的限制也套用於之後突變的過程中。之所以採取如此限制的原因在於防範**過度擬合** (**overfitting**)，也就是避免程式僅能對於預測訓練資料有良好的結果，但是對於未知資料的結果卻不盡理想。

5. 資料由 S&P 於 1983 年 4 月 4 日至 2004 年 6 月 18 日期間之收盤價。以一個給定的樹狀結構分析前 4750 日的資料，這棵樹從第一天 $1 開始，若在某一特定日中該樹回傳大於 0 的數值則買進，否則賣出。當要買進時手頭上所有現金將會全部用於投資中，或是當要賣出時總是賣出全額債券，實驗中不會有半買半賣的情況。若前一天為買入且當天又判斷買入，則沒有任何動作發生。同樣的若前一天為賣出且當天又判斷賣出，則沒有任何動作發生。當 4750 日的交易結束時，第一天的金額減去最後一天的金額得到最後盈額。為了避免過度擬合，適應度會被扣除與交易量成正比的數值。

6. 族群中，適應度較差的 25% 個體會被刪去，進而被更符合的個體所取代。

7. 互換藉由隨機交換兩子樹達成，**點突變 (node mutation)** 則是先由隨機選定幾個結點，再來從它們原本參數中隨機的改變數值。函數節點的突變改變執行的動作，而終端節點則改變指標或是常數值。**樹突變 (tree mutation)** 則是以隨機抽選子樹，並且藉由重新生成子樹的方式替換。

   組群中前 10% 的個體保持不變，並且隨機抽選 50% 的個體和前 10% 個體進行互換，另外 20% 的個體進行點突變，最後 10% 被選作為樹突變對象，還有另外 10% 被刪去並隨機替換。

8. 決定演算法終止條件，這裡設定最大世代數介於 300 到 500 之間。

   圖 10.11 展示了其中一次實驗中適應度最高的樹狀結構。

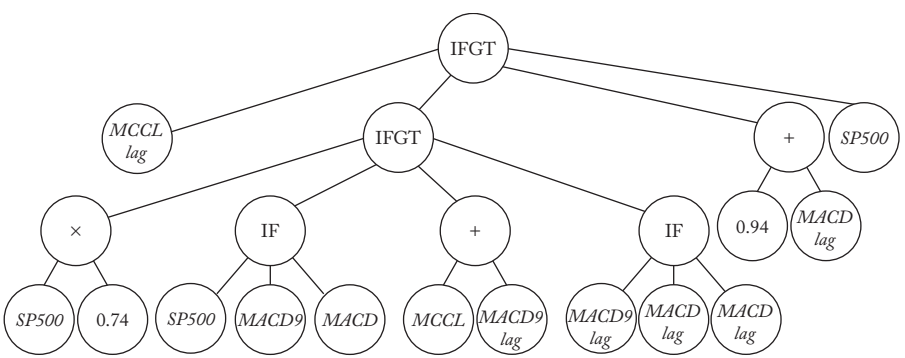


圖 10.11 其中一次實驗中適應度最高且在評估中表現很好的樹狀結構

### 評估

回想起資料是 S&P 於 1983 年 4 月 4 日至 2004 年 6 月 18 日期間之收盤價，且前 4750 天的資料被用於系統學習上，也就是決定適應度，之後剩下的 500 天資料被用於計算適應度以評估系統。此交易系統如圖 10.11 示，並且在於未知資料中有 0.397 的適應度。我們再來看另一種投資市場中的基本策略，**買入並持有** (**buy-and-hold**)，購入債券並持有，這種策略僅有 0.1098 的適應度。

## 10.4 討論與深入閱讀

對於演化式演算我們還有兩個領域尚未介紹 : 演化式規劃和演化策略。**演化式規劃**與基因演算法相似，它們相似處在於它也透過演化候選解答族群的方式尋找問題的答案，演化式規劃和基因演算法不同在於它注重於發展可觀測系統與環境之互動的行為模型，此方法由 Fogel 於 1994 年提出。**演化策略**將問題的答案視為物種的方式模型化，Rechenberg 於 1994 年指出演化策略的領域是基於演化上的演化。讀者可詳讀由 Kennedy 與 Eberhart 於 2001 年的著作，進而對於演化式演算的四個領域皆有更深入的理解。

## 習題演練

### 10.1 節

1. 描述二倍體生物的有性生殖，單倍體生物的二分裂法和單倍體生物的細胞融合之間的差異。
2. 假設有二倍體生物，其一個基因組內包含 10 個染色體，則
   (a) 其體細胞內有多少個染色體？
   (b) 其配子內有多少個染色體？
3. 假設有兩個成年的二倍體生物是藉由細胞融合繁衍，則
   (a) 有多少子代可能被繁衍出來？
   (b) 這些子代是否都擁有相同的基因組成？
4. 人類的眼睛顏色是由 bey2 基因所決定的，並且棕色眼睛的對偶基因是顯性的。對於下列各個父母的對偶基因組合，決定子代的眼睛顏色。

## 10.2 節

5. 參照表 10.1，假設 8 個體的適應度分別為：.61, .23, .85, .11, .27, .36, .55 和 .44。試算正規化後的適應度與累計的正規適應度。

6. 假設我們如同表 10.3 般執行互換，其中親代分別是 01101110 和 11010101 且開始和結束位置分別是 3 和 7，根據上述試推演出兩個子代。

7. 參閱 10.2.2 小節，實作基因演算法用於尋找函數 $f(x) = sin(x\pi/256)$ 的極大值。

8. 參照圖 10.12 中的 TSP 範例，假設往來兩節點之邊權重相同，試找出最短的旅行路線。

9. 假設兩親代分別為 3 5 2 1 4 6 8 7 9 和 5 3 2 6 9 1 8 7 4，並且起始和結束點分別為 4 和 7，試推演出互換後產生之子代結果。

10. 參照圖 10.12 中的 TSP 範例，以每個節點起始套用近鄰演算法，是否有任一個結果為最短的旅行路線。

11. 將圖 10.13 中兩個旅行路線聯集成另外一圖，並且以初始節點為 $v5$ 推演套用近鄰演算法於之上的結果。

12. 試以貪婪邊演算法用於圖 10.12 之 TSP 範例上，試問其結果是否為一最短的旅行路線？

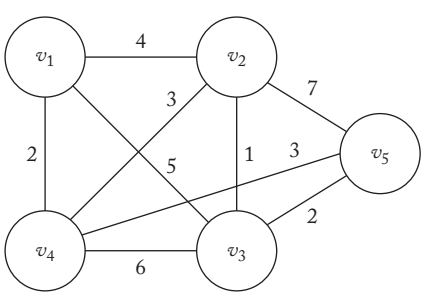


圖 10.12　TSP 的一個問題實例

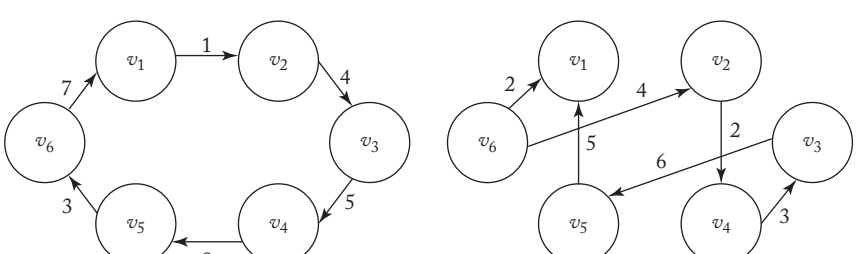


圖 10.13　兩種旅行路線

### 10.3 節

13. 根據圖 10.8 中兩個樹狀結構，試將左樹中以 4 起始的子樹和右樹中以 + 起始的子樹互換，並推演出新的樹狀圖。
14. 根據圖 10.10 中的個體（規劃），推演出圖 10.9 中的聖菲小道前十步的過程。
15. 實作基因規劃法用以解決 10.3.2 小節提及的聖菲小道問題。