



Manufacturing Data Science

Metaheuristic (第 16 章 元啟發式演算法)

Chia-Yen Lee, Ph.D. (李家岩 博士)

Department of Information Management (資訊管理學系)
National Taiwan University (國立臺灣大學)

- 第一章 製造數據科學
- 第二章 製造系統分析與管理
- 第三章 數據科學基礎與模型評估
- 第四章 數據科學分析架構與系統運算決策
- 第五章 數據預處理與製造數據特性
- 第六章 線性分類器
- 第七章 無母數迴歸與分類
- 第八章 決策樹與集成學習
- 第九章 特徵挑選與維度縮減
- 第十章 類神經網路與深度學習
- 第十一章 集群分析
- 第十二章 特徵工程、數據增強與數據平衡
- 第十三章 故障預測與健康管理
- 第十四章 可解釋人工智慧
- 第十五章 概念漂移
- **第十六章 元啟發式演算法**
- 第十七章 強化學習

藍：老師課堂講授
綠：學生自學

- 附錄A 線性迴歸
- 附錄B 支持向量機
- 附錄C 統計製程管制與先進製程控制
- 附錄D 超參數最佳化

- 應用涵蓋
產能規劃、瑕疵檢測、製程監控與診斷、機台保養、需求預測、生產排程、電腦視覺、自動光學檢測、原料價格預測與採購等

- 最佳化基礎與求解搜尋
- 禁忌搜尋法與排程問題
- 基因演算法
- 進階議題
 - 迷因演算法
 - 多目標最佳化
 - 最佳資源預算分配 (學生自學)
- 結語

Material source: Gen, M. and R. Cheng, Genetic Algorithms & Engineering Design, John Wiley & Sons, New York, 1997.

□ 處方性構析 (prescriptive analytics)

- 將問題以 **最佳化的形式** 描述，包含欲求解的決策變數 (decision variable) 、最佳化的目標函數 (objective function) 以及資源與時間上的限制 (constraint) 。
- 根據問題特性 (例如：確定性或隨機性、問題可否結構化) ，可將不同最佳化問題所對應的方法整理。

表 16.1 最佳化問題的類型 (Amaran et al., 2016)

問題	可建構數學模型	未知或複雜的問題結構
確定性	數學規劃 (線性、整數、非線性) (Hillier and Lieberman, 2021)	元啟發式演算法 (求解具有隨機性) [] (Abdel-Basset et al., 2018)
隨機性	隨機規劃 (Birdge and Louveaux, 2011) 、穩健最佳化 (robust optimization) (Bertsimas et al., 2011; Gorissen et al., 2015)	模擬最佳化 (可結合元啟發式演算法) (Carson and Maria, 1997; Amaran et al., 2016) 、實驗設計與反應曲面法 (Montgomery, 2020)

□ 處方性構析 (prescriptive analytics)

- 在實務上，多數的最佳化問題具備隨機性或過於複雜非線性結構〔如非凸性 (nonconvex) 特性〕
- 此類問題包含多目標且多限制的生產排程最佳化或是製造現場交互作用複雜的機台參數最佳化等
- 因此實務上常使用「元啟發式演算法」 (metaheuristic algorithm)



□ 元啟發式演算法

- 是指一種主策略 (master strategy)，該策略可指導和修改其他啟發式方法，以產生跳出陷入區域最佳解 (local optimum) 的情境，進而搜尋更好的解。

□ 最佳化的方法論



- 區域與全域最佳解、求解速度與品質、開採與探索權衡思維三個層面來評估最佳化演算法的適用性與本質

□ 區域與全域最佳解

- 對於一個非凸性的最小化問題，其函數包含區域最小值（local minimum）及全域最小值（global minimum），全域最小值是想搜尋解的目標。
- 然而，典型的「啟發式演算法」，（又稱區域（local）搜尋，使用基於規則或經驗的方法（rule-based method），因而經常落入區域最小值。
- 例如：梯度下降法(gradient descent)，學習法則如公式

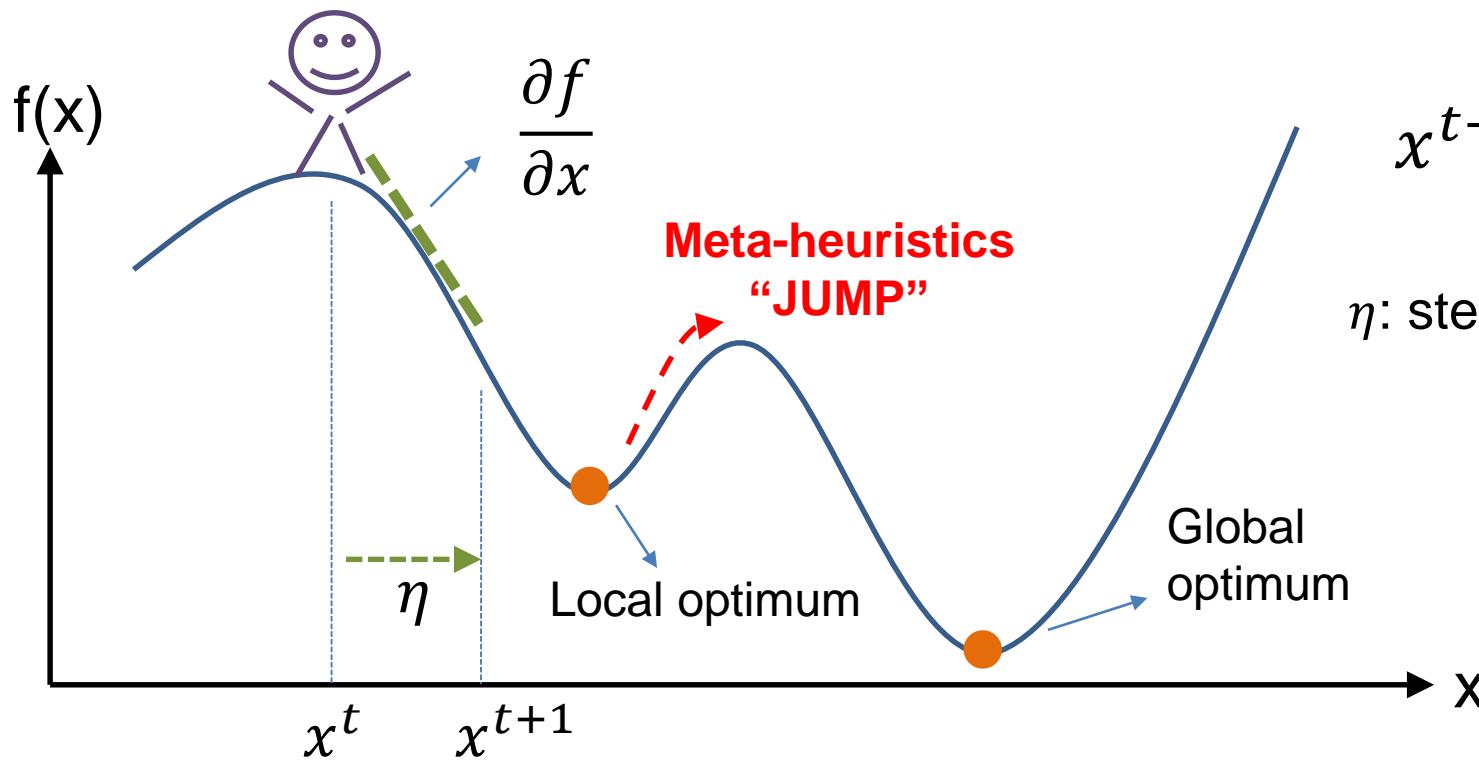
$$x^{t+1} \leftarrow x^t + \eta \frac{\partial f}{\partial x}$$

— 其中 x 為決策變數(解)， $f(x)$ 為適應值， η 為學習率(learning rate)，也就是每一回合更新解的步伐大小(step size)。 t 為演算法學習疊代回合(epoch)。

- 若以梯度下降法進行運算，當解落入區域最小解時，此時進行鄰近的梯度計算，發現皆無法讓解變得更小，因此演算法停止（跳出學習迴圈）。然而，元啟發式演算法提供了一個機制讓解有機會跳脫區域最小值，進而試著搜尋更好的解。

□ Definition of Meta-heuristics (Glover and Laguna 1997)

- A meta-heuristic refers to a **master strategy** that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for **local optimality**.
- For the “Minimization Problem”...



Gradient Descent (梯度下降法)

$$x^{t+1} \leftarrow x^t + \eta \frac{\partial f}{\partial x}$$

η : step size (learning rate)

□ 求解品質與運算速度

- 最佳化方法包含數學規劃（含動態規劃）、元啟發式與啟發式演算法

— 運算速度

- 數學規劃所需的時間複雜度通常較大；啟發式演算法以規則求解速度快

— 求解品質

- 數學規劃學理上求解全域最佳解；啟發常對於複雜問題會收斂至區域最佳解

— 總結來說，元啟發式演算法的實用性是因為在速度與品質皆能有所兼顧與權衡，尤其對於**大型且複雜的最佳化問題**。

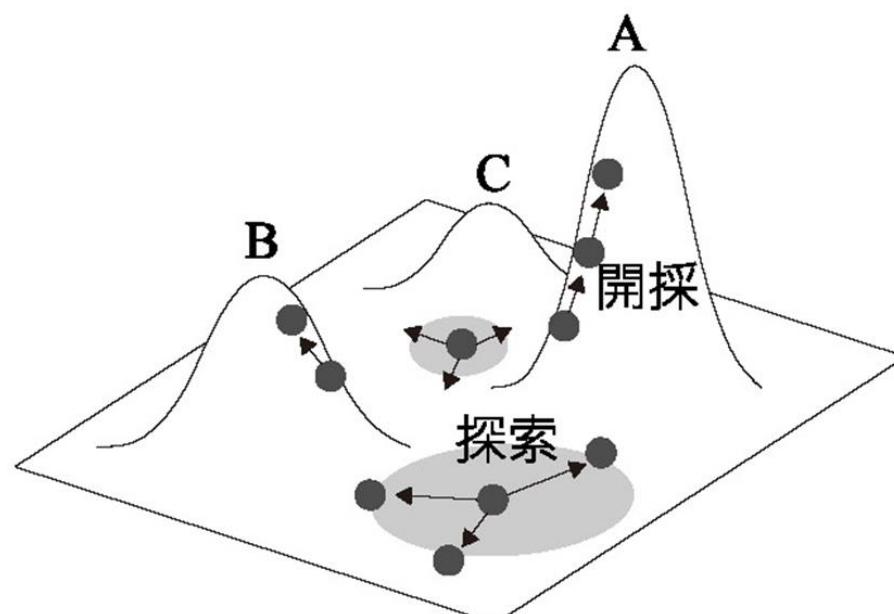
表 16.2 求解品質與運算速度於方法論上的比較

	數學規劃	元啟發式演算法	啟發式演算法
設計本質	學理上的最佳化	搜尋解的演算程序+開採與探索的權衡機制（跳出區域最佳解的機制）	搜尋解的演算程序
求解品質	高（全域最佳解）	中（近似最佳解；學理上無法保證全域最佳解）	低（複雜問題下常是區域最佳解）
運算速度	慢（尤其是混整數規劃或大型問題）	中	快
方法	線性規劃、非線性規劃、動態規劃等	禁忌搜尋法、基因演算法、模擬退火法、粒子群最佳化等	基於規則或經驗方法、典型的梯度下降法、貪婪演算法等

□ 開採與探索權衡思維

- 在最佳化演算法找尋全域最佳解的過程中，一方面期望在潛在最佳解的鄰近區域進行**開採** (exploitation)，一方面試著往潛在最佳解可能的方向搜尋未曾探索過的區域進行**探索** (exploration)，此為兩者權衡思維

— 如圖所示，我們考慮一個最大化問題，若將可行解域視覺化呈現出一個三維度空間，圖中的三個山峰包含了兩個區域最佳解 (B, C) 以及一個全域最佳解 (A)，而元啟發式演算法的其中一種思維是在空間中散佈許多粒子（構成母體），並利用他們同時開採與探索整個可行解空間。



A : 全域最佳解 B,C : 局部最佳解

□ 元啟發式演算法

- 元啟發式演算法是依據啟發式演算法所延伸的思維與命名。
- 啟發式演算法的啟發式 (heuristics) 源自於古希臘文「heuriskein」，意味著「發現，尋找」(“I find; discover”)，是一種在解空間 (solution space) 隨機搜尋 (random search) 的過程。
 - 其思維是設計一種規則化的程序求解問題，並同時兼顧最佳性 (optimality)、精確性 (accuracy)、運算效率 (computational efficiency) 以及完整性 (completeness)。
- 然而，對於具有隨機性與非凸性特性的最佳化問題而言，規則化的程序例如「爬山演算法」(hill climbing algorithm) 或梯度下降法往往落入區域最佳解。
- 因此元啟發式演算法興起，除了有機會跳脫區域最佳解，並同時權衡運算運算速度與求解品質。

□ 元啟發式演算法

- 多數方法源自於大自然中的生物特性、本能以及群體行為互動等啟發，來設計演算法程序。
- 一般我們可將這些方法分為兩種類型，分別為單一解式搜尋（single-solution search）〔也就是軌跡式（trajectory）〕與母體式搜尋（population-based search）。
- **單一解式搜尋**是經由反覆地調整一組解來找尋最佳解；而**母體式搜尋**則是反覆地藉由多組解的資訊共享與交換來找尋最佳解。
- **母體式搜尋的方法**
 - 「基因演算法」（genetic algorithm, GA）
 - 「基因規劃」（genetic programming, GP）
 - 「粒子群最佳化」（particle swarm optimization, PSO）
 - 「蟻群最佳化」（ant colony optimization, ACO）



□ 禁忌搜尋法(Glover and Laguna, 1998)

- 透過使用記憶(memory)來記住與忘記過去曾搜尋過的解，建立相對簡單的鄰近(neighbor)搜尋機制，通常適用於求解「組合最佳化問題」(combinatorial optimization)。
 - 例如巡視廠區內一百台機台，並期望著以一個最短路徑巡視並回到原點(旅行推銷員問題TSP))
 - 首先，隨機或依過去經驗先巡視一輪，其次，在未來多次巡視時微調部分巡視的順序，並記憶調整後最好的路線。接著，也記憶被調整機台的順序盡可能不再去調整(避免重複嘗試)，最後經反覆上述程序可得一個相對短的路徑
 - 上述策略的關鍵在於記憶過去的變動，唯有這樣的記憶才能使得新路徑得以探索未搜尋過的解區域，而避免走回頭路(避免再次搜尋過去已經搜尋過的區域)，同時不被過去最好的經驗(區域最佳解)所禁錮，這便是禁忌搜尋法的核心思維。換句話說，是將過去的變動視為禁忌，而把這些變動紀錄於「禁忌名單」(Tabu list)之中。
- 短期記憶促使解的開採，長期記憶則具有探索的本質。
 - 短期記憶：過去的變動，也就是禁忌名單，是上述案例順序微調的記錄。
 - 長期記憶：每次更新後所得到最好的解，是上述案例每輪順序微調後的最短路徑

□ 禁忌搜尋法

表 16.3 禁忌搜尋法

演算法 禁忌搜尋演算法 (Tabu Search Algorithm)

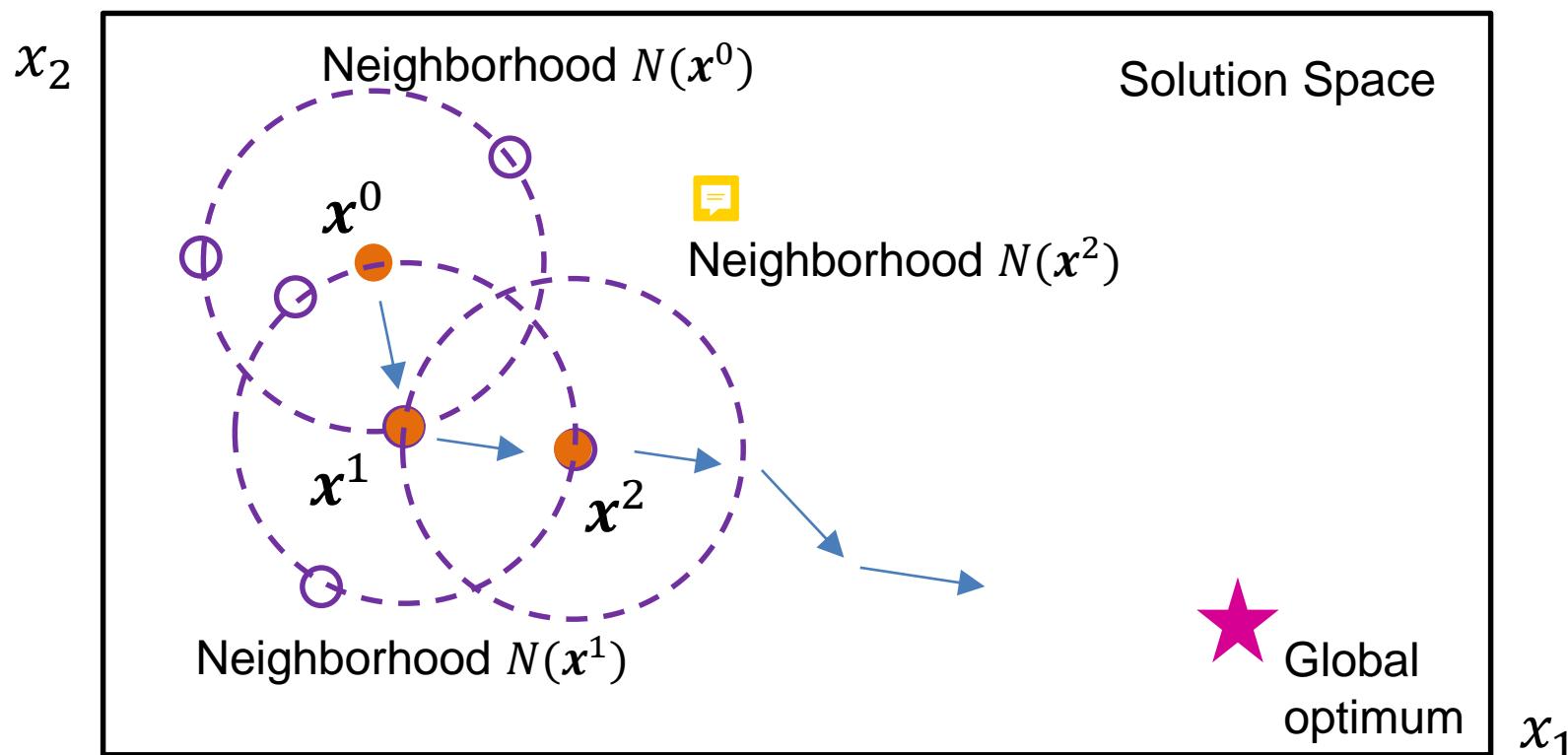
輸入：禁忌名單大小 s ，鄰近解個數 n ，很大的正數 M ，鄰近解生成函數 $N(\cdot)$ ，目標函數 Fitness，停止條件為最大疊代次數 $Iter^{stop}$

輸出：最小化問題的最佳解 x^{best}

1. 設定一個空的禁忌名單 T ，並以隨機或基於經驗法則產生一個初始解 $x^{best} = x^{now} = x_0$
2. 開始迭代（重複此步驟直到滿足停止條件 $Iter^{stop}$ ）
 3. [開採階段] 生成鄰近解 $N(x^{now}) = \{x_1, \dots, x_n\}$ ，以產生一組候選名單 C ， $x^{next} = M$
 4. For $i = 1$ to n
 5. 若鄰近解 x_i 落於禁忌名單 T 中，則將其從候選名單 C 中剔除
 6. 若目標值 $Fitness(x_i) < Fitness(x^{next})$ ，則令 $x^{next} = x_i$
 7. End;
 8. [探索階段] $x^{now} = x^{next}$ ，並將 x^{now} 更新至禁忌名單 T （若禁忌名單 T 已達大小 s 時，則剔除名單中最早的解）
 9. 若目標值 $Fitness(x^{now}) < Fitness(x^{best})$ ，則令 $x^{best} = x^{now}$
 10. End;
 11. Return最佳解 x^{best} .

□ 禁忌搜尋法的隨機搜尋過程 Stochastic/Local/Random Search

- 如圖所示，呈現禁忌搜尋法在解空間（ solution space ）隨機搜尋的過程，透過迭代與更新，慢慢逼近全域最佳解。值得留意的是，禁忌尋法為了逃脫區域最佳解，是可通過放寬搜尋規則來接受一個相對不好的解，例如就算 x^{next} 沒有比 x^{best} 來得好，依然會有機會（或以某種機率）來接受一個相對較差的解，更新 $x^{now} = x^{next}$ 。



□ 禁忌搜尋法

- 在禁忌搜尋法中，鄰近解的生成方式（ neighborhood searching scheme ）與禁忌名單大小（ Tabu list size ）對於最佳解的搜尋有顯著的影響，此二者決定了可行解探索的方式與程度。
- 常見的鄰近解的生成方式包含了鄰近互換（ neighborhood swap ）以及兩兩互換（ random swap ）等，而這通常會需依據問題的特性來設定。
- 禁忌名單大小的設定
 - 若名單大小設定較大，則將使得鄰近解強調比對過去所搜尋過的路徑，不走回頭路，使其探索的程度相對大於開採（解搜尋會收斂往特定的方向前行），降低局部的搜尋
 - 相對地，若禁忌名單大小設定較小時，將經常遺忘過去所走過的路徑，加強鄰近區域開採的能力，而減少探索未知區域。
 - 因此，在禁忌搜尋法中適當的選擇禁忌名單大小（超參數）會是取得探索與開採權衡的關鍵。

□ 單機生產排程案例

- 依據下列四個工件 (job) 的特性包括了加工時間 P_j (processing time) 、交期 D_j (due date) 以及權重 W_j (weights) (訂單的重要性，例如顧客重要性) 。此處，我們探討四個工件的排序 (sequencing) 問題，以禁忌搜尋法最小化總加權延遲時間 (total weighted tardiness, TWT)
- 令 C_j 為工件 j 的完成時間 (completion time) ，因此總加權延遲時間的計算如公式 $TWT = \sum_{j=1}^4 W_j \max\{C_j - D_j, 0\}$

表 16.4 單機排程的工件數據

工件 j	1	2	3	4
加工時間 P_i	9	11	6	14
交期 D_i	3	6	12	2
權重 W_i	10	13	11	3

□ 單機生產排程案例

● 初始化

- 紿定禁忌名單大小為2，先隨機產生一組初始解
- 其工件加工順序為(1,2,3,4)，則其總加權延遲時間計算如公式所示
- $TWT(1,2,3,4) = W_1 \max\{P_1 - D_1, 0\} + W_2 \max\{P_1 + P_2 - D_2, 0\} + W_3 \max\{P_1 + P_2 + P_3 - D_3, 0\} + W_4 \max\{P_1 + P_2 + P_3 + P_4 - D_0, 0\} = 510$

● 第一次迭代

- 對初始解(1,2,3,4)透過鄰近互換的方式產生三組鄰近解 (neighbor) 作為候選解 (candidate)，例如交換工件(1,2)、(2,3)與(3,4)，我們並計算其分別的總加權延遲時間如公式所示
 - $TWT(2, 1, 3, 4) = 503;$
 - $TWT(1, 3, 2, 4) = 467;$
 - $TWT(1, 2, 4, 3) = 646$
- 其中，對工件(2,3)交換後得到(1,3,2,4)具有最小的總加權延遲時間467，因而將這組解設定為當前的最佳解，並放入禁忌名單中，也就是 $Tabu=\{(2,3)\}$ 。

□ 單機生產排程案例

● 第二次迭代

- 對當前最佳解(1,3,2,4)再由鄰近互換的方式產生三組鄰近解，其分別的總加權延遲時間如公式所示。
 - $TWT(3, 1, 2, 4) = 494;$
 - $TWT(1, 2, 3, 4) = 510$ (Tabu); 
 - $TWT(1, 3, 4, 2) = 616$
- 其中工件(3,2)交換後形成(2,3)存在於禁忌名單中，因此將其剔 (不可行解)
- 剩下來的候選解(3,1,2,4)與(1,3,4,2)的總加權延遲時間為494與616，都比目前的解(1,3,2,4)為467來得差。
- 若根據典型的啟發式演算法或梯度下降法，周圍鄰近解沒有比目前的最佳解好，因此演算法停止並輸出當前的最佳解 (常是區域最佳解)。
 - 然而，對於禁忌搜尋法來說，**有機會接受較差的解**，讓演算法繼續計算下去，以試著搜尋更好的解。
- 而由於工件(1,3)交換後得到(3,1,2,4)在候選解中具有最小的總加權延遲時間，於是將這組解設定為當前的最佳解，並放入禁忌名單中，也就是 $Tabu=\{(1,3),(2,3)\}$ 。

□ 單機生產排程案例

● 第三次迭代

- 對當前最佳解(3,1,2,4)再由鄰近互換的方式產生三組鄰近解，其分別的總加權延遲時間如公式所示。
 - $TWT(\mathbf{1}, \mathbf{3}, 2, 4) = 467$ (Tabu);
 - $TWT(3, \mathbf{2}, \mathbf{1}, 4) = \mathbf{487}$;
 - $TWT(3, 1, \mathbf{4}, \mathbf{2}) = 643$
- 其中工件(1,2)交換後得到(3,2,1,4)具有最小的總加權延遲時間，因而將這組解設定為當前的最佳解，並放入禁忌名單中。
- 然而。由於禁忌名單大小為2，只能記憶兩條禁忌規則，因此把最早的記憶刪除，將新的記憶加入，也就是更新名單為 $\text{Tabu} = \{(1,2), (1,3)\}$ 。

□ 單機生產排程案例

● 第四次迭代

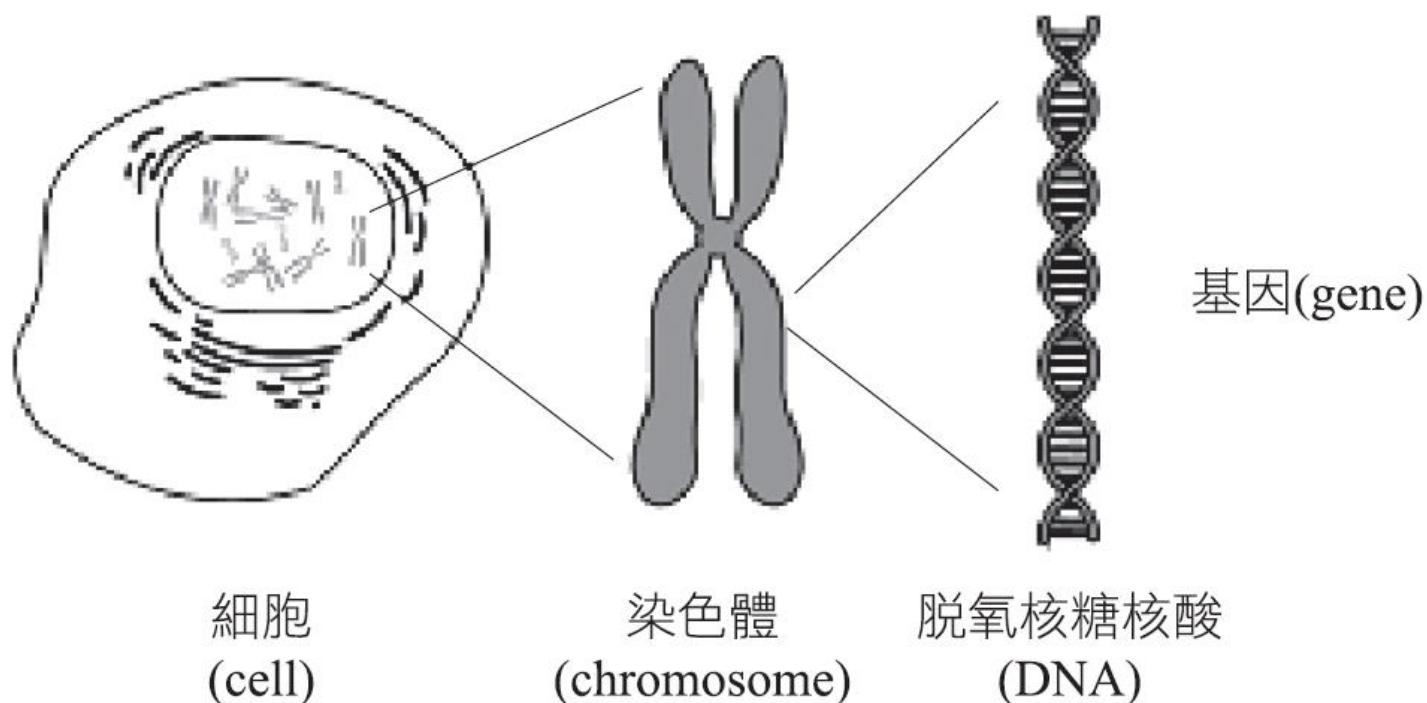
- 對當前最佳解(3,2,1,4)再由鄰近互換的方式產生三組鄰近解，其分別的總加權延遲時間如公式所示。
 - $TWT(2, 3, 1, 4) = 464;$
 - $TWT(3, 1, 2, 4) = 494$ (Tabu);
 - $TWT(3, 2, 4, 1) = 600$
- 其中，對工件(3,2)交換後得到(2,3,1,4)具有最小的總加權延遲時間，因而將這組解設定為當前的「最佳解」，並更新名單為 $Tabu = \{(3,2), (1,2)\}$ 。
- 此處可以發現到，當前得到的最佳解(3,2,1,4)其總加權延遲時間為464，比前面找到的區域最佳解(1,3,2,4)為467來得好。
- 這也說明了禁忌搜尋法跳脫了區域最佳解的狀況，而搜尋到更好的解。
- 在案例中僅解說到第四次迭代，後續可以繼續以此類推計算下去，直到滿足演算法的停止條件(例如目前搜尋到的最佳解經過100次迭代後依然沒有改善)。

□ 基因演算法

- 在元啟發式演算法中，母體式搜尋根據所設定的母體大小 (**population size**)，以多個解透過一般化且高度運算的同步搜尋機制去求解更複雜的最佳化問題，其中基因演算法 (**genetic algorithm, GA**) (Holland, 1975; Goldberg and Holland, 1988) 為適用性廣泛使用的方法之一。
- 由於其適合於求解組合最佳化問題，例如
 - 背包問題、裁切問題 (**cutting stock**) 、裝箱問題 (**bin packing**)
 - 旅行推銷員問題
 - 整數規劃 (**integer programming**)
 - 最小生成樹 (**minimum spanning tree**)
 - 車輛途程問題 (**vehicle routing**)
 - 集合覆蓋問題 (**set covering**)
 - 排程問題
- 在產業中有著廣泛應用。

□ 基因演算法

- 的思維源自於生物圈中的**演化機制**(evolutionary mechanism)。在大自然中，**染色體(chromosome)**的**交配(crossover)**與**突變(mutation)**是生物的遺傳與演化的重要關鍵，從而更好地適應環境。其中染色體又是由**去氧核糖核酸(DNA)**和**蛋白質**所組成的，而每個DNA片段則是控制某一性狀的**基因(gene)**，因而我們可以說染色體是由許多基因所構成的，如圖所示。

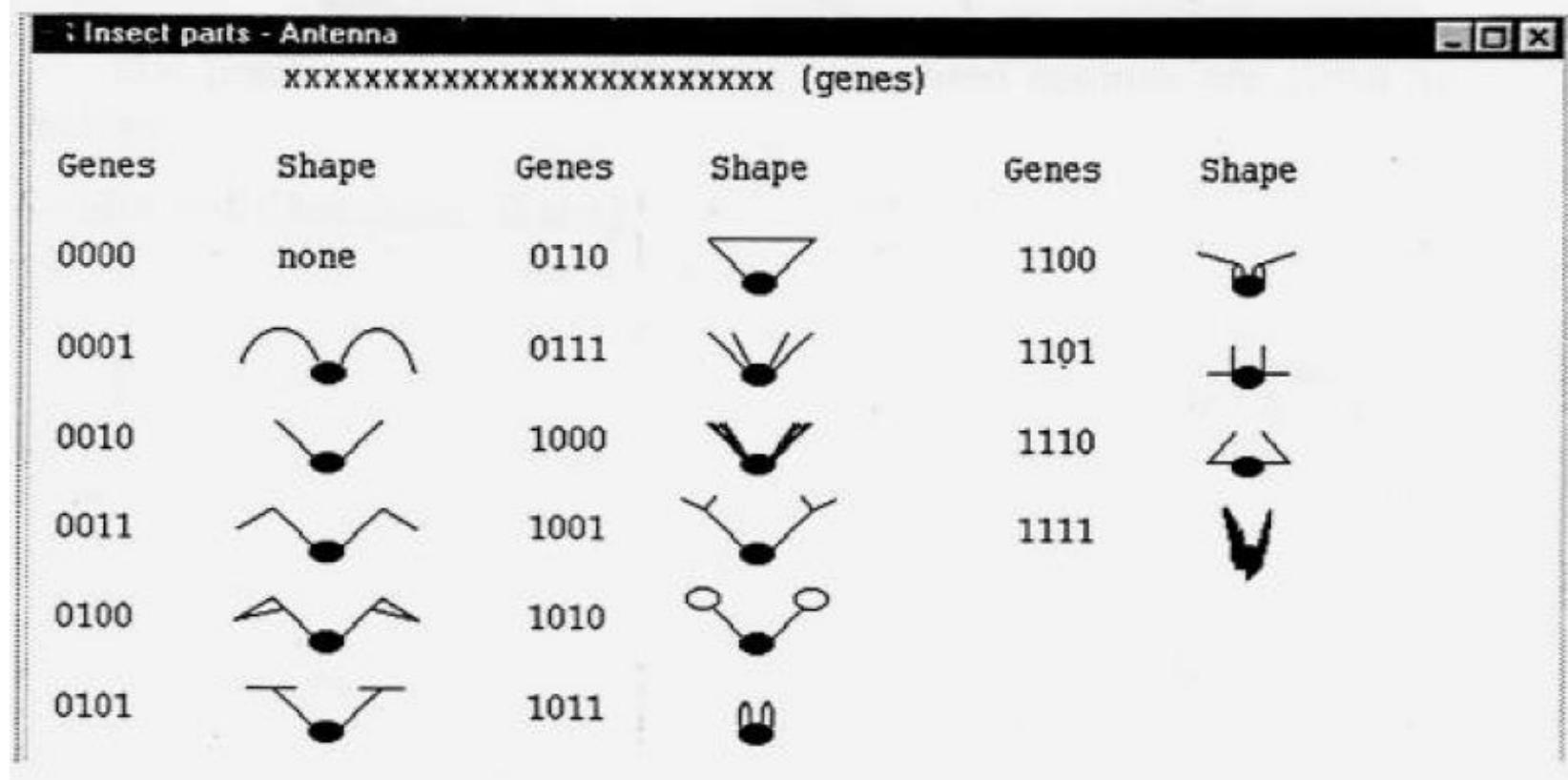


Example- a Butterfly

□ Chromosome Representation

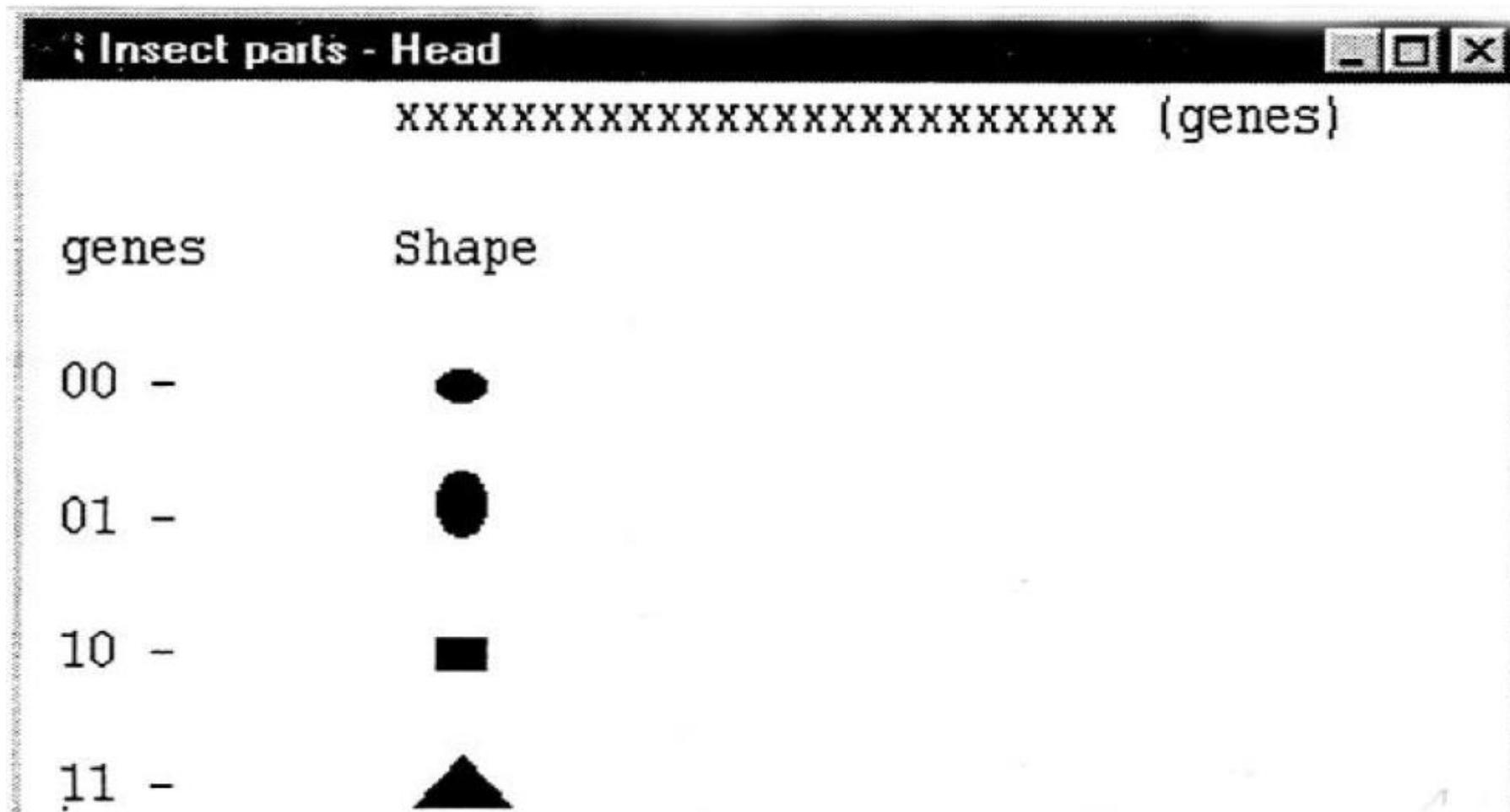
antennae	head	wing	body	feet	body color	size	head color						
1	4	5	6	7	10 11	16	17	18	20	21	22	23	25

Fig. Chromosome coding



Example- a Butterfly

□ Chromosome Representation



Example- a Butterfly

□ Chromosome Representation

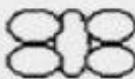
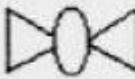
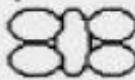
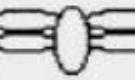
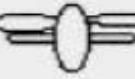
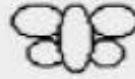
: Insect parts - Wings					
XXXXXXXXXXXXXXXXXXXX (genes)					
Genes	Shape	Genes	Shape	Genes	Shape
0000		0110		1100	
0001		0111		1101	
0010		1000		1110	
0011		1001		1111	
0100		1010			
0101		1011			

Fig. ... Wings representation

□ GA Philosophy

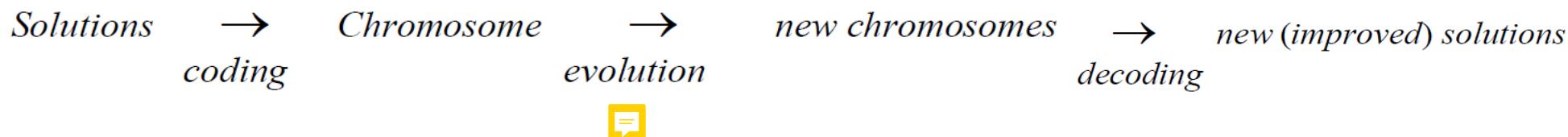


Table **Explanation of Genetic Algorithm Terms**

Genetic Algorithms	Explanation
Chromosome (string, individual)	Solution (coding)
Genes (bits)	Part of solution
Locus	Position of gene
Alleles	Values of gene
Phenotype	Decoded solution
Genotype	Encoded solution

- *Single point crossover:* 

A single crossover point with the range [1, 24] can be randomly selected.
An example of single point crossover is shown in Fig.

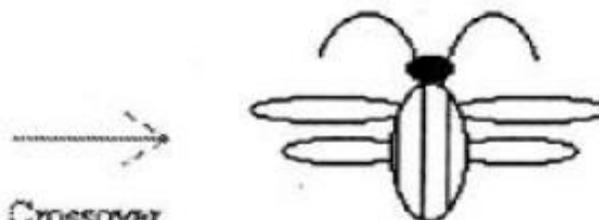
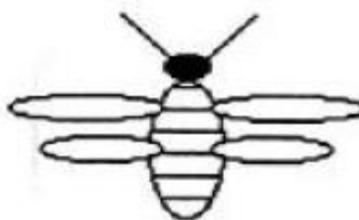
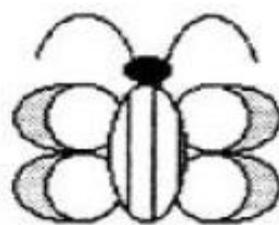


Fig. One point crossover

- *Multi-point crossover:*

Rather than be restricted to the use of single point crossover, a multi-point crossover is allowed. This programming provides 2-point, 3-point and 4-point crossover which can be selected by the user. For demonstration purposes, only a chromosome (insect) with 3-point crossover is shown in Fig. The user may use the other two crossover methods for further investigation.

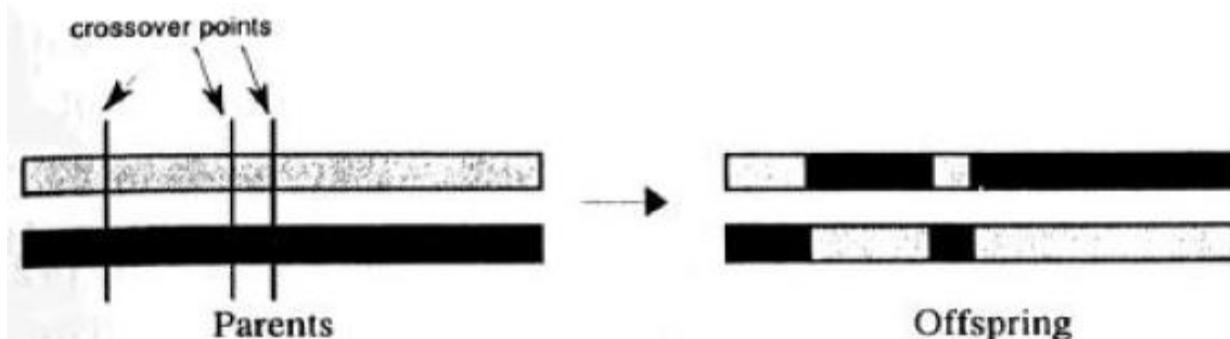


Fig. Example of multi-point crossover

Another approach is the uniform crossover. This generates offspring from the parents, based on a randomly generated crossover mask. The operation is demonstrated in Fig.

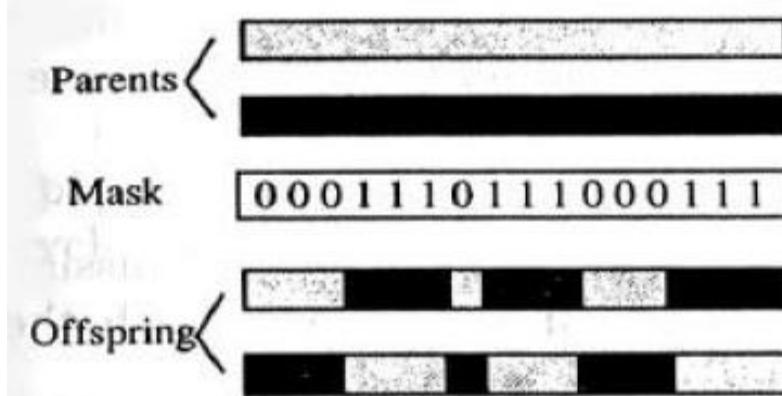


Fig. Example of uniform crossover

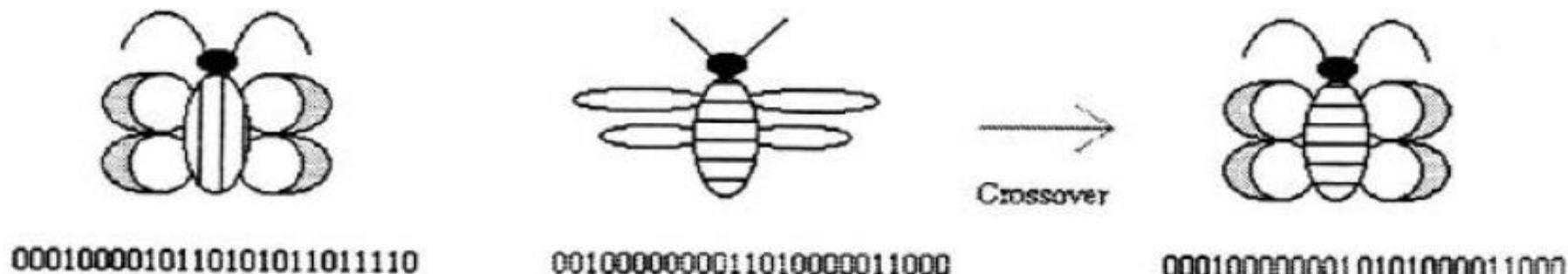


Fig. Three point crossover

- *Uniform crossover:*

As explained in Sect. a uniform crossover mask is required during uniform crossover operations. Such a mask with the same length of the chromosome is randomly generated. In Fig. an example of the mask “1111000010001111100000000” is given to illustrate this principle.

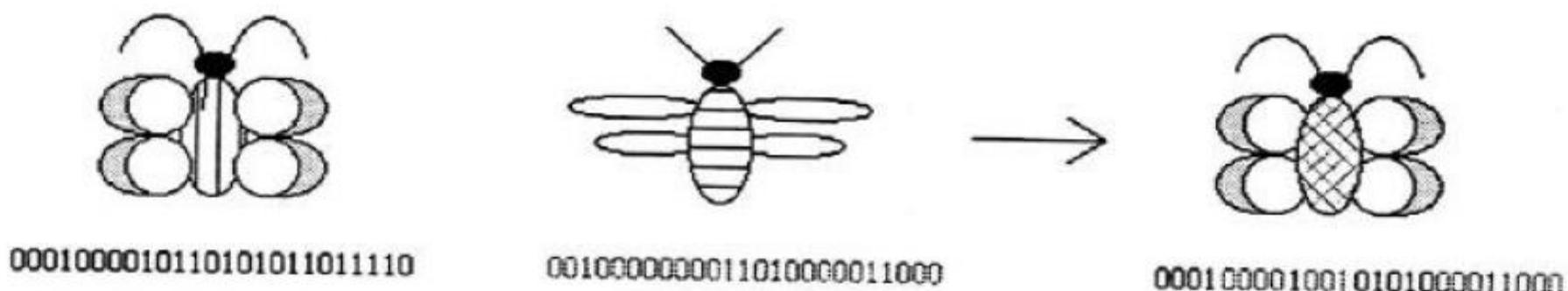


Fig. Uniform crossover

□ Fitness Function

- Head Shape triangle
- Body Color no preference
- Wings no preference
- Size biggest

□ Parameters

- Crossover single point
- Crossover rate 80%
- Mutation random change bit value
- Mutation rate 1%
- Selection Roulette Wheel
- Stopping Best fitness
- Number of generations 10

Example- a Butterfly

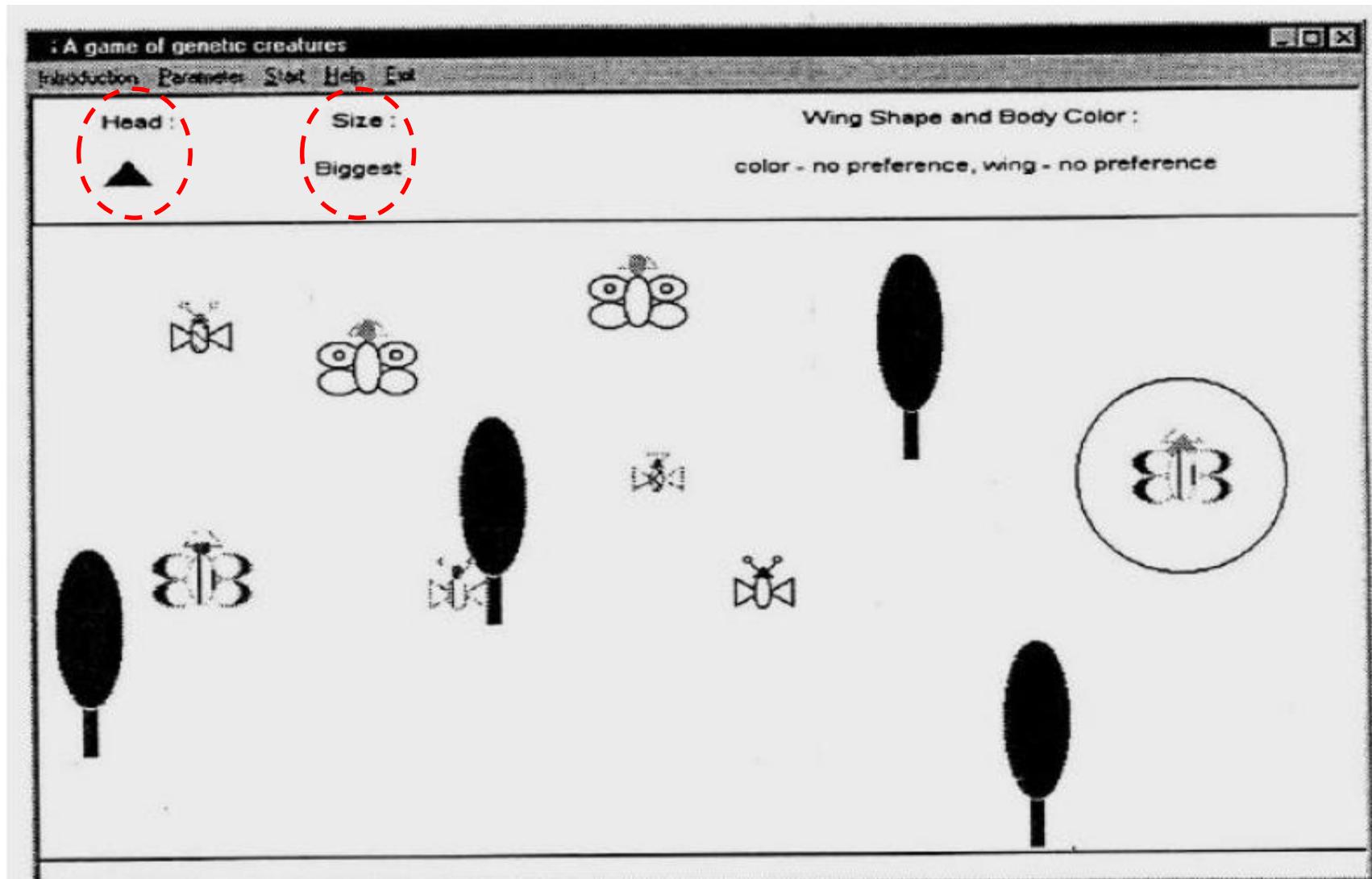


Fig.

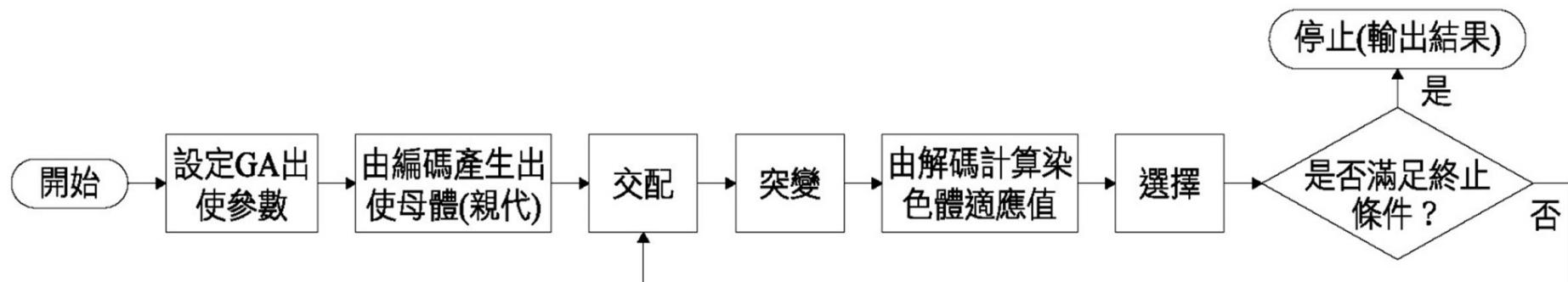
A sample run of the program

□ 基因演算法

- 將最佳化問題的一組解，透過設計**染色體表現**（chromosome representation）以編碼（encoding）的方式把解轉換成染色體表示
 - 由多個基因的組合所構成，並根據**母體大小**產生多個染色體，進行同步搜尋
- 核心思維
 - 在**編碼空間**（encoding space）中利用演化機制，將多個染色體不斷地**交配與突變**產生下一代（offspring）。
 - 再依據**解碼**（decoding）過程將染色體還原成原始問題的解，以計算該組解的**適應值**（fitness value）（又稱目標函數或損失函數）。
 - 最後，透過**選擇機制**（selection mechanism）依據染色體對應的適應值來選擇能保留下來的子代，也就是「適者生存，不適者淘汰」的概念，直到演算法停止條件被滿足。
 - 因此，若將演化機制對應到元啟發式演算法的探索與開採權衡，**交配與突變是對解的探索，而選擇則是對解的開採**。

□ 演算法程序

- 第一：初始化(initialization)階段需要設定母體大小、交配率、突變率、演算法終止條件等。
- 第二：將欲求解的最佳化問題的解進行編碼(encoding)，並根據母體大小隨機產生多個染色體作為母體，形成初始的親代(parents)。
- 第四：開始展開迭代，染色體進行交配與突變從而產生子代(offspring)。
- 第五：每個染色體經解碼(decoding)後分別計算對應到的適應值，並選擇優良的保留。
- 最後，若終止條件被滿足則演算法收斂。



□ 編碼與解碼

- 在基因演算法的迭代過程中，解空間（solution space）與編碼空間（coding space）中不斷地切換運行。解空間中執行的是適應值評估與選擇，編碼空間中執行的則是突變與交配，如圖所示。為了能在編碼空間進行基因的操作，我們必須設計特定的機制將解轉換到編碼空間，也就是編碼機制。

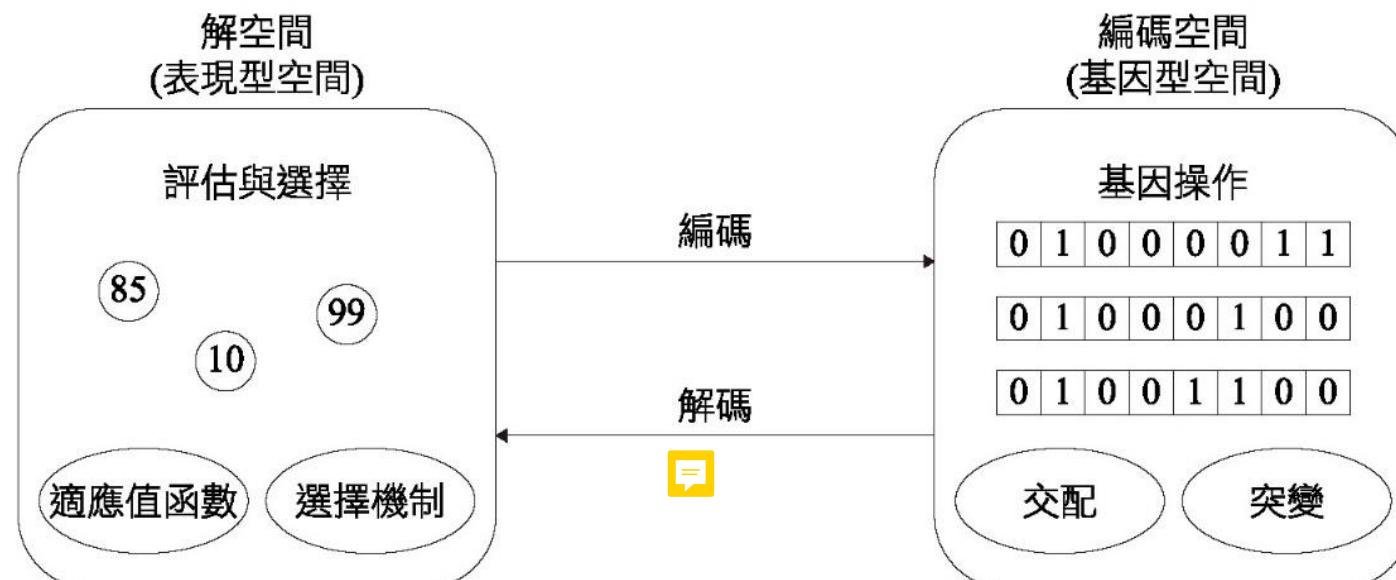


圖 16.5 編碼與解碼的空間轉換

□ 編碼與解碼

- 編碼機制主要有四種類型

- 「二元編碼」(binary encoding)

— 以二元編碼為例，若想設計一個染色體代表一個連續的數值，並且介於0~10之間，可設計一個具有8個二元編碼基因的染色體，如圖所示。這便是將此範圍的數值切割成 $2^8=256$ 個位元，因此每個位元約表示連續數值的 $10/256 \approx 0.04$ ，可推導出該染色體的解碼如公式所示，將編碼還原到原始解空間。

— $10 \cdot \frac{(0 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 0 \times 2^7)}{2^{8-1}} = 1.72$

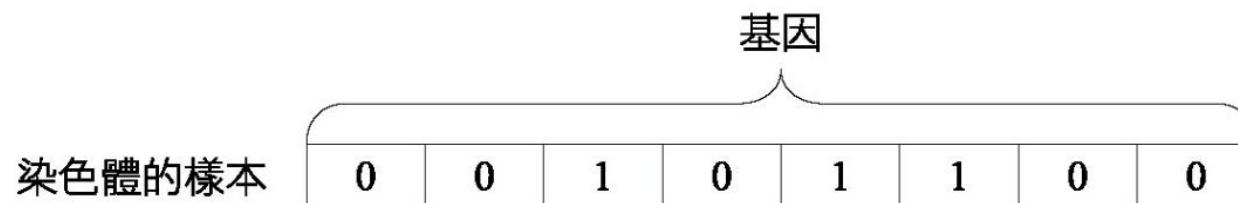


圖 16.6 二元編碼

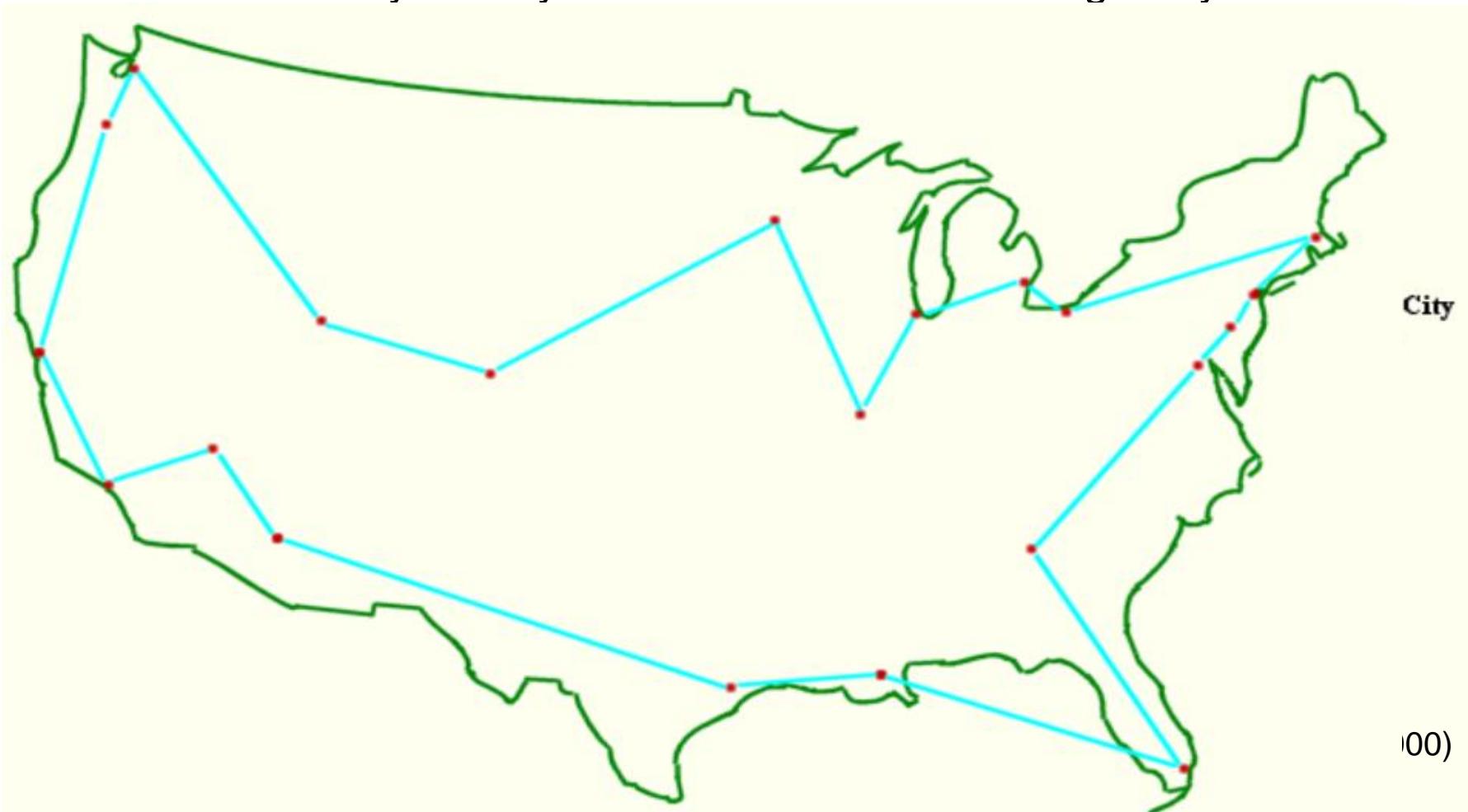
- 因此，當染色體的基因越多，則解析度 (resolution) 越高（可精確到小數點後愈多位數），但這也同時增加了執行基因演算法所需的計算資源。

□ 編碼與解碼

- 「實數編碼」 (real value encoding)
- 「排序編碼」 (permutation encoding)
 - 在旅行推銷員問題中，我們可以排序編碼設計一個染色體代表一個路徑 (path)，其中每個基因以整數表示節點序號，代表所有需要經過的地點。
 - 例如，一個染色體為(1,2,3,4,5,6,7,8)代表行經路徑為從地點1出發，依著地點 2、接著的地點3的順序一直到地點8之後，再回到原地點1。此時計算總行經距離作為適應值，評估該組解 (染色體) 所呈現路徑的好壞，以最小化總距離。
 - 又或可以試想設計一個染色體代表一個單機「流程式排程」 (flow shop scheduling) 的解，以整數代表工件序號，而基因順序便可直接代表加工順序。
- 「樹狀編碼」 (tree encoding)

□ Travel Salesman Problem (TSP)

- Given a set of cities and pairwise distances, to find the shortest possible route that visits each city exactly once and returns to the origin city



- 從城市1出發，必須拜訪每個城市且僅能拜訪一次，然後再回到城市1，最佳路徑為何？

表 14.1 各城市間的距離（單位：公里）

	1	2	3	4	5	6	7	8
1	—	91.8	105.2	89.9	189.9	76.2	278.3	54.4
2	91.8	—	187.2	38.9	271.3	162.9	363.3	88.4
3	105.2	187.2	—	194.1	182.3	31.4	176.1	153.8
4	89.9	38.9	194.1	—	249.4	166.1	368.3	63.6
5	189.9	271.3	182.3	249.4	—	168.0	243.0	185.9
6	76.2	162.9	31.4	166.1	168.0	—	202.2	122.8
7	278.3	363.3	176.1	368.3	243.0	202.2	—	320.0
8	54.4	88.4	153.8	63.6	185.9	122.8	320.0	—

□ 交配與突變

- 基因操作 (genetic operations) 是以交配與突變兩種機制使親代的染色體產生子代的染色體，也就是**產生新的候選解**，藉由增加解的多樣性來實踐對解的探索，同時也能使解逃脫出區域最佳解。
 - 交配或突變的形式沒有一定的或最好的，會依據最佳化問題的特性有著相對合適的形式。

□ 交配

- 兩個染色體的部分基因進行交換重組，產生新染色體（候選解）
 - 「單點交配」 (single point crossover)
 - 隨意選取某個基因位置（又稱為等位基因 *allele*）作為交配點（crossover point），並依此交配點將親代染色體切成兩段，接著將某段固定，另一段互相交換，產生兩個新的子代。
 - 「多點交配」 (multi-point crossover)
 - 多點交配：多點交配的概念與單點交配相似，只是拓展為一次選取多個基因位置當成交配點，並設定交換與固定的區段，以產生新的子代。
 - 「均勻交配」 (uniform crossover)
 - 先隨機產生一條與親代染色體等長的二元編碼作為交配遮蔽（crossover mask），當交配遮蔽內的值為1時，則其對應兩親代的基因進行交換，以產生新的子代。
 - 然而，實際上並非所有親代的染色體都需進行交配，我們會設定一個交配率（crossover rate）作為超參數（hyperparameter），作為決定任意兩條染色體是否要進行交配的機率。這是因為交配通常會使解產生大幅度的變動，而導致解難以收斂。因此，會以某個比例的交配率使解進行探索的同時，**保留部分親代**可能較好結構。

□ 交配與突變

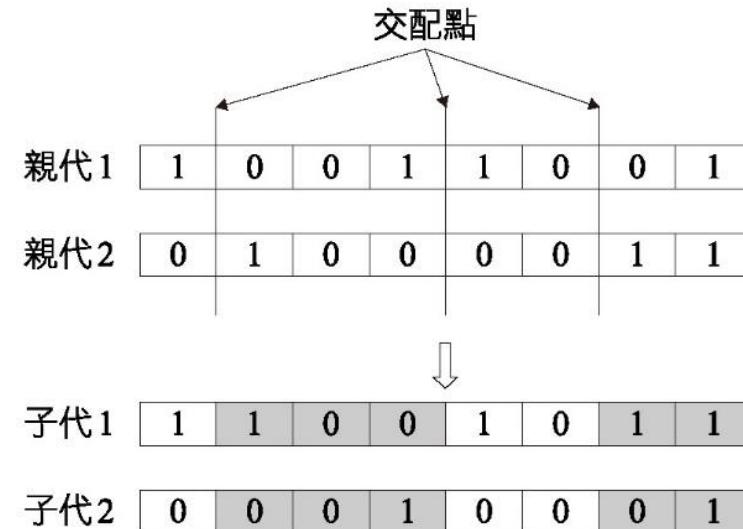
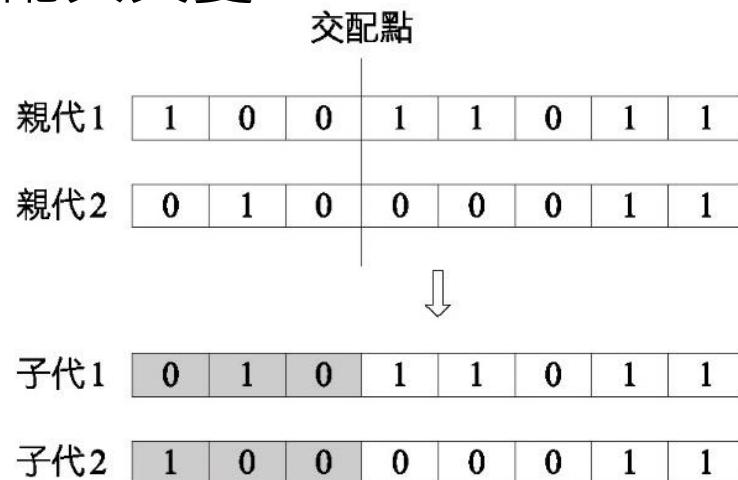


圖 16.7 單點交配與多點交配

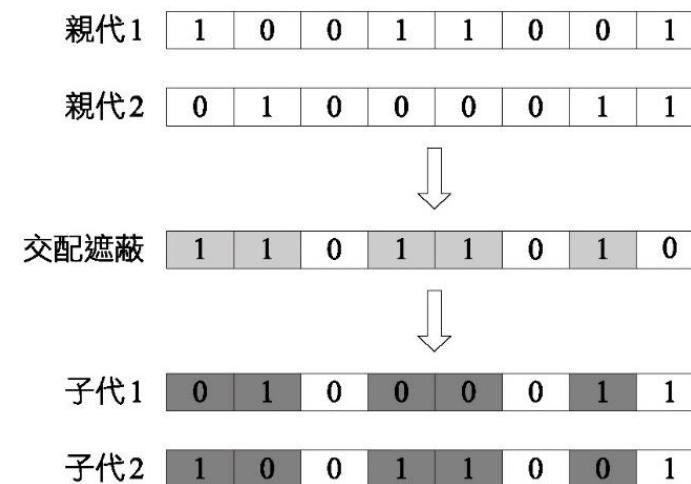


圖 16.8 均匀交配

□ 突變

- 是隨機地改變單一染色體內的部分基因，而同樣地為了不讓所有基因均被突變（過度地探索而導致發散），我們會設定另一個突變率（mutation rate）作為超參數，以決定每個基因被突變的機率。常見的形式有「單點突變」（single point mutation），例如二元編碼中將突變的基因由0變1或由1變0，如圖所示。

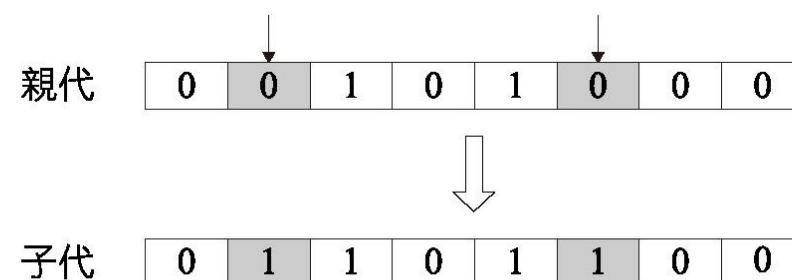


圖 16.9 單點突變

□ 適應值評估與選擇機制

- 基於上述對染色體的交配與突變後，將產生多樣化的子代，進行解空間的探索。
- 下一步即是決定要保留哪些染色體作為下一個親代，也就是進行解空間的開採
 - 「適應值評估」 (fitness evaluation)
 - 「選擇機制」 (selection)

□ 適應值評估

- 是將子代的染色體進行解碼後還原到解空間，再代入給定的目標函數計算得到對應的適應值。很直觀地，在最大化問題中適應值越大越好，相反地在最小化問題中則適應值越小越好。
- 此外，在基因演算法進行交配與突變後，可能會產生不可行解（無法解碼或不滿足限制式）
 - 修復 (repair)：需進行染色體的修復，而修復一般需藉由領域知識來建構規則化法則，將不可行解調整至可行解
 - 懲罰 (penalize)：我們可對適應值進行懲罰

□ 適應值評估-對不可行解的懲罰

- 透過限制式的轉換，可於適應值中加入一懲罰項(penalty term)。其懲罰大小可透過計算該解離可行解域的距離，愈是不可行其懲罰愈大。因此在後續演算法迭代的選擇過程，不可行解被選到的機會愈小。
- 舉例說明，在一個有限制式的最小化問題

$$\text{Minimize } f(x_1, x_2) = (2x_1^2 - x_2 + 4)^2 - (x_1 - 0.4x_2^2 - 8)^3$$

subject to $x_1^2 + 2x_2 \leq 20$

$5x_1 + 3x_2^2 \leq 40$

$-5 \leq x_1, x_2 \leq 5$



- 我們可設計一個染色體中包含兩段基因序列，分別代表 x_1 與 x_2 。若經過染色體間交配突變產生的候選解不滿足限制式，這時可設計懲罰項以懲罰適應值
- Fitness:** $(2x_1^2 - x_2 + 4)^2 - (x_1 - 0.4x_2^2 - 8)^3 + \lambda_1[(x_1^2 + 2x_2 - 20)_+]^2 + \lambda_2[(5x_1 + 3x_2^2 - 40)_+]^2$
 - 將兩條限制式轉換到適應值函數作為懲罰項，此處以**二次項**作為懲罰，以在選擇機制過程避免挑選到不可行解。其中 λ_1 與 λ_2 為非負的懲罰係數對應到兩條限制式，作為接受不可行解的**容忍度(tolerance)**，且函數 $(\cdot)_+ = \max\{(\cdot), 0\}$ 為非負函數

基因演算法

□ 選擇機制

- 通常在每個世代演進的過程，首先會把歷史迭代中最好或較好的染色體予以保留，此稱作「**菁英保留策略**」(elite preservation strategy)。然而，若僅篩選出適應值最好的染色體並非一個好的選擇機制，這將造成對解的開採過快而可能過早收斂至區域最佳解，又稱為**早熟** (premature)。
- 因此，通常適當地加入**隨機性**於選擇機制中

— 「輪盤法」 (Roulette wheel selection)

➤ 以最大化問題為例，輪盤法是基於每個染色體的適應值作為被選擇的機率，從而透過**隨機亂數**(random number)進行挑選。這好比是將一個輪盤依各個染色體的適應值劃分圓餅圖的面積，並以固定數量的飛鏢射向轉動輪盤，決定所選擇的染色體，如圖所示。

➤ 此外，也可將染色體依適應值先做**排序**(ranking)，在最大化問題中，可將最差的染色體排序為1、次差的為2，以此類推，再用排序結果來進行機率的計算。若是最小化問題，則可由適應值的**倒數**，或是適應值由小到大給予1、2、3...的排序後再取倒數，作為機率計算的依據。

— 「競爭選取法」 (tournament selection)

➤ 以最大化問題為例，競爭選取法是從子代中隨機挑選若干個染色體出來從中選擇適應值最大的，**加入到下一世代的親代中**。並重複此動作直到選擇的染色體數量達到設定的母體大小。

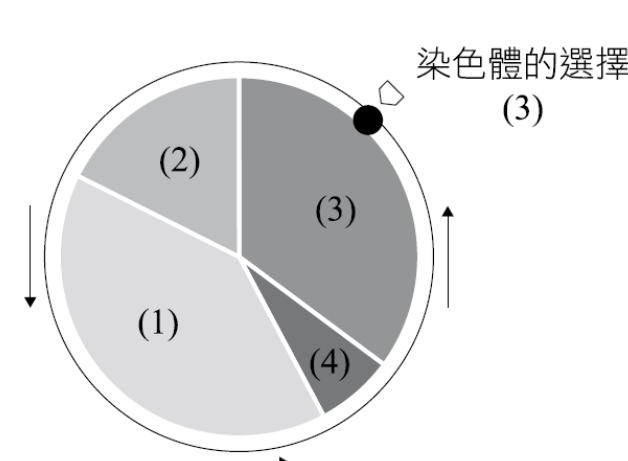


圖 16.10 輪盤法

□ 終止條件

- 終止條件是決定演算法收斂的機制，一旦尚未達到所設定的終止條件，則會不斷地產生新的子代，也就是進行基因操作與選擇機制，直到終止條件被滿足，並由所有染色體中找出可能的近似全域最佳解。常見的終止條件如下所示，
 - 設定固定的迭代次數 (number of iterations)
 - **設定當前最佳解的適應值在連續經歷幾個世代後依然不變的次數**
 - 設定當前最佳解的適應值小於某個閥值 (或改善幅度小於某個百分比)
- 最後，基因演算法的績效往往跟染色體表現的**設計** (design of chromosome representation) 十分相關。
- 此外，對於元啟發式演算法中的**超參數**，也可適當地加入**自適應參數控制** (adaptive control) 機制，使得超參數隨著迭代過程可動態調整，來取得開採與探索間的平衡。
 - 舉例來說，交配率或突變率可以隨著迭代過程動態調整，當母體中染色體相似時太早收斂，可以提高交配率或突變率來增加探索的機會；相反地，如果母體中染色體差異很大而導致演算法難以收斂，這時可調低交配率或突變率來達到收斂（這思維接近於**模擬退火法**的溫度於每個迭代乘上一個冷卻率）。

Unconstrained Nonlinear Optimization Problem

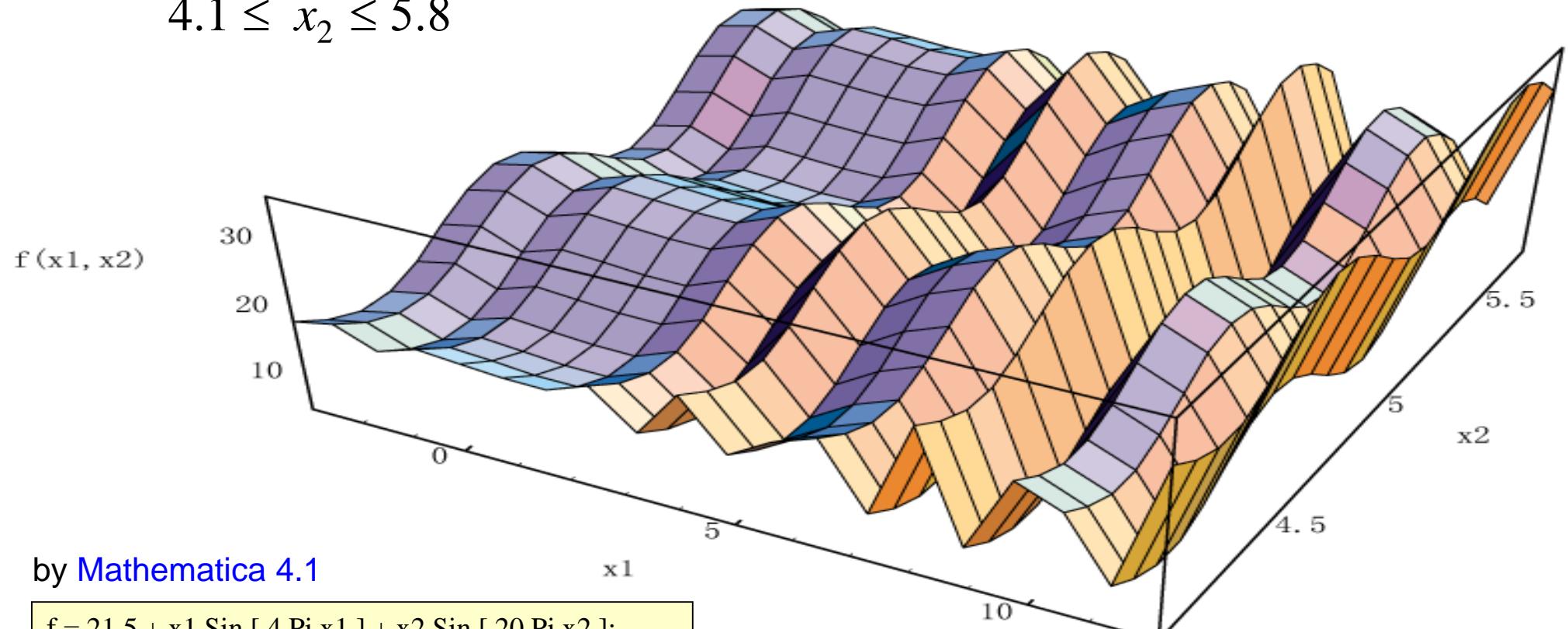
□ Unconstrained Optimization Problem

- We explain in detail about how a genetic algorithm actually works with a simple examples.
- We follow the approach of implementation of genetic algorithms given by Michalewicz.
 - Michalewicz, Z.: *Genetic Algorithm + Data Structures = Evolution Programs*. 3rd ed., Springer-Verlag: New York, 1996.
- The numerical example of unconstrained optimization problem is given as follows:

$$\begin{aligned} \max \quad & f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2) \\ \text{s. t.} \quad & -3.0 \leq x_1 \leq 12.1 \\ & 4.1 \leq x_2 \leq 5.8 \end{aligned}$$

Unconstrained Optimization

$$\begin{aligned} \text{max } & f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2) \\ \text{s. t. } & -3.0 \leq x_1 \leq 12.1 \\ & 4.1 \leq x_2 \leq 5.8 \end{aligned}$$



by Mathematica 4.1

```
f = 21.5 + x1 Sin [ 4 Pi x1 ] + x2 Sin [ 20 Pi x2 ];
Plot3D[f, {x1, -3, 12.1}, {x2, 4.1, 5.8},
PlotPoints -> 19,
AxesLabel -> {x1, x2, "f(x1, x2)"}];
```

- For solving the numerical example:

$$\begin{aligned} \max \quad & f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2) \\ \text{s. t.} \quad & -3.0 \leq x_1 \leq 12.1 \\ & 4.1 \leq x_2 \leq 5.8 \end{aligned}$$

- We give the Binary String Representation

- The domain of x_j is $[a_j, b_j]$ and the required precision is five places after the decimal point.
- The precision requirement implies that the range of domain of each variable should be divided into at least $(b_j - a_j) \times 10^4$ size ranges.
- The required bits (denoted with m_j) for a variable is calculated as follows:

$$2^{m_j-1} < (b_j - a_j) \times 10^4 \leq 2^{m_j} - 1$$

- The mapping from a binary string to a real number for variable x_j is completed as follows:

$$x_j = a_j + \text{decimal}(\text{substring}_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$$

- Binary String Encoding

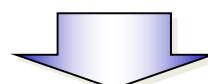
- The precision requirement implies that the range of domain of each variable should be divided into at least $(b_j - a_j) \times 10^4$ size ranges.
- The required bits (denoted with m_j) for a variable is calculated as follows:

$$x_1 : (12.1 - (-3.0)) \times 10,000 = 151,000$$

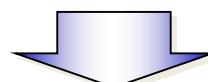
$$2^{17} < 151,000 \leq 2^{18}, \quad m_1 = 18 \text{ bits}$$

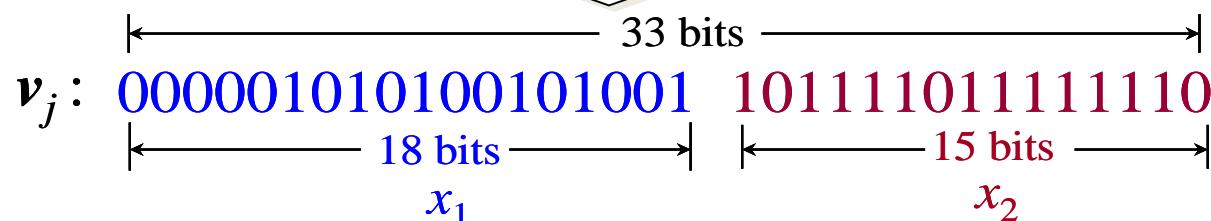
$$x_2 : (5.8 - 4.1) \times 10,000 = 17,000$$

$$2^{14} < 17,000 \leq 2^{15}, \quad m_2 = 15 \text{ bits}$$



precision requirement: $m = m_1 + m_2 = 18 + 15 = 33 \text{ bits}$

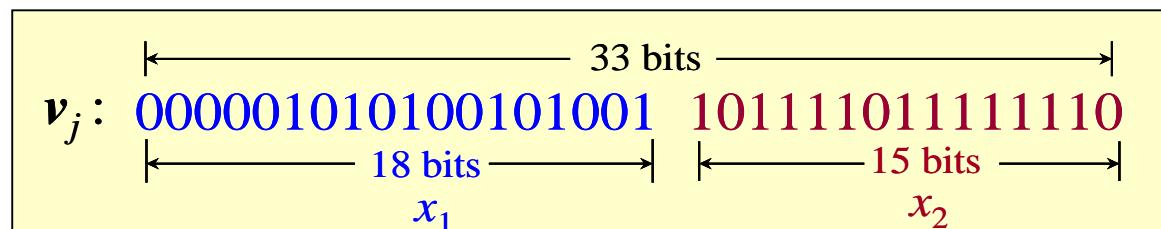


$v_j :$ 

x_1 x_2

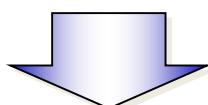
- Binary String Decoding

- The mapping from a binary string to a real number for variable x_j is completed as follows:



	Binary Number	Decimal Number
x_1	000001010100101001	5417
x_2	101111011111110	24318

$$x_j = a_j + \text{decimal}(\text{substring}_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$$



$$x_1 = -3.0 + 5417 \times \frac{12.1 - (-3.0)}{2^{18} - 1} = -2.687069$$

$$x_2 = 4.1 + 24318 \times \frac{5.8 - 4.1}{2^{15} - 1} = 5.361653$$

- Initial Population
 - In general, there are two ways to generate the initial population satisfying system constraints, **heuristic initialization** and **random initialization**.
 - Although the mean fitness of the heuristic initialization is already high so that it may help the GAs to find solutions faster.
 - Unfortunately, in most large scale problems, it may just explore a small part of the solution space and it is difficult to find global optimal solutions because of the lack of diversity in the population.
 - C. W. Ahn and R. S. Ramakrishna, “A genetic algorithm for shortest path routing problem and the sizing of populations,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 6, pp. 566-579, Dec. 2000.
 - Therefore, random initialization satisfying system constraints is effected in the most of **NP hard** optimization problems.

- Initial population satisfying system constraints is randomly generated as follows:

$$v_1 = [0000010101001010011011101111110] = [x_1 \ x_2] = [-2.687969 \ 5.361653]$$

$$v_2 = [00111010110011000000010101001000] = [x_1 \ x_2] = [0.474101 \ 4.170144]$$

$$v_3 = [111000111000001000010101001000110] = [x_1 \ x_2] = [10.419457 \ 4.661461]$$

$$v_4 = [100110110100101101000000010111001] = [x_1 \ x_2] = [6.159951 \ 4.109598]$$

$$v_5 = [000010111101100010001110001101000] = [x_1 \ x_2] = [-2.301286 \ 4.477282]$$

$$v_6 = [111110101011011000000010110011001] = [x_1 \ x_2] = [11.788084 \ 4.174346]$$

$$v_7 = [11010001001111000100110011101101] = [x_1 \ x_2] = [9.342067 \ 5.121702]$$

$$v_8 = [001011010100001100010110011001100] = [x_1 \ x_2] = [-0.330256 \ 4.694977]$$

$$v_9 = [111110001011101100011101000111101] = [x_1 \ x_2] = [11.671267 \ 4.873501]$$

$$v_{10} = [111101001110101010000010101101010] = [x_1 \ x_2] = [11.446273 \ 4.171908]$$

- **Evaluation**

- The process of **evaluating the fitness** of a chromosome consists of the following three steps:

input: chromosome v_k , $k=1, 2, \dots, popSize$

output: the fitness $eval(v_k)$

step 1: Convert the chromosome's genotype to its phenotype, i.e., convert binary string into relative real values $x_k = (x_{k1}, x_{k2}), k = 1, 2, \dots, popSize$.

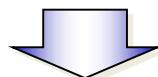
step 2: Evaluate the fitness by the objective function $f(x_k), k = 1, 2, \dots, popSize$.

$$eval(v_k) = f(x_k), k = 1, 2, \dots, popSize$$

step 3: When treating the minimization problem, the fitness is simply equal to the value of $1 / f(x_k), k = 1, 2, \dots, popSize$.

$$eval(v_k) = f(x_{ki}) \quad (k = 1, 2, \dots, popSize) \\ (i = 1, 2, \dots, n)$$

$$f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$



Example: $(x_1 = -2.687969, x_2 = 5.361653)$

$$eval(v_1) = f(-2.687969, 5.361653) = 19.805119$$

- An evaluation function plays **the role of the environment**, and it rates chromosomes in terms of **their fitness**.
- The fitness function values of above chromosomes are as follows:

$$\text{eval}(v_1) = f(-2.687969, 5.361653) = 19.805119$$

$$\text{eval}(v_2) = f(0.474101, 4.170144) = 17.370896$$

$$\text{eval}(v_3) = f(10.419457, 4.661461) = 9.590546$$



$$\text{eval}(v_4) = f(6.159951, 4.109598) = 29.406122$$

$$\text{eval}(v_5) = f(-2.301286, 4.477282) = 15.686091$$

$$\text{eval}(v_6) = f(11.788084, 4.174346) = 11.900541$$

$$\text{eval}(v_7) = f(9.342067, 5.121702) = 17.958717$$

$$\text{eval}(v_8) = f(-0.330256, 4.694977) = 19.763190$$

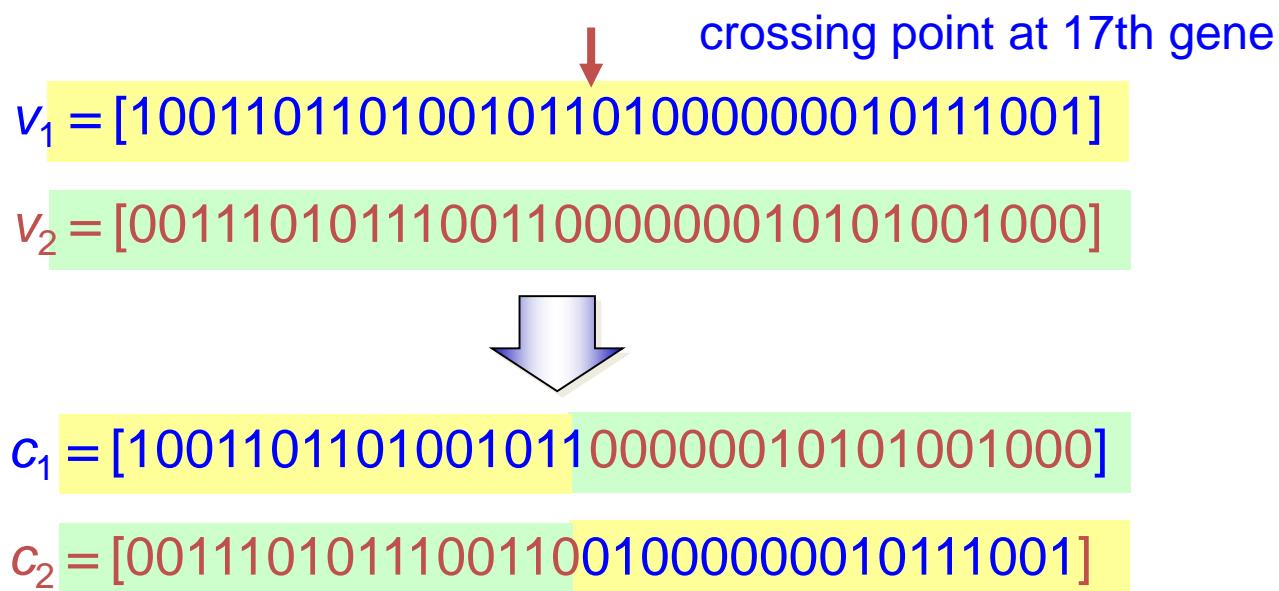
$$\text{eval}(v_9) = f(11.671267, 4.873501) = 26.401669$$

$$\text{eval}(v_{10}) = f(11.446273, 4.171908) = 10.252480$$

- It is clear that chromosome v_4 is the strongest one and that chromosome v_3 is the weakest one.

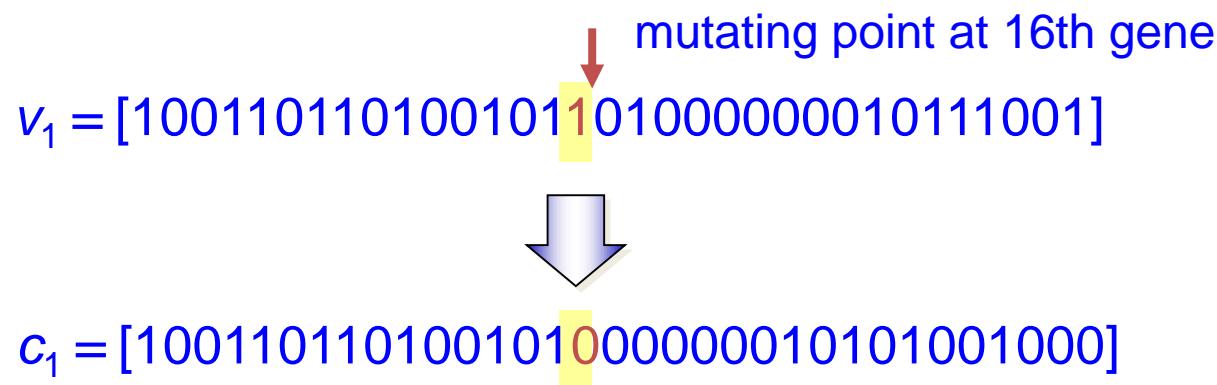
- Crossover (One-cut point Crossover)

- Crossover used here is one-cut point method, which random selects one cut point.
- **Exchanges** the right parts of two parents to generate offspring.
- Consider two chromosomes as follow and the cut point is randomly selected after the 17th gene:



- Mutation

- Alters one or more genes with a probability equal to the mutation rate.
- Assume that the 16th gene of the chromosome v_1 is selected for a mutation.
- Since the gene is 1, it would be flipped into 0. So the chromosome after mutation would be:



Assume that $p_M = 0.01$

<i>bitPos</i>	<i>chromNo k</i>	<i>bitNo j</i>	<i>randomNum r</i>
105	4	6	0.009857
164	5	32	0.003113
199	7	1	0.000946
329	10	32	0.001282

- Selection:

- In most practices, a **roulette wheel approach** is adopted as the selection procedure, which is one of the fitness-proportional selection and can select a new population with respect to the probability distribution based on fitness values.
- The **roulette wheel** can be constructed with the following steps:

input: population $P(t-1)$, $C(t-1)$

output: population $P(t)$, $C(t)$

step 1: Calculate the total fitness for the population

$$F = \sum_{k=1}^{\text{popSize}} \text{eval}(\mathbf{v}_k)$$

step 2: Calculate selection probability p_k for each chromosome \mathbf{v}_k

$$p_k = \frac{\text{eval}(\mathbf{v}_k)}{F}, \quad k = 1, 2, \dots, \text{popSize}$$

step 3: Calculate cumulative probability q_k for each chromosome \mathbf{v}_k

$$q_k = \sum_{j=1}^k p_j, \quad k = 1, 2, \dots, \text{popSize}$$

step 4: Generate a random number r from the range $[0, 1]$.

step 5: If $r \le q_1$, then select the first chromosome \mathbf{v}_1 ; otherwise, select the k th chromosome \mathbf{v}_k ($2 \le k \le \text{popSize}$) such that $q_{k-1} < r \le q_k$.

- Illustration of Selection:

input: population $P(t-1)$, $C(t-1)$

output: population $P(t)$, $C(t)$

step 1: Calculate the total fitness F for the population.

$$F = \sum_{k=1}^{10} eval(v_k) = 178.135372$$

step 2: Calculate selection probability p_k for each chromosome v_k .

$$\begin{aligned} p_1 &= 0.111180, & p_2 &= 0.097515, & p_3 &= 0.053839, & p_4 &= 0.165077, \\ p_5 &= 0.088057, & p_6 &= 0.066806, & p_7 &= 0.100815, & p_8 &= 0.110945, \\ p_9 &= 0.148211, & p_{10} &= 0.057554 \end{aligned}$$

step 3: Calculate cumulative probability q_k for each chromosome v_k .

$$\begin{aligned} q_1 &= 0.111180, & q_2 &= 0.208695, & q_3 &= 0.262534, & q_4 &= 0.427611, \\ q_5 &= 0.515668, & q_6 &= 0.582475, & q_7 &= 0.683290, & q_8 &= 0.794234, \\ q_9 &= 0.942446, & q_{10} &= 1.000000 \end{aligned}$$

step 4: Generate a random number r from the range $[0,1]$.

$$\begin{array}{ccccc} 0.301431, & 0.322062, & 0.766503, & 0.881893, & 0.350871, \\ 0.583392, & 0.177618, & 0.343242, & 0.032685, & 0.197577 \end{array}$$

- Illustration of Selection:

step 5: $q_3 < r_1 = 0.301432 \leq q_4$, it means that the chromosome v_4 is selected for new population; $q_3 < r_2 = 0.322062 \leq q_4$, it means that the chromosome v_4 is selected again, and so on. Finally, the new population consists of the following chromosome.

$$v'_1 = [10011011010010110100000010111001] \quad (v_4)$$

$$v'_2 = [10011011010010110100000010111001] \quad (v_4)$$

$$v'_3 = [001011010100001100010110011001100] \quad (v_8)$$

$$v'_4 = [111110001011101100011101000111101] \quad (v_9)$$

$$v'_5 = [10011011010010110100000010111001] \quad (v_4)$$

$$v'_6 = [11010001001111000100110011101101] \quad (v_7)$$

$$v'_7 = [00111010111001100000001010100100] \quad (v_2)$$

$$v'_8 = [10011011010010110100000010111001] \quad (v_4)$$

$$v'_9 = [00000101010010100110111101111110] \quad (v_1)$$

$$v'_{10} = [00111010111001100000001010100100] \quad (v_2)$$

- Next Generation

 $v_1' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122$ $v_2' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122$ $v_3' = [001011010100001100010110011001100], f(-0.330256, 4.694977) = 19.763190$ $v_4' = [1111000101110110001110100011101], f(11.907206, 4.873501) = 5.702781$ $v_5' = [100110110100101101000000010111001], f(8.024130, 4.170248) = 19.91025$ $v_6' = [11010001001111000100110011101101], f(9.34067, 5.121702) = 17.958717$ $v_7' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122$ $v_8' = [100110110100101101000000010111001], f(6.159951, 4.109598) = 29.406122$ $v_9' = [000001010100100110111011111110], f(-2.687969, 5.361653) = 19.805199$ $v_{10}' = [001110101110011000000010101001000], f(0.474101, 4.170248) = 17.370896$

- Final Result (Sample Case)

- The test run is terminated after 1000 generations.
- We obtained the best chromosome in the 884th generation as follows:

$$\max \quad f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

$$\begin{aligned} \text{s. t. } & -3.0 \leq x_1 \leq 12.1 \\ & 4.1 \leq x_2 \leq 5.8 \end{aligned}$$

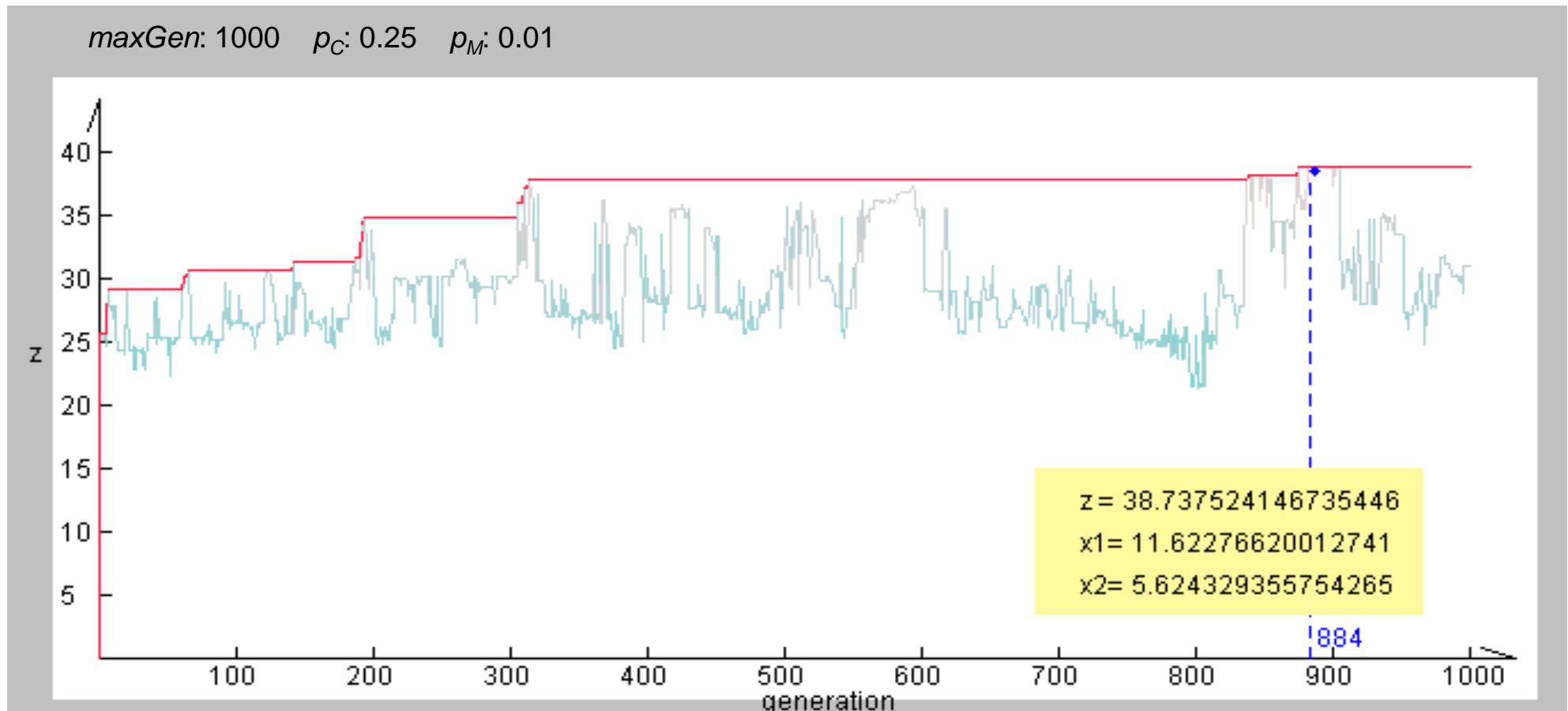
$$\begin{aligned} eval(v^*) &= f(11.622766, 5.624329) \\ &= 38.737524 \end{aligned}$$

$$x_1^* = 11.622766$$

$$x_2^* = 5.624329$$

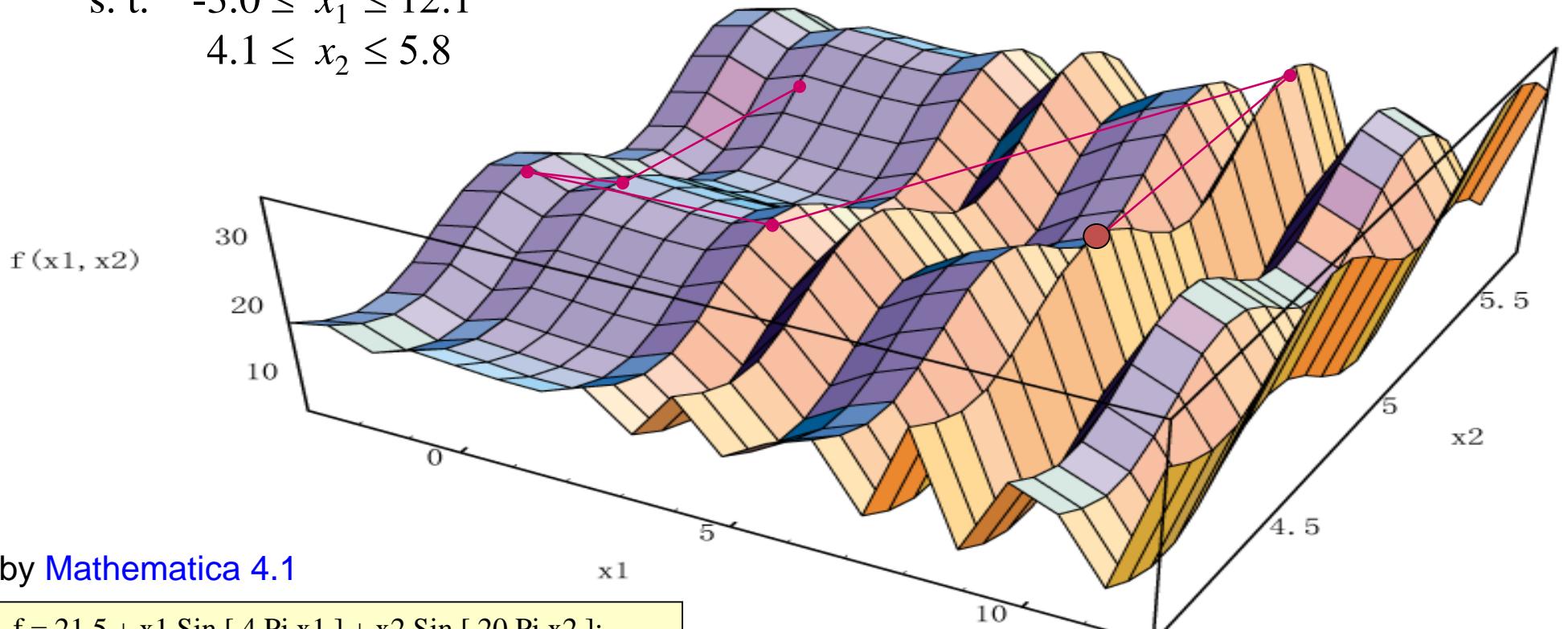
$$f(x_1^*, x_2^*) = 38.737524$$

- Evolutional Process (Sample Case)



- **Evolutional Process (Sample Case)**

$$\begin{aligned} \max \quad & f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2) \\ \text{s. t.} \quad & -3.0 \leq x_1 \leq 12.1 \\ & 4.1 \leq x_2 \leq 5.8 \end{aligned}$$



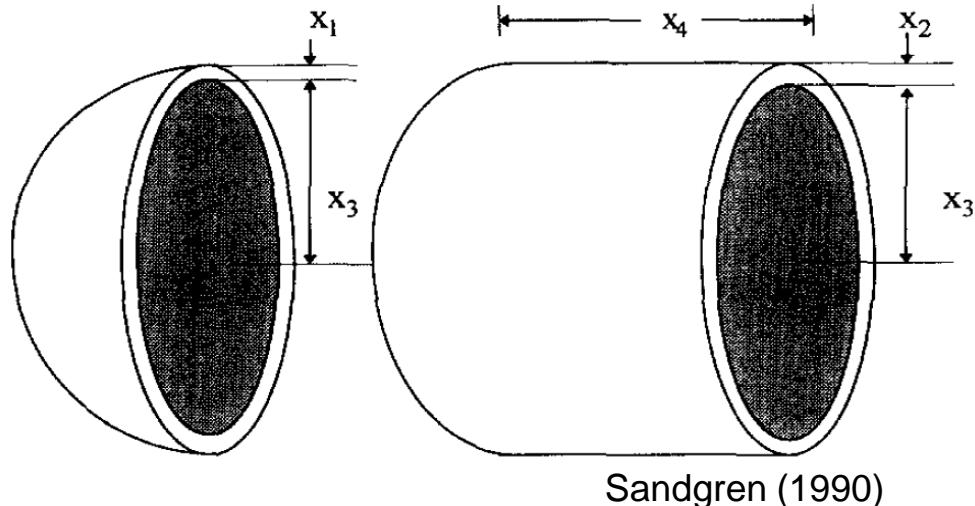
by [Mathematica 4.1](#)

```
f = 21.5 + x1 Sin [ 4 Pi x1 ] + x2 Sin [ 20 Pi x2 ];
Plot3D[f, {x1, -3.0, 12.1}, {x2, 4.1, 5.8},
PlotPoints -> 19,
AxesLabel -> {x1, x2, "f(x1, x2)"}];
ContourPlot[ f, {x, -3.0, 12.1},{y, 4.1, 5.8}];
```

Can you solve this problem now?

□ Mechanical Design Problem

- Tube and Pressure Vessel
- Objective function
 - Minimize the total volume



$$\text{Minimize } 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

$$\text{subject to } -x_1 + 0.0193x_3 \leq 0,$$

$$-x_2 + 0.00954x_3 \leq 0,$$

$$-\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0,$$

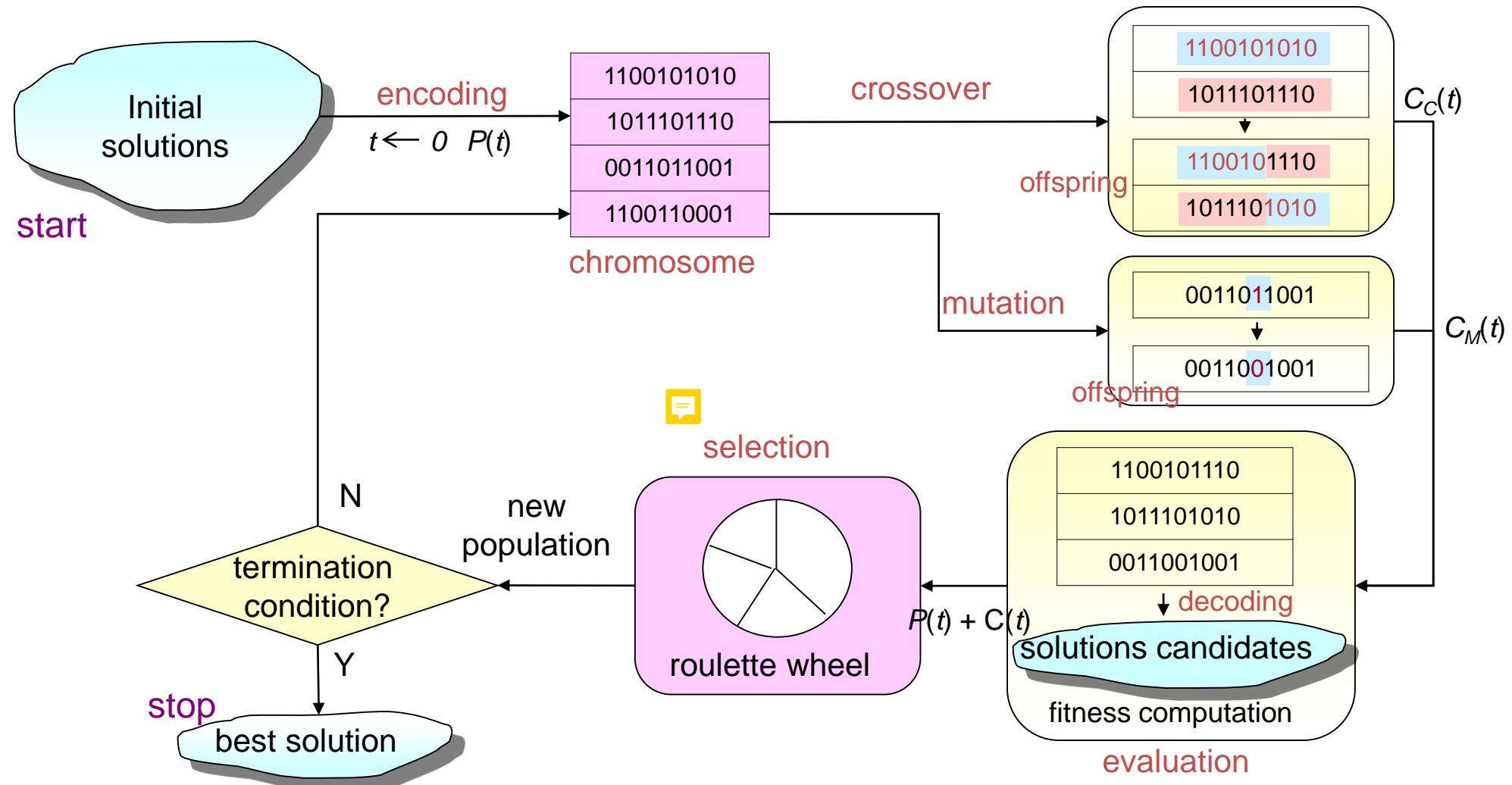
$$-240 + x_4 \leq 0,$$

$$1 \leq x_1 \leq 1.375,$$

$$0.625 \leq x_2 \leq 1, 48 \leq x_3 \leq 52, \quad 90 \leq x_4 \leq 112,$$

What is Genetic Algorithm?

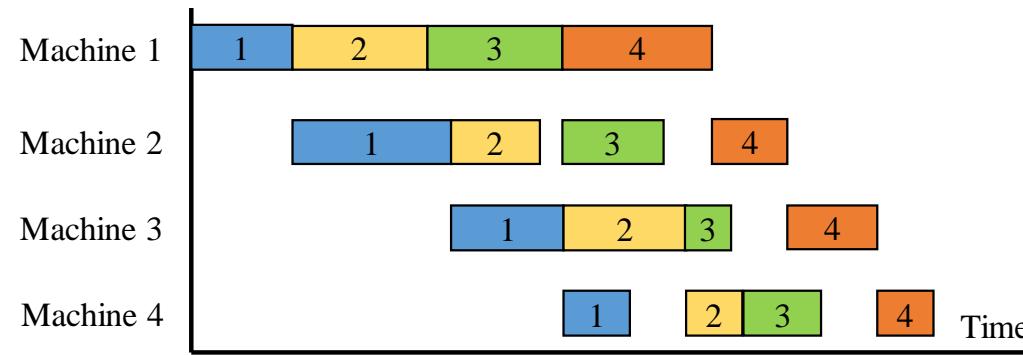
□ The general structure of genetic algorithms



Gen and Cheng (1997)

□ Flow shops

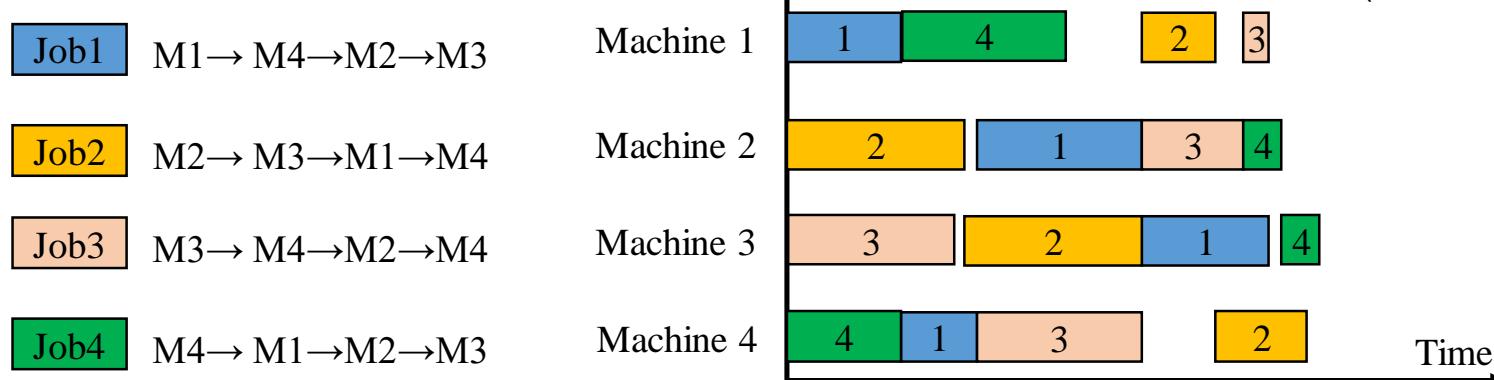
- Each job must follow the same route. (There is a sequence of machines.)



□ Job shops

- When the routes are fixed, but not necessarily the same for each job, the model is called a **job shop**.
- If a job in a job shop has to visit certain machines more than once, the job is said to **recirculate**.

(Yu-Wen Teng, 2019)



□ 工單式排程案例 (job shop scheduling)

- 常見目標為最小化總加權延遲時間，或最大完工時間（makespan）
 - 後者其意義為所有工件加工完成所需的時間，也就是從第一個工件加工開始算起到最後一個工件加工完成為止所經過的全部時間。
- 考慮一個具有三個工件與三台機台的工單式排程為例
 - 其工件根據作業程序所經過的加工時間與機台(P_{ijk}, M_k)。

表 16.5 工件需經過的加工時間與機台

工件 \ 作業程序	O1	O2	O3
Job 1	(P_{112}, M_2)	(P_{123}, M_3)	(P_{131}, M_1)
Job 2	(P_{211}, M_1)	(P_{223}, M_3)	(P_{232}, M_2)
Job 3	(P_{313}, M_3)	(P_{322}, M_2)	(P_{331}, M_1)

□ 工單式排程案例 (job shop scheduling)

- 「基於作業的染色體表示」(operation-based representation)
- 在解空間可將工單式排程以 O_{ijk} 表示(第*i*個工件在第*j*個作業程序使用第*k*台機台)，而在編碼空間則是以一個基因代表要加工的一個工件。
- 其次，由於每個工件有三個作業程序，因而會在不同基因中重複出現三次，而在染色體中同一工件出現的次數代表其作業程序。最後，在給定工件與作業程序後，自然可對應到指定加工的機台。依此邏輯則可一對一在解空間與編碼空間中轉換，從而在基因演算法中進行交配、突變與選擇的演化機制。本案例的解同時可對應至圖「甘特圖」(Gantt chart)

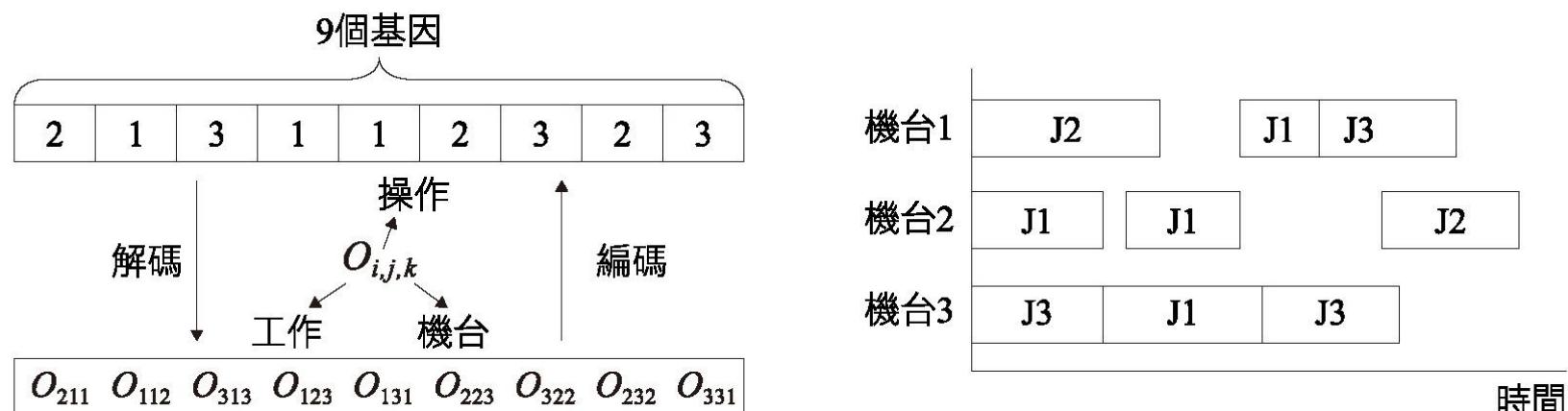
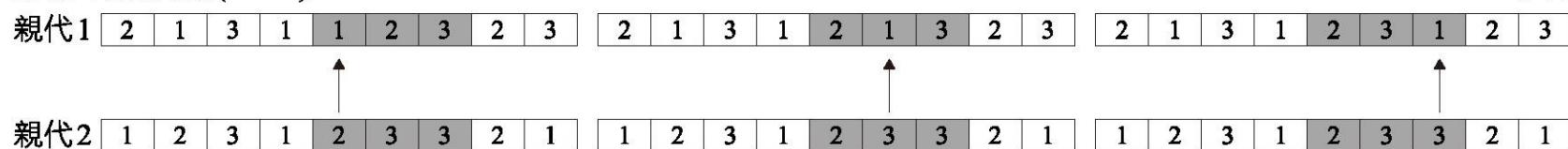


圖 16.11 染色體編碼

□ 工單式排程案例 (job shop scheduling)

- 不同的基因編碼方式，其所使用的交配與突變也可能有所不同。
- 在排程問題中
 - 常用的交配包括「部分映射交配」(partially mapped crossover, PMX) 、「順序交配」(order crossover, OX)
 - 部分映射交配上圖，將親代2的隨機選取區段影射到親代1，一開始親代1中1的位置一旦置換了2，則在第一個染色體中便多了2但少了1，因此從後面去找有2的位置換為1。
 - 順序交配如下圖，將親代1隨機選取區段固定住後，需要補足的基因為一個1、兩個2與兩個3，將從另一個親代依順序補足。

部分映射交配(PMX)



順序交配(OX)

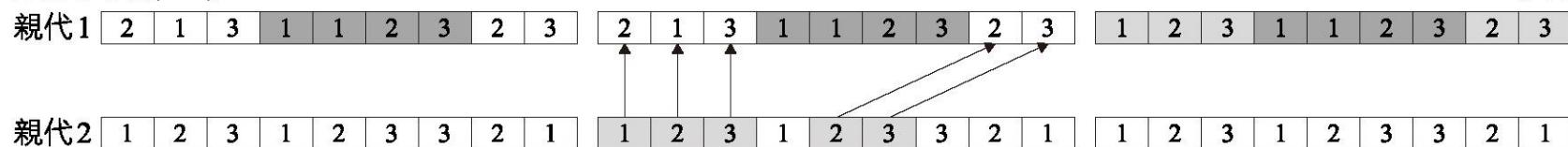


圖 16.12 PMX 與 OX

□ 工單式排程案例 (job shop scheduling)

- 在排程問題中

- 其他常用的突變包括「置換突變」(displacement mutation)、「反轉突變」(reversion mutation)、「插入突變」(insertion mutation)等



圖 16.13 置換突變、反轉突變與插入突變

□ 元啟發式演算法相關的進階議題

- 「迷因演算法」 (memetic algorithm)
 - 提升最佳化精度
- 「多目標最佳化問題」 (multi-objective optimization problem)
 - 具有多個目標函數的
- 「最佳資源預算分配」 (optimal computing budget allocation, OCBA)
 - 適應值具有隨機性時如何將優化運算資源的

□ 迷因演算法

- 迷因演算法 (Krasnogor and Smith, 2005; Nguyen et al., 2007) 是在演化機制的架構下，在迭代的循環中嵌入區域搜尋 (local search) 的能力。簡單來說，迷因演算法是在元啟發式演算法中引入了啟發式演算法。
- 迷因演算法的思維源自於一個族群長時間的演化以及在生命週期中的「迷因學習」 (individual learning)，此方法的命名則是受到迷因 (meme) 一詞的啟發 (迷因指的是文化的傳承、散播，並會依據特定的環境有所進化)。
- 我們經常關注於探索與開採的權衡，舉例來說，為了逃脫區域最佳解所採取的探索策略已被充分地設計於原有的基因演算法中，然而開採策略一般僅用了簡單的選擇機制。因此，**迷因演算法以啟發式演算法強化開採策略**，能更有效率地開採出多個潛在最佳解 (可能包含多個區域最佳解或全域最佳解)，而避免收斂時落入特定的區域最佳解。
- 若以基因演算法作為迷因演算法的基底，是在交配與突變後以及選擇前，引入**區域搜尋**的方法，例如梯度下降法或貪婪演算法，以更新染色體到區域最佳解。

□ 多目標最佳化

- 多目標最佳化問題是指有一個以上的目標函數之最佳化問題
 - $\min/\max f_m(x), m = 1, 2, \dots, M.$
- 通常由於多個目標間存在著衝突（conflict），也就是一個目標變好會導致另一個目標變差。例如從投資觀念裡我們常聽的高風險高報酬、希望薪資所得愈高愈好但工時也隨之拉長、希望機台產出率高（例如車台轉速調升）但可能導致良率不穩或下降（例如產生不良品）、希望在疫情中維持經濟發展但又需要保持社交距離避免群聚等。
- 這些實務問題皆不容易以單一的適應值來判定解的優劣（尤其在未知目標間的重要性或權重），因此其核心思維是如何在多目標之間找到適當的權衡（trade-off）或妥協解（compromise solution）。為了比較解之間的優劣，多目標最佳化引入「凌越解」（dominated solution）的思維，它是用於比較解在不同目標上與其他解的關係。

□ 多目標最佳化

- 如前所述，假設一車台製造五金扣件（例如螺絲、螺帽），欲同時考量轉速（speed）與良率（yield）兩項目標（具有衝突），轉速代表單位時間的生產量，良率代表生產的產品品質是否滿足規格，兩目標皆為愈大愈好（望大特性）。
- 理想上，我們期望設定在具有最大轉速與良率的情況。首先，當我們比較設定D與設定E時，會發現後者在兩目標上均大於前者，我們稱後者被前者所凌越（dominate）。另一方面，設定F被設定E所凌越，因此可表示如 $F < E < D$ 所示。

設定	轉速	良率
A	140	60%
B	130	80%
C	120	85%
D	110	97%
E	100	93%
F	90	88%

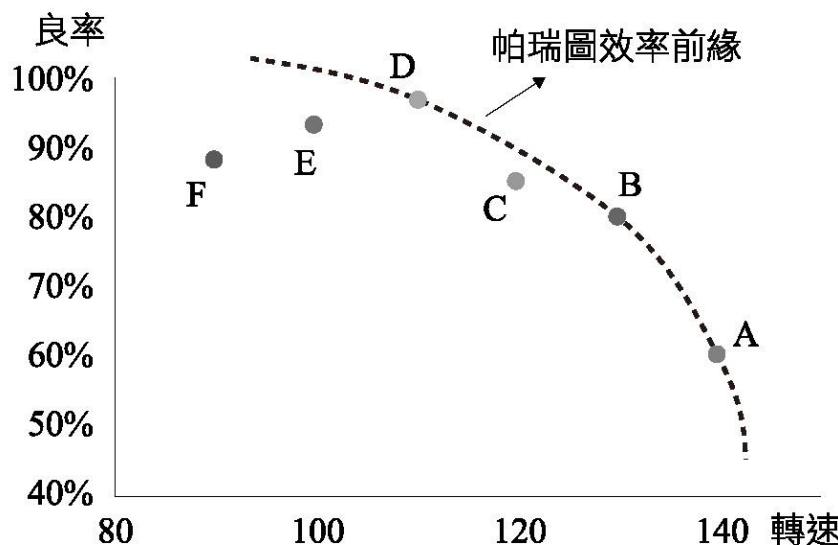


圖 16.14 轉速與良率設定所形成的多目標問題

□ 多目標最佳化

- 當我們比較設定A、B、C與 D時，會發現四個解各自有各自的優勢，在目標上是相互不凌越，因此四者被稱為「非凌越解」（non-dominated solutions）或「帕瑞圖最佳解」（Pareto optimal solutions）〔此並非古希臘哲學家柏拉圖（Plato）〕。也就是這些解同時是多目標最佳化問題中的最佳解。其他常用使用的作法包括：
 - **目標轉換成限制式**：若某些目標是可給定門檻，只要超過或低於此閥值即可滿足的話，便可將此**目標函數轉換為限制式**，用以呈現可行解域，簡化多目標問題。例如給定良率在大於等於90%的情況下來最大化車台轉速，以提高產能。
 - **整合成單一目標**：若能給定各個目標的權重（以呈現重要性），可將多個目標函數以權重整合成單一目標函數，並求解此單目標問題。值得注意的是，由於每個目標的尺度（scale）不同，通常需要先經過歸一化（normalization）後，再進行加權的目標整合。
 - **多準則決策分析（multi-criteria decision analysis, MCDA）**：求得多目標的適應值後，對每個解進行評估。例如使用理想解相似度排序偏好法（TOPSIS）（Hwang and Yoon, 1981），先產生每個目標皆最佳的最優理想解以及每個目標最差的最劣理想解。接著評估每個解距離最優理想解愈近愈好與距離最劣理想解愈遠愈好，來將非凌越解加以排序。此方法用於在每個解已計算出多目標適應值向量的**事後**整合多目標評估方法，與上述一開始就整合成單一目標的事前整合多目標評估方法有所不同。

□ 非凌越排序基因演算法

- 以基因演算法為核心的多目標方法—「非凌越排序基因演算法」(non-dominated sorting genetic algorithm, NSGA-II) (Deb et al., 2002)
- 求解帕瑞圖最佳解的方法
- 它在原有基因演算法的選擇機制前增加了兩個衡量指標
 - 「非凌越排序」(nondominated sorting)
 - 「擁擠距離計算」(crowding-distance calculation)
- 最後，非凌越排序基因演算法依據每個染色體的非凌越層級(nondomination rank)以及擁擠距離進行**選擇機制**，其規則為越高層級的解具有較高的被選擇機率。
 - 接著，若存在解之間的層級相同時，則越大的擁擠距離具有較高的被選擇機率(提升解在多目標之間的多樣性)。
- 這樣的機制使得非凌越排序基因演算法能求解出多目標最佳化問題可能的帕瑞圖前緣。

□ 非凌越排序基因演算法

● 非凌越排序

- 是為了同時衡量多個目標而進行非凌越前緣 (non-dominated front) 的排序，主要包含四個步驟。第一，計算每個解被凌越的個數 n 與其凌越解的集合 S ；第二，找出沒有被凌越的解，形成非凌越前緣，並另其排序等級 (rank) 為 1；第三步依照凌越的個數來排序 (sorting) 等級；最後，我們將得到不同等級的非凌越前緣，如圖所示。

	n	S	Rank
A	0	F	1
B	0	D, E, F	1
C	0	F	1
D	1	F	2
E	1	F	2
F	5	5	3

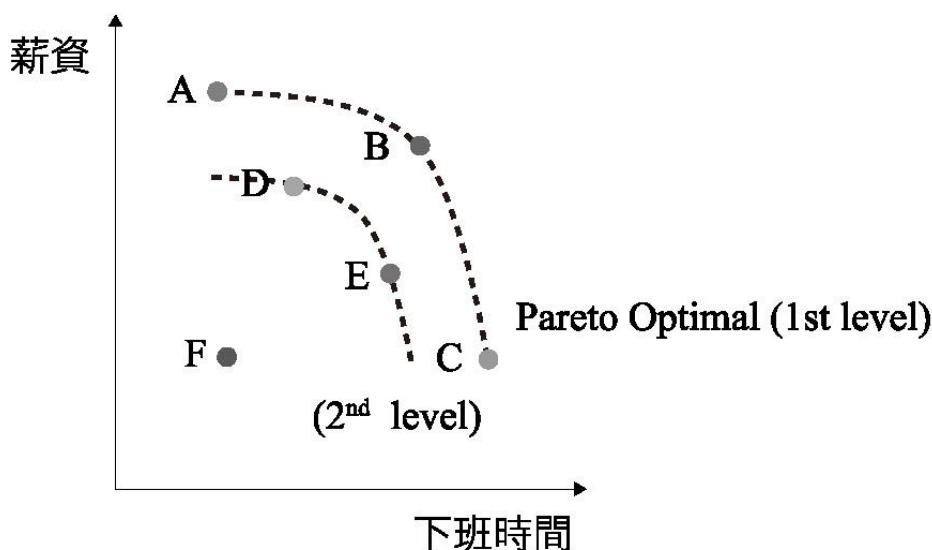


圖 16.15 非凌越前緣的層級排序

□ 非凌越排序基因演算法

● 擁擠距離

- 是為了在開拓最佳的非凌越前緣時，保有多目標在不同權重下的解的**多樣性**，此指標避免非凌越前緣只傾向於特定的目標收斂。而第*i*個解的擁擠距離 $CD(i)$ 的計算如公式所示

$$- D_p(i) = \frac{|f_p(i+1) - f_p(i-1)|}{|\max(f_p) - \min(f_p)|}$$

$$- CD(i) = \sum_{i=1}^p D_p(i)$$

- 其中每個目標以 p 表示，適應值則以函數 f 表示，並且在計算前須依據給定的目標將其適應值由小到大排序，再計算其與前後兩個鄰近解($i \pm 1$)的距離 $D_p(i)$ ，最後再進行每個目標的 $D_p(i)$ 加總。

- 以兩個最大化的目標為例，其第*i*個非凌越前緣解的擁擠距離如圖所示，是長方形虛線的一邊長與寬之和(也就是兩點的**曼哈頓距離**(rectilinear distance, L₁ norm))。

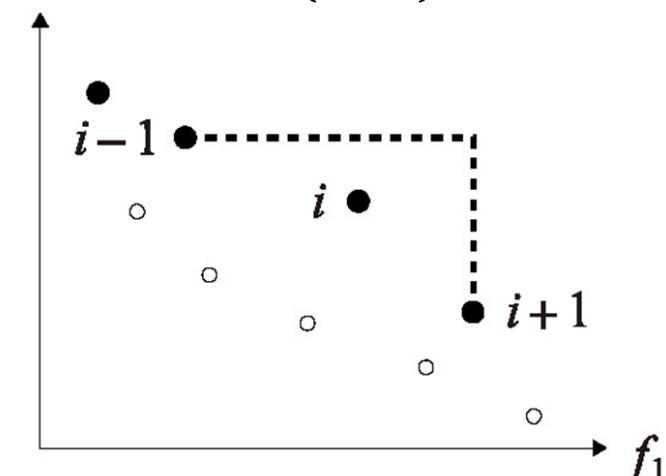


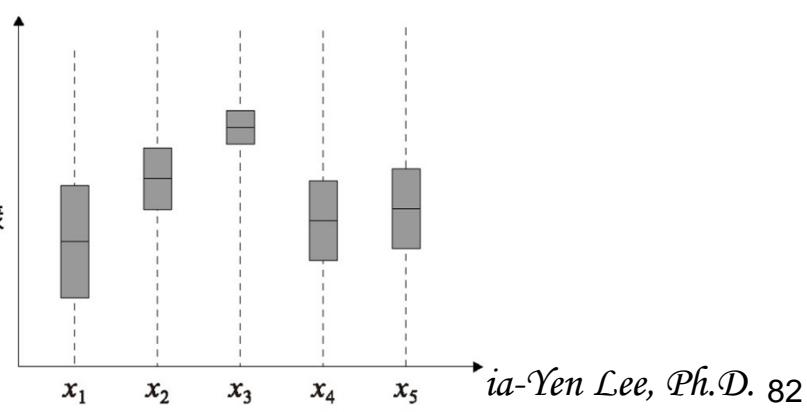
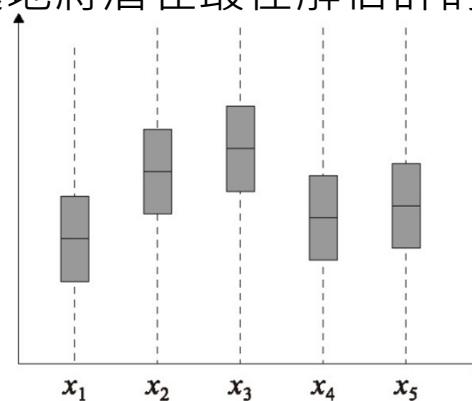
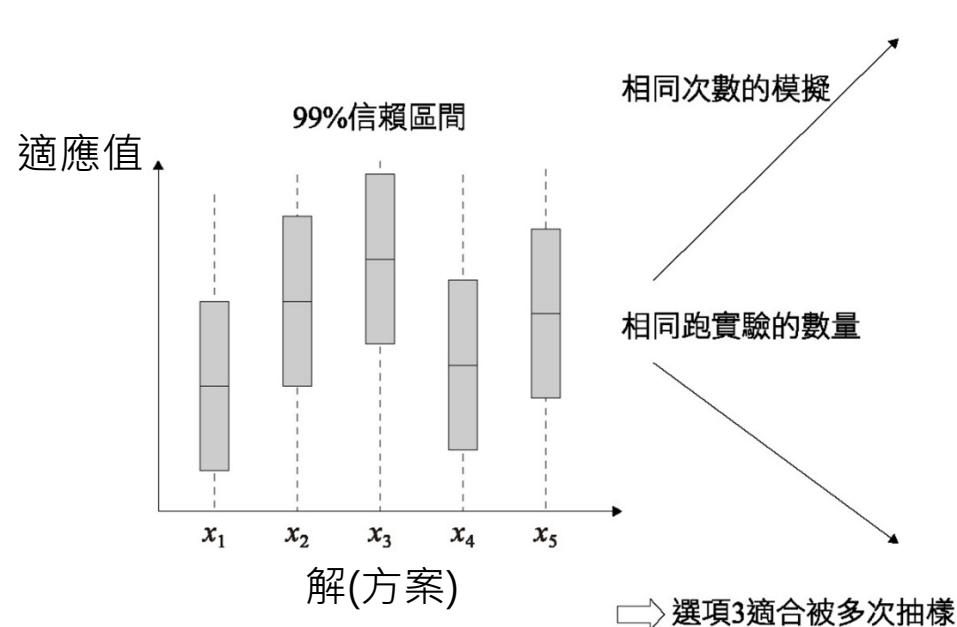
圖 16.16 擁擠距離的計算

□ 最佳資源預算分配

- 在具有隨機性的系統或環境中〔隨機最佳化 (stochastic optimization) 問題，或稱模擬最佳化 (simulation optimization)〕，需藉由重複抽樣才得以穩健地評估適應值，例如同時估計其平均值與變異數。
- 以典型實驗設計 (design of experiments, DOE) 的觀點，會盡可能的對所有的解（或稱方案、實驗、參數設計）以相同樣本數（又稱實驗次數、重複次數）進行抽樣估計。然而在資源有限下的最佳化，平均分配資源（例如計算資源）是相對沒有效率的。
- 因此，「最佳資源預算分配」(optimal computing budget allocation, OCBA) (Chen et al., 2000; Lee et al., 2010) 是一種「順序型重複抽樣」(sequential resampling) 策略，將有限的資源有效地分配在最需要被評估的解上，更精確地說，此方法的目標是在有限資源下最大化對潛在最佳解的抽樣機率。
- 此方法在使用上呈現兩個觀點
 - 在有限的預算下最大化正確解的機率 (probability of correct solution, PCS)
 - 在可接受的正確解的機率下最小化資源預算

□ 最佳資源預算分配

- 一個隨機最佳化的最大化問題，經由對五組解的評估後，得出各別的信賴區間（左圖）。若以平均樣本數的重複抽樣（右上圖），可將對所有解估計的變異縮小，但也同時浪費資源去計算適應值較小的解（即便考慮其變異也不太可能為最佳解）。
- 若在重複抽樣中僅著重於較可能為最佳解的第二與第三組解（右下圖），將本來要分配給其他方案的計算資源挪到第二與第三組，則能快速縮小這兩組的信賴區間以比較出差異。換句話說，試圖以較少的資源快速地將潛在最佳解估計的變異縮小，而這便是最佳資源預算分配的思維。



□ 最佳資源預算分配

- OCBA重複抽樣是如何決定每次抽樣樣本數呢？如何有效地分配資源呢？

- 實際上，我們一方面期望能將資源配置在最可能成為最佳解的地方；另一方面，也須同時考慮每組的變異大小（不確定性）。因此，OCBA的抽樣策略是考慮對最佳解與不確定性的解進行更多的抽樣。
- 令 N_i 為第*i*組解的抽樣樣本數， T 是全部可分配的運算資源， \bar{x}_i 與 s_i 為樣本平均數與標準差，第*b*組解為當前的最佳解(目標最大值)，因而最佳資源預算分配透過最大化正確解的機率來找到最佳解，如公式所示

$$\max_{N_1, \dots, N_k} PCS = \sum_{i=1, i \neq b}^k P\{\bar{x}_b > \bar{x}_i\},$$

➤ subject to $\sum_{i=1}^k N_i \leq T,$
 $N_i \geq 0, \forall i \in \{1, 2, \dots, k\}$

- 求解上述最佳化問題可得到各組解的抽樣樣本數分配如公式所示

➤ $\frac{N_i}{N_j} = \left(\frac{s_i / (\bar{x}_i - \bar{x}_b)}{s_j / (\bar{x}_j - \bar{x}_b)} \right)^2, \forall i, j \in \{1, 2, \dots, k\} \text{ 與 } i \neq j \neq b$

➤ $N_b = s_b \sqrt{\sum_{i=1, i \neq b}^k \frac{N_i^2}{s_i^2}}$

- 其中 N_b 為當前最佳解被分配到的計算資源。換言之，當某一組解的變異越大或與當前最佳解越接近時，其被分配到的樣本數(資源)越多。

□ 最佳資源預算分配

- 舉例來說，若將最佳資源預算分配應用在基因演算法上，其流程可參考圖所示，在每次對解評估時，逐步對當前所有解的評估，再傳遞回基因演算法進行交配與突變。

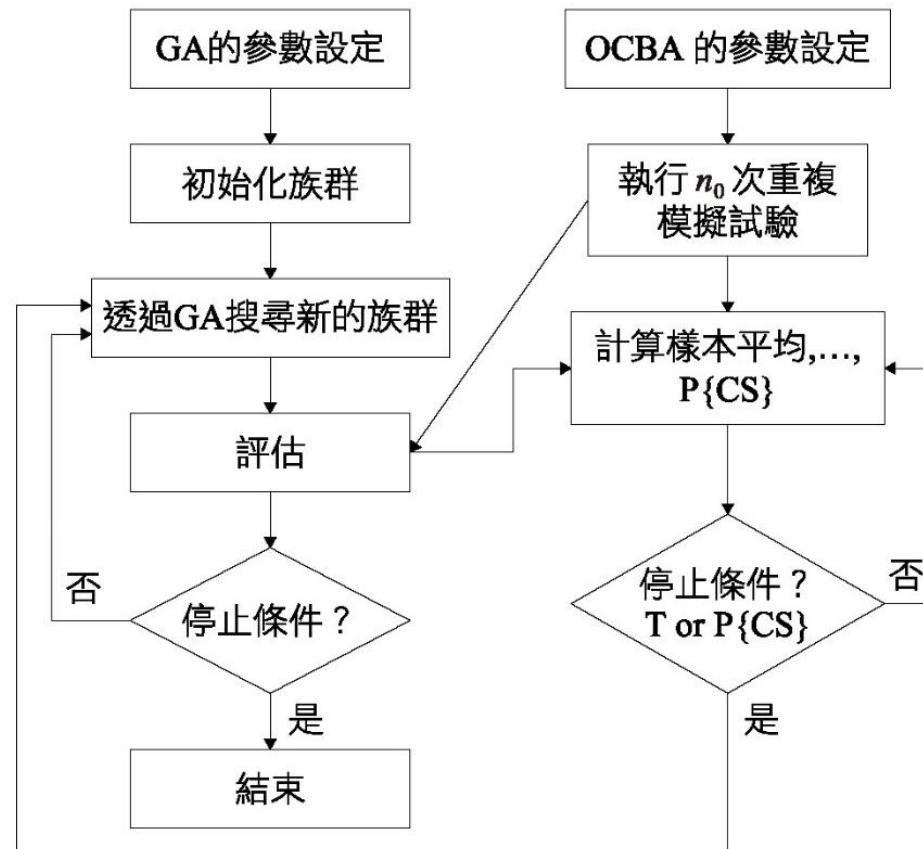


圖 16.18 基因演算法內嵌最佳資源預算分配

□ 此章節介紹元啟發式演算法的核心思維與經典的演算法

- 禁忌搜尋法與基因演算法，以及相關進階議題
- 實務應用層面相當廣泛，尤其是對於**組合最佳化**的困難問題，能在有限時間內提供近似最佳解。
- 其實是提供了一個較佳的**試誤法(trial-and-error)**機制，在不需要太多數學理論基礎下，易理解且容易使用。更特別的是，其具有良好的彈性，可以跟其他方法論整合，例如與上述提及的**區域搜尋法**、**多目標最佳化**與**最佳資源預算分配**進行整合等。

□ 演算法是一門與設計有關的學問

- 透過基礎理論的學習，理解**開採與探索間的權衡**。
- 當轉換到實務應用時，必須根據**實際問題特性**、**目標限制**、以及**計算資源**，進行客製化的設計
- 例如在基因演算法中客製化染色體編碼、交配與突變機制、選擇、與超參數動態調整等，每個元素都需要依現場問題作適當地調整。
- 換言之，能夠融會貫通、懂得舉一反三，就是元啟發式演算法所帶來的重要啟示。

Thanks for your attention



NTU Dept. of Information Management
name: 李家岩 (FB: Chia-Yen Lee)
phone: 886-2-33661206
email: chiayenlee@ntu.edu.tw
web: <https://polab.im.ntu.edu.tw/>