國立臺灣師範大學理學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering
College of Science
National Taiwan Normal University

Master's Thesis

以雙族群遺傳演算法求解旅行小偷問題

A Dual-Population Genetic Algorithm for the Travelling Thief Problem

王碩英

Wang, Shuo-Ying

指導教授：蔣宗哲 博士

Advisor: Chiang, Tsung-Che, Ph.D.

中華民國 110 年 8 月

August 2021

致謝

誠摯感謝指導教授—蔣宗哲教授，謝謝蔣教授兩年間的細心指導與鼓勵，不斷與我討論修改研究與實驗內容，使得論文能夠更加完整。從蔣教授身上，我也學習到許多事情，例如寫程式的技巧，如何分析實驗結果等等，讓我能夠在研究領域中更細膩的思考研究內容。感謝實驗室的同窗們，在最後衝刺階段的日子裡，謝謝吳培弘與楊雅茹學姊一起奮鬥，一起度過最後難關。特別謝謝柯炫任適時地在研究中幫助我許多事情，也謝謝實驗室其他的同窗們適時地提供協助及支援。最後感謝我的家人兩年來的鼓勵與支持，讓我能夠專心的完成學業，在此特別感謝，謝謝你們的支持。

中文摘要

旅行推銷員問題與 0-1 背包問題都是著名的離散最佳化問題。旅行推銷員問題欲求旅行推銷員行走各城市的最短路徑。0-1 背包問題求解物品挑選利益最大化。旅行小偷問題將旅行推銷員問題與 0-1 背包問題結合，使得需同時求解路徑排序與物品組合。在旅行推銷員問題與 0-1 背包問題各自的文獻中，常見以啟發式演算法求解。然而，若是以啟發式演算法直接求解旅行小偷問題，將面臨巨大的搜尋空間。因此，我們提出雙族群式的遺傳演算法，分別專注搜尋兩個子問題，以此縮小搜尋空間。每間隔一段時間，兩個族群使用遷移機制進行資訊的交換，達到互相幫助的效果。搜尋單一子問題過程中，好的解碼函式能夠讓個體在另一子問題中獲得更好的基因。因此，我們針對解碼函式中的參數使用自適應控制機制，以提升個體解碼後的品質。本研究詳細探討自適應控制機制的作用與效果，實驗結果證明使用自適應控制機制能夠提升演化搜尋最佳解，並且我們觀察自適應控制的參數演化圖，在不同測試資料中，都能演化收斂在合理的數值。本研究探討雙族群演化方式的作用與效果，實驗證明相較單一族群的演化方式，使用雙族群式遺傳演算法效果更好。我們觀察雙族群遺傳演算法中，兩個族群在演化互助的過程，也確認遷移機制能夠提升雙族群遺傳演算法的演化結果。最後我們所提出之雙族群遺傳演算法相比近年文獻演算法更為優秀，代表雙族群遺傳演算法在近年求解旅行小偷問題的演算法中頗具競爭力。

關鍵字 : 演化演算法、遺傳演算法、旅行小偷問題、自適應控制

Keywords: evolutionary algorithm, genetic algorithm, travelling thief problem

目錄

致謝 i

中文摘要 ii

目錄 iii

附表目錄 v

附圖目錄 vii

第一章 緒論 1

1.1 研究背景與動機 1

1.2 研究問題定義 2

1.3 研究問題範例 3

1.4 研究目的與方法 4

1.5 論文架構 4

第二章 文獻探討 5

2.1 旅行推銷員問題演算法 5

2.1.1 區域搜尋與鄰域函式 5

2.1.2 Lin–Kernighan heuristic (LKH) 7

2.1.3 Chained Lin–Kernighan heuristic (CLKH) 7

2.2 背包問題演算法 8

2.3 單目標旅行小偷問題演算法 9

2.3.1 經驗法則 9

2.3.2 單一子問題搜尋 10

2.3.3 交替式搜尋 13

2.3.4 同步搜尋 16

2.4 多目標旅行小偷問題演算法 16

第三章 雙族群遺傳演算法 18

3.1 雙族群遺傳演算法 (dpGA) 架構 18

3.2 個體的編碼與評估方式 20

3.3 tspGA 演算法 21

3.3.1 個體的解碼 21

3.3.2 族群初始化 22

3.3.3 交配策略 24

3.3.4 自適應控制機制 25

3.3.5 突變策略 26

3.4 kpGA 演算法 27
3.4.1 個體的解碼 27
3.4.2 修復機制 27
3.4.3 族群初始化 28
3.4.4 交配策略 29
3.4.5 突變策略 30
3.5 遺傳演算法通用機制 30
3.5.1 親代選擇策略 30
3.5.2 環境選擇策略 31
3.6 遺傳演算法虛擬碼 32
3.7 遷移機制與繼承機制 33
3.8 dpGA 虛擬碼 37
第四章 實驗設計與結果 38
4.1 測試資料集 38
4.2 演算法參數設定 40
4.3 實驗環境設定 40
4.4 CLKH 路徑差異影響實驗 41
4.5 tspGA 相關實驗 42
4.5.1 自適應控制機制實驗 42
4.5.2 自適應控制機制參數調整實驗 47
4.5.3 子代數量調整實驗 47
4.5.4 突變策略與族群多樣性維護實驗 48
4.6 kpGA 參數調整實驗 49
4.7 dpGA 相關實驗 49
4.7.1 遷移機制相關實驗 49
4.7.2 雙族群與單一族群的比較 52
4.7.3 族群大小影響實驗 53
4.7.4 族群初始化 CLKH 路徑個數實驗 53
4.8 比較文獻演算法 54
第五章 結論與未來研究方向 58
參考文獻 60

附表目錄

表 1：2-change 虛擬碼 (2-change)..... 6
表 2：2-opt 虛擬碼 (2-opt)..... 7
表 3：Bit-flip 虛擬碼 (Bit-flip)..... 9
表 4：Pack 演算法虛擬碼 (Pack)..... 11
表 5：Pack-Iterative 演算法虛擬碼 (Pack-Iterative)..... 12
表 6：解個體評估函式虛擬碼 (Evaluation)..... 21
表 7：隨路選物之 Pack 解碼函式虛擬碼 (PackCompletion)..... 22
表 8：隨路選物之 Pack-Iterative 解碼函式虛擬碼 (Pack-IterativeCompletion)..... 22
表 9：CLKH 路徑個體初始化虛擬碼 (CLKH-Initialization)..... 23
表 10：tspGA 族群初始化虛擬碼 (tspGA-Initialization)..... 23
表 11：自適應控制虛擬碼 (Self-AdaptiveControl)..... 25
表 12：tspGA 交配策略虛擬碼 (tspGA-Crossover)..... 26
表 13：tspGA 突變策略虛擬碼 (tspGA-Mutation)..... 26
表 14：隨物選路解碼函式虛擬碼 (ClkhCompletion)..... 27
表 15：背包組合基因修復機制虛擬碼 (KnapsackRepair)..... 28
表 16：kpGA 族群初始化虛擬碼 (kpGA-Initialization)..... 28
表 17：kpGA 交配策略虛擬碼 (kpGA-Crossover)..... 29
表 18：機率性突變策略虛擬碼 (ProbabilityMutation)..... 30
表 19：kpGA 突變策略虛擬碼 (kpGA-Mutation)..... 30
表 20：等級輪盤法虛擬碼 (RankBased-RouletteWheelSelection)..... 31
表 21：環境選擇策略虛擬碼 (EnvironmentalSelection)..... 31
表 22：tspGA 演算法虛擬碼 (tspGA)..... 32
表 23：kpGA 演算法虛擬碼 (kpGA)..... 33
表 24：遷移機制虛擬碼 (Migration)..... 34
表 25：物品繼承函式虛擬碼 (ItemInheritance)..... 34
表 26：路徑繼承函式虛擬碼 (TourInheritance)..... 34
表 27：tspGA 交配策略虛擬碼 (tspGA-Crossover)..... 35
表 28：kpGA 交配策略虛擬碼 (kpGA-Crossover)..... 35
表 29：tspGA 突變策略虛擬碼 (tspGA-Mutation)..... 36
表 30：kpGA 突變策略虛擬碼 (kpGA-Mutation)..... 36

表 31：dpGA 演算法虛擬碼 (dpGA) 37
表 32：測試資料集與背包容量參數 *C* 39
表 33：dpGA 演算法參數設定 40
表 34：CLKH 路徑差異影響實驗統計檢定結果 41
表 35：自適應控制機制實驗統計檢定結果 43
表 36：三種「Pack 參數控制機制」正規化平均數比較表 44
表 37：自適應控制機制參數調整實驗統計檢定結果 47
表 38：族群擴張大小調整實驗統計檢定結果 48
表 39：族群多樣性維護實驗統計檢定結果 48
表 40：機率性突變策略實驗統計檢定結果 49
表 41：遷移機制實驗統計檢定結果 50
表 42：雙族群優勢實驗統計檢定結果 52
表 43：族群大小影響實驗統計檢定結果 53
表 44：族群初始化 CLKH 路徑個數差異統計檢定結果 53
表 45：dpGA 與文獻演算法比較結果表 55
表 46：文獻演算法與 dpGA 在類別 1 測試資料的平均值 55
表 47：文獻演算法與 dpGA 在類別 2 測試資料的平均值 56
表 48：文獻演算法與 dpGA 在類別 3 測試資料的平均值 57

附圖目錄

圖 1：旅行小偷問題範例圖 3
圖 2：Insertion 示意圖 5
圖 3：Swap 示意圖 5
圖 4：2-change 示意圖 6
圖 5：不適當 2-change 示意圖 8
圖 6：double bridge 示意圖 8
圖 7：CoSolver 架構流程圖 13
圖 8：雙族群遺傳演算法流程圖 19
圖 9：遺傳演算法解個體編碼 20
圖 10：Order Crossover 親代排序基因列 24
圖 11：Order Crossover 子代排序基因列 (步驟一) 24
圖 12：Order Crossover 子代排序基因列 (步驟二) 24
圖 13：Order Crossover 子代排序基因列 (步驟三) 24
圖 14：two-point crossover 親代組合基因列 29
圖 15：two-point crossover 子代組合基因列 (步驟一) 29
圖 16：two-point crossover 子代組合基因列 (步驟二) 29
圖 17：測資類別 1 正規化平均數比較圖 43
圖 18：測資類別 2 正規化平均數比較圖 43
圖 19：測資類別 3 正規化平均數比較圖 44
圖 20：自適應控制實驗中位數演化圖 45
圖 21：測試資料 d15112 之 α 值演化圖與相異 α 值 Pack 函式之適應值曲線圖 . 46
圖 22：測試資料 pla7397 之 α 值演化圖與相異 α 值 Pack 函式之適應值曲線圖 46
圖 23：測試資料 u724 之 α 值演化圖與相異 α 值 Pack 函式之適應值曲線圖 47
圖 24：測試資料 usa13509 之雙族群演化曲線圖 50
圖 25：測試資料 fnl4461 之雙族群演化曲線圖 51
圖 26：測試資料 fl1577 之雙族群演化曲線圖 52

第一章　緒論

1.1 研究背景與動機

旅行推銷員問題 (Travelling Salesman Problem) 是一個離散最佳化問題，欲求一位旅行推銷員在行進速度為等速的情況下，拜訪所有城市各一次所行經的總距離最短，使用暴力法求 n 座城市最佳解的時間複雜度為 O(n!)。0-1 背包問題 (0-1 Knapsack Problem) 也是一個離散最佳化問題，在 m 個物品中挑選出最大利益物品的組合且物品總重不得超過指定上限 W_{max}，經典的動態規劃法求解 m 項物品最佳解的時間複雜度為 O($m \times W_{max}$)。此兩問題都難於多項式時間內找出最佳解，因此許多研究利用啟發式演算法 (meta-heuristic) 或是經驗法則 (heuristic) 來搜尋解。

旅行小偷問題 [1] 是由 Bonyadi 等人於 2013 年提出的問題模型，它是由旅行推銷員問題與 0-1 背包問題組合而來：一位小偷帶著有限容量的背包從初始城市出發，每拜訪一座城市，可選擇是否帶走該城市中的某些物品，最後回到初始城市。問題目標是最大化背包中物品總價值以及最小化行走的總時間。由於背包中物品的重量會影響到小偷行走的速度，因此兩個子問題的關聯性高，導致問題的難度上升。

旅行小偷問題曾於著名國際會議 IEEE CEC 2014、 2015、2017 年 [2] 以及 GECCO 2017 年舉辦過單目標問題模型的競賽，並於 2019 年在 EMO 與 GECCO 舉辦過雙目標旅行小偷問題的競賽。因此，除了此問題的有趣性之外，多次競賽的舉辦代表此問題有被探討與研究的價值，並決定深入研究問題細節，期望能夠對此問題的研究領域往前推動一步。

1.2 研究問題定義

旅行小偷問題定義如下：在二維平面空間中，給定 n 座城市，m 項物品散落在除起始城市之外的城市中，每項物品 k 皆有其價值 p_k 與重量 w_k，小偷行走速度在背包為空與滿容量時分別為 v_{max} 與 v_{min} 。一位小偷帶著重量上限為 W_{max} 的背包拜訪每座城市一次並且偷取物品，假設城市 i 與城市 j 距離為 $d_{i,j}$，背包此時重量為 W_{cur} ，小偷此時行走的速度 $v_{i,j}$ 如式 (1)，而城市 i 與城市 j 行走耗費的時間 $t_{i,j}$ 即為 $d_{i,j} / v_{i,j}$ 。

$$v_{i,j} = v_{max} - (v_{max} - v_{min}) \times W_{cur} / W_{max} \tag{1}$$

假設小偷行徑路線 $X = (x_1,\ldots,x_n)$，代表小偷拜訪城市的順序，物品挑選組合 $Y = (y_1,\ldots,y_m)$，其中 $y_k \in \{0, 1\}$ 代表著物品 k 被選取與否。當小偷完成所有城市拜訪後，期望偷取的物品總價值 $P(Y)$ 越高，並且行走耗費的時間 $T(X, Y)$ 越少。於 Bonyadi 等人 [1] 提出的單目標模型中，設定背包每單位時間的租借費用為 r，以此來結合兩個目標 $P(Y)$ 與 $T(X, Y)$ 形成單目標問題，期望目標函式 $F(X, Y)$ 最大化。前述物品總價 $P(Y)$、行走時間 $T(X, Y)$ 與合併目標函式 $F(X, Y)$ 之定義如式 (2)–(4)。

$$P(Y) = \sum_{k=1}^{m} p_k y_k \tag{2}$$

$$T(X, Y) = \sum_{i=1}^{n-1} t_{x_i,x_i+1} + t_{x_n,x_1} \tag{3}$$

$$F(X, Y) = P(Y) - r \times T(X, Y) \tag{4}$$

1.3 研究問題範例

假設現有四座城市且每座城市放置兩件物品（起點城市除外），物品 y_j 的價值與重量以 (p_j, w_j) 表示，如圖 1 所示。此時小偷帶著租借費用 $r = 1$ 且重量上限 $W_{max} = 10$ 的背包旅行，行走速度上下限 $v_{max} = 1$ 與 $v_{min} = 0.1$，拜訪城市的行走路線 $X = (1, 3, 4, 2)$，物品挑選組合 $Y = (1, 1, 0, 0, 1, 0)$。由此可以計算旅行結束後，小偷拿取 y_1、y_2、y_5 三件物品，可得總價值 $P(Y) = 50+30+10 = 90$。小偷依照 1→3→4→2→1 的拜訪順序，從城市 1 到城市 3 距離 $d_{1,3}$ 為 3，小偷此時背包重量 W_{cur} 為 0，行走速度 $v_{1,3}$ 為 1，因此耗費時間 $t_{1,3} = \frac{3}{1} = 3$。在城市 3 並無物品被拾取，因此城市 3 到 4 行走速度 $v_{3,4}$ 也是 1，耗費時間 $t_{3,4} = \frac{1}{1} = 1$，接著小偷於城市 4 拾取了物品 y_5，行走速度 $v_{4,2}$ 由式 (1) 可得為 0.91，因此城市 4 到 2 耗費時間 $t_{4,2} = \frac{2}{0.91} \approx 2.2$，最後小偷在城市 2 拾取兩物品 y_1、y_2，行走速度 $v_{2,1}$ 因此降為 0.28，回到城市 1 需耗費時間 $t_{2,1} = \frac{2}{0.28} \approx 7.14$，所以小偷於旅行所耗費的時間 $T(X, Y)$ 總共為 3+1+2.2+7.14 = 13.34。目標函式 $F(X, Y)$ 可由式 (4) 計算而得 $90 - (1 \times 13.34) = 76.66$。

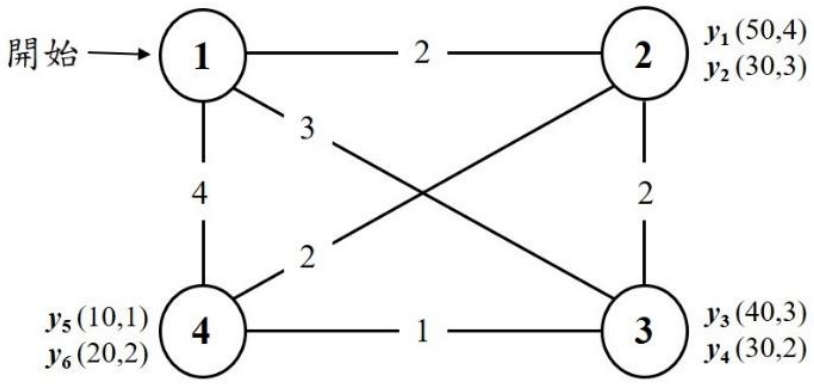

圖 1：旅行小偷問題範例圖

1.4 研究目的與方法

如 1.3 節所定義，旅行小偷問題需同時解決旅行推銷員問題和背包問題兩個相互關聯的經典離散最佳化問題。此二問題都屬於 NP-Complete 問題，在兩個問題各自的文獻中，常見以啟發式演算法求解。然而，若要以啟發式演算法同時求解這兩個問題，將面對極為巨大的搜尋空間。因此，本論文將提出一個雙族群式的遺傳演算法來求解，研究議題如下：

1. 雙族群遺傳演算法：我們的演算法將以兩個族群求解旅行小偷問題，兩個族群分別專注於求解旅行推銷員問題與背包問題，意即一個族群中交配與突變操作僅著眼於單一子問題，另一個子問題則透過經驗法則求解。
2. 自適應性控制機制：針對旅行推銷員子問題搜尋的遺傳演算法中，於演化過程中根據路徑的演進，控制背包子問題解碼函式中的參數，以此找尋到更適合此路徑的背包物品。

本論文所提出之方法，將以文獻中 [3] 測試問題集進行測試，與多個近年提出之演算法比較，以檢視本文方法之求解品質與計算效率。

1.5 論文架構

本論文共分為五個章節，第二章為文獻探討，第一部分介紹常用的旅行推銷員演算法以及背包問題演算法，第二部份介紹單目標旅行小偷問題演算法，第三部分介紹雙目標旅行小偷問題演算法。第三章詳細介紹我們所提出之雙族群遺傳演算法。第四章我們逐步探討雙族群遺傳演算法中的機制策略，了解演算法中各項機制的效益，最後與文獻演算法比較，評估本研究演算法的強度。第五章為研究結論，並提供未來相關學者研究方向。

第二章　文獻探討

2.1 旅行推銷員問題演算法

2.1.1 區域搜尋與鄰域函式

以區域搜尋法求解旅行推銷員問題時，經常以城市序列為編碼方式，透過鄰域函式改變造訪城市的順序，以此搜尋解空間。常用的旅行推銷員鄰域函式為 Insertion、Swap 與 λ-changes 三種類型，以下分別介紹三種鄰域函式：

1. Insertion：從完整路徑中選取一城市，並將此城市插入其餘路段之中即為 Insertion。範例如圖 2 所示，假設有一路徑 $X_1 = (1, 3, 5, 4, 6, 2)$，選取城市 3 插入城市 4 與城市 6 之間，即可得到新的路徑 $X_2 = (1, 5, 4, 3, 6, 2)$。

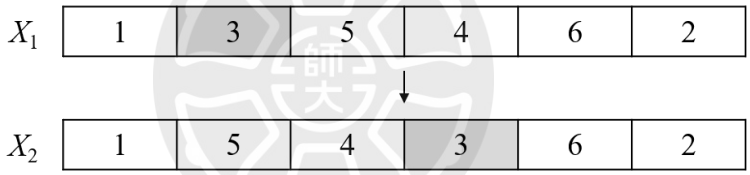

圖 2：Insertion 示意圖

2. Swap：交換機制為選取兩城市，交換在路徑上的行走順序。假設從路徑 $X_1 = (1, 5, 3, 2, 4)$ 選取城市 3 與 4 交換，即得路徑 $X_2 = (1, 5, 4, 2, 3)$，如圖 3 所示。

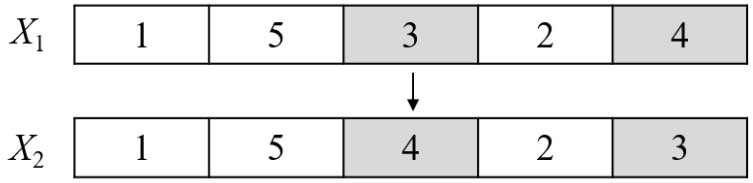

圖 3：Swap 示意圖

3. λ-changes：此操作為在一條路徑上移除 λ 條邊，接著以新的方式重新連接使其路徑完整，當沒有新的 λ-changes 存在使得路徑更短，則稱之為 λ-opt [4]。圖 4 以 2-change 為例，假設有一路徑 $X = (x_1,\ldots, x_i, x_{i+1},\ldots, x_j, x_{j+1},\ldots, x_n)$，首先選定移除兩邊 (x_i, x_{i+1}) 與 (x_j, x_{j+1})，再重新連接 (x_i, x_j) 與 (x_{i+1}, x_{j+1}) 兩條邊使得路徑完整。圖中顯示 2-change 後的路徑，拜訪 x_{i+1} 至 x_j 的城市順序會顛倒。

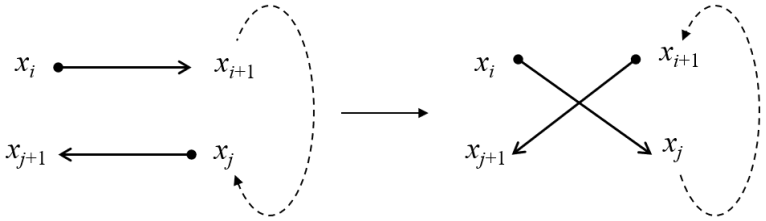

圖 4：2-change 示意圖

表 1 顯示 2-change 虛擬碼，f、s 為拜訪城市的順位，4–6 行表示 2-change 在拜訪城市的順序上，會顛倒原本拜訪路徑中 f 至 s 的順序，其餘城市拜訪順序不變。表 2 為 2-opt 的虛擬碼，由第 4 行取得 2-change 的路徑，5–8 行表示若是此路徑有改善，則從第一個城市重新搜尋所有可能，直到沒有改善的情況才結束演算法。

表 1：2-change 虛擬碼

Function 2-change (*X*, *f*, *s*)
Input: *X*, an input tour
f, *s*, the position of tour sequence
Output: *X-change*, a 2-change tour
Notations
n: number of cities

```
01  for i = 1 to f − 1 do
02      X-change_i = X_i
03  end for
04  for i = f to s do
05      X-change_i = X_{s−(i−f)}
06  end for
07  for i = s + 1 to n do
08      X-change_i = X_i
09  end for
10  return X-change
```

表 2：2-opt 虛擬碼

Function 2-opt (X)
Input: X, an input tour
Output: X_{best}, best tour
Notations
X: genes of tour
n: number of cities

```
01  X_best = X
02  for i = 1 to n − 1 do
03     for j = i + 1 to n do
04        X = 2-change(X_best, i, j)
05        if X is better than X_best do
06           X_best = X
07           Start with i = 1 again
08        end if
09     end for
10  end for
11  return X_best
```

2.1.2 Lin–Kernighan heuristic (LKH)

LKH 搜尋法 [5] 在 1973 年被提出， 它是目前求解旅行推銷員最著名的演算法之一，利用類似求取 λ-opt 路徑的方式，不斷的在新產生的隨機初始個體中，找到其區域最佳解。在旅行推銷員問題中，3-opt 的區域搜尋法即可得到非常好的效果，而 LKH 搜尋法與一般 λ-opt 固定 λ 的求取方式不同，求取過程中並不會有固定的 λ 值，LKH 將路徑中的路段集合 $X = \{x_1,\ldots, x_k\}$ 依序替換成 $X' = \{x_1',\ldots, x_k'\}$，並且利用增益參數 $g_i = \mathrm{d}(x_i) - \mathrm{d}(x_i')$ 判斷路段 x_i 替換成 x_i' 的好壞，其中 $\mathrm{d}(x_i)$ 為路段 x_i 的長度。若是 $g_i > 0$，代表 $\mathrm{d}(x_i')$ 比 $\mathrm{d}(x_i)$ 短，替換為正增益。因此若是 $\sum_1^k g_i > 0$，代表路段集合 X 替換成 X' 能使得路徑更短，而若是 $\sum_1^k g_i < 0$，代表替換結果更差。

2.1.3 Chained Lin–Kernighan heuristic (CLKH)

CLKH 搜尋法 [6][7] 基於 LKH 搜尋法改善加強，有別於 LKH 找到區域最佳解後重新初始化新的隨機解個體，CLKH 則是把 LKH 得到的區域最佳解進行擾動，以脫離區域最佳解的搜尋區域。CLKH 使用 “double bridge” 進行擾動，double bridge 由兩個不適當的 2-change 合併而成，圖 5 表示一個不適當的 2-

change，刪除兩條邊 (a, b)、(c, d) 之後，連接新的兩條邊 (a, c)、(b, d) 使得路徑分成兩個區塊。圖 6 顯示合理的 double bridge 行為，兩次不適當 2-change 需要合併成一合理的路徑。LKH 與 CLKH 常被應用於旅行小偷問題的初始個體中，使得求解旅行小偷問題能在初始化的步驟中獲得相當不錯的解個體。

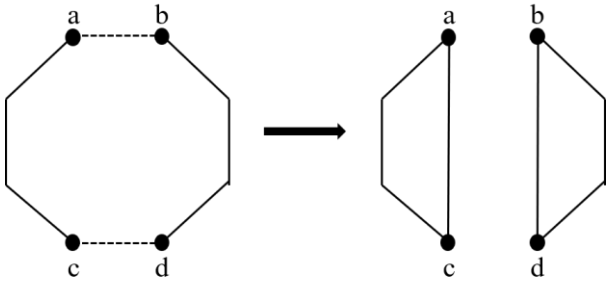

圖 5：不適當 2-change 示意圖

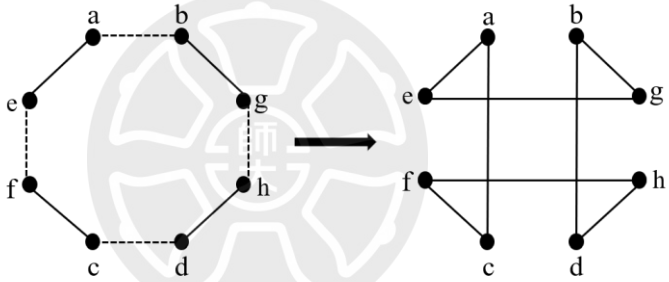

圖 6：double bridge 示意圖

2.2 背包問題演算法

背包問題以組合的編碼方式搜尋，假設現有 *m* 項物品，則基因列長度為 *m*，每項物品皆以 0 或 1 表示是否被拾取。鄰域函式 bit-flip 常被應用於旅行小偷問題中的背包問題，概念為翻轉物品的挑選來改善解個體。Bit-flip 用於區域搜尋時，會查看每一物品翻轉後是否會改善，若是有改善，則會重新查看每一物品，直到所有物品翻轉皆沒有改善的情況才停止。表 3 為 bit-flip 虛擬碼，傳入背包物品列 *Y* 翻轉第 *k* 個物品，1–2 行表示若是此物品存在於背包中，則從背包移除，3–4 則是反過來，將物品放於背包中。

表 3：Bit-flip 虛擬碼

Function Bit-flip (*Y, k*) Input: *Y*, an input picking plan *k*, item's index Output: *Y*, a picking plan after bit-flip Notations *m*: number of items
01 **if** Y_k **equals** 1 **do** 02 $Y_k = 0$ 03 **else** 04 $Y_k = 1$ 05 **end if** 06 **return** *Y*

2.3 單目標旅行小偷問題演算法

2.3.1 經驗法則

由於旅行小偷問題的搜尋空間極大，因此部份文獻專注於求解背包子問題，旅行推銷員子問題則以 LKH 或 CLKH 求得好的路徑。此類型經常以評分物品的方式求取背包問題。

Polyakovskiy 等人提出 Simple Heuristic (SH) [3]，SH 使用 CLKH 生成路徑，並利用物品 k 的價值 p_k、重量 w_k 與物品被挑選後行走的距離 d_k 等資訊，對各項物品進行評分，接著挑選分數高的物品直到背包裝滿為止。SH 另對每一物品設計一個 u 值，以此預防選擇到太差的物品。Gupta 與 Prakash 提出兩種貪婪式經驗法則 G1、G2 [8]，G1 考慮物品 k 的重量與價值 p_k/w_k 來評分，以此挑選評分高的物品，G2 希望小偷保持最大速度 v_{max} 旅行的城市越多越好，因此優先挑選靠近路徑尾端城市的物品。G1 與 G2 皆有背包參數 *KF* 與重量參數 *WF*，兩參數皆介於 0 與 1 之間；在挑選物品 k 時，背包重量上限設定為原容量 W_{max} 的 *KF* 倍，且物品重量 w_k 不得超過所有物品中最大重量的 *WF* 倍才得以被挑選。G1 與 G2 利用反覆搜尋不同 *KF* 與 *WF* 組合來求得最佳解。

Faulkner 等人於 2015 年提出 Pack 經驗法則與 Pack-Iterative 搜尋法 [9]，並提出新的評分公式 (5)。Pack 搜尋法考慮物品 k 的價值 p_k、重量 w_k 以及物品被攜帶行走的距離 d_k 來進行物品的評分，並給予固定指數權重 α 來調整物品價值與重量的影響力。表 4 顯示 Pack 搜尋法虛擬碼，首先於第一行由公式 (5) 計算所有物品的分數，接著於第 2 行依照分數由大到小排列放入物品序列 L 中。第 3 行設定頻率參數 μ，此參數由物品數量 m 與固定參數 τ 生成。9–12 行由分數高的物品持續放入 μ 個至背包中。13–26 行判斷此 μ 個物品同時放入是否增益：若是有，則繼續放入物品列 L 中的下 μ 個物品；若是變差，則取出此 μ 個物品，改為放入 $\mu / 2$ 個物品來判斷是否合適。Pack-Iterative 通過反覆調整 Pack 搜尋法中的指數權重 α 大小進行區域搜尋，以此找到更好的背包組合，虛擬演算法於表 5。首先 1–4 行以三個不同權重 α 值實施 Pack 搜尋法，6–20 行選出三者中最佳的 α 值，並於 21-24 行縮減偏離參數 δ 並計算新的權重值以實施 Pack 搜尋法。

$$\frac{{p_k}^{\alpha}}{{w_k}^{\alpha} \times d_k} \tag{5}$$

2.3.2 單一子問題搜尋

面對搜尋空間極大的問題，除了以經驗法則以外，部分文獻以區域搜尋法針對背包子問題求解。先以 LKH 或 CLKH 求得路徑，接著再使用區域搜尋背包組合。

表 4：Pack 演算法虛擬碼

Function Pack (X, α)
Input: X, a fixed tour
α, an exponent value
Output: Y, a picking plan
Notations
L: a score list for all items
μ: frequency parameter
W_{max}: knapsack capacity
W_{cur}: current knapsack's weight
f: fitness value
f_{best}: fitness of the best picking plan

```
01  Compute score for each item I_k
02  Sort items' list L in non-decreasing order by their score
03  μ = ⌊m / τ⌋  //Set frequency μ
04  Y = ϕ, Y_best = ϕ
05  W_cur = 0, W_best = 0
06  k = 1, k_best = 1
07  f_best = F(X, Y_best)  //evaluate fitness function
08  while W_cur < W_max and μ > 1 and k ≤ m do
09     Get item I_k from list L position k
10     if W_cur + w_k ≤ W_max do
11        Y_cur = Y_cur ∪ {I_k}
12        W_cur = W_cur + w_k
13        if k mod μ = 0 do
14           f = F(X, Y_cur)  //evaluate fitness function
15           if f < f_best do
16              Y_cur = Y_best
17              W_cur = W_best
18              k = k_best
19              μ = ⌊μ / 2⌋
20           else
21              Y_best = Y_cur
22              k_best = k
23              f_best = f
24              W_best = W_cur
25           end if
26        end if
27     end if
28     k = k + 1
29  end while
30  return Y
```

Polyakovskiy 等人除了提出 Simple Heuristic (SH) 之外，另提出 Random Local Search (RLS)、(1+1) Evolutionary Algorithm (EA) [3] 兩種演算法。 RLS 與 (1+1) EA 為疊代搜尋法，RLS 通過反覆的隨機翻轉物品是否挑選，尋找疊代過程中的最佳解。(1+1) EA 與 RLS 極為類似，由隨機翻轉物品變為每一物品給予機率性翻轉的可能。

表 5：Pack-Iterative 演算法虛擬碼

Function Pack-Iterative (X, c)
Input: X, a fixed tour
　　c, an initial exponent value
Output: Y, a picking plan
Notations
q: iteration number
δ: an offset value
f: fitness value
Y_l, Y_m, Y_r: picking plans come from Pack function in different exponent value

```
01  Y_l = Pack(X, c − δ)
02  Y_m = Pack(X, c)
03  Y_r = Pack(X, c + δ)
04  f_l, f_m, f_r = F(X, Y_l), F(X, Y_m), F(X, Y_r)
05  for i = 1 to q do
06     if f_l > f_m and f_r > f_m do
07        if f_l > f_r do
08           f_m = f_l, Y_m = Y_l
09           c = c − δ
10        else
11           f_m = f_r, Y_m = Y_r
12           c = c + δ
13        end if
14     else if f_l > f_m do
15        f_m = f_l, Y_m = Y_l
16        c = c − δ
17     else if f_r > f_m do
18        f_m = f_r, Y_m = Y_r
19        c = c + δ
20     end if
21     δ = δ / 2
22     Y_l = Pack(X, c − δ)
23     Y_r = Pack(X, c + δ)
24     f_l, f_r = F(X, Y_l), F(X, Y_r)
25     if |f_l − f_m| < ε and |f_r − f_m| < ε do
26        break
27     end if
28  end for
29  return Y_m
```

在 bit-flip 區域搜尋時，每次翻轉物品都需評估個體，然而旅行小偷問題目標函式 $F(X, Y)$ 公式 (4) 時間複雜度為 $O(m + n)$，m 為物品數量，n 為城市數量。為了降低計算時間，在操作一次 bit-flip 時，僅翻轉一件物品，因此可將計算物品總價值 $P(Y)$ 公式 (2) 時間複雜度降為 $O(1)$，目標函式 $F(X, Y)$ 時間複雜度降為 $O(n)$。Faulkner 等人提出線性時間搜尋 m 項物品的方法 [9]，不管是否改善，翻轉各項物品一次即終止搜尋，算上目標函式的計算時間，時間複雜度為 $O(mn)$。

Maity 和 Das 利用 CLKH 所產生的路徑搭配評分物品的經驗法則來初始化個體 [10]，接著再逐步翻轉單一物品的偷取與否 (bit-flip) 進行區域搜尋來獲得更佳的解，不同於 Faulkner 等人 [9] 對 m 項物品逐一嘗試的區域搜尋方法，Maity 與 Das 進行 m 次隨機取一項物品翻轉的區域搜尋方法。

2.3.3 交替式搜尋

Bonyadi 等人於 2014 年提出 CoSolver 架構 [11]，其將旅行小偷問題拆解成兩個子問題，分別為攜帶背包的旅行推銷員問題 (Travelling Salesman with Knapsack Problem, TSKP) 與路徑上的背包問題 (Knapsack on the Route Problem, KRP)。TSKP 目標在已確定背包內容的情況下，尋求較佳的路經；KRP 則是在已確定的路徑下，找尋更適合的物品組合。如圖 7 所示，CoSolver 輪流求解 KRP 與 TSKP，使得解的品質逐步上升，並於解的品質未獲得改善時終止演算法。此架構下的演算法皆為單一個體式的搜尋方式，由於搜尋單一子問題時，會固定另一子問題的基因列，因此在交替的搜尋過程中，容易受到前者搜尋到的解的影響。

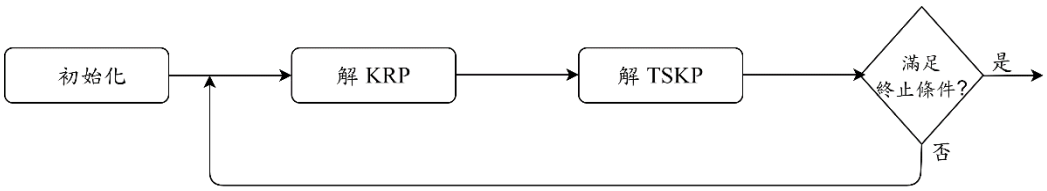

圖 7：CoSolver 架構流程圖

Yafrani 與 Ahiod 於 2015 年基於 CoSolver 的架構提出 Cosolver2B 演算法 [12]，分別使用 2-opt 與 bit-flip 解 TSKP 與 KRP 兩個子問題，其中 2-opt 利用 Delaunay Triangulation [13] 來減少區域搜尋範圍。隔年 Yafrani 等人根據 CoSolver 架構改動，提出 CS2SA 演算法 [14]，利用 2-opt 與模擬退火法交替求

解兩子問題，並得出 CS2SA 於大型測試資料有較佳的結果。接著於 2018 年，Yafrani 與 Ahiod 對 CS2SA 強化模擬退火法中的參數並稱為 CS2SA* [15]。CoSolver 架構終止演算法於解停止改善的情況，因此當演算法停止時，仍有多餘的時間可以利用，重新啟動演算法的機制因此而生，Yafrani 與 Ahiod 提出 CS2SA-R [15]，以 CS2SA 為基底，於解停止改善時，重新啟動演算法。Nieto-Fuentes 等人提出的 RES_GLS 演算法，以 CoSolver 為基底，除了 2-opt 與 bit-flip 的區域搜尋，其強調在一定次數未改善解的情形下，重新啟動演算法，此機制能夠有效提升 CoSolver 解品質。Yafrani 與 Ahiod 提出 JNB 與 J2B 區域搜尋演算法 [17]，其中 JNB 與 J2B 分別以相鄰城市交換函式 (adjacent city swap) 與 2-opt 作為排列路徑的鄰域函式，bit-flip 為兩者的背包鄰域函式，兩個演算法使用兩子問題的鄰域函式嘗試區域搜尋當前路徑與背包的所有可能。

Mei 等人於 2014 年提出協同演化演算法 (cooperative co-evolution algorithm) 求解旅行小偷問題 [18]，協同演化演算法與 CoSolver 類似，其將旅行小偷問題拆解成 TSKP 與 KRP 兩子問題，再利用協同演算法進行兩子問題的交替式搜尋。不同於 CoSolver 為單一個體式的搜尋法，協同演化演算法進行兩子問題搜尋時，採用族群式搜尋法。協同演算法中設有一個合作池 (collaboration pool)，將單一子問題搜尋完畢後，最佳個體放置於合作池中。假設在搜尋旅行推銷員子問題的路徑時，背包的選擇會以合作池中個體的背包基因作為解碼函式。反過來說，在搜尋背包子問題時，會以合作池中個體的路徑作為解碼函式。此種交替搜尋的過程與 CoSolver 架構類似，極容易受到前個子問題搜尋求得的個體影響。

遺傳演算法搭配區域搜尋法即可統稱為瀰集演算法 (memetic algorithm)，通常以遺傳演算法搜尋旅行路徑，再搭配經驗法則求解背包子問題，並以區域搜尋法補強背包組合，如此的交替搜尋來降低同時搜尋兩子問題的解空間。Alharbi 結合遺傳演算法與禁忌搜尋法 [19]，僅對排序基因實行交配策略求得更佳的路徑，搭配 Pack-Iterative 經驗法則獲取適合該路徑的物品組合，使得搜尋空間下降，能夠在有限時間內找尋到更好的路徑。它於每個世代 (generation) 的最後，對所有解個體的背包子問題進行禁忌搜尋，找尋更適合該路徑上的物品選擇。Yafrani 與 Ahiod 除了提出 CS2SA 的演算法之外，同時也提出 MA2B 啟發式演算法 [14]，以瀰集演算法為架構。使用 Maximal Preservative Crossover (MPX) 進行交配，背包物品跟隨城市遺傳到子代，例如子代的城市 A 遺傳自親代中的父親，則城市 A 中包含的物品都由父親遺傳到子代。在突變策略中，使用兩次 “double bridge” 突變搭配貪婪式經驗法則決定背包物品，接著再使用 2-opt 與 bit-flip 的區域搜尋來補強突變後的個體。Mei 等人提出 MATLS 演算法 [20] 來求解旅行小偷問題，架構為兩層式區域搜尋的瀰集演算法。此演算法每個世代以 Order Crossover 產生子代，接著對此子代進行兩層式的區域搜尋。第一層區域搜尋期望獲得較佳的路徑（忽略背包子問題），以較短路徑為搜尋目標。第二層根據生成的路徑，使用經驗法則解碼挑選物品。接著再利用僅放入物品的 bit-flip 進行區域搜尋，由於經驗法則選擇物品後，剩餘背包空間不多，所以區域搜尋耗費的時間減少。

蟻群演算法 (ant colony optimization algorithm) 是一熱門求解旅行推銷員的演算法類別，而 Wagner [21] 於 2016 年使用最大最小螞蟻系統 (max-min ant system) [22] 得出行走路徑，配合 Pack-Iterative 經驗法則求得該路徑上的物品組合，並以旅行小偷問題的目標函式 $F(X, Y)$ 更新費洛蒙資訊，於每個世代的最後進行區域搜尋。

2.3.4 同步搜尋

Vieira 等人於 2017 年使用遺傳演算法求解旅行小偷問題 [23]，其中同時對兩個子問題的基因列實行交配策略，並使用競爭選擇法來篩選存活的個體。由於搜尋空間較大，因此在有限時間下，難於在大型測試資料演化收斂，獲得好的結果。

Mei 等人於 2014 使用瀰集演算法求解旅行小偷問題 [18]，其架構與 Vieira 等人 [23] 提出的遺傳演算法類似，皆同時對排序與組合基因列實行交配，其中不同在於交配生成的子代，加入機率性的區域搜尋。

除了遺傳演算法與瀰集演算法之外，遺傳規劃法 (genetic programming) 也被嘗試求解旅行小偷問題，遺傳規劃法概念在於利用遺傳演算法去搜尋出一個好的演算法架構。Yafrani 等人於 2017 提出 GPHS* [24]，利用遺傳規劃法去組合不同的鄰域函式，例如 2-opt、bit-flip、模擬退火法等，形成搜尋法求取旅行小偷問題解空間。

2.4 多目標旅行小偷問題演算法

多目標問題不同於單目標問題存在單一最佳解，求解時常會生成柏拉圖前緣 (Pareto Front)。柏拉圖前緣上的所有個體無法於所有目標上勝過對方，或稱無

法凌越 (dominate) 對方。多目標問題利用柏拉圖前緣來判斷族群的好壞，第一步為判斷柏拉圖前緣上的個體能否凌越另一前緣上的個體；若是能，代表此前緣較佳；若是不能，則會觀察兩條前緣中個體的分佈情況，能夠越均勻且涵蓋範圍廣的較佳。NSGA-II [25] 為經典的多目標演化演算法，其核心概念為基於非凌越排序 (non-dominated sorting) 的環境篩選，將族群中的個體以 rank 的方式區分優劣，將無法互相凌越對方的個體放在同一個 rank 中，最後篩選留下 rank 較佳的個體。然而在留下的個體中，最差 rank 中經常發生有多餘的個體需要淘汰，因此，NSGA-II 設計了 crowding distance 來計算此 rank 中個體之間的密集程度，用以淘汰較密集的個體，使得個體能夠均勻分佈。Blank 等人即使用 NSGA-II 求解雙目標旅行小偷問題 [26]。

Laszczyk 和 Myszkowski 使用 NTGA 求解旅行小偷問題 [27]。相較於 NSGA-II，NTGA 對其進行四項改動，第一是分開親代與子代的族群；第二為使用非凌越排序之排名 (rank) 進行競爭式選擇 (tournament selection) 來選擇親代；第三為族群多樣性的維護，若交配後產生的子代已經存在於子代的族群中，則強制進行突變；最後則是比 NSGA-II 多一個額外族群 (archive)，存放演化過程中產生的非凌越關係個體。Chagas 等人於 2020 年提出 NDS-BRKGA (Non-Dominated Sorting Biased Random-Key Genetic Algorithm) [28]，此演算法也是基於 NSGA-II 進行改動，使用非凌越排序來區分菁英與非菁英族群，並利用兩族群來選擇親代進行交配。此演算法不同於一般使用離散編碼來搜尋旅行小偷問題，它將離散的排列與組合基因，轉化成連續型的值，並用於交配與突變策略之中。演算法最後每過一定次數的世代做區域搜尋，區域搜尋的方法為離散相關的機制。

第三章　雙族群遺傳演算法

3.1 雙族群遺傳演算法 (dpGA) 架構

旅行小偷問題由旅行推銷員問題與背包問題結合而來。然而，若是以啟發式演算法直接求解旅行小偷問題，將面臨巨大的搜尋空間。因此，本論文擬定以兩個遺傳演算法分別專注求解旅行推銷員問題與背包問題：解旅行推銷員子問題的遺傳演算法簡稱為 tspGA，而解背包子問題的遺傳演算法則簡稱為 kpGA。我們利用遺傳演算法來專注搜尋單一子問題解空間，而另一子問題則是使用解碼函式來求解，使得搜尋解能在另一子問題上高效率獲得較合適的解。tspGA 以隨路選物的解碼方式補足背包子問題的物品，kpGA 則以隨物選路的解碼方式補足旅行推銷員子問題的路徑。雙族群遺傳演算法 dpGA 結合 tspGA 以及 kpGA 兩個遺傳演算法，使得 tspGA 或是 kpGA 除了能在自己擅長的範圍搜尋之外，也能夠互相補強自己不足的部分。dpGA 架構為島嶼模型 (island model) [29]，將一個大族群拆成兩個族群平行演化，並利用遷移機制達到傳遞訊息的目的。圖 8 為雙族群遺傳演算法流程圖，中間為 dpGA 主要架構，首先初始化兩個遺傳演算法的族群，接著以 tspGA 和 kpGA 平行演化兩個族群一段時間後，再進行兩族群的遷移策略，使得兩族群在此時間交換資訊，接著再持續演化兩族群，如此循環至終止條件觸發。左右兩邊為遺傳演算法 tspGA 和 kpGA 的架構，經過疊代 *PS* 次的親代選擇與交配，以此生成一定數量的子代探索解空間，其中 *PS* 為族群大小。交配後實行環境篩選，淘汰較差的個體，接著對族群中的個體實行突變以維護族群的多樣性。圖中灰底的機制策略為雙族群遺傳演算法新增的機制，遷移機制將對方族群最佳的個體加入族群中，配合繼承函式即可在演化過程中獲得遷移個體的基因列。

雙族群遺傳演算法 (dpGA) 利用遷移機制使得兩族群分別搜尋單一子問題時，達到互相幫助的效果。而協同演算法則是利用合作池，交替式的搜尋兩個子問題。以 Mei 等人提出的協同演算法 [18] 為例，協同演算法在搜尋旅行推銷員子問題時，其背包子問題由合作池中的個體取得背包基因。在搜尋的過程中，搜尋到的路徑會受限於合作池中的個體背包基因。然而雙族群遺傳演算法在搜尋路徑時，除了有繼承機制繼承另一族群的背包基因外，還有解碼函式找尋適合此路徑的背包物品，使得搜尋過程不會完全受限於另一族群的搜尋解。

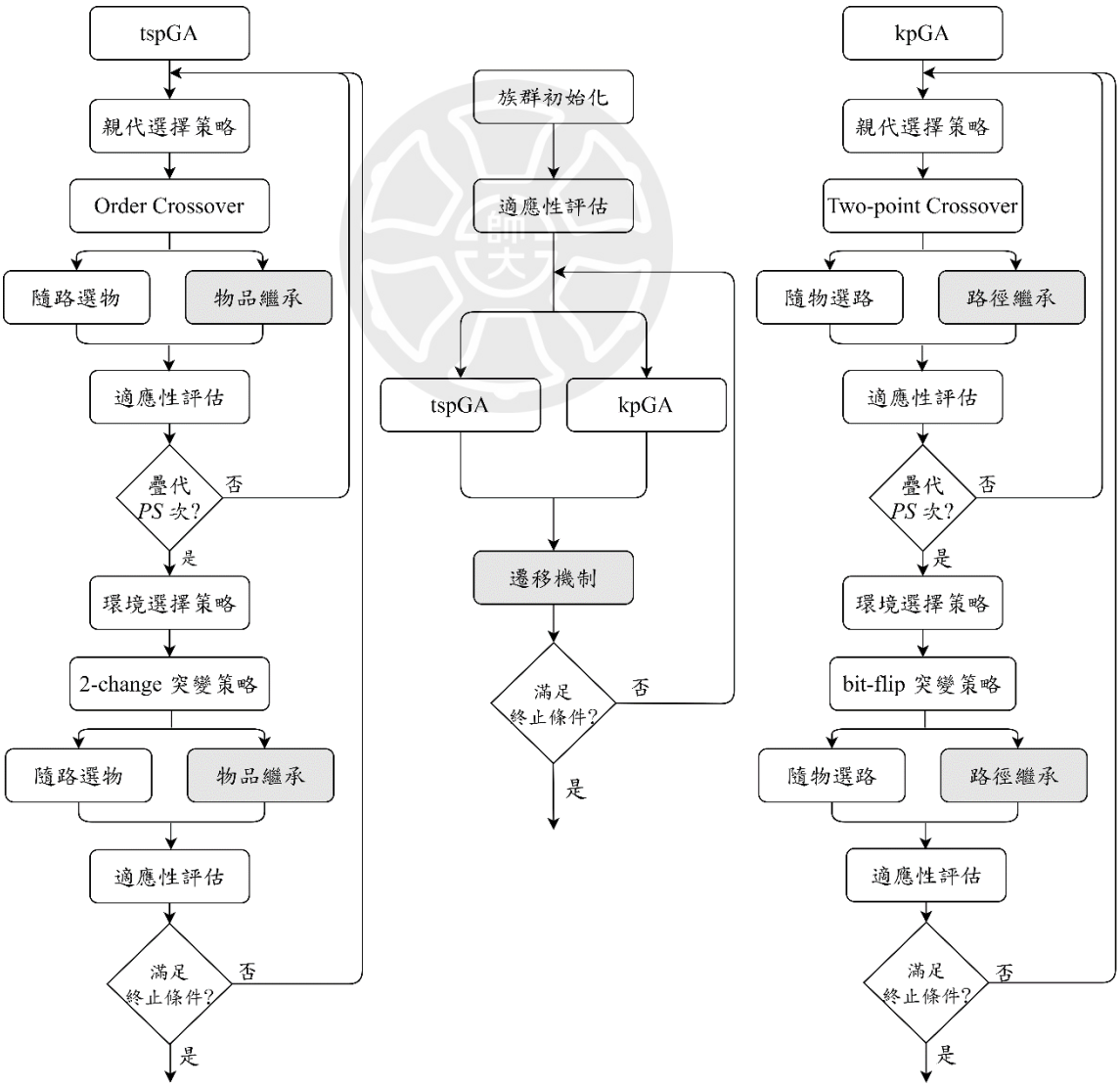

圖 8：雙族群遺傳演算法流程圖

3.2 個體的編碼與評估方式

個體的編碼有兩條基因列，第一條為長度 n 的排序列 X，代表著 n 座城市的拜訪順序，X_1 為固定起始城市 1。另一條為長度 m 的組合列 Y，代表 m 項物品的選取與否。如圖 9 所示，假設現有 7 座城市、6 項物品放置於城市之中，排序列 X 意謂小偷從固定起點城市 1 出發，並依序經過城市 3、5、4、7、6、2，最後回到城市 1 結束旅程。組合列 Y 中的 0 與 1 代表物品是否被小偷取走，圖中小偷從六項物品中，取走編號 2、3、6 的物品。圖中 α 值為自適應控制參數，細節於 3.3.7 小節說明。

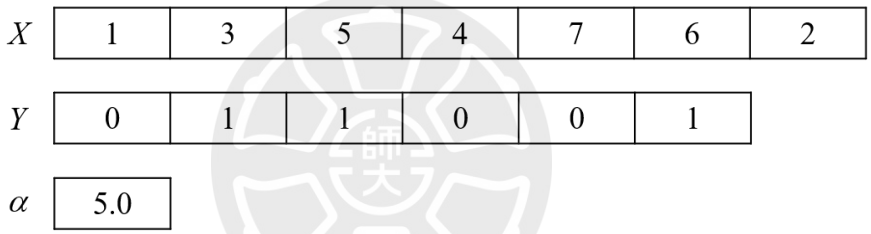

圖 9：遺傳演算法解個體編碼

解個體評估需由兩條基因列 X 以及 Y 計算而來，將小偷行走的路徑 X 與物品選擇 Y 帶入問題目標函式 $F(X, Y)$ 公式 (4) 中，其回傳的數值我們稱之為適應值 (fitness value)，並以此值來判斷個體的優劣，適應值越高代表此個體越好。表 6 為評估函式虛擬碼，4–12 行分別將各座城市 c_k 中有被選取的物品重量總和存至 $cw(c_k)$，並且計算背包物品價值總和 p_{total}。13–18 行計算小偷行走耗費的時間，每經過一座城市 X_i，則將該城市取走的物品重量總合 $cw(X_i)$ 加入當前背包重量 W_{cur} 中，並且計算由城市 X_i 走至 X_{i+1} 耗費的時間。第 19 行即由公式 (4) 得出適應值。

表 6：解個體評估函式虛擬碼

Function Evaluation(*indv*)
Input: *indv*, an input individual
Output: *indv*, a output individual
Notations
X: genes of tour
Y: genes of picking plan
n: number of cities
m: number of items
W_{max}: knapsack capacity
v_{max}, v_{min}: the thief's maximum and minimum velocity
r: knapsack's renting rate
f: fitness value

```
01  X = indv.X
02  Y = indv.Y
03  p_total = 0, t = 0, W_cur = 0
04  for i = 1 to n do
05     cw(i) = 0
06  end for
07  for k = 1 to m do
08     if Y_k = 1 do
09        cw(c_k) = cw(c_k) + w_k     //item k is in city c_k
10        p_total = p_total + p_k
11     end if
12  end for
13  for i = 1 to n − 1 do
14     W_cur = W_cur + cw(X_i)
15     t = t + distance(X_i, X_{i+1}) / (v_max − (v_max − v_min) × W_cur / W_max)
16  end for
17  W_cur = W_cur + cw(X_n)
18  t = t + distance(X_n, X_1) / (v_max − (v_max − v_min) × W_cur / W_max)
19  f(indv) = p_total − r × t
20  return indv
```

3.3 tspGA 演算法

3.3.1 個體的解碼

tspGA 專注於搜尋路徑，而物品則根據路徑以經驗法則挑選。本論文中，隨路選物的解碼方式有兩種，分別為 Pack 函式 (表 4) 與 Pack-Iterative 函式 (表 5)，以此挑選適合路徑的背包物品。表 7 為對個體隨路選物之 Pack 解碼函式虛擬碼，將個體的路徑代入 Pack 函式取得背包物品，並以此背包物品作為此個體的物品選擇。表 8 為對個體隨路選物之 Pack-Iterative 解碼函式虛擬碼，第 1 行 c 值為 Pack-Iterative 函式初始 α 值，其數值採用原文 [9] 設定之數值，接著以 Pack-

Iterative 函式取得個體 *indv* 的物品選擇。

表 7：隨路選物之 Pack 解碼函式虛擬碼

Function PackCompletion(*indv*) Input: *indv*, an input individual Output: *indv*, an output individual Notations *X*: genes of tour *Y*: genes of picking plan α: scoring function parameter
01 *indv.Y* = Pack(*indv.X*, *indv.*α) 02 **return** *ind.Y*

表 8：隨路選物之 Pack-Iterative 解碼函式虛擬碼

Function Pack-IterativeCompletion(*indv*) Input: *indv*, an input individual Output: *indv*, an output individual Notations *indv*: individual *X*: genes of tour *Y*: genes of picking plan α: scoring function parameter *c*: initial α value of Pack-Iterative function
01 *c* = 5 02 *indv.Y* = Pack-Iterative(*indv.X*, *c*) 03 **return** *ind.Y*

3.3.2 族群初始化

使用 CLKH 搜尋法來初始化個體的路徑是許多文獻常用的方法，本論文也會在初始族群時以此路徑 (簡稱為 CLKH 路徑) 配合 Pack-Iterative 經驗法則決定背包物品來生成解。旅行小偷問題不同於對稱的旅行推銷員問題，順時針或逆時針行走同樣的路線也會造成不同的結果，因此在生成初始族群的個體時，會基於每條 CLKH 搜尋法生成的路線，以順時針或逆時針行走方向產生兩個解並皆會加入族群之中。其餘個體皆由隨機的方式生成。

表 9 為 CLKH 路徑個體初始化虛擬碼，第 1 行表示我們會以 CLKH 搜尋法生成 *CT* 條路徑，2–3 行中 X_1、X_2 分別為 CLKH 搜尋法生成之路徑的順時針與逆時針走法，第 4 行將生成的 CLKH 路徑儲存進 *CTS* 集合中，5–6 行由隨路選物之 Pack-Iterative 解碼函式挑選適合該路徑的背包物品，並於第 7 行使用目標函式評估個體，在第 8 行將個體加入族群中。

表 9：CLKH 路徑個體初始化虛擬碼

Function CLKH-Initialization(*pop*, *CT*)
Input: *pop*, an input population
CT, times of running CLKH algorithm
Output: *pop*, an output population
CTS, the set of all CLKH tour generate by CLKH algorithm
Notations
indv: individual
X: genes of tour
CT: times of running CLKH algorithm

```
01  for i = 1 to CT do
02      X1 = Run CLKH algorithm
03      X2 = Reverse(X1)
04      CTS = CTS ∪ { X1, X2 }
05      indv_cw.X, indv_ccw.X = X1, X2
06      indv_cw, indv_ccw = Pack-IterativeCompletion(indv_cw), Pack-IterativeCompletion(indv_cw)
07      indv_cw, indv_ccw = Evaluation(indv_cw), Evaluation(indv_ccw)
08      pop = pop ∪ { indv_cw, indv_ccw }
09  end for
10  return pop, CTS
```

表 10 為 tspGA 族群初始化虛擬碼，此函式接收由表 9 CLKH-Initialization(*pop*, *CT*) 函式初始化的族群，第 2 行生成隨機排序的路徑，第 3 行給予個體 α 值介於 [1, 10] 之間的隨機實數值，4–5 行以隨路選物之 Pack 函式挑選背包物品並評估此個體，以此不斷生成新的個體直到滿足初始族群大小 N^{init}。

表 10：tspGA 族群初始化虛擬碼

Function tspGA-Initialization(*pop*)
Input: *pop*, an input population after CLKH-Initialization
Output: *pop*, an output population
Notations
N^{init}: initial population size
indv: individual
X: genes of tour

```
01  while |pop| < N^init do
02      indv.X = random permutation tour
03      indv.α = rand(1, 10)
04      indv = PackCompletion(indv)
05      indv = Evaluation(indv)
06  end while
07  return pop
```

3.3.3 交配策略

tspGA 採用 Davis 提出的 Order Crossover [30] 來對排列基因交配，假設有兩親代 P_1、P_2 如圖 10 所示，首先隨機選取親代中的一段基因直接植入子代中，選取範圍為兩個切點符號 | 之間，並將 P_1 中該段基因植入 O_2 中，同理，P_2 植入 O_1 如圖 11 所示。接著 O_1 子代依照 P_1 由第二個切點 | 後的城市拜訪順序 3→6→1→4→7→5→2，移除已存在於 O_1 中的城市成 1→7→5→2，並依序由 O_1 中第二個切點 | 後填入，如圖 12 所示，子代 O_2 可同理得之。圖 13 中，將固定起始城市 1 位移回基因列中第一格。

$P_1 = (1\ \ 4 \mid 7\ \ 5\ \ 2 \mid 3\ \ 6)$

$P_2 = (1\ \ 5 \mid 6\ \ 3\ \ 4 \mid 7\ \ 2)$

圖 10：Order Crossover 親代排序基因列

$O_1 = (\times\ \ \times \mid 6\ \ 3\ \ 4 \mid \times\ \ \times)$

$O_2 = (\times\ \ \times \mid 7\ \ 5\ \ 2 \mid \times\ \ \times)$

圖 11：Order Crossover 子代排序基因列 (步驟一)

$O_1 = (5\ \ 2 \mid 6\ \ 3\ \ 4 \mid 1\ \ 7)$

$O_2 = (3\ \ 4 \mid 7\ \ 5\ \ 2 \mid 1\ \ 6)$

圖 12：Order Crossover 子代排序基因列 (步驟二)

$O_1 = (1\ \ 7\ \ 5\ \ 2\ \ 6\ \ 3\ \ 4)$

$O_2 = (1\ \ 6\ \ 3\ \ 4\ \ 7\ \ 5\ \ 2)$

圖 13：Order Crossover 子代排序基因列 (步驟三)

3.3.4 自適應控制機制

tspGA 於 Order Crossover 交配後使用隨路選物 Pack 函式 (表 4) 挑選物品，由於評分公式 (5) 中的 α 值會對 Pack 函式挑選物品造成影響，因此本論文針對此 α 值提出自適應控制，幫助族群於子代路徑解碼步驟中，挑選到更合適的物品。本論文採用 jDE [31] 提出的自適應參數控制方式，將參數 α 編碼在個體上，如圖 9 所示，使得參數跟隨族群的演化淘汰或留存。子代參數值依指定機率由親代遺傳或是隨機產生，由公式 (6) 選擇子代 α 值 (簡稱 α_{child})，$rand_1$ 為介於 [1, 10] 之間的隨機小數，$rand_2$ 為介於 [0, 1] 之間的隨機小數，pr 為介於 [0, 1] 之間的固定小數值。

$$\alpha_{\text{child}} = \begin{cases} rand_1, & \text{if } rand_2 < pr \\ \alpha_{\text{parent}}, & \text{otherwise} \end{cases} \tag{6}$$

表 11 為自適應控制函式虛擬碼，1–2 行表示子代使用之 α 值有 p_r 機率為隨機值 $rand_1$，否則 (3–9 行) 子代 α 值遺傳自親代 α 值。遺傳演算法親代個體數目有兩個，本研究令子代 α 值繼承兩親代個體中適應值較佳個體的 α 值 (4–8 行)。

表 11：自適應控制虛擬碼

Function Self-AdaptiveControl(*child*, $parent_1$, $parent_2$)
Input: $parent_1$, $parent_2$, parental individuals
child,offspring individual
Output: *child*, offspring individual
Notations
α: scoring function parameter
f: fitness value

```
01  if rand(0, 1) < pr do
02     child.α = rand(1, 10)
03  else
04     if f(p1) > f(p2) do
05        child.α = parent1.α
06     else
07        child.α = parent2.α
08     end if
09  end if
10  return child
```

表 12 為 tspGA 完整交配策略虛擬碼，1–3 行由 Order Crossover 生成兩個子代，並以自適應控制機制選擇子代個體的 α 值，4–5 行以隨路選物之 Pack 解碼函式挑選物品並評估，即可得到完整的子代個體。

表 12：tspGA 交配策略虛擬碼

Function tspGA-Crossover(p_1, p_2) Input: p_1, p_2, parental individuals Output: c_1, c_2, offspring individuals	
01	c_1, c_2 = OrderCrossover(p_1, p_2)
02	c_1 = Self-AdaptiveControl(c_1, p_1, p_2)
03	c_2 = Self-AdaptiveControl(c_2, p_1, p_2)
04	c_1, c_2 = PackCompletion(c_1), PackCompletion(c_2)
05	c_1, c_2 = Evaluation(c_1), Evaluation(c_2)
06	**return** c_1, c_2

3.3.5 突變策略

在演化過程中，交配策略使得族群往菁英個體靠近，最終導致族群個體相近，失去演化的動力。突變的目的在於維護族群的多樣性，讓族群保有演化搜尋的能力。本論文中，於環境篩選後選取適應值相同的個體進行突變，以此維護族群的多樣性。tspGA 對排列基因採用兩次 2-change (表 1) 的突變策略。表 13 為 tspGA 突變策略虛擬碼，首先第 1 行選擇 4 個不同的隨機值，範圍介於 [2, n] 之間，n 為城市數量，2–3 行以此 4 個隨機值執行兩次 2-change 的突變，第 4 行以隨路選物之 Pack 函式作為此路徑的解碼函式 (表 7)。

表 13：tspGA 突變策略虛擬碼

Function tspGA-Mutation(*indv*) Input: *indv*, an input individual Output: *indv*, an output individual Notations *indv*: individual *X*: genes of tour *n*: number of cities	
01	r_1, r_2, r_3, r_4 = rand-Int(2, n), rand-Int(2, n), rand-Int(2, n), rand-Int(2, n)
02	*indv.X* = 2-change(*indv.X*, r_1, r_2)
03	*indv.X* = 2-change(*indv.X*, r_3, r_4)
04	*indv* = PackCompletion(*indv*)
05	*indv* = Evaluation(*indv*)
06	**return** *indv*

3.4 kpGA 演算法

3.4.1 個體的解碼

kpGA 為著重搜尋背包子問題的遺傳演算法，本論文以 CLKH 路徑作為背包組合基因的解碼方式，嘗試 *CTS* (CLKH 路徑集合) 中所有的路徑，並選擇最佳的 CLKH 路徑作為解個體路徑。*CTS* 路徑集合由表 9 CLKH-Initialization(*pop*, *CT*) 函式回傳取得。表 14 為隨物選路解碼函式虛擬碼，首先於 1–2 行取出 *CTS* 集合中第一條 CLKH 路徑作為解個體 *indv* 的路徑並評估，4–9 行評估其餘 *CTS* 集合中的 CLKH 路徑作為解個體新的路徑，並將最佳的個體存於解 *indv_best* 中。

表 14：隨物選路解碼函式虛擬碼

Function ClkhCompletion(*indv*, *CTS*)
Input: *indv*, an input individual
CTS: the set of all CLKH tour running by CLKH algorithm
Output: *indv*, an output individual
Notations
indv_best: current best individual
X: genes of tour

```
01  indv.X= CTS_1
02  indv = Evaluation(indv)
03  indv_best = indv
04  for i = 2 to |CTS| do
05     indv.X = CTS_i
06     indv = Evaluation(indv)
07     if f(indv) > f(indv_best) do
08        indv_best = indv
09     end if
10  end for
11  return indv_best
```

3.4.2 修復機制

背包組合的基因需遵守偷取的物品重量不能超過背包的重量之限制，本論文的修復策略隨機移除偷取的物品直到符合背包的承受重量。表 15 為虛擬碼，W_{cur} 為此時背包重量，若超出重量上限 $W_{\max}$，則隨機移除背包中的物品。

表 15：背包組合基因修復機制虛擬碼

Function KnapsackRepair(*indv*) Input: *indv*, an input indvidual Output: *indv*, an output individual Notations W_{cur}: current weight $W_{\max}$: maximum knapsack's capacity
01 W_{cur} = Calculate total weight of *indv.Y* 02 **while** $W_{cur} > W_{\max}$ **do** 03 remove random item from *indv.Y* 04 W_{cur} = Calculate total weight of *indv.Y* 05 **end while** 06 **return** *ind*

3.4.3 族群初始化

表 16 為 kpGA 族群初始化虛擬碼，此函式傳入由表 9 CLKH-Initialization(*pop*, *CT*) 函式初始化的族群以及 CLKH 路徑集合 *CTS*，2–4 行隨機挑選物品 Y_i 是否放入背包，其中 $Y_i \in \{0, 1\}$，0 代表不選，1 代表選擇放入背包。第 6 行對此背包物品執行修復機制，7–8 行以隨物選路解碼函式挑選 CLKH 路徑並評估此個體。

表 16：kpGA 族群初始化虛擬碼

Function kpGA-Initialization(*pop, CTS*) Input: *pop*, an input population after CLKH-Initialization *CTS*, the set of all CLKH tours generated by CLKH algorithm Output: *pop*, an output population Notations N^{init}: initial population size *indv*: individual *Y*: genes of picking plan *m*: number of items
01 **while** $\|pop\| < N^{\text{init}}$ **do** 02 **for** *i* = 1 **to** *m* **do** 03 Y_i = rand-Int(0, 1) 04 **end for** 05 *indv.Y* = *Y* 06 *indv* = KnapsackRepair(*indv*) 07 *indv* = ClkhCompletion(*indv*, *CTS*) 08 *indv* = Evaluation(*indv*) 09 **end while** 10 **return** *pop*

3.4.4 交配策略

kpGA 選用 two-point crossover 作為背包組合基因的交配策略。首先隨機選取兩切點 (|)，如圖 14 所示，將親代基因切出一段來，植入該段基因於子代中。如圖 15 所示，親代 P_1 取出中間一段植入子代 O_2 中，同理，P_2 取出中間一段植入子代 O_1 中，接著其餘基因由另一方親代直接植入，結果如圖 16 所示。

$$P_1 = (0\ \ 1 \mid 0\ \ 0\ \ 1 \mid 1\ \ 1)$$
$$P_2 = (1\ \ 0 \mid 1\ \ 0\ \ 0 \mid 0\ \ 1)$$

圖 14：two-point crossover 親代組合基因列

$$O_1 = (\times\ \ \times \mid 1\ \ 0\ \ 0 \mid \times\ \ \times)$$
$$O_2 = (\times\ \ \times \mid 0\ \ 0\ \ 1 \mid \times\ \ \times)$$

圖 15：two-point crossover 子代組合基因列 (步驟一)

$$O_1 = (0\ \ 1 \mid 1\ \ 0\ \ 0 \mid 1\ \ 1)$$
$$O_2 = (1\ \ 0 \mid 0\ \ 0\ \ 1 \mid 0\ \ 1)$$

圖 16：two-point crossover 子代組合基因列 (步驟二)

表 17 為 kpGA 完整交配策略虛擬碼，1–2 行由 two-point crossover 生成兩個子代，並修復子代個體中的背包物品。3–4 行完成隨物選路解碼函式並評估，即可得到完整的子代個體。

表 17：kpGA 交配策略虛擬碼

Function kpGA-Crossover(p_1, p_2, *CTS*) Input: p_1, p_2, parental individuals *CTS*, the set of all CLKH tours generated by CLKH algorithm Output: c_1, c_2, offspring individuals	
01	c_1, c_2 = TwoPointCrossover(p_1, p_2)
02	c_1, c_2 = KnapsackRepair(c_1), KnapsackRepair (c_2)
03	c_1, c_2 = ClkhCompletion(c_1, *CTS*), ClkhCompletion(c_2, *CTS*)
04	c_1, c_2 = Evaluation(c_1), Evaluation(c_2)
05	**return** c_1, c_2

3.4.5 突變策略

kpGA 對每項物品以 *pm* 機率進行 bit-flip 突變，虛擬演算法由表 18 顯示。表 19 為 kpGA 突變策虛擬碼，第 1 行針對背包物品執行機率性突變策略，第 2 行修復背包物品成合法解，意即背包重量不得超過規定上限，3–4 行以隨物選路解碼函式挑選路徑並評估。

表 18：機率性突變策略虛擬碼

Function ProbabilityMutation(*Y*) Input: *Y*, an input picking plan Output: *Y*, a picking plan after mutate Notations *m*: number of items W_{max}: maximum knapsack's capacity
01 **for** *i* = 1 **to** *m* **do** 02 **if** rand(0, 1) < *pm* **do** 03 $Y_i = (Y_i + 1) \bmod 2$ 04 **end if** 05 **end for** 06 **return** *Y*

表 19：kpGA 突變策略虛擬碼

Function kpGA-Mutation(*indv*, *CTS*) Input: *indv*, an input individual *CTS*, the set of all CLKH tour generate by CLKH algorithm Output: *indv*, an output individual Notations *indv*: individual *Y*: genes of picking plan *n*: number of cities
01 *indv.Y* = ProbabilityMutation(*indv.Y*) 02 *indv* = KnapsackRepair(*indv*) 03 *indv* = ClkhCompletion(*indv*, *CTS*) 04 *indv* = Evaluation(*indv*) 05 **return** *indv*

3.5 遺傳演算法通用機制

3.5.1 親代選擇策略

交配的策略需先選出兩個親代個體才得以進行。本論文採用等級輪盤法與

隨機的方式各選一個親代。等級輪盤法有高機率選取到較佳的個體，利於整個族群往好的個體方向靠近；而另一個親代採用隨機方式選取可以讓族群往不同的方向搜尋，而非一味地靠近族群中好的個體。等級輪盤法以適應值對個體進行排名，排名越高被選到的機率越高，舉例來說，假如族群中有 40 個個體，排名 1 至排名 40 被選重的比例依序由 40 遞減至 1，因此，排名第 1 的就有 40 / (40 × (40+1) / 2) 的機率被選中。表 20 為等級輪盤法虛擬碼，第 1 行首先隨機選出總比例中的一位數 *r*，接著 2–7 行觀察此數位於哪一個個體的範圍內，並於第 6 行回傳被選中的個體。

表 20：等級輪盤法虛擬碼

Function RankBased-RouletteWheelSelection(*pop*)
Input: *pop*, an input population sorted by fitness in decreasing order
Output: *indv*, output individual
Notations
|*pop*|: population size

```
01  r = rand-Int(1, (|pop| + 1) × |pop| / 2)
02  cnt = 0
03  for i = 1 to |pop| do
04     cnt = cnt + i
05     if cnt >= r do
06        return pop[|pop| – i + 1]
07  end for
```

3.5.2 環境選擇策略

本論文環境選擇策略較為單純，淘汰較差的個體直到初始族群的大小 N^{init}，虛擬碼由表 21 顯示，保留前 N^{init} 個好的個體，其餘全部淘汰。

表 21：環境選擇策略虛擬碼

Function EnvironmentalSelection (*pop*,N^{init})
Input: *pop*, an input population
N^{init}: initial population size
Output: *pop*, a population after selection
Notations
|*pop*|: population size

```
01  while |pop| > N^init do
02     remove worst individual from pop
03  end while
04  return pop
```

3.6 遺傳演算法虛擬碼

表 22 為遺傳演算法 tspGA 虛擬碼，首先傳入經過表 10 tspGA-Initialization(*pop*) 初始化的族群 *pop*。2–7 行為族群的繁衍，合計執行 N^{init} 次交配，給予族群一段時間搜尋好的個體再淘汰差勁的個體，3–4 行由等級輪盤法和隨機法選出兩個親代個體，5–6 行執行交配策略生成兩個子代並加入族群中，第 8 行為環境篩選，淘汰族群中不良的個體，並在 9–13 行針對族群實行突變策略以此維護族群多樣性，10–11 行突變策略選擇族群中兩個相同適應值個體之一 $indv_1$ 進行突變，第 12 行將突變後的個體 $indv_2$ 取代原先族群中的個體 $indv_1$。表 23 為遺傳演算法 kpGA 虛擬碼，整體架構與 tspGA 類似，2–7 行為交配演化，第 8 行執行環境篩選，9–13 行對族群相同適應值之個體實行突變策略以維護族群多樣性。

表 22：tspGA 演算法虛擬碼

Function tspGA (*pop*)
Input: *pop*, a population after initialization
Output: *pop*, a population after evolution
Notations
N^{init}: initial population size
p, parental individual
c, offspring individual
indv, individual

```
01  while did not meet the stopping criterion do
02    for i = 1 to N^init do
03      p1 = RankBased-RouletteWheelSelection(pop)
04      p2 = random choose individual from pop
05      c1, c2 = tspGA-Crossover(p1, p2)
06      pop = pop ∪ {c1, c2}
07    end for
08    EnvironmentalSelection (pop, N^init)
09    while pop contains same-fitness individuals do
10      indv1 = one of same-fitness individuals
11      indv2 = tspGA-Mutation(indv1)
12      Replace indv1 by indv2 in pop
13    end while
13  end while
15  return pop
```

表 23：kpGA 演算法虛擬碼

```
Function kpGA (pop, CTS)
Input: pop, a population after initialization
       CTS, the set of all CLKH tour generate by CLKH algorithm
Output: pop, a population after evolution
Notations
N^init: initial population size
01  while did not meet the stopping criterion do
02    for i = 1 to N^init do
03      p1 = RankBased-RouletteWheelSelection(pop)
04      p2 = random choose individual from pop
05      c1, c2 = kpGA-Crossover(p1, p2, CTS)
06      pop = pop ∪ {c1, c2}
07    end for
08    EnvironmentalSelection (pop, N^init)
09    while pop contains same-fitness individuals do
10      indv1 = one of same-fitness individuals
11      indv2 = kpGA-Mutation(indv1, CTS)
12      Replace indv1 by indv2 in pop
13    end while
13  end while
15  return pop
```

3.7 遷移機制與繼承機制

雙族群遺傳演算法中，兩子族群專注求解的方向不同，因此適時的從對方中獲取資訊能夠有效幫助演化。表 24 為遷移機制虛擬碼，1–2 行中 *indv_tsp*、*indv_kp* 分別為兩子族群的最佳個體。第 3 行為了配合自適應控制機制，*indv_kp* 的 α 值以 tspGA 子族群中最佳個體 *indv_tsp* 的 α 值代替，此作法用意在於保護 tspGA 的自適應控制 α 值的演化，若是 kpGA 子族群遷移過來的菁英個體帶有差勁的 α 值，則此 α 值有可能因為此菁英個體足夠強大而無法被環境篩選淘汰，進而影響到 tspGA 子族群的演化，4–5 行將 *indv_tsp*、*indv_kp* 分別加入兩族群 *pop_kp*、*pop_tsp* 中，並取代兩族群中最差的個體。

表 24：遷移機制虛擬碼

Function Migration(*pop_tsp*, *pop_kp*)
Input: *pop_tsp*, an population for tspGA
pop_kp, an population for kpGA
Output: *pop_tsp*, *pop_kp*, two output population
Notations
α: scoring function parameter
indv: individual

01 *indv_tsp* = best individual in *pop_tsp*
02 *indv_kp* = best individual in *pop_kp*
03 *indv_kp*.α = *indv_tsp*.α
04 Replace worst individual in *pop_tsp* by *indv_kp*
05 Replace worst individual in *pop_kp* by *indv_tsp*
06 **return** *pop_tsp*, *pop_kp*

為了能有效的使用遷移之個體，我們新增物品繼承以及路徑繼承機制，使得 tspGA 專注演化旅行推銷員子問題時，也能得到好的背包基因列，反過來說，kpGA 也能夠得到好的路徑基因列。表 25 為物品繼承函式虛擬碼，1–2 行個體 *c* 繼承個體 *p* 的背包基因列形成新的個體 *indv*，第 3 行評估新個體 *indv*，並於第 4 行回傳新個體。表 26 為路徑繼承函式虛擬碼，個體 *c* 繼承個體 *p* 的路徑基因列形成新的個體 *indv*，評估新個體後回傳。

表 25：物品繼承函式虛擬碼

Function ItemInheritance(*c*, *p*)
Input: *p*, parental individuals
c, offspring individuals
Output: *indv*, an output individual
Notations
Y: genes of picking plan

01 *indv* = *c*
02 *indv.Y* = *p.Y*
03 *indv* = Evaluation(*indv*)
04 **return** *indv*

表 26：路徑繼承函式虛擬碼

Function TourInheritance(*c*, *p*)
Input: *p*, parental individuals
c, offspring individuals
Output: *indv*, an output individual
Notations
X: genes of tour

01 *indv* = *c*
02 *indv.X* = *p.X*
03 *indv* = Evaluation(*indv*)
04 **return** *indv*

加入繼承機制的 tspGA 以及 kpGA 在交配策略與突變策略會有改動，表 27 以及表 28 為新定義之交配策略，2–4 行 tspGA 交配策略中除了以 Pack 解碼函式挑選物品生成子代 c_{11}, c_{21}，5–9 行選擇繼承親代中適應值較佳的背包物品形成子代 c_{12}, c_{22}，並於 12–22 行判斷兩種挑選物品的方式何者較佳並存留。kpGA 交配策略架構與 tspGA 相同，除了 CLKH 解碼函式生成路徑之外，也會繼承親代個體的路徑，並判斷兩種生成路徑的方式何者較佳以存留。表 29、30 為 tspGA 以及 kpGA 新定義之突變策略虛擬碼，選擇以解碼函式或是繼承函式中適應值較佳的留存。

表 27：tspGA 交配策略虛擬碼

Function tspGA-Crossover(p_1, p_2)
Input: p_1, p_2, parental individuals
Output: c_1, c_2, offspring individuals
Notations
f: fitness value

01 c_1, c_2 = OrderCrossover(p_1, p_2)
02 c_1 = Self-AdaptiveControl(c_1, p_1, p_2)
03 c_2 = Self-AdaptiveControl(c_2, p_1, p_2)
04 c_{11}, c_{21} = PackCompletion(c_1), PackCompletion(c_2)
05 **if** $f(p_1) > f(p_2)$ **do**
06 c_{12}, c_{22} = ItemInheritance(c_1, p_1), ItemInheritance (c_2, p_1)
07 **else**
08 c_{12}, c_{22} = ItemInheritance(c_1, p_2), ItemInheritance (c_2, p_2)
09 **end if**
10 c_{11}, c_{21} = Evaluation(c_{11}), Evaluation(c_{21})
11 c_{12}, c_{22} = Evaluation(c_{11}), Evaluation(c_{21})
12 **if** $f(c_{11}) > f(c_{12})$ **do**
13 $c_1 = c_{11}$
14 **else**
15 $c_1 = c_{12}$
16 **end if**
17 **if** $f(c_{21}) > f(c_{22})$ **do**
18 $c_2 = c_{21}$
19 **else**
20 $c_2 = c_{22}$
21 **end if**
22 **return** c_1, c_2

表 28：kpGA 交配策略虛擬碼

Function kpGA-Crossover(p_1, p_2, *CTS*)
Input: p_1, p_2, parental individuals
CTS, the set of all CLKH tours generated by CLKH algorithm
Output: c_1, c_2, offspring individuals

Notations
f: fitness value

01 c_1, c_2 = TwoPointCrossover(p_1, p_2)
02 c_1, c_2 = KnapsackRepair(c_1), KnapsackRepair (c_2)
03 c_{11}, c_{21} = ClkhCompletion(c_1, *CTS*), ClkhCompletion(c_2, *CTS*)
04 **if** $f(p_1) > f(p_2)$ **do**
05 c_{12}, c_{22} = TourInheritance(c_1, p_1), TourInheritance (c_2, p_1)
06 **else**
07 c_{12}, c_{22} = TourInheritance(c_1, p_2), TourInheritance (c_2, p_2)
08 **end if**
09 c_{11}, c_{21} = Evaluation(c_{11}), Evaluation(c_{21})
10 c_{12}, c_{22} = Evaluation(c_{11}), Evaluation(c_{21})
11 **if** $f(c_{11}) > f(c_{12})$ **do**
12 $c_1 = c_{11}$
13 **else**
14 $c_1 = c_{12}$
15 **end if**
16 **if** $f(c_{21}) > f(c_{22})$ **do**
17 $c_2 = c_{21}$
18 **else**
19 $c_2 = c_{22}$
20 **end if**
21 **return** c_1, c_2

表 29：tspGA 突變策略虛擬碼

Function tspGA-Mutation(*indv*)
Input: *indv*, an input individual
Output: *indv*, an output individual
Notations
indv: individual
X: genes of tour
n: number of cities
f: fitness value

01 r_1, r_2, r_3, r_4 = rand-Int(2, *n*), rand-Int(2, *n*), rand-Int(2, *n*), rand-Int(2, *n*)
02 *indv.X* = 2-change(*indv.X*, r_1, r_2)
03 *indv.X* = 2-change(*indv.X*, r_3, r_4)
04 $indv_1$ = PackCompletion(*indv*)
05 $indv_2$ = ItemInheritance(*indv*)
06 $indv_1$, $indv_2$ = Evaluation($indv_1$), Evaluation($indv_2$)
07 **if** $f(indv_1) > f(indv_2)$ **do**
08 $indv = indv_1$
09 **else**
10 $indv = indv_2$
11 **end if**
12 **return** *indv*

表 30：kpGA 突變策略虛擬碼

Function kpGA-Mutation(*indv*, *CTS*)
Input: *indv*, an input individual
CTS, the set of all CLKH tours generated by CLKH algorithm
Output: *indv*, an output individual
Notations
indv: individual
Y: genes of picking plan
n: number of cities
f: fitness value

```
01  indv.Y = ProbabilityMutation(indv.Y)
02  indv = KnapsackRepair(indv)
03  indv1 = ClkhCompletion(indv, CTS)
04  indv2 = TourInheritance(indv)
05  indv1, indv2 = Evaluation(indv1), Evaluation(indv2)
06  if f(indv1) > f(indv2) do
07     indv = indv1
08  else
09     indv = indv2
10  end if
11  return indv
```

3.8 dpGA 虛擬碼

表 31 為雙族群遺傳演算法虛擬碼，首先於 1–4 行初始化兩族群，7–8 行分別執行 tspGA 以及 kpGA 兩種遺傳演算法對兩族群進行演化，9–11 行表示兩個族群演化過程每隔 *MT* 個世代，也就是兩演算法各自執行 *MT* 個世代，即會進行一次族群的遷移機制。最後第 14 行回傳兩族群中的最佳解個體，此個體即為此次演化搜尋得出之最佳結果。

表 31：dpGA 演算法虛擬碼

```
Function dpGA(pop1, pop2)
Input: pop1, an empty population for tspGA
       pop2, an empty population for kpGA
Output: indv, best individual after evolution
Notations
MT: migration timing
g: evolution generation
CTS: the set of all CLKH tour generate by CLKH algorithm
MaxTime: algorithm maximum computing time
01  pop1, CTS = CLKH-Initialization(pop1)
02  pop2 = pop1
03  pop1 = tspGA-Initialization(pop1)
04  pop2 = kpGA-Initialization(pop2, CTS)
05  g = 1
06  while computing time < MaxTime do
07     pop1 = tspGA(pop1)
08     pop2 = kpGA(pop2, CTS)
09     if g mod MT equals 0 do
10        Migration(pop1, pop2)
11     end if
12     g = g + 1
13  end while
14  return best individual from pop1, pop2
```

第四章　實驗設計與結果

4.1 測試資料集

在 2014 年，Polyakovskiy 等人提供旅行小偷問題測試資料集 [3]，設定計算時間為 10 分鐘。測試資料集中，每組測試資料包含物品參數 $F \in \{1, 3, 5, 10\}$ 與背包容量參數 $C \in \{1,\ldots, 10\}$。物品參數 F 代表每座城市放置的物品數量，且每座城市物品數量相同；背包容量參數 C 限制背包容量，每組測試資料的背包重量上限為所有物品總重量 W_{total} 的 $\frac{C}{11}$ 倍。

測試資料集以“城市數_物品數_資料型態_背包容量參數”來表示。以 a280_n279_bounded-strongly-corr_01 為例，a280 為旅行推銷員的測試資料，代表此二維空間中存在 280 座城市；此測試資料集中，每座城市擁有的物品數量相同 (起始城市無物品放置)，n279 代表有 279 件物品數量，因此每座城市有一件物品供小偷偷取。最後是背包容量參數 $C = 1$，限制著背包容量上限為所有物品總重量 W_{total} 的 $\frac{1}{11}$ 倍。

資料型態分為物品價值重量強相關 (bounded strongly correlated)、物品價值重量無相關且物品重量相近 (uncorrelated with similar weights) 與物品價值重量無相關 (uncorrelated) 三種。三種資料型態差別在於物品的重量與價值的產生方式，「物品價值重量強相關」的物品重量均勻分佈於 [1, 1000] 之間，而物品價值為其重量值加 100。「物品價值重量無相關且物品重量相近」的測試資料其物品重量與價值分別均勻分佈於 [1000, 1000+10] 與 [1, 1000] 之間。「物品價值重量無相關」的測資較為單純，重量與價值皆均勻分佈於 [1, 1000] 之間。

本文實驗測試資料集依照於 2020 年 Maity 與 Das [10] 採用的實驗測試資料集，其從 Polyakovskiy 等人提供的旅行小偷問題測試資料集 [3] 中選取 84 組測試資料，其中分為三種類別的測試資料，每種類別皆包含 28 筆不同城市數量的測試資料，範圍從 52 座城市到 33810 座城市，以下為三種類別：

類別 1：每座城市一項物品, 物品價值重量強相關
類別 2：每座城市五項物品, 物品價值重量無相關且物品重量相近
類別 3：每座城市十項物品, 物品價值重量無相關

此測試資料集包含了大小型測試資料與種類，利於判斷演算法的強項與弱項，所有測試資料附於下表 32，第一列為旅行推銷員問題的測試資料集種類，而二到四列為各類別測試資料的背包容量參數 C。

表 32：測試資料集與背包容量參數 C

測試資料集	背包容量參數		
	類別 1	類別 2	類別 3
a280	2	1	1
berlin52	2	1	1
bier127	1	1	1
brd14051	2	1	1
ch130	1	1	4
ch150	4	1	1
d2103	1	1	1
d15112	1	1	2
d18512	1	1	1
dsj1000	1	1	1
eil51	1	1	1
eil76	2	1	1
fl1577	1	1	1
fnl4461	1	1	2
kroA100	1	1	1
kroA200	1	1	2
lin318	1	1	1
pcb3038	1	1	1
pla7397	1	1	1
pla33810	1	1	1
pr124	1	1	1
rl11849	1	1	1
rl1304	1	1	1
ts225	5	1	2
u159	2	1	3
u574	2	1	1
u724	2	1	1
usa13509	1	1	1

4.2 演算法參數設定

本研究所提出演算法 dpGA 所有參數設定如表 33 所示，總共分為四個部分，分別是 (1) dpGA 架構內參數 N^{init}、*MT*、*MaxTime* 與 *CT*，(2) kpGA 突變策略參數 *pm*，(3) tspGA 自適應控制參數 *pr*，(4) Pack-Iterative 經驗法則參數 τ、c、φ、q、ε。其中參數 N^{init}、*MT*、*CT*、*pm* 以及 *pr* 經本章節後續實驗調整後設定之，而其他參數則沿用原文獻之建議設定。

表 33：dpGA 演算法參數設定

參數名稱	用途	值	備註	參數虛擬碼位置
N^{init}	子族群初始大小	30		表 10、16
CT	族群初始化 CLKH 路徑個數	5	本研究實驗後設定	表 9
MT	兩個子族群遷移時機	50		表 31
MaxTime	最大可用的計算時間	600	文獻 [3] 規定	表 31
pm	機率性突變策略中，每個基因突變機率	0.005	本研究實驗後設定	表 18
pr	自適應控制機制中選擇隨機的機率	0.1	本研究實驗後設定	表 11
τ	用於計算頻率參數的固定整數	100	Pack-Iterative [9] 原文參數設定	表 5
c	評分公式 (5) 初始指數	5		
φ	評分公式 (5) 初始指數偏移量	2.5		
q	Pack 函式疊代次數	20		
ε	閾值常數	0.1		

4.3 實驗環境設定

本研究之演算法以 C++ 編寫，並在 Windows 10 作業系統下執行，中央處理器為 Intel i7-8700 (3.2 GHz)，記憶體為 24 GB。4.4–4.6 小節 tspGA 以及 kpGA 演算法相關實驗以 5 分鐘為實驗終止條件，4.7–4.8 小節 dpGA 演算法相關實驗以及文獻演算法以 10 分鐘為實驗終止條件。

本研究實驗皆以 10 次獨立執行各版本演算法，並採用獨立雙樣本中位數差異檢定 (Wilcoxon rank-sum test) 進行比較。

4.4 CLKH 路徑差異影響實驗

本論文以 [32] 所提供之 CLKH 搜尋法程式產生 CLKH 路徑，由於每次 CLKH 生成之路徑不盡相同，因此本小節探討使用不同 CLKH 路徑初始化族群，是否影響族群演化的結果。本小節以自適應控制遺傳演算法 tspGA 做為測試演算法，以 CLKH 隨機生成兩組 CLKH 路徑，作為測試演算法 tspGA 的初始化路徑，而兩組 CLKH 路徑分別稱為第 1 組與第 2 組。此實驗設定 CLKH 路徑初始化個數 *CT* 值為 1。

表 34 為 CLKH 路徑差異影響實驗統計檢定結果，分別以第 1 組、第 2 組初始化 CLKH 路徑個體。類別 1 檢定結果 5 / 5 代表第 1 組與第 2 組各贏對方 5 個測試資料，其餘測試資料的結果並無差異；類別 2 中第 1 組贏對方 6 個測試資料，輸 7 個測試資料。下表可以說明在每個問題類別 28 筆測試資料下，兩組 CLKH 初始化路徑在類別 1、2、3 各有 10、13、14 個測試資料的求解結果是有差異的。為了避免 CLKH 路徑影響我們觀察演化演算法各項機制的作用與影響性，後續實驗 4.5–4.7.3 小節皆採用固定的 CLKH 路徑 (第 1 組)。

表 34：CLKH 路徑差異影響實驗統計檢定結果

第 1 組	vs. 第 2 組
測資類別 1	5 / 5
測資類別 2	6 / 7
測資類別 3	7 / 7

4.5 tspGA 相關實驗

本研究提出自適應控制機制來調整 Pack-Iterative 背包求解方法中的評分參數 α，以下將自適應控制機制分為二個實驗討論。第一個實驗討論自適應控制的效果，第二個實驗討論自適應控制相關參數設定。另外還有兩個實驗測試單一世代子代個數對於演化的影響性，以及討論維護族群的重要性。以下實驗遺傳演算法初始族群大小皆為 30。

4.5.1 自適應控制機制實驗

此實驗比較「使用自適應控制」與「使用 Pack-Iterative」調整 Pack 函式中的 α 值之求解品質，同時納入使用「隨機 α 值」之版本。以下總共為三個版本：

版本 S (Self-adaptive):以自適應控制 α 值的 Pack 函式為背包問題的解碼函式
版本 R (Random): 以隨機 α 值的 Pack 函式為背包問題的解碼函式
版本 I (Iterative): 以 Pack-Iterative 函式為背包問題的解碼函式

在版本 S 與版本 R 中，族群初始化會加入 CLKH 路徑搭配 Pack-Iterative 的個體，其餘個體的背包基因皆用隨機 α 值 Pack 函式求得，版本 I 則是所有的初始化個體皆採用 Pack-Iterative 求得背包組合。

表 35 為實驗統計檢定結果，中間為版本 S 與版本 R 比較，版本 S 於類別 1、2、3 三個類別分別有 9、12、12 個測試資料較為優秀 (33/84 ≒ 40%)，代表自適應控制 α 值相較使用隨機 α 值更佳。版本 S 與版本 I 比較中，使用自適應控制 Pack 明顯比疊代搜尋 Pack (Pack-Iterative) 更好 (超過一半測資較好，無測資較差)，可能是因為疊代搜尋的解碼方式較為耗費計算資源，導致搜尋不同路徑的次數下降。

表 35：自適應控制機制實驗統計檢定結果

版本 S	vs. 版本 R	vs. 版本 I
測資類別 1	9 / 0	12 / 0
測資類別 2	12 / 0	19 / 0
測資類別 3	12 / 2	14 / 0

圖 17–19 與表 36 為三個版本在測資類別 1、2、3 的比較圖表，以三個版本中最佳平均數作正規化，意即將各版本的平均數除以三個版本中最佳的平均數再乘上 100，因此圖表中 100 即為最佳的平均數。圖 17–19 中，版本 S 在大多數測試資料中擁有較佳的平均數。表 36 為各項測試資料正規化平均數，表中 + 號為我們所提出的版本 S 統計檢定獲勝的測試資料，– 號為版本 S 輸的測試資料。

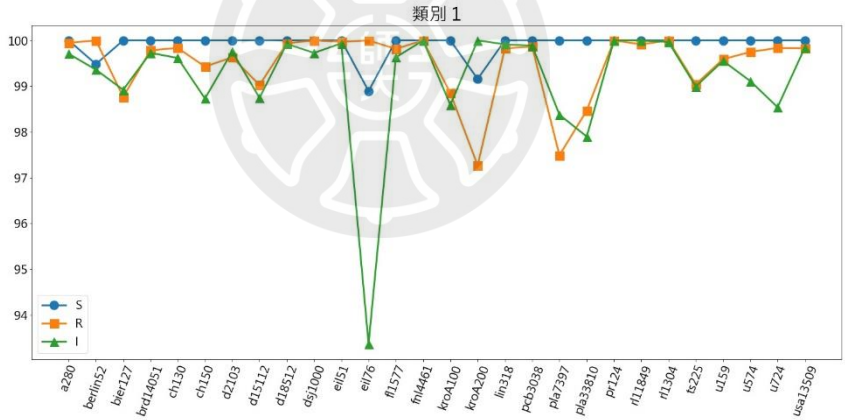

圖 17：測資類別 1 正規化平均數比較圖

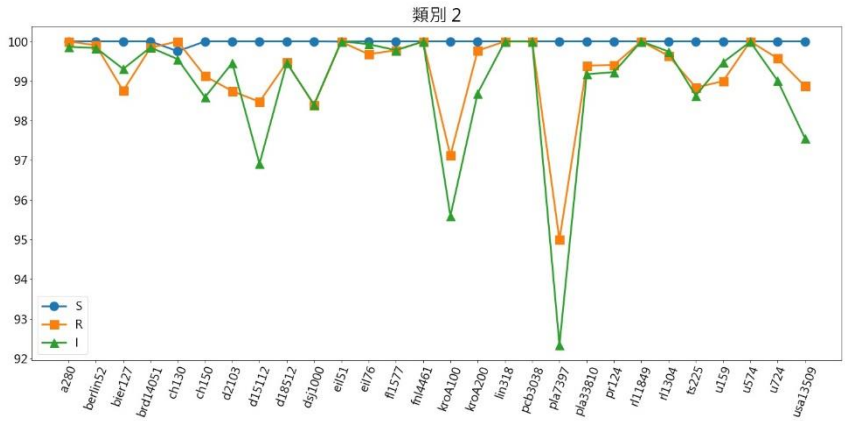

圖 18：測資類別 2 正規化平均數比較圖

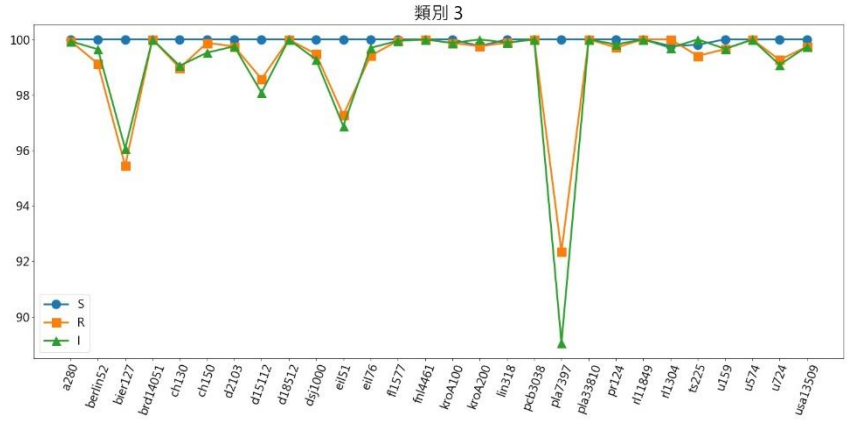

圖 19：測資類別 3 正規化平均數比較圖

表 36：三種「Pack 參數控制機制」正規化平均數比較表

	測資類別 1			測資類別 2			測資類別 3		
	S	R	I	S	R	I	S	R	I
a280	100	99.9	99.7 +	100	100	99.9 +	100	99.9 +	99.9 +
berlin52	99.5	100	99.4	100	99.9	99.8 +	100	99.1	99.6
bier127	100	98.8 +	98.9	100	98.8 +	99.3 +	100	95.4 +	96.1 +
brd14051	100	99.8 +	99.7 +	100	99.8	99.8	100	100	100
ch130	100	99.8	99.6	99.8	100	99.6	100	99 +	99 +
ch150	100	99.4	98.7	100	99.1	98.6 +	100	99.9 +	99.5 +
d2103	100	99.6 +	99.8	100	98.7 +	99.5	100	99.7 +	99.7 +
d15112	100	99 +	98.7 +	100	98.5 +	96.9 +	100	98.6 +	98.1 +
d18512	100	99.9 +	99.9 +	100	99.5 +	99.5 +	100	100	100
dsj1000	100	100	99.7	100	98.4 +	98.4 +	100	99.5	99.3 +
eil51	100	100	99.9	100	100	100	100	97.3 +	96.9 +
eil76	98.9	100	93.4 +	100	99.7	99.9	100	99.4 +	99.7 +
fl1577	100	99.8	99.6	100	99.8	99.8	100	99.9	99.9
fnl4461	100	100	100 +	100	100	100 +	100	100 –	100
kroA100	100	98.8	98.6	100	97.1	95.6 +	100	99.9	99.9
kroA200	99.2	97.3	100	100	99.8	98.7 +	99.8	99.8	100
lin318	100	99.8	99.9	100	100	100	100	99.9	99.9 +
pcb3038	100	99.9	99.9	100	100	100 +	100	100	100
pla7397	100	97.5 +	98.4 +	100	95 +	92.3 +	100	92.3 +	89.1 +
pla33810	100	98.5 +	97.9 +	100	99.4 +	99.2 +	100	100	100
pr124	100	100	100	100	99.4 +	99.2 +	100	99.7	99.8
rl11849	100	99.9	100	100	100	100	100	100	100
rl1304	100	100	99.9 +	100	99.6 +	99.7 +	99.8	100 –	99.7
ts225	100	99 +	99 +	100	98.8 +	98.6 +	99.8	99.4 +	100
u159	100	99.6	99.6	100	99 +	99.5	100	99.7 +	99.7 +
u574	100	99.7	99.1	100	100	100 +	100	100	100 +
u724	100	99.8	98.5 +	100	99.6	99 +	100	99.3 +	99.1 +
usa13509	100	99.8 +	99.8 +	100	98.9 +	97.6 +	100	99.7	99.7

圖 20 為三個版本 S、R、I 演化曲線比較圖，x 軸為計算時間，y 軸為適應值，圖中將各版本執行 10 次的演化過程，在每個時間點取中位數繪出而成。三張圖由左至右分別為求解測試資料 d15112 (類別 1)、pla7397 (類別 2)、u724 (類別 3) 的中位數演化圖。圖中可以發現三組測資中版本 S 皆較為優秀，在 d15112 與 u724 測試問題中，S 保持領先求得好的解，R 次之，I 則是移動緩慢，而在 pla7397 測試問題中，R 在前期較快搜尋到的較佳的解，但在後期當 S 依靠合適的 α 值，能夠找尋到更優秀的解。

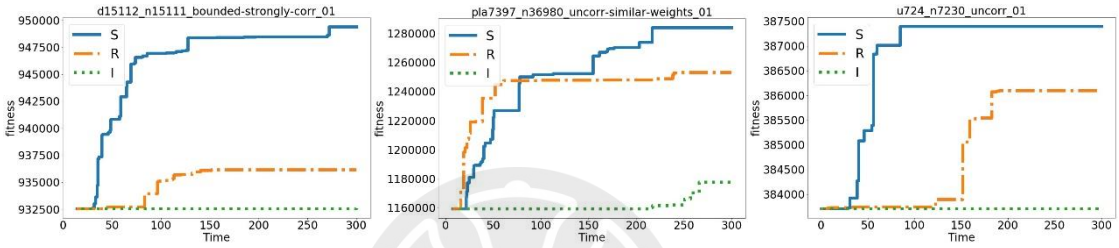

圖 20：自適應控制實驗中位數演化圖

上述實驗得知自適應控制 α 值相較隨機 α 值與疊代搜尋 α 值更為優秀，接下來我們將檢驗自適應控制機制在演化過程中是否合理的控制 α 值。首先我們在固定的 CLKH 路徑下，將不同 α 值代入 Pack 函式，評估求得的背包物品適應值，以此觀察 Pack 函式代入不同 α 值的效能差異，如圖 21–23 中的右圖 (相異 α 值 Pack 函式之適應值曲線圖)。相異 α 值 Pack 函式之適應值曲線圖中，x 軸為 α 值，y 軸為適應值，圖中兩條曲線分別為 CLKH 路徑順時針走法與逆時針走法的相異 α 值 Pack 函式之適應值曲線。由圖中可以看出，以不同 α 值計分來選取背包物品，會影響個體適應值的好壞。接著我們以相異 α 值 Pack 函式之適應值曲線圖來檢驗在不同的測試資料中，自適應控制機制是否合理的控制 α 值。我們以測試資料 d15112 (類別 1)、pla7397 (類別 2)、u724 (類別 3) 的 α 值演化圖 (圖 21–23 中的左圖)，檢驗自適應控制 α 值收斂值是否與相異 α 值 Pack 函式之

適應值曲線圖中較佳的 α 值相吻合。α 值演化圖中，x 軸為演化世代，y 軸為 α 值，圖中每間隔固定代數，繪製當個世代中所有子代個體使用之 α 值；若子代個體沒有比親代更好，我們以綠色 × 繪製 α 值，若是子代個體能勝過親代個體，則以黑色 ▲ 繪製。圖 21中，左方 α 值演化圖顯示測資 d15112 於演化過程中，α 值收斂在 7.6 附近，並且能以相近數值生成優於親代的子代個體。在右方相異 α 值 Pack 函式之適應值曲線圖中，α 值介於 [7, 8] 之間表現較佳。由兩圖對照可知自適應控制能夠有效演化出合理之 α 值。圖 22 的 α 值演化圖中，α 值演化收斂在 6.8 附近，雖然稍微高於相異 α 值 Pack 函式測試圖中最好的 α 值，但 α 值演化圖顯示演化前期仍有找尋到此範圍之 α 值，而且後續演化過程中，發現 6.8 更為適合。圖 23 的 α 值演化圖中，α 值收斂於 2.6 附近，與相異 α 值 Pack 函式測試圖相差無幾。

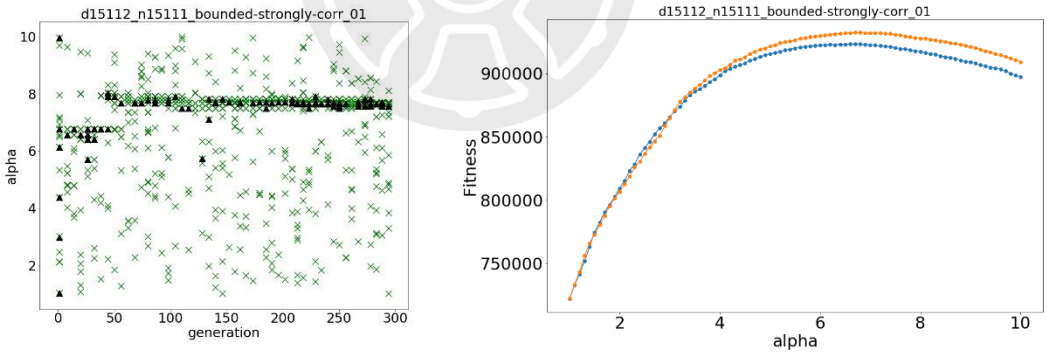

圖 21：測試資料 d15112 之 α 值演化圖與相異 α 值 Pack 函式之適應值曲線圖

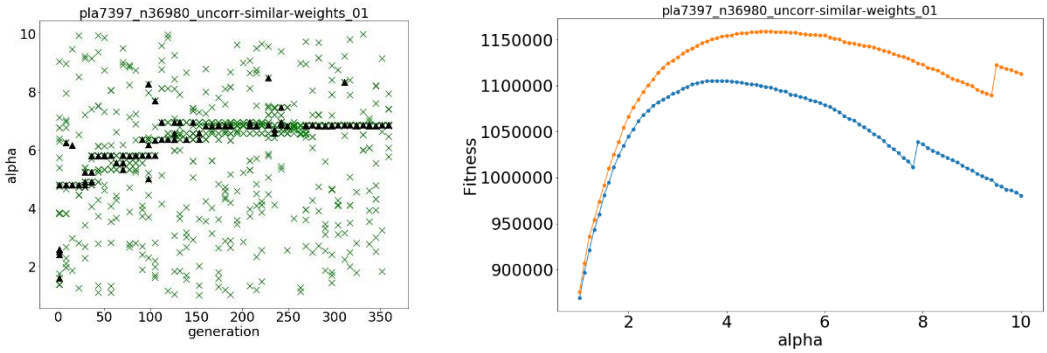

圖 22：測試資料 pla7397 之 α 值演化圖與相異 α 值 Pack 函式之適應值曲線圖

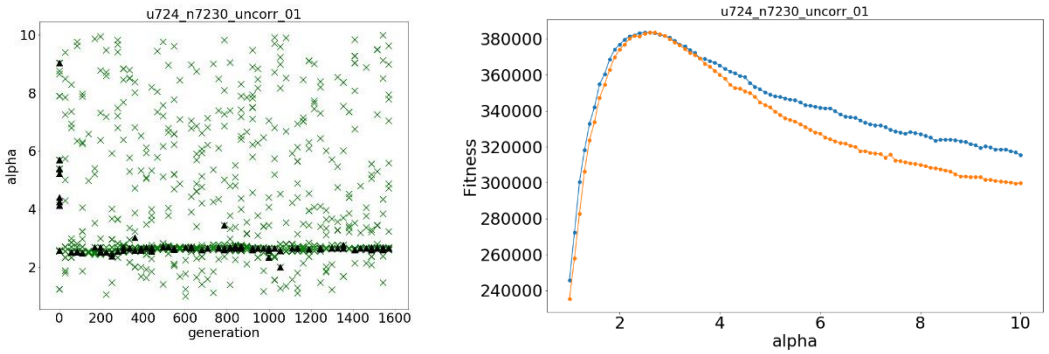

圖 23：測試資料 u724 之 α 值演化圖與相異 α 值 Pack 函式之適應值曲線圖

4.5.2 自適應控制機制參數調整實驗

本實驗探討自適應控制機制參數 *pr* 的設定，依次設定 *pr* = {0, 0.1, 0.3, 0.5, 1} 進行比較。下表 37 為 *pr* = 0.1 (即前節實驗中的版本 S) 比較其他設定值的統計檢定結果。和 *pr* = 0.3 以及 *pr* = 0.5 的比較中，相差結果不大，僅少量的測試資料勝過兩者。雖然 *pr* 設為 0.1 至 0.5 差異不大，但明顯贏過 *pr* = 0 與 1；贏過 *pr* = 0 (無隨機) 代表演化過程中需要隨機生成新的 α 值幫助族群演化，勝過 *pr* = 1 (純隨機) 代表本研究所提之自適應控制機制是有效果的。

表 37：自適應控制機制參數調整實驗統計檢定結果

pr = 0.1 (版本 S)	vs. *pr* = 0	vs. *pr* = 0.3	vs. *pr* = 0.5	vs. *pr* = 1 (版本 R)
測資類別 1	7 / 0	0 / 2	2 / 1	9 / 0
測資類別 2	7 / 1	2 / 0	2 / 0	12 / 0
測資類別 3	3 / 0	2 / 0	2 / 1	12 / 2

4.5.3 子代數量調整實驗

3.1 小節提到遺傳演算法每一世代會進行 N^{init} 次交配生成子代找尋菁英個體，再執行環境篩選與突變。下表 38 檢視不同族群擴張大小 [1, $N^{init}/2$, N^{init}, $2\times N^{init}$] 的差異，數據結果顯示 $N^{init}/2$、N^{init}、$2\times N^{init}$ 並無太大差異，而 N^{init} 與 1 的比較中，顯示每個世代仍需要生成一定數量的子代來探索解空間，尤其是測資類別 2。

表 38：族群擴張大小調整實驗統計檢定結果

N^{init}	vs. $N^{init}/2$	vs. $2\times N^{init}$	vs. 1
測資類別 1	1 / 0	0 / 0	3 / 0
測資類別 2	0 / 1	1 / 0	7 / 0
測資類別 3	1 / 0	2 / 1	2 / 0

4.5.4 突變策略與族群多樣性維護實驗

在初始化族群時，CLKH 路徑加上 Pack-Iterative 背包策略可達到相當好的效果，導致族群在演化時容易快速的逼近此菁英個體而快速收斂，造成向外探索能力下降。因此，適時的維護族群多樣性相當的重要。在本論文中，採用突變策略使相同適應值之個體強制突變，僅保留一個，以此維護族群多樣性的能力。此實驗以 4.5.1 小節版本 S 作為演算法之基準，比較有無突變策略對於演化的重要性。表 39 顯示突變策略對於演化過程影響性很大，三個類別合計 53 個測試資料 (53/84 ≒ 63%) 在有利用突變策略進行族群多樣性維護時之結果較佳。

表 39：族群多樣性維護實驗統計檢定結果

版本 S	vs. 無突變策略版本 S
測資類別 1	22 / 0
測資類別 2	19 / 0
測資類別 3	12 / 0

4.6 kpGA 參數調整實驗

此實驗研究機率性突變策略中參數 *pm* 的設置，比較 *pm* = {0, 0.005, 0.01, 0.05} 對於 kpGA 演化結果的差異，本實驗以最終參數設定 *pm* = 0.005 比較其他三個數值。*pm* = 0 代表無突變策略，表 40 顯示無突變策略對結果的影響性極大，*pm* = 0.005 大勝 *pm* = 0 合計 70 筆測試資料 (70/84 ≒ 83%)，表示確實需要突變來維護族群的多樣性用以幫助演化。*pm* = 0.01 與 0.05 輸給 *pm* = 0.005，代表此兩數值對於演化過程中，突變機率相較 0.005 而言過大。

表 40：機率性突變策略實驗統計檢定結果

p_m = 0.005	vs. p_m = 0	vs. p_m = 0.01	vs. p_m = 0.05
測資類別 1	24 / 0	5 / 1	12 / 2
測資類別 2	20 / 0	4 / 0	8 / 0
測資類別 3	26 / 0	10 / 0	13 / 1

4.7 dpGA 相關實驗

4.7.1 遷移機制相關實驗

本實驗分為兩個部分，第一部分探討遷移時機參數 *MT* 的影響性，第二部分觀察有無遷移機制的影響性。首先比較 *MT* = {1, 50, 100}，表 41 為遷移機制實驗統計檢定結果，*MT* = 1 (即每代遷移) 代表頻繁地進行族群的資訊交換，統計檢定結果 *MT* = 1 相比於 *MT* = 50 合計 14 筆測試資料會因此受到影響而變差，表示遷移太頻繁對族群演化造成不好的影響。*MT* = 50 與 *MT* = 100 比較中，兩者差異不大，最終選擇少量贏得 4 筆測試資料的 *MT* = 50 作為參數設定。我們觀察 *MT* = 50 與無遷移的比較中，*MT* = 50 合計贏 16 筆測試資料，可以看出遷移機制對於演化過程是有助益的。

表 41：遷移機制實驗統計檢定結果

MT = 50	vs. *MT* = 1	vs. *MT* = 100	vs. 無遷移
測資類別 1	6 / 0	1 / 0	5 / 0
測資類別 2	4 / 1	0 / 0	4 / 0
測資類別 3	5 / 0	3 / 0	7 / 0

接著我們觀察雙族群遺傳演算法中，兩個族群的演化曲線圖。圖 24–26 為三組測資 usa13509 (類別 1)、fnl4461 (類別 2)、fl1577 (類別 3) 的雙族群演化曲線圖，圖中黑色垂直線代表兩族群遷移的時機點，藍色虛線為 tspGA 族群每代最佳解，橘色實線為 kpGA 族群每代最佳解，圖中的圓點代表族群最佳解更新時間點。在 usa13509 的測資中，於接近 150 世代時，tspGA 更新族群最佳解 (虛線向上)。經過遷移機制後，長時間無法更新最佳解的 kpGA 依靠遷移個體的幫助，於接近 200 世代時找到新的最佳解 (實線向上)。接著 200 至 250 世代中，tspGA 基於 kpGA 最佳解更新了兩次最佳解，然後 kpGA 又基於 tspGA 的最佳解持續找到更好的解。

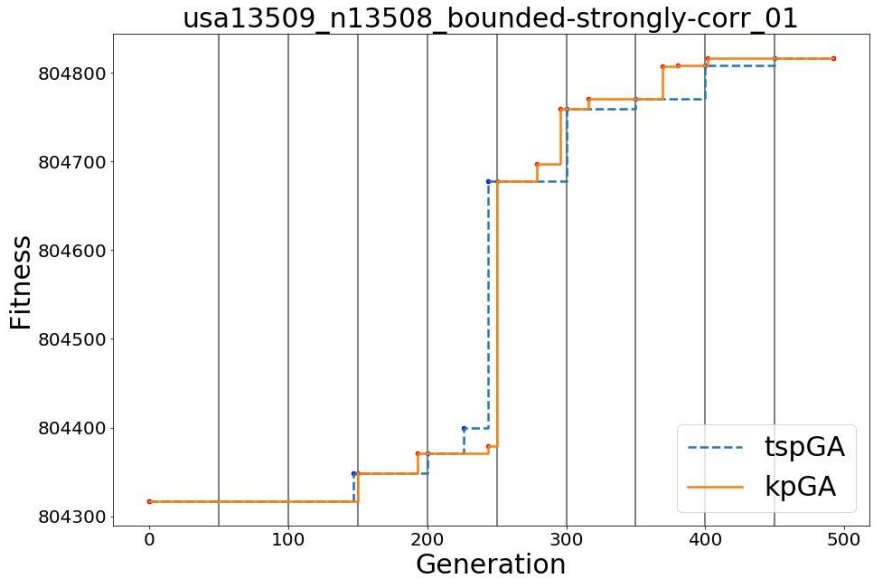

圖 24：測試資料 usa13509 之雙族群演化曲線圖

圖 25 為求解測試資料 fnl4461 的雙族群演化曲線圖，kpGA 從 450 世代至 800 世代都無法演化出新的解，直到靠近 800 世代 tspGA 找到新的最佳解，kpGA 藉由第 800 世代 tspGA 遷移之個體，幫助族群演化出更好的解。

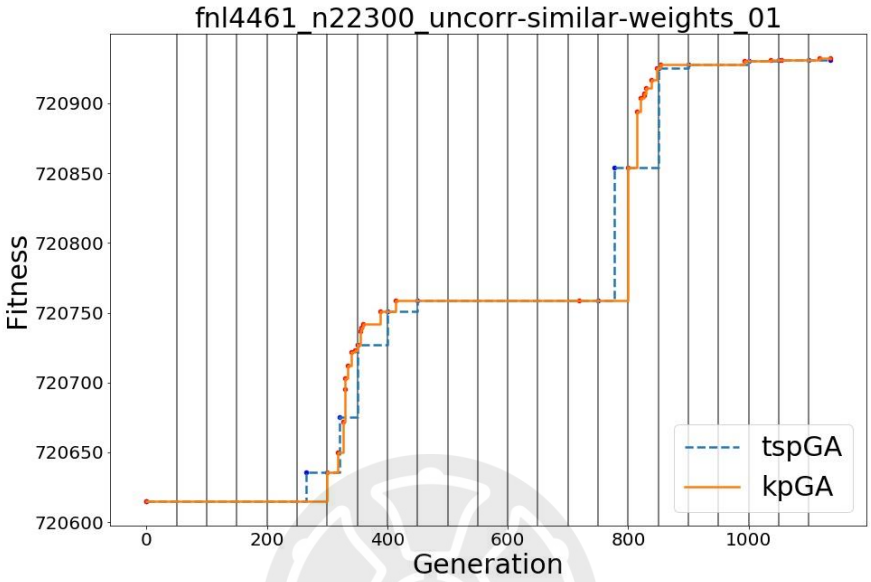

圖 25：測試資料 fnl4461 之雙族群演化曲線圖

圖 26 為求解測試資料 fl1577 的雙族群演化曲線圖，kpGA 於前期緩慢提升，在 400 世代附近，tspGA 大幅更新解。經過第 400 世代的遷移之後，可以看出 kpGA 藉由遷移個體直接拉升族群最佳解。tspGA 藉由於 450 世代 kpGA 遷移的菁英個體，持續找到新的最佳解，kpGA 又藉由於 500 與 550 世代 tspGA 遷移之個體，持續更新族群最佳解。總結來說，由圖 24–26 中可以看出兩個族群能夠互相幫助，由遷移機制獲得對方的菁英個體，協助自身的演化。

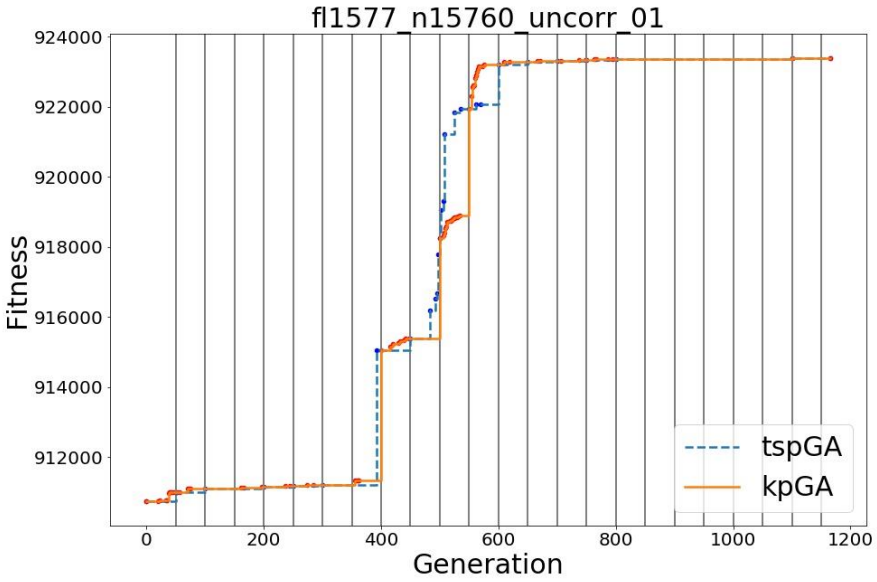

圖 26：測試資料 fl1577 之雙族群演化曲線圖

4.7.2 雙族群與單一族群的比較

本實驗將雙族群遺傳演算法 dpGA 與專注搜尋單一子問題的遺傳演算法 tspGA (版本 S) 以及 kpGA 進行比較，以此來觀察雙族群是否相較單族群更加優秀，演算法終止條件皆設為 10 分鐘。表 42 統計檢定結果中，顯示 dpGA 相較 tspGA 與 kpGA 更加優秀，問題類別 1 相較類別 3 的測資中旅行推銷員子問題的成份較重，因此 dpGA 與 tspGA 的比較中，類別 1 勝出的較少，而問題類別 3 的測資中背包子問題的比重上升，dpGA 贏的相對更多。dpGA 明顯勝過 kpGA，與 kpGA 比較結果與 tspGA 剛好相反，kpGA 在問題類別 3 輸得比較少。

表 42：雙族群優勢實驗統計檢定結果

dpGA	vs. tspGA	vs. kpGA
測資類別 1	5 / 0	24 / 3
測資類別 2	6 / 1	21 / 0
測資類別 3	10 / 0	12 / 6

4.7.3 族群大小影響實驗

此實驗探討 dpGA 子族群大小對於搜尋能力的影響，我們比較不同子族群大小 N^{init} = [2, 30, 60]。表 43 實驗統計結果中顯示 N^{init} = 30 比 N^{init} = 2 贏 17 筆測試資料，代表族群式的搜尋方式在此問題上是有效的，族群太小會降低搜尋能力。N^{init} = 30 與 60 的比較中，顯示 60 族群數相對太大，效果較差。

表 43：族群大小影響實驗統計檢定結果

N^{init} = 30	vs. N^{init} = 2	vs. N^{init} = 60
測資類別 1	6 / 1	4 / 0
測資類別 2	7 / 0	2 / 0
測資類別 3	4 / 1	1 / 0

4.7.4 族群初始化 CLKH 路徑個數實驗

在 4.4 小節的實驗中，我們說明 CLKH 因有隨機性所以生成之路徑不盡相同，也得出使用不同 CLKH 路徑初始化族群對於演化結果是有影響的。本小節以 MATLS [20] 瀰集演算法中初始 CLKH 路徑個體數目 10 進行實驗，我們設定 CT = 5 生成 10 個 CLKH 路徑個體。表 44 比較 CT = 1 與 CT = 5，由結果中可看出，CT = 5 結果較為優秀，加入不同的 CLKH 路徑能夠在初始化時獲得較好的路徑。

表 44：族群初始化 CLKH 路徑個數差異統計檢定結果

CT = 5	vs. CT = 1
測資類別 1	9 / 0
測資類別 2	12 / 0
測資類別 3	17 / 0

4.8 比較文獻演算法

前述實驗我們已檢驗並證實本論文所提出之各種機制 (自適應參數控制、多樣化維持、雙族群與遷移) 有助於提升求解品解。最後一個實驗中，我們將以本論文提出的完整版 dpGA 和 2020 年文獻 [10] 前三名之演算法比較求解品質。此三種演算法為 MATLS [20]、CS2SA [14] 和 EHLS [10]。由於 MATLS 和 CS2SA 均有公開程式碼可下載 [33][34]，我們同樣執行這兩個演算法求解每筆測資 10 次，並基於統計檢定呈現比較結果。EHLS 因無法取得程式碼，所以我們只能比較該方法文獻中求解各測資之平均值和我們的 dpGA 求解各測資 10 次的平均值。表 45 為 dpGA 與文獻演算法比較結果表，表中明顯看出我們提出的 dpGA 演算法較為優秀。雙族群遺傳演算法藉由兩個族群分別搜尋兩個單一子問題，以此縮小搜尋空間。在搜尋單一子問題時，好的解碼函式很重要，能夠幫助個體在另一子問題上獲得好的基因。在 tspGA 中，我們以 Pack 解碼函式挑選物品，其評分公式中物品價值與重量會有一個 α 指數的權重，會影響到背包物品的好壞。MATLS 以經驗法則作為隨路選物的解碼函式，其解碼函式中並未考慮物品價值與重量的權重對於評分的影響性 (其作法類似於本研究之解碼函式，但 α 值固定為 1)，因此 MATLS 在個體解碼方面， 較弱於我們提出之演算法。接著 MATLS 以區域搜尋法調整背包物品，而 dpGA 以族群式的方式搜尋背包子問題，探索空間較完整。整體來說，dpGA 在 80% 以上的測資都勝過 MATLS。dpGA 表現也比 CS2SA 更好，CS2SA 為單一個體的交替搜尋方式，而 dpGA 為族群式搜尋法，搜尋解空間比起單一個體式較為完整。EHLS 為單一子問題搜尋法，其只使用區域搜尋法求解背包子問題，比起 EHLS，dpGA 能夠搜尋更大的解空間，並且在大部分測資搜尋到更好的解。

表 46–48 為 MATLS、CS2SA、EHLS 以及我們提出的演算法 dpGA 執行 10 次在問題類別 1 到 3 各項測試資料的平均值，其中黑色粗體為四個演算法中最佳的平均值。在類別 1 與類別 2 的測試資料中，僅一筆與兩筆測試資料輸給其他三者。在類別 3 中，dpGA 在 28 筆測試資料中有 15 筆最佳。問題類別 3 相較類別 1 與類別 2 的物品數量較多，所以在遺傳演算法的機制策略與目標函式的運算上需要耗費更多的時間，因此在相同計算時間下，dpGA 相較發揮的沒有在類別 1 與類別 2 出色，但也已有一半測資表現較佳。表 48 中，fnl4461 測試資料在原論文中並不符合類別 3 規定的測試資料類型，因此此處不放置數據。CS2SA-R 程式碼在某些測試資料會有錯誤的情形，因此此處不予比較。

表 45：dpGA 與文獻演算法比較結果表

dpGA	vs. MATLS	vs. CS2SA	vs. EHLS
測資類別 1	27 / 0	27 / 1	28 / 0
測資類別 2	21 / 1	27 / 1	26 / 2
測資類別 3	21 / 3	23 / 3	21 / 6

表 46：文獻演算法與 dpGA 在類別 1 測試資料的平均值

	MATLS	CS2SA	EHLS	dpGA
a280	31616.8	29519.0	32157	**32576.2**
berlin52	6743.5	5767.6	5789	**7140.5**
bier127	5564.1	4283.7	5216	**5936.0**
brd14051	1503667.0	1457213.6	1502300	**1556836.0**
ch130	8888.7	8953.2	9262	**9576.9**
ch150	21996.9	20491.0	21233	**22928.0**
d2103	112897.5	118832.7	116677	**122022.2**

d15112	880207.8	870473.8	911388	**951484.9**
d18512	993457.8	964668.5	1053165	**1087620.0**
dsj1000	143280.0	**144219.0**	137724	143836.0
eil51	3935.6	3445.0	3841	**4414.6**
eil76	7217.1	6162.0	5648	**8138.8**
fl1577	88223.3	87947.9	88519	**93896.9**
fnl4461	248784.5	239817.5	257684	**265403.4**
kroA100	4626.7	4436.0	4321	**4853.7**
kroA200	9730.1	9353.2	9648	**10612.3**
lin318	16818.3	14047.6	14025	**18324.8**
pcb3038	148630.5	145633.8	153615	**162495.2**
pla7397	367251.5	314617.0	321361	**392218.5**
pla33810	1750000.0	1779510.2	1702492	**1858166.0**
pr124	6261.8	6703.4	7433	**7494.2**
rl11849	663643.1	658970.3	683402	**707321.6**
rl1304	75756.5	75827.2	77078	**79835.2**
ts225	44667.0	43631.0	43243	**44788.1**
u159	14964.6	14797.5	15143	**15554.7**
u574	49517.9	47715.1	48071	**51689.8**
u724	70945.7	65682.4	69709	**72053.7**
usa13509	745886.2	683900.8	742225	**806791.1**

表 47：文獻演算法與 dpGA 在類別 2 測試資料的平均值

	MATLS	CS2SA	EHLS	dpGA
a280	38090.7	36195.3	37283	**39500.6**
berlin52	10847.2	10171.9	**13550**	10941.1
bier127	25684.0	21523.5	22232	**25802.5**
brd14051	2425434.0	2231863.8	2421320	**2536977.0**
ch130	22898.2	20143.8	21931	**23006.2**
ch150	19745.6	19282.2	19682	**20798.7**
d2103	305431.8	309157.5	298527	**317318.2**
d15112	2522001.0	2359119.7	2513037	**2612296**
d18512	2892955.0	2636686.4	2980263	**3021685**
dsj1000	147533.8	138085.1	140228	**148215.6**
eil51	5617.2	5349.9	5410	**6531.0**
eil76	11451.3	11638.7	9630	**12883.4**
fl1577	247369.7	241010.5	242721	**258902.6**
fnl4461	697312.5	643399.1	706454	**718377**
kroA100	13421.3	13160.0	12052	**13846.7**
kroA200	35150.5	32667.3	33757	**36413.7**
lin318	**89964.7**	81515.9	77541	50615.4
pcb3038	505624.3	488298.4	513089	**526122.7**
pla7397	1380332.0	1216073.1	1039157	**1409067.0**
pla33810	5771101.0	5351922.5	5936408	**5977317.0**
pr124	20257.6	20462.6	20641	**21501.0**
rl11849	1893920.0	1772483.7	1929624	**1934132**
rl1304	215848.6	203507.5	218950	**219834.9**
ts225	36189.2	31611.1	35235	**36206.4**
u159	25229.2	24446.4	24209	**25840.2**
u574	103537.1	97802.6	101480	**103786.6**
u724	114395.9	115295.3	112719	**116504.1**
usa13509	2489707.0	2068174.3	2380024	**2580898**

表 48：文獻演算法與 dpGA 在類別 3 測試資料的平均值

	MATLS	CS2SA	EHLS	dpGA
a280	137684.9	139132.0	100800	**142795.2**
berlin52	39782.2	37988.0	30204	**40036.8**
bier127	**68146.0**	61512.6	53476	68038.8
brd14051	8456536.0	8642335.8	6799618	**8755605**
ch130	**130808.1**	119901.0	129065	126120.0
ch150	73822.9	**75323.0**	72257	74780.6
d2103	1118354.0	1119867.5	1136320	**1149978**
d15112	12507290.0	**13362584.5**	12199219	12894830
d18512	9996043.0	10273481.1	10048678	**10371540**
dsj1000	497939.5	487117.8	**503440**	495642.8
eil51	22051.8	20052.0	19484	**25065.3**
eil76	35183.2	32148.0	35947	**39789.7**
fl1577	870091.6	899767.7	616099	**913798.1**
fnl4461	3222239.0	3198988.0		**3327468**
kroA100	40681.1	40997.0	29888	**41284.9**
kroA200	**141905.4**	138602.0	115144	139647.0
lin318	161071.6	161073.0	164427	**165331.0**
pcb3038	1599425.0	1632043.3	1119380	**1655324.0**
pla7397	5021711.0	4448583.3	**5074983**	5049701.0
pla33810	19766030.0	19243035.8	**20406279**	20211500.0
pr124	**66891.8**	59940.0	64747	65231.5
rl11849	6480857.0	**6649771.3**	6607124	6575109
rl1304	747547.7	**759445.1**	757380	758027.1
ts225	172376.4	170201.0	125297	**174065.8**
u159	153176.5	**153850.0**	134013	153805.4
u574	334512.1	326216.0	337936	**341628.2**
u724	378521.4	381876.0	386856	**388706.0**
usa13509	9197273.0	8123190.5	**9306518**	9291269.0

第五章　結論與未來研究方向

旅行小偷問題是一個同時求解城市排列以及物品組合的離散最佳化問題。此問題為 NP-hard，文獻中經常以啟發式演算法求解。然而，城市排列和物品組合兩個子問題交疊而成巨大的搜尋空間，即便運用啟發式演算法求解也具有相當的挑戰性。本論文提出以雙族群遺傳演算法求解旅行小偷問題，兩個族群分別專注搜尋求解一個子問題，但演化過程中同時以高效率的解碼函式解決另一子問題。tspGA 以 Pack 解碼函式挑選物品，其評分公式 α 值會對 Pack 函式挑選物品造成影響，因此我們使用自適應控制機制控制 α 值，使得個體解碼時能夠挑選到合適的物品。實驗證明自適應控制機制能合理的控制 α 值，並且提升演算法效能。另外實驗證明突變策略與族群多樣性維護的重要性，使得族群保有演化搜尋的能力。接著我們探討雙族群遺傳演算法的作用與效益，兩個族群藉由遷移機制傳遞最佳解個體，使得族群基於遷移個體，更新族群的最佳解。相比單一族群的遺傳演算法，實驗證明雙族群式遺傳演算法效果更好。最後我們所提出之 dpGA 與近年文獻演算法比較，實驗結果說明 dpGA 效能更好，代表 dpGA 在近年求解旅行小偷問題的演算法中頗具競爭力。

後續研究方向可以再加強兩個族群的遺傳演算法搜尋能力，以此提升解的品質，例如加入區域搜尋法幫助族群更快的搜尋區域最佳解，但是使用區域搜尋法時，需要思考另一子問題的解碼方式：若是以經驗法則解碼勢必會消耗許多計算時間，但能提升個體的適應值；若是不考慮處理另一子問題 (即固定另一子問題基因不變動)，搜尋空間會受限於固定的另一子問題基因。一個旅行小偷問題中，兩個子問題所佔的成份比重不一定相同，若是能適當的給予兩個子問題搜尋演算法不同的計算資源，或許能提升解品質，例如我們可以用適應性控

制機制分配兩族群的演化世代數，給予兩族群不同的計算資源。最後，期望本研究提出之演算法能夠協助更多研究兩子問題結合模型的學者，也希冀未來能夠被應用於實際問題中。

參考文獻

[1] M. R. Bonyadi, Z. Michalewicz, and L. Barone, “The travelling thief problem: The first step in the transition from theoretical problems to realistic problems,” IEEE Congress on Evolutionary Computation, Cancun, Mexico, pp. 1037–1044 , 2013.

[2] IEEE Congress on Evolutionary Computation 2017 3rd Competition on Optimisation of Problems with Multiple Interdependent Components, url: https://cs.adelaide.edu.au/~optlog/TTP2017Comp/

[3] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann, “A comprehensive benchmark set and heuristics for the traveling thief problem,” Genetic and Evolutionary Computation Conference, pp. 477–484, July 2014.

[4] S. Lin, “Computer solutions of the traveling salesman problem,” The Bell System Technical Journal, vol. 44, no. 10, pp. 2245–2269, 1965.

[5] S. Lin and B. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” Operations Research, vol. 21, no 2, pp. 498–516, 1973.

[6] O. Martin, S. W. Otto, and E. W. Felten, “Large-step markov chains for the traveling salesman problem,” Complex Systems, vol. 5, no. 3, pp. 299–326, 1991.

[7] D. Applegate, W. Cook, and A. Rohe, “Chained lin-kernighan for large traveling salesman problems,” INFORMS Journal on Computing, vol. 15, no. 1, pp. 82–92, 2003.

[8] B. C. Gupta and V. P. Prakash, “Greedy heuristics for the travelling thief problem,” National Systems Conference, Greater Noida, India, pp. 1–5, 2015.

[9] H. Faulkner, S. Polyakovskiy, T. Schultz, and M. Wagner, “Approximate approaches to the traveling thief problem,” Genetic and Evolutionary Computation Conference, pp. 385–392, July 2015.

[10] A. Maity and S. Das, “Efficient hybrid local search heuristics for solving the travelling thief problem,” Applied Soft Computing, vol. 93, 2020. url: https://doi.org/10.1016/j.asoc.2020.106284

[11] M. R. Bonyadi, Z. Michalewicz, M. R. Przybylek, and A. Wierzbicki, “Socially inspired algorithms for the travelling thief problem,” Genetic and Evolutionary Computation Conference, pp. 421–428, July 2014.

[12] M. El Yafrani and B. Ahiod, “Cosolver2B: An efficient local search heuristic for the travelling thief problem,” IEEE/ACS 12th International Conference of Computer Systems and Application, Marrakech, Morocco, pp. 1–5, 2015.

[13] B. Delaunay, “Sur la sphère vide,” Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk, pp. 793–800, 1934.

[14] M. El Yafrani and B. Ahiod, “Population-based vs. single-solution heuristics for the travelling thief problem,” Genetic and Evolutionary Computation Conference, pp. 317–324, July 2016.

[15] M. El Yafrani and B. Ahiod, “Efficiently solving the traveling thief problem using hill climbing and simulated annealing,” Information Sciences, vol. 432, pp. 231–244, 2018.

[16] R. Nieto-Fuentes, C. Segura, and S. I. Valdez, “A guided local search approach for the travelling thief problem,” IEEE Congress on Evolutionary Computation, Rio de Janeiro, Brazil, pp. 1–8, 2018.

[17] M. El Yafrani and B. Ahiod, “A local search based approach for solving the travelling thief problem: the pros and cons,” Applied Soft Computing, vol. 52, pp. 795–804, March 2017.

[18] Y. Mei, X. Li, and X. Yao, “On investigation of interdependence between sub-problems of the travelling thief problem,” Soft Computing, vol. 20, pp. 157–172, 2016.

[19] S. T Alharbi, “A hybrid genetic algorithm with tabu search for optimization of the traveling thief problem,” International Journal of Advanced Computer Science and Applications, vol. 9, no. 11, pp. 276–287, 2018.

[20] Y. Mei, X. Li, and X. Yao, “Improving efficiency of heuristics for the large scale traveling thief problem,” Simulated Evolution and Learning, pp. 631–643, 2014.

[21] M. Wagner, “Stealing items more efficiently with ants: a swarm intelligence approach to the travelling thief problem,” Swarm Intelligence, ANTS, pp. 273–281, 2016.

[22] T. Stützle and H. H. Hoos, “Max-min ant system,” Future Generation Computer Systems, vol. 16, no. 8, pp. 889–914, June 2020.

[23] D. K. S. Vieira, G. L. Soares, J. A. Vasconcelos, and M. H. S. Mendes, “A genetic algorithm for multi-component optimization problems: the case of the travelling thief problem,” Evolutionary Computation in Combinatorial Optimization, pp. 18–29, 2017.

[24] M. El Yafrani, Yafrani, M. Martins, M. Wagner, B. Ahiod, M. Delgado, and R. Lüders, “A hyperheuristic approach based on low-level heuristics for the travelling thief problem,” Genetic Programming and Evolvable Machines, vol. 19, pp. 121–150, 2018.

[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, April 2002.

[26] J. Blank, K. Deb, and S. Mostaghim, “Solving the bi-objective traveling thief problem with multi-objective evolutionary algorithms,” Evolutionary Multi-Criterion Optimization, pp. 46–60, 2017.

[27] M. Laszczyk and P. B. Myszkowski, “A specialized evolutionary approach to the bi-objective travelling thief problem,” Federated Conference on Computer Science and Information Systems, Leipzig, Germany, pp. 47–56, 2019.

[28] J. B. C. Chagas, J. Blank, M. Wagner, M. J. F. Souza, and K. Deb, “A non-dominated sorting based customized random-key genetic algorithm for the bi-objective traveling thief problem,” Journal of Heuristics, pp. 267–301, 2021.

[29] Y. J. Gong, W. N. Chen, Z. H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J. J. Li, “Distributed evolutionary algorithms and their models: A survey of the state-of-the-art,” Applied Soft Computing, vol. 34, pp. 286–300, 2015.

[30] L. Davis, “Applying adaptive algorithms to epistatic domains,” International Joint Conference on Artificial intelligence, vol. 1, pp. 162–164, August 1985.

[31] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, “Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems,” IEEE Transactions on Evolutionary Computation, vol. 10, no. 6, pp. 646–657, 2006.

[32] Chained Lin–Kernighan heuristic (CLKH) algorithm source code, url: http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

[33] MATLS source code, url: https://homepages.ecs.vuw.ac.nz/~yimei/Research-LSGO.html

[34] CS2SA source code, url: https://github.com/yafrani/ttplab