

以 TSP 為載體進行演算法比較進而學習形式驗證優化

1. 引言 (Introduction)

1.1 目的概述與問題選定

本報告旨在學習較為日常可見之 NP 問題的演算法原理，並自形式驗證 (Formal Verification, FV) 領域的相關文獻之研讀發想，嘗試實作相關演算法之基礎模型且結合現今文獻，進行相關演算法之於電路驗證領域的討論。

由上，本文選定旅行推銷員問題 (Traveling Salesman Problem, TSP) 為主要研讀與實作主題。其定義為：給定一系列城市以及各城市之間的距離，目標是找出訪問每一座城市恰好一次，並最終返回出發城市的最短可能路徑。該問題已被證明為 NP-hard 問題，這意味著隨著城市數量的增加，找到絕對最優解所需的時間會呈指數級增長，使得暴力窮舉法對於實際規模的問題變得不可行。

雖說看似並非日常，但 TSP 實際卻在現實世界中仍具有廣泛的應用價值。現代數位電路形式驗證已發展出多元化的演算法體系，包括 SAT/SMT 求解器、BDD 符號模型檢查、插值技術 (Interpolation)、抽象細化(CEGAR)、機器學習導向方法，以及演化式演算法等¹²³。這些方法相互補充，形成了完整的驗證工具鏈，在多個層面發揮著重要作用。

1.2 報告目的與範疇 (Purpose and Scope of the Report)

此次報告將涵蓋了兩部份，分別為：

1. 簡要解釋五種常用於解決 TSP 的演算法原理，並對其進行比較分析
2. 再將基因演算法的結果與開源且較為精確的模型⁴進行比較，嘗試分析基礎實作模型與改良或優化版之模型雙方的差異。

報告的分析將基於實際運作所提供的實驗數據，聚焦於這些演算法在四個特定的 TSPLIB 標準測試案例上的表現，分別是 att48、burma14、ch130 及 ulysses16。主要的比較指標包括：演算法計算所需的 CPU 時間、找到的路徑總成本，以及該成本與已知的官方最優解或理想解之間的差異百分比。透過這些比較，期望能揭示不同演算法在處理不同規模和特性問題時的優勢與劣勢。

1.3 納入比較的演算法 (Algorithms Included in Comparison)

本報告將深入探討以下五種 TSP 求解演算法：

- 基因遺傳演算法 (Genetic Algorithm, GA)
一種模擬生物進化過程的元啟發式搜索演算法。⁵⁶
- Held-Karp (Dynamic Programming, DP)
一種能夠在特定條件下找到最優解的精確演算法。
- 模擬退火演算法 (Simulated Annealing, SA)
一種基於物理退火過程的概率性元啟發式演算法。⁷
- 蟻群最佳化演算法 (Ant Colony Optimization, ACO)
一種模擬螞蟻覓食行為的元啟發式演算法。⁸
- 最小生成樹的近似演算法 (MST-based Approximation Algorithm, MST)
一種利用圖的最小生成樹特性來構造近似解的經典演算法。⁹¹⁰

選擇這五種演算法進行比較，是因為它們代表了求解最佳化問題的不同思路、技術範疇和問題範圍。DP 在各類小範圍的問題中是最常見的，也是教科書內最常出現的演算法之一；最小生成樹近似法則是因其在特定條件下具有時間複雜度保證的方法；而遺傳演算法、模擬退火和蟻群最佳化則是近期較為廣泛應用的元啟發式方法。藉由對這一系列演算法的比較與實作，相信這有助於理解在問題規模、期望解的品質以及可用計算資源之間進行權衡的思維訓練。

2. 演算法原理 (Algorithm Principles)

本節旨在提供對五種演算法核心機制的簡要說明。

2.1 遺傳演算法 (Genetic Algorithm, GA)¹¹¹²¹³

2.1.1 核心思想

遺傳演算法 (GA) 是一種啟發式搜索技術，其靈感來源於生物進化論中的自然選擇和遺傳機制。它將潛在解視為「個體 (individuals)」，每個個體擁有一套「染色體 (chromosomes)」來編碼解的特徵 (例如，在 TSP 中，染色體可以是一條城市訪問序列)。通過「適應度函數 (fitness function)」(例如，TSP 中的路徑總長度) 來評估每個個體的優劣。通過遺傳操作，產生具有更高適應度的新一代個體群體，逐步逼近最優解。

2.1.2 步驟¹⁴

典型的遺傳演算法包含以下步驟

1. 初始化族群 (Initialization)

隨機生成一定數量的初始解 (個體)，構成初始族群。

2. 適應度評估 (Fitness Evaluation)

根據適應度計算族群中個體的適應度。

3. 選擇 (Selection)

根據個體的適應度值，選出優良的個體，使其有更大的機會進入下一代。

4. 交叉 (Crossover)

從選出的父代個體中配對，並使其染色體片段進行交換，從而產生子代個體。

5. 突變 (Mutation)

子代個體以一定的較小機率發生隨機變動，

6. 重複步驟 2 至 5，直到滿足預設的終止條件，例如達到最大進化代數或解的品質不再顯著提升。

2.2 Held-Karp 動態規劃 (Dynamic Programming, DP)¹⁵

2.2.1 核心思想

Held-Karp 動態規劃演算法是解決 TSP 的一種精確演算法，對於 TSP，DP 的核心思想是將問題分解為一系列重疊的子問題，並儲存這些子問題的解以避免重複計算。狀態通常定義為 $dp[j]$ ，表示從指定的起點城市出發，訪問過城市子集 S 中的所有城市，並最終停留在城市 j 的最短路徑長度 1。

2.2.2 演算法步驟¹⁶

1. 初始化 (Initialization)

設定起始狀態。將狀態 $dp[mask][pos]$ 定義為 $(cost, prev_city)$ ，其中 $mask$ 是一個位元遮罩表示已訪問的城市集合， pos 是當前所在的城市。

2. 迭代與狀態轉移 (Iteration and State Transition)

依序遍歷所有可能的已訪問城市子集 $mask$ (從只包含一個城市到包含所有城市)。對於每個 $mask$ ，遍歷 $mask$ 中所有已被訪問的城市 po ，並更新 $dp[new_mask][nxt]$ 為 (new_cost, pos) 。

3. 計算最終結果 (Final Result)

當所有城市都被訪問過時 (即 $mask$ 為 FULL_MASK，所有位均為 1)， $dp[j]$ 儲存了遍訪所有城市並停留在城市 j 的最低成本。最終解的成本是枚舉所有可能的終點城市 j ，並取其中的最小值。

4. 路徑重建 (Path Reconstruction)

利用在 dp 表中儲存的前一個城市 ($prev_city$) 的資訊，從最終選擇的終點開始，反向回溯，直到回到起點，從而重建出完整的最佳路徑。

2.2.3 簡要分析

Held-Karp DP 演算法的顯著優點是它能夠保證找到 TSP 的最優解。然而，其代價是較高的計算複雜度。該演算法的時間複雜度為 $O(n^2 \cdot 2^n)$ ，空間複雜度為 $O(n \cdot 2^n)$ ，其中 n 是城市的數量。這種指數級的複雜度使得 DP 演算法僅適用於城市數量較少的情況。

2.3 模擬退火演算法 (Simulated Annealing, SA)¹⁷

2.3.1 核心思想¹⁸

模擬退火 (SA) 演算法是一種概率性的元啟發式方法，其設計靈感來源於冶金學中固體材料的退火過程。在退火過程中，金屬被加熱至高溫然後緩慢冷卻，使得其內部分子能夠排列到低能量的穩定晶體結構。SA 演算法模擬這一過程，允許在搜索解空間的過程中，不僅接受能夠改進當前解的移動 (即走向成本更低的解)，而且以一定的機率接受可能使解變差的移動。這種接受「壞解」的機制使得演算法有能力跳出局部最優解的束縛，從而有更大的機會找到全局最優解。接受差解的機率與一個稱為「溫度」的控制參數相關，溫度越高，接受差解的機率越大；隨著溫度的逐漸降低，演算法趨向於只接受優解，最終收斂。

2.3.2 演算法步驟

1. 初始化 (Initialization)

- 隨機生成一個初始解。
- 設定一個較高的初始溫度 T_{start} 和一個冷卻率 α

2. 迭代搜索 (Iterative Search)

在達到預設的迭代次數或溫度降至某個值之前，重複以下步驟

- 產生鄰近解 (Generate Neighboring Solution)

對當前解施加一個微小的擾動，產生一個新的鄰近解。在 TSP 中，常用的鄰域操作是 2-opt 交換，即隨機選取路徑中的兩個邊，斷開它們，然後以不同的方式重新連接以形成一條新路徑。

- 計算成本差 (Calculate Cost Difference, ΔE)

計算新解的成本與當前解的成本之差

$$\Delta E = \text{cost}(\text{new_solution}) - \text{cost}(\text{current_solution})$$

- 接受準則 (Metropolis Criterion):

- 如果 $\Delta E < 0$ (即新解更優) , 則接受新解作為當前解。
- 如果 $\Delta E \geq 0$ (即新解更差或相同) , 則以機率 $P = e^{-\Delta E/T}$ 接受新解。其中 T 是當前溫度。

3. 冷卻 (Cooling)

按照預定的冷卻策略降低溫度，例如 $T=T\cdot\alpha$ 。

4. 結束 (Termination)

當滿足終止條件時，演算法結束，回傳在整個搜索過程中找到的最佳解。

2.3.3 簡要分析

模擬退火演算法的有效性在很大程度上取決於其參數的設定，特別是初始溫度、冷卻速率和迭代次數。初始溫度需要足夠高，以確保在搜索初期有足夠的探索能力，能夠廣泛地搜索解空間。冷卻速率 α 控制著降溫的速度， α 越接近 1，降溫越慢，搜索越精細，但需要更多的計算時間。

Metropolis 接受準則 $e^{-\Delta E/T}$ 是 SA 的核心機制。在早期高溫階段，即使新解比較差，接受的機率也相對較高，鼓勵演算法探索。隨著溫度的降低，接受差解的機率減小，演算法逐漸將搜索集中在已發現的較優解附近，進行更精細的利用。如果過早收斂，表現會類似於 greedy；如果冷卻過慢，則可能浪費計算資源。

2.4 蟻群最佳化演算法 (Ant Colony Optimization, ACO)¹⁹

2.4.1 核心思想

蟻群最佳化 (ACO) 演算法是一種模擬自然界中螞蟻覓食行為的元啟發式演算法。真實的螞蟻在尋找食物的過程中，會在走過的路徑上釋放一種稱為「費洛蒙 (pheromone)」的化學物質。其他螞蟻在選擇路徑時，會傾向於選擇費洛蒙濃度較高的路徑。這種正回饋機制使得整個蟻群能夠逐漸發現從蟻巢到食物源的最短路徑。

ACO 將這一生物學現象應用於解決組合最佳化問題，如 TSP。在 ACO 中，「虛擬螞蟻」負責建構問題的解 (即 TSP 路徑)。對於 TSP 來說距離越短，吸引力越大。

2.4.2 演算法步驟

1. 初始化 (Initialization)

- 在所有可能的路徑 (城市間的邊) 上設置少量且均勻的初始費洛蒙濃度。
- 將一定數量的虛擬螞蟻放置在隨機選擇的起始城市，或者都從同一個城市出發。

2. 螞蟻建構路徑 (Ant Solution Construction)

- 每一隻螞蟻獨立地、逐步地建構一條完整的 TSP 路徑。
- 在路徑的每一步，螞蟻 k 從當前城市選擇下一個未訪問的城市的機率通常由邊上的費洛蒙濃度、邊的啟發式資訊值和控制費、洛蒙啟發式資訊，在決策中的相對重要性的參數。

3. 費洛蒙更新 (Pheromone Update):

- 當所有螞蟻都完成路徑的建構後，對所有路徑上的費洛蒙進行更新。
- 費洛蒙蒸發 (Evaporation):

所有邊上的費洛蒙濃度都會按照一定的比例 ρ ($0 < \rho \leq 1$) 減少。蒸發有助於避免演算法過早收斂到次優解，並鼓勵探索新的路徑。

4. 費洛蒙沉積 (Deposition)

每一隻螞蟻會根據其建構路徑的品質 (通常是路徑長度的倒數) 在其走過的邊上增加費洛蒙。路徑越短，增加的費洛蒙越多。

5. 迭代 (Iteration)

重複步驟 2 和 3，直到達到預設的終止條件 (如最大迭代次數或解的品質長時間沒有改善)。整個過程中找到的最短路徑即為最終的近似最優解。

2.4.3 簡要分析:

ACO 的核心在於其通過環境實現所謂的「標記」(stigmergy)。單個螞蟻的行為相對簡單，但群體的集體行為卻能導致複雜問題的有效解決。費洛蒙的蒸發和沉積機制是 ACO 的關鍵組成部分。

參數的設定對 ACO 的性能有顯著影響。兩者分別決定費洛蒙的歷史經驗和啟發式信息的影響程度。適當平衡這兩者，以及調整螞蟻數量、迭代次數和費洛蒙蒸發率等參數，才能獲得更優質的解解。

2.5 基於最小生成樹的近似演算法 (MST-based Approximation Algorithm)²⁰

2.5.1 核心思想

基於最小生成樹 (MST) 的近似演算法是一種經典的 TSP 求解方法，尤其適用於滿足三角不等式 (triangle inequality) 的度量 TSP (Metric TSP)。

該演算法的核心思想是利用圖的最小生成樹。MST 的總權重是連接所有城市的一個子圖的最小可能權重，因此它天然地構成了 TSP 最優路徑長度的一個下界 (儘管 TSP 路徑是一個環路，而 MST 是一棵樹)。對於滿足三角不等式的 TSP，通過對 MST 進行特定操作，可以構造出一條 TSP 路徑，其長度被證明不超過最優 TSP 路徑長度的兩倍 (即 2-近似保證)。

2.5.2 演算法步驟

1. 建立最小生成樹 (Construct MST)

給定所有城市及其間的距離，使用標準的 Prim 演算法或 Kruskal 演算法找到連接所有城市的最小生成樹。

2. 前序走訪 (Preorder Traversal)

從 MST 的任意一個節點 (城市) 開始，對該 MST 進行一次前序深度優先搜索 (DFS)，並記錄節點被訪問的順序。

3. 形成哈密瓜迴路 (Form Hamiltonian Circuit):

- 將前序走訪得到的節點序列視為一條初步的 TSP 路徑。
- 更簡單的理解是，按照前序走訪的順序連接城市，如果某個城市已經在當前構建的路徑中 (除了作為路徑的終點去閉合環路)，則跳過它，直接連接到序列中的下一個新城市。
- 最後，將序列中的最後一個城市與第一個城市相連，形成一個完整的哈密頓迴路。

2.5.3 簡要分析:

MST 近似演算法的主要優勢在於其理論上的性能保證和計算效率 (Prim 或 Kruskal 演算法通常具有多項式時間複雜度，如 $O(N^2)$ 或 $O(E\log N)$ ，其中 N 是城市數， E 是邊數)。這使得它成為一種快速獲得有界解的方法。

然而，這個性能保證嚴格依賴於三角不等式的成立。如果問題不滿足三角不等式，MST 近似演算法的理論保證將完全失效，其產生的解的品質可能會非常差，遠超最優解的兩倍。

即使在滿足三角不等式的情況下，2-近似也意味著解的成本可能高達最優解的兩倍，這在許多實際應用中可能仍然不夠理想。另外，演算法的實際性能還取決於 MST 的結構與最優 TSP 路徑結構的相似程度。

3. 實驗數據分析 (Experimental Data Analysis)

3.1 實驗設置與數據來源 (Experimental Setup and Data Sources)

本報告的實驗數據分析基於四個選自 TSPLIB 的標準測試案例：

- att48：包含 48 個城市，其坐標源於美國電話電報公司 (AT&T) 的一個實際問題。
- burma14：包含 14 個城市，代表緬甸的地理位置。
- ch130：包含 130 個城市，數據來源於瑞士。
- ulysses16：包含 16 個城市，其名稱來源於荷馬史詩《奧德賽》中的地點。
 - gr17 : 特地為 DP 多測試的數據，為了測試基礎模組有限時間(3min)內所能解的上限

所有用於分析的數據，包括 CPU 運行時間、演算法求得的總成本以及官方最優成本，均從提供的研究材料²¹中提取。且所有實驗皆以不超過 3 min 為前提

正百分比表示演算法成本高於最優成本，0% 表示達到最優，負百分比（在本報告數據中未出現，但理論上可能因最優成本記錄問題而產生）表示低於記錄的最優成本。

3.2 數據匯總與缺失說明 (Data Summary and Missing Data Notes)

為了清晰地比較各演算法在選定測試案例上的性能，所有相關數據被匯總於下表。

表格 1: 各演算法在選定測試案例上的性能數據匯總

演算法 (Algorithm)	測試案例 (Test Case)	城市數 (N)	CPU 時間 (秒) (CPU Time (s))	演算法總成本 (Total Cost (Algorithm))	官方最優成本 (Official Optimal Cost)	與最優解差異百分比 (%) Difference from Optimal)

GA	att48	48	2.4300	14178	10628	33.40%
GA	burma14	14	0.7500	3323	3323	0.00%
GA	ch130	130	10.9100	14264	6110	133.45%
GA	ulysses16	16	0.7900	7091	6859	3.38%
DP	att48	48	N/A	N/A	10628	N/A
DP	burma14	14	0.0900	3323	3323	0.00%
DP	gr17 (原 ch130)	17	1.1217	2085	2085	0.00%
DP	ulysses16	16	0.4500	6859	6859	0.00%
SA	att48	48	1.1200	10815	10628	1.76%
SA	burma14	14	0.5700	3323	3323	0.00%
SA	ch130	130	1.0000	6527	6110	6.82%
SA	ulysses16	16	0.6800	6859	6859	0.00%
ACO	att48	48	2.6403	11217	10628	5.54%

ACO	burma14	14	0.6700	3346	3323	0.69%
ACO	ch130	130	8.0900	6818	6110	11.59%
ACO	ulysses16	16	0.7800	6909	6859	0.73%
MST	att48	48	0.0007	13926	10628	31.03%
MST	burma14	14	0.0001	4003	3323	20.46%
MST	ch130	130	0.0074	8279	6110	35.50%
MST	ulysses16	16	0.0001	7788	6859	13.54%

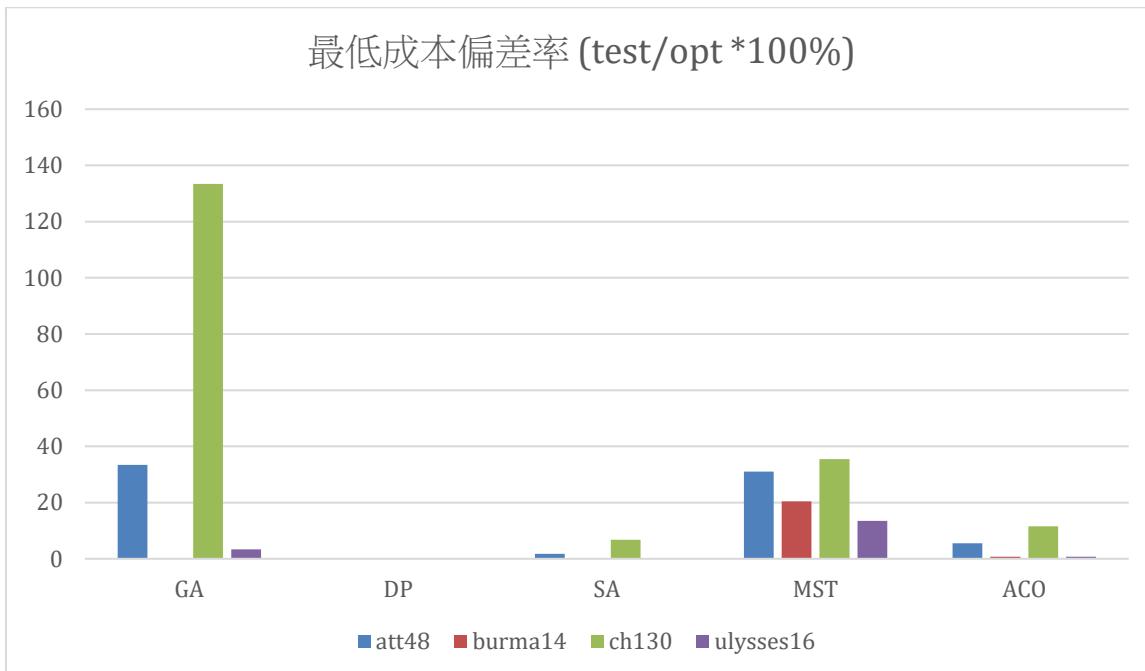
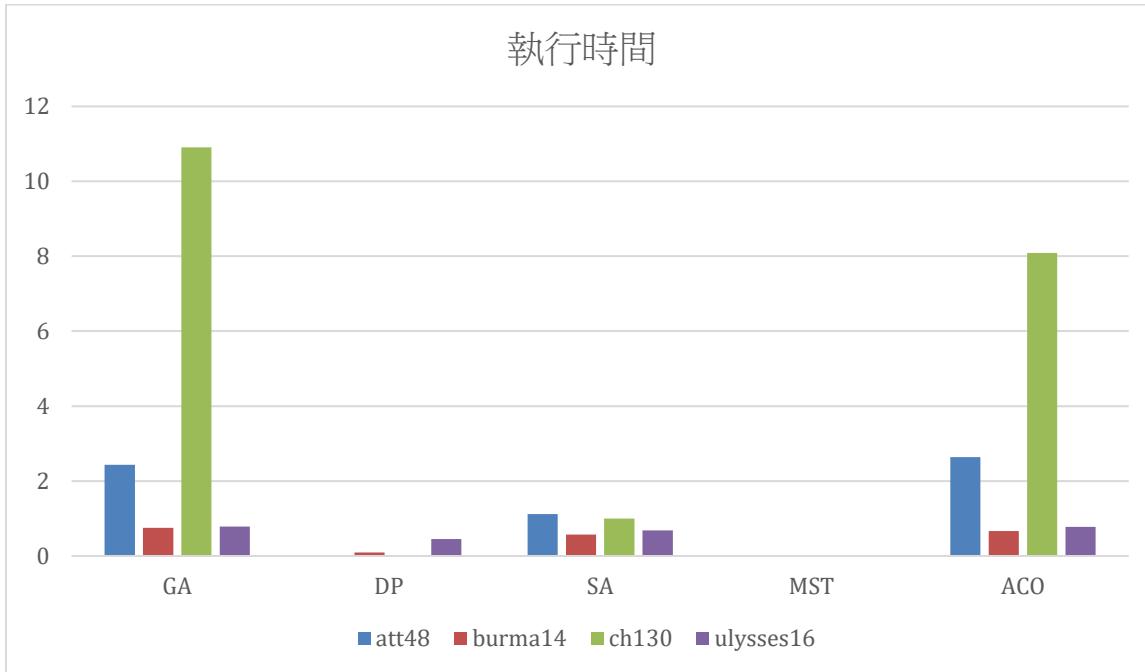
缺失數據說明

由於其 Held-Karp 動態規劃 (DP) $O(N^2 2^n)$ 的時間複雜度，DP 演算法通常不適用於城市數量較大的問題實例。因此，在表格中，att48 ($N=48$) 和 ch130 ($N=130$) 案例的 DP 數據標註為 N/A (不適用)，而為了求有限時間能解的上限，將原 ch130 改為 gr17

此表格集中呈現了所有後續分析的基礎數據，便於進行跨演算法和跨案例的比較。

3.3 各演算法於不同測試案例下的表現比較 (Performance Comparison of Each Algorithm Across Test Instances)

本節將針對每種演算法，分析其在不同規模和特性的測試案例 (att48, burma14, ch130, ulysses16) 上的 CPU 時間、總成本以及與最優解差異百分比



3.3.1 遺傳演算法 (GA) 表現

參數設置

```
n: int,  
population_size: int = 100,  
generations: int = 1000,  
mutation_rate: float = 0.02,  
seed: int = random
```

討論

- CPU 時間

GA 的 CPU 時間隨著城市數量的增加而增加。burma14 ($N=14$) 和 ulysses16 ($N=16$) 的耗時均在 1 秒以內 (分別為 0.7500s 和 0.7900s)。對於 att48 ($N=48$)，耗時增加至 2.4300s。規模最大的 ch130 ($N=130$) 耗時最長，為 10.9100s。

- 總成本與差異百分比:

- 在 burma14 上，GA 找到了最優解 3323，差異百分比為 0.00%。
- 在 ulysses16 上，GA 得到的成本為 7091，相較於最優解 6859，差異百分比為 3.38%。
- 在 att48 上，GA 成本為 14178，與最優解 10628 相差 33.40%。
- 在 ch130 上，GA 的表現最差，成本高達 14264，與最優解 6110 的差異百分比達到了驚人的 133.45%。這一結果應與 2.1 節中討論的 ch130 實例的幾何特性、GA 參數設定以及缺乏局部搜索有關。

3.3.2 Held-Karp 動態規劃 (DP) 表現 (僅分析 burma14, ulysses16)

- CPU 時間

DP 在 burma14 (N=14) 上的 CPU 時間為 0.0900s，在 ulysses16 (N=16) 上為 0.4500s。

隨著城市數量從 14 增加到 16，CPU 時間增加了約 5 倍，初步體現了其指數級複雜度的影響，儘管在這些小規模問題上絕對時間仍然很短。

- 總成本與差異百分比:

- 在 burma14 上，DP 找到了最優解 3323，差異百分比為 0.00%。
- 在 ulysses16 上，DP 找到了成本為 6859 的解，這與公認的最優解一致，因此差異百分比為 0.00%。

DP 在其適用的極小規模問題上，能夠可靠地找到最優解，且計算時間尚可接受。但到大規模問題時，執行時間指數上升的特性導致資源消耗過多。

3.3.3 模擬退火 (SA) 表現

參數設置

```
cost_mat, n,  
iterations: int = 100_000,  
start_temp: float = 1_000.0,  
alpha: float = 0.9995 (降溫)
```

- CPU 時間

SA 的 CPU 時間同樣隨問題規模增加而增加。burma14 (N=14) 耗時 0.5700s，ulysses16 (N=16) 耗時 0.6800s，ch130 (N=130) 耗時 1.0000s，而 att48 (N=48) 耗時 1.1200s。值得注意的是，att48 的 CPU 時間略高於規模更大的 ch130，這可能與 TSP 內部結構和具體的迭代次數設定或問題的收斂難度有關。

- 總成本與差異百分比:
 - 在 burma14 和 ulysses16 上，SA 均找到了最優解，差異百分比為 0.00% 。
 - 在 att48 上，SA 成本為 10815，與最優解 10628 相差僅 1.76% 。
 - 在 ch130 上，SA 成本為 6527，與最優解 6110 相差 6.82% 。
- 模擬退火在這些測試案例中普遍表現出良好的求解品質，差異百分比均控制在較低水平，且 CPU 時間相對合理。

3.3.4 蟻群最佳化 (ACO) 表現

參數設置

```
ants: int = 10,  
iterations: int = 100,  
alpha: float = 1.0,(費洛蒙影響度)  
beta: float = 2.0, (啟發式)  
rho: float = 0.5,(蒸發率)  
Q: float = 1.0,(沉積常數)
```

- CPU 時間
 - ACO 的 CPU 時間也隨問題規模增加。burma14 (N=14) 耗時 0.6700s，ulysses16 (N=16) 耗時 0.7800s，att48 (N=48) 耗時 2.6403s，規模最大的 ch130 (N=130) 耗時最長，為 8.0900s。整體而言，ACO 的 CPU 時間似乎略高於 SA。
- 總成本與差異百分比:
 - 在 ulysses16 上，ACO 成本為 6909，與最優解 6859 相差 0.73% 。
 - 在 burma14 上，ACO 成本為 3346，與最優解 3323 相差 0.69% 。
 - 在 att48 上，ACO 成本為 11217，與最優解 10628 相差 5.54% 。
 - 在 ch130 上，ACO 成本為 6818，與最優解 6110 相差 11.59% 。

ACO 在這些案例中也提供了質量較好的解，但其誤差百分比和 CPU 時間在部分案例上略遜於 SA。這可能與 ACO 參數較多、調優更為複雜有關。

3.3.5 MST 近似演算法表現 (不含 att48)

- CPU 時間

MST 近似演算法的 CPU 時間極短，對於 burma14、ch130 和 ulysses16，記錄的 CPU 時間均為 0.0000s（或極小，如 ch130 的 wall-clock time 為 0.0022s）。這體現了其作為快速近似演算法的特性。

- 總成本與差異百分比:

- 在 ulysses16 上，MST 成本為 7788，與最優解 6859 相差 13.54%。
- 在 burma14 上，MST 成本為 4003，與最優解 3323 相差 20.46%。
- 在 ch130 上，MST 成本為 8279，與最優解 6110 相差 35.50%。

MST 演算法速度非常快，但其求解品質相對於其他元啟發式演算法來說較低，差異百分比較高。這符合其作為一個具有理論近似保證（在滿足三角不等式時）但實際精度可能有限的快速估計演算法的定位。

3.4 各測試案例於不同演算法下的表現比較 (Performance Comparison on Specific Test Instances Across Algorithms)

本節將針對每個選定的測試案例，橫向比較不同演算法在 CPU 時間、總成本和與最優解差異百分比上的表現。

3.4.1 att48 案例 (N=48) (DP 不適用)

- CPU 時間

在 att48 上，最快仍是 MST(0.0007s)SA 的 CPU 時間 (1.1200s)，再來是 GA (2.4300s) 和

ACO 的 CPU 時間最長 (2.6403s)。

- 總成本與差異百分比:
 - MST 雖然快，但與最佳解偏差第二大 31.05 %
 - SA 表現最好，成本為 10815，與最優解 10628 的差異僅為 1.76% 。
 - ACO 次之，成本為 11217，差異為 5.54% 。
 - GA 表現最差，成本為 14178，差異高達 33.40%。對於 att48 這個中等規模問題，SA 在解的品質和計算效率上均展現出優勢。

3.4.2 burma14 案例 (N=14)

- CPU 時間

MST 演算法幾乎不耗時 (0.0001s)。其次是 DP (0.0900s)、SA (0.5700s)、ACO (0.6700s) 和 GA (0.7500s) 的 CPU 時間相近，均在 1 秒以內。
- 總成本與差異百分比:
 - DP、SA 和 GA 均找到了最優解 3323，差異百分比為 0.00% 。
 - ACO 的解成本為 3346，差異為 0.69%，也非常接近最優。
 - MST 的解成本為 4003，差異為 20.46%，是所有演算法中最高的。

對於 burma14 這樣的小規模問題，DP 能夠高效地找到最優解。重要的是，這類小問題凸顯了精確演算法的可行性。

3.4.3 ch130 案例 (N=130)

(DP 不適用)

- CPU 時

MST 速度最快 (0.0074s)。SA 的 CPU 時間為 1.0000s，相對較快。ACO 耗時 8.0900s，而 GA 耗時最長，為 10.9100s 。
- 總成本與差異百分比
 - SA 在此案例中表現最好，成本為 6527，與最優解 6110 的差異為 6.82% 。
 - ACO 次之，成本為 6818，差異為 11.59% 。

- MST 的成本為 8279，差異為 35.50%。
- GA 表現最差，成本高達 14264，差異達到 133.45%¹。ch130 案例似乎是一個較難的實例，它更清晰地放大了不同啟發式演算法之間的性能差異。特別是 GA 在此案例上的不佳表現，再次印證了其對問題特性和自身配置的敏感性。因此，不只模組本身，問題的幾何結構和演算法參數的失配也是原因。

3.4.4 ulysses16 案例 (N=16)

- CPU 時間

MST 依然最快 (0.0001s)。DP 耗時 0.4500s。SA (0.6800s)、ACO (0.7800s) 和 GA (0.7900s) 的 CPU 時間相近。

- 總成本與差異百分比:

- DP 和 SA 均找到了最優解 6859，差異百分比為 0.00%。
- ACO 的成本為 6909，差異為 0.73%。
- GA 的成本為 7091，差異為 3.38%。
- MST 的成本為 7788，差異為 13.54%。與 burma14 類似，在 ulysses16 這個小規模問題上，DP 和 SA 均能達到最優，其他啟發式演算法也表現良好。

4. 與開源模型比較(GA)

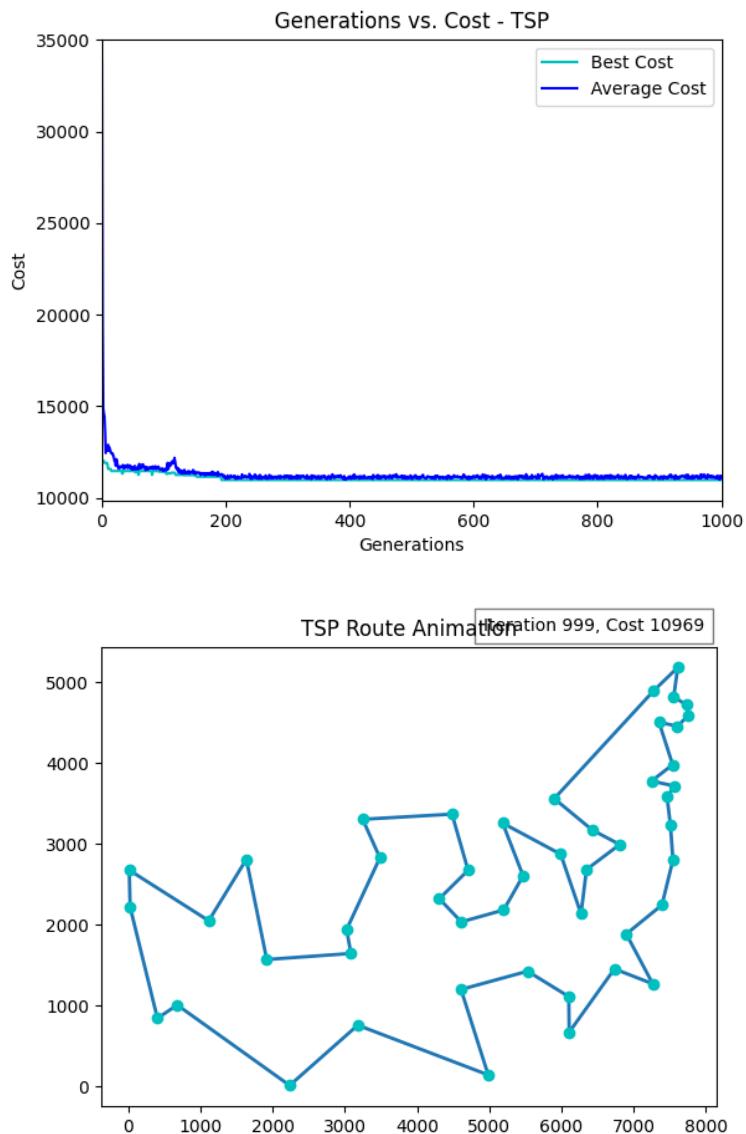
4.1 模型 TSP-Genetic-Algorithm²²

- 簡單版本採用基本的選擇 (selection)、交配 (crossover)、突變 (mutation)
- 進階 GA 則包含更大的族群規模 (population size)、更多的世代數 (generations)，並可整合局部搜尋步驟 (例如 2-Opt) 來改善解
- 提供多種突變／擾動 (mutation/perturbation) 方法，包括：
 - 隨機交換兩個城市的位置。
 - 交換相鄰城市。
 - 反轉某段子路徑 (sub-route)。
 - 使用 2-Opt 進行局部改良 (目前註解掉，但可以啟用)。

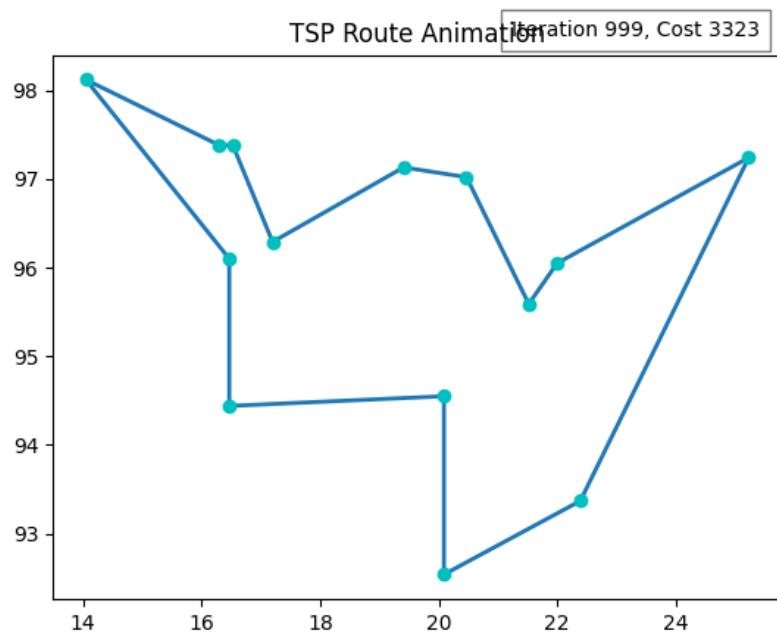
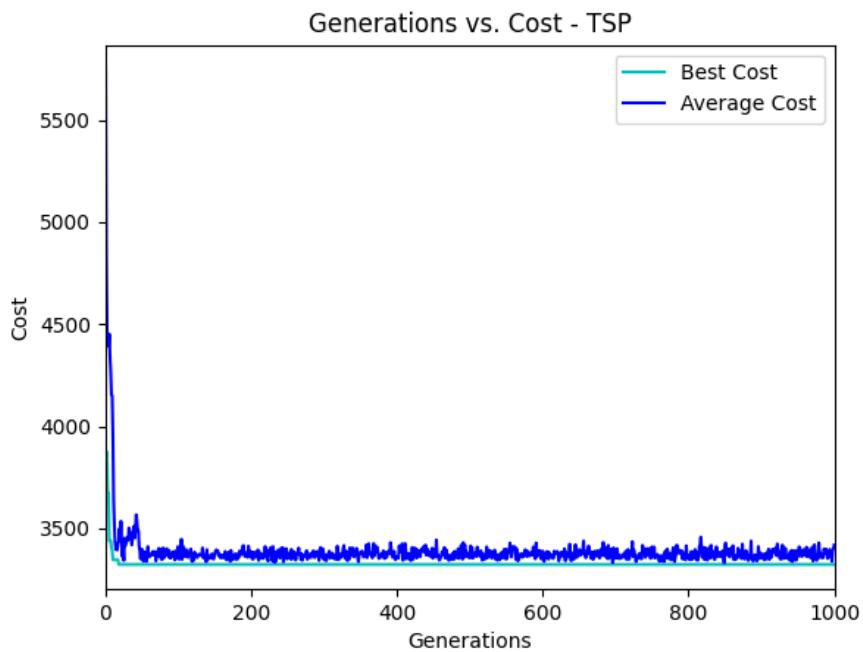
值得注意的是，作者 Renato Maynard²³所實作的此 GA 模型，除了有 MIT 授權，還具有視覺化功能，繪製每一世代的最佳成本 (best cost) 與平均成本 (average cost) 變化圖。若在 Jupyter/Python 環境中執行，還可選擇動畫顯示路線演化過程。

4.2 執行結果 (random seed = 42 for all)

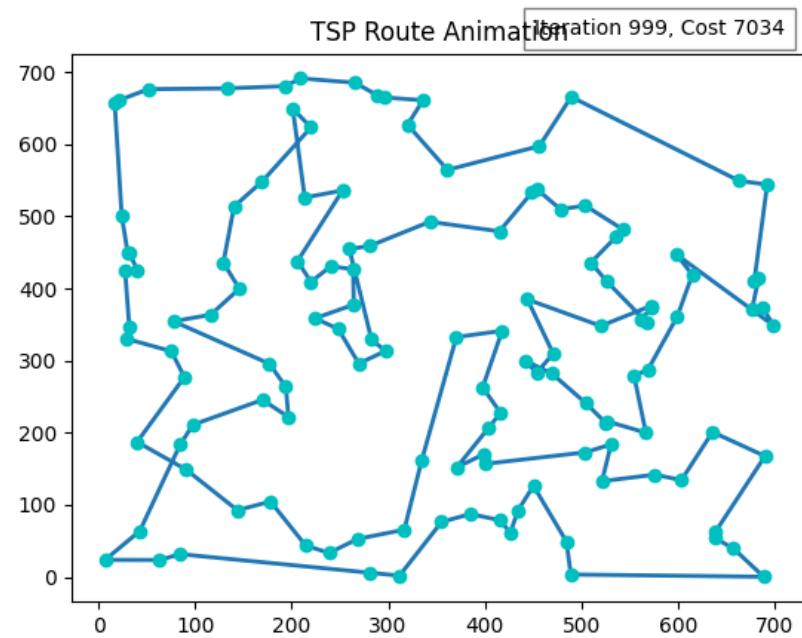
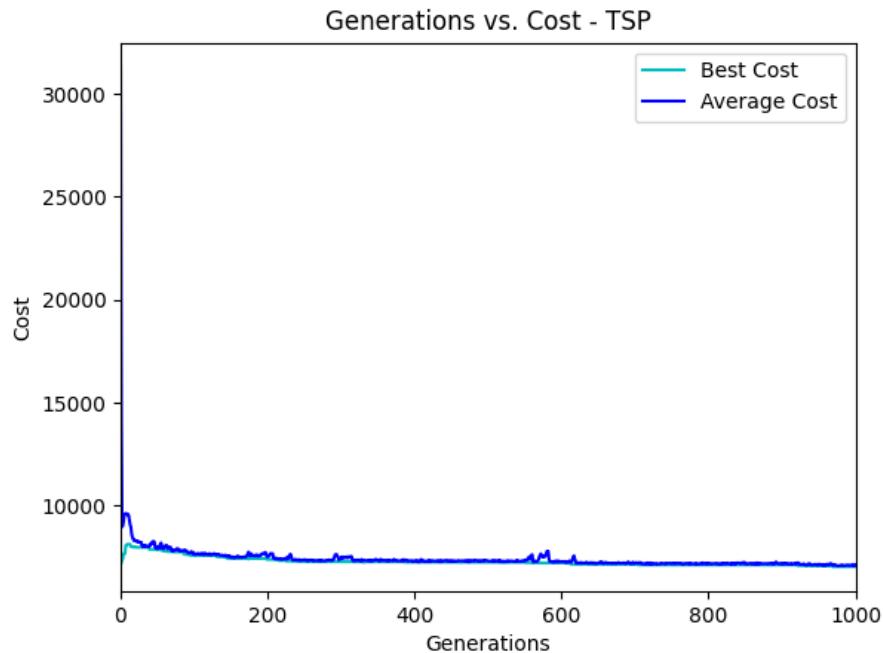
att48 : opt = 10628 testresult = 10969



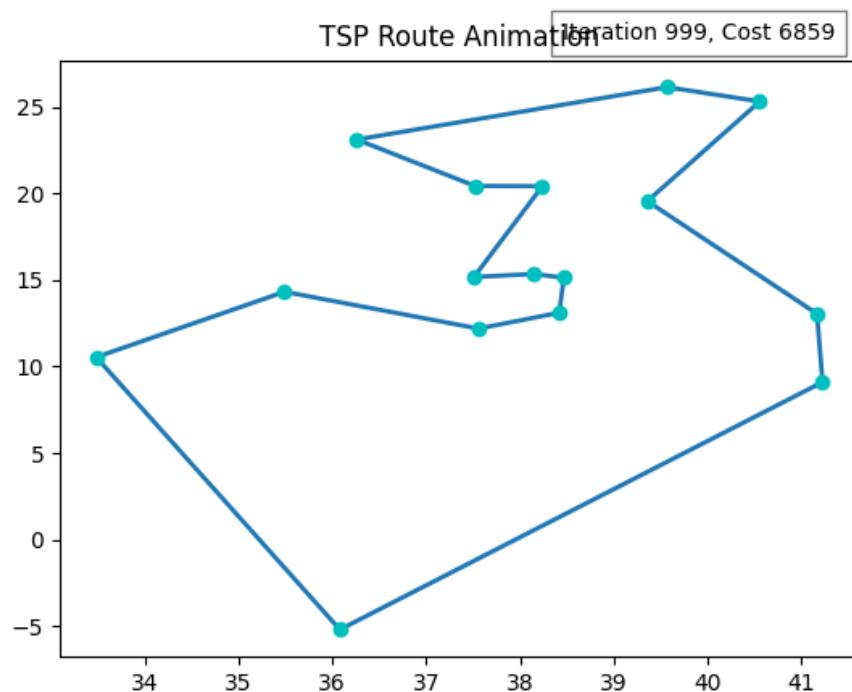
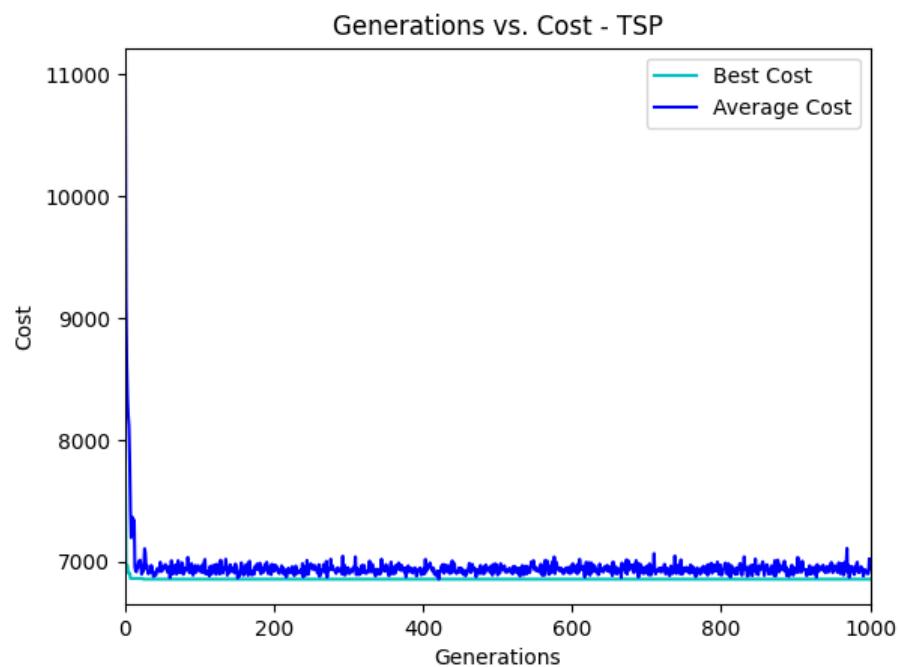
burma14 : opt = 3323 testresult = 3323



ch130 : opt = 6110 testresult = 7034



ulysses16 : opt = 6859 testresult : 6859



4.3 結果討論

4.3.1 程式碼分析

基因編碼與初始族群

面向	自製版	DEAP 版	分析
染色體表示	0-base 城市排列	同	同
初始族群生成	random.sample() → 純隨機排列	60 % 隨機 + 40 % Nearest-Neighbor 啟發式	B 初始適應度平均較低，加速早期收斂

基因編碼與初始族群

面向	自製版	DEAP 版	分析
交叉	自寫 Order Crossover (OX1)	DEAP 內建 cxOrdered (同 OX)	功能等價，但自製版額外加 parent2_ptr 容錯；B 以框架呼叫
突變	單一 Swap Mutation (交換兩點)	預設 Sub-route reverse ; 註解可切換 swap / neighbor swap ; 可選擇在內部呼叫 two_opt()	反轉或 2-opt 更容易一次修正多條錯邊，對簇狀圖效果佳
局部搜尋 (LS)	N/A	two_opt() 實作就緒 (目前註解) ; 可作 hybrid GA+LS	若開啟 2-opt，屬於 Memetic Algorithm，品質通常優於純 GA

選擇機制與策略

面向	自製版	DEAP 版	觀察
父母選擇	Truncation：先排序取前 50 % 當選擇池，再隨機配對	Tournament (size = 4)	維持多樣性較佳，且不必全量排序
精英保留	強制把當代最佳個體複製到下一代第 0 位	以 DEAP Hall of Fame (HoF) 保存，並在世代交替時更新	邏輯等價，HoF 方便日後分析

終止條件與參數

面向	自製版	DEAP 版
族群大小	100	50 (Simple) ; 100 (Advanced)
代數上限	1 500	200 (Simple) ; 1000 (Advanced)
突變率	2%	40 % (Simple ; 但突變較輕量) 10 % (Advanced)
自適應/提前停止	無	無；Advanced 版可自行在 while 內加「連續 k 代未改善終止」

4.3.2 DEAP 版 : Gen VS Cost

對於低於一定數量的迭代數，cost 的增加結果是顯而易見的，但結果再次顯現，對於不同的測資而言，其資料結構的差異與城市數量對結果的影響度不會都是一面倒的情況。

最能說明此現象的便是 att48 與 ch130 兩測資所產生的結果，從一開始低迭代數所造成的 cost 影響，ch130 幾乎都是在 10000 以下，但低城市數的 att48 却大多都大於等於 10000，甚至快觸及到 15000。

種種現象都證明著以往 TSP 其實在許多情況下並不是一面的以 N 的數值來決定，求解的複雜度，而是需要同時考慮實際資料結構。

4.3.3 小結

演算法結構而言自製版為「輕量、純 GA、手工控流程」，簡單來說是簡陋；DEAP 屬「框架化 GA，易插混合式局部搜尋」。兩者的求解品質還是有極大的差異，DEAP 優於自製版幾乎是肉眼可見的，但是問題是為什麼？

1. 在具大量局部極小的實例（如 ch130），DEAP 有啟發式初始化的優化
2. DEAP 能夠快速試驗多種 GA/LS 配方、做大規模實驗或可視化展示，更具生產力。
3. 甚至在開啟 two_opt() 並調整 mut_prob (如 0.2)，理論上能更提高效率。
4. Tournament + 較高突變率，維持族群探索，但自製版的 Trunction 相比之下過早收斂，容易忽略甚至找不到最佳解
5. 交互、選擇、突變、局部搜索，算法相同，但實際操作下，DEAP 保持了基因多樣性，這幾乎說明了一切，雖然 GA 在定義時就被賦予了一定的隨機性，但自製版在多樣性因突變率、交互、過早收束，能查找的最優解的 set 數必定少於 DEAP，而這也就影響了最終結果
6. 涵蓋基礎互動式介面，可選是否使用 Numpy
7. 程式碼寫作能力落差過大，從結構到算法實現，且自製版並無經過大量嚴謹測試(如特定 edge case)，容易在某些測試回報 error

5. 結論 (Conclusion)

5.1 主要發現總結 (Summary of Key Findings)

本報告主要對五種常用的旅行推銷員問題 (TSP) 求解演算法——遺傳演算法 (GA)、Held-Karp 動態規劃 (DP)、模擬退火 (SA)、蟻群最佳化 (ACO) 及基於最小生成樹的近似演算法 (MST)——的原理進行了闡述，並基於提供的實驗數據，對它們在 att48、burma14、ch130 和 ulysses16 四個測試案例上的性能進行了比較分析。主要發現總結如下

- Held-Karp 動態規劃 (DP)

在其適用的極小規模問題 (如 burma14, ulysses16) 上，能夠找到最優解，但其指數級的時間複雜度使其無法應用於稍大規模的問題 (如 att48, ch130)。

- MST 近似演算法

計算速度極快，幾乎不消耗 CPU 時間。然而，其求解品質相對較低，與最優解的差異百分比較大。它適用於對速度要求極高且允許一定誤差的場景，並需注意其理論保證依賴於三角不等式。

- 模擬退火 (SA)

在本次比較的測試案例中，SA 整體表現出最佳的綜合性能。它在多個小規模問題上找到了最優解，在中等規模問題 (att48, ch130) 上也獲得了與最優解差異較小 (1.76% 至 6.82%) 的高品質解，且 CPU 時間消耗相對合理。

- 蟻群最佳化 (ACO)

ACO 也能提供良好的近似解，其求解品質僅次於 SA，與最優解的差異百分比在 0.69% 至 11.59% 之間。但其 CPU 時間成本在部分案例中 (尤其是 ch130) 明顯高於 SA。

- 遺傳演算法 (GA)

GA 的表現具有顯著的可變性。雖然在小規模問題 burma14 上找到了最優解，但在 att48 和 ulysses16 上的誤差相對較大，在 ch130 上的表現則遠遜於其他啟發式演算法，誤差高達 133.45%。

5.2 演算法選擇的考量 (Considerations for Algorithm Selection)

基於上述分析，選擇 TSP 求解演算法時應綜合考量以下因素

1. 問題規模 (Problem Scale)
 - 對於城市數量非常少 (例如 $N < 20-25$) 且追求絕對最優解的情況，DP 是可行的選擇。
 - 對於更大規模的問題，必須依賴近似演算法或啟發式方法。
2. 對解品質的要求 (Solution Quality Requirement):
 - 如果必須找到最優解，且問題規模允許，則選擇 DP。
 - 如果可以接受一定程度的誤差，啟發式演算法如 SA、ACO 通常能提供高品質的近似解。SA 在本次測試中表現更為穩健。
 - 如果只需要一個快速的、粗略的估計，且問題滿足三角不等式，MST 近似演算法可以提供一個有理論界限的解。
3. 可用的計算資源與時間限制 (Computational Resources and Time Constraints):
 - MST 演算法對計算資源要求最低，速度最快。
 - SA 在本次測試中顯示了較好的時間效率平衡。ACO 的時間成本略高。GA 的時間成本則因問題而異。
 - DP 的時間成本隨規模急劇上升。
4. 演算法的複雜性與實現難度 (Algorithm Complexity and Implementation Effort)
 - MST 演算法相對簡單直觀。
 - 元啟發式演算法 (GA, SA, ACO) 的實現和參數調優可能更為複雜，需要一定的經驗。DP 的遞歸結構也需要仔細實現。
5. 問題特性 (Problem Characteristics):
 - 如 ch130 案例所示，某些問題的特殊結構可能對特定演算法（如基礎 GA）構成挑戰。理解問題特性有助於選擇更合適的演算法或對所選演算法進行針對性調整。
 - MST 演算法的性能保證依賴於三角不等式。

總而言之，不存在一種普適性的「最佳」TSP 演算法。實踐者應根據具體的應用需求和約束條件，權衡各種因素，做出明智的選擇。對於元啟發式演算法，還應考慮進行細緻的參數調優和可能的結構性改進（如混合局部搜索），以充分發揮其潛力。

6. 演算法在數位電路形式驗證中的應用擴展

6.1 遺傳演算法在硬體驗證中的「雙層架構」模式

6.1.1 外層搜尋器與內層驗證器的協同

根據 Vasíček & Sekanina 的研究²⁴，遺傳演算法在數位電路後合成優化中採用「外層 GA／內層 Model Checker」的架構模式。在每次迭代中，GA 負責產生候選電路設計，而 SAT 求解器作為內層驗證器判斷功能等價性。這種模式的核心優勢在於將全域搜尋能力與嚴格的形式驗證相結合，確保優化過程中不引入功能錯誤。

6.2 模擬退火在驗證流程優化中的關鍵作用

6.2.1 BDD 變數排序優化

由於 BDD 變數排序問題一直都被視為是優化 SAT 的其中一種方式²⁵²⁶。而 SA 提供了有效的近似解決方案²⁷。實驗結果顯示，SA 優化的變數排序可以使 BDD 節點數下降一個數量級，大幅降低記憶體消耗並提升驗證效率。

6.3.2 強化學習增強的 SAT 求解

GQSAT 系統通過分析問題實例的結構特徵，在搜尋初期做出更佳的分支決策，並能泛化到比訓練集大 5 倍的問題規模²⁸。

6.3.3 神經網路輔助的模型檢查

Neural Model Checking 技術代表了形式驗證的最新發展方向。該方法使用神經網路作為線性時序邏輯的形式證明證書，通過隨機執行軌跡訓練神經排名函數，然後利用 SAT 求解器驗證其有效性。實驗結果表明，在 194 個標準硬體模型檢查問題上，該方法平均比學術工具多完成 60 個任務，比商業工具多完成 11 個任務。²⁹

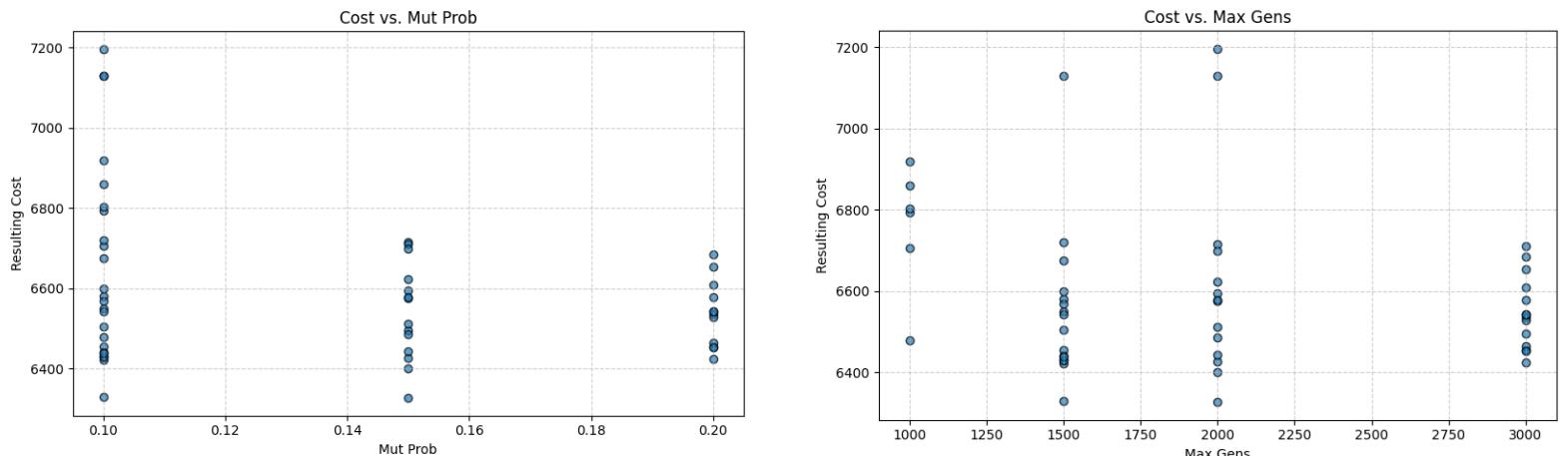
7. 額外補充

一開始在規劃期末計畫時，我參考了一些比較近期且有關於 RTL 電路驗證的論文，其中”*SoC connectivity specification extraction using incomplete RTL design: An approach for Formal connectivity Verification*”³⁰ 的內容是我比較感興趣的。

而又由於我自身在這學期的期許是學習演算法，所以我選擇了演算法為主題的報告內容。一開始其實考慮過自作 NP-hard 相關議題的 RTL 模型，並透過 abc³¹ 工具進行 UBMC、INT、PDR 的驗證，並根據數據分析各驗證方式的優劣。但由於模組化的過程受挫，因此轉向了使用其他方式進行演算法與形式驗證相關議題的專題題目。

後續在查找過程後，才確認了以 TSP 問題為模型，進行形式驗證方面的演算法研究。故以此而選擇了 DP、MST 作為課堂所學的代表，SA、GA、ACO 作為延伸內容，進行後續的規劃。

而在前述的主題作完後，我思考是否能使用適應優化的方式，對 GA、SA、ACO 這幾個比較有趣的模組作效能改良。故後續才有了這一章節，雖然只是簡單的調整與改良，但我還是覺得挺有趣的，所以在最後不當成正文補充。



由一開始對 GA 的簡介可知，GA 的演算法主要由幾個參數控制，所以我將原模組改成求最近似解為前提，對程式碼進行自適應更改參數，以逼近輸入參數。其中有因應 GA 容易過早收斂的情況，因此加入激勵機制，使自適應程序幫助 GA 在一定情況下能夠逃出過早收斂踢出特定解的問題。而以上是將自適應迴圈設為 50 次，得出的其中兩個參數對 cost 的影響圖表。

Final Work :

Github : <https://github.com/jimmy01081122/EDA-Final-Project>

Reference

¹ Understanding Formal Verification

<https://blogs.sw.siemens.com/verificationhorizons/2024/09/05/understanding-formal-verification/>

² The Rise of Formal Verification in Hardware Design: Adoption Trends and Future Projections
<https://www.linkedin.com/pulse/rise-formal-verification-hardware-design-adoption-trends-alwin-dsouza-ipqtc>

³ Vasicek, Z., Sekanina, L. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genet Program Evolvable Mach* 12, 305–327 (2011). <https://doi.org/10.1007/s10710-011-9132-7>

⁴ TSP-Genetic-Algorithm Model : https://github.com/RenatoMaynard/TSP-Genetic-Algorithm?utm_source=chatgpt.com

⁵ Adewole, Philip & Akinwale, Adio & Otubamowo, Kehinde. (2011). A Genetic Algorithm for Solving Travelling Salesman Problem. *International Journal of Advanced Computer Sciences and Applications*. 2. 10.14569/IJACSA.2011.020104

⁶ Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) Optimization by Simulated Annealing. *Science*, 220, 671-680. <https://doi.org/10.1126/science.220.4598.671>

⁷ Dorigo, Marco & Maniezzo, Vittorio & Colomi, Alberto. (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Trans Syst Man Cybernetics - Part B. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*. 26. 29-41. 10.1109/3477.484436.

⁸ van Bevern, René; Slugina, Viktoriia A. (2020), "A historical note on the 3/2-approximation algorithm for the metric traveling salesman problem", *Historia Mathematica*, 53: 118–127, [arXiv:2004.02437](https://arxiv.org/abs/2004.02437), doi:[10.1016/j.hm.2020.04.003](https://doi.org/10.1016/j.hm.2020.04.003), S2CID 214803097

⁹ Christofides, Nicos. (2022). Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem. Carnegie Mellon University. 3. 10. 10.1007/s43069-021-00101-z.

¹⁰ Larranaga, Pedro & Kuijpers, Cindy & Murga, R. & Inza, Inaki & Dizdarevic, S.. (1999). Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review*. 13. 129-170. 10.1023/A:1006529012972.

¹¹ Haojie Xu, Yisu Ge, and Guodao Zhang. 2023. Genetic algorithm for Traveling Salesman Problem. In Proceedings of the 2022 5th International Conference on Computational Intelligence and Intelligent Systems (CIIS '22). Association for Computing Machinery, New York, NY, USA, 33–40. <https://doi.org/10.1145/3581792.3581798>

¹² Larranaga, Pedro & Kuijpers, Cindy & Murga, R. & Inza, Inaki & Dizdarevic, S.. (1999). Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review*. 13. 129-170. 10.1023/A:1006529012972.

¹³ Traveling Salesman Problem using Genetic Algorithm

<https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/>

¹⁴ Held, M. and Karp, R. (1962) A Dynamic Programming Approach to Sequencing. Journal for the Society for Industrial and Applied Mathematics, 10, 1-10. <https://doi.org/10.1137/0110015>

¹⁵ Travelling Salesman Problem using Dynamic Programming
<https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/>

¹⁶ Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983) Optimization by Simulated Annealing. Science, 220, 671-680.
<http://dx.doi.org/10.1126/science.220.4598.671>

¹⁷ Luping Fang, Pan Chen, and Shihua Liu. 2007. Particle swarm optimization with simulated annealing for TSP. In Proceedings of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases - Volume 6 (AIKED'07). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 206–210.

¹⁸ M. Dorigo, V. Maniezzo and A. Colorni, "Ant system: optimization by a colony of cooperating agents," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 26, no. 1, pp. 29-41, Feb. 1996, doi: 10.1109/3477.484436.

keywords: {Ant colony optimization;Traveling salesman problems;Intelligent robots;Feedback;Distributed computing;Simulated annealing;Robustness;Artificial intelligence;Computational modeling;Data structures},

¹⁹ Approximate solution for Travelling Salesman Problem using MST
<https://www.geeksforgeeks.org/approximate-solution-for-travelling-salesman-problem-using-mst/>

²⁰ <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/>

²¹ <https://github.com/RenatoMaynard/TSP-Genetic-Algorithm.git>

²² <https://github.com/RenatoMaynard>

²³ Vasicek, Z., Sekanina, L. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genet Program Evolvable Mach 12, 305–327 (2011). <https://doi.org/10.1007/s10710-011-9132-7>

²⁴ TSP-Genetic-Algorithm Model :
<https://github.com/RenatoMaynard/TSP-Genetic-Algorithm/tree/main>

²⁵ Orna Grumberg, Shlomi Livne, and Shaul Markovitch. 2003. Learning to order BDD variables in verification. J. Artif. Int. Res. 18, 1 (January 2003), 83–116.

²⁶ Aloul, Fadi & Markov, Igor & Sakallah, Karem. (2004). MINCE: A static global variable-ordering heuristic for SAT search and BDD manipulation. JOURNAL OF UNIVERSAL COMPUTER SCIENCE. 10. 1562-1596.

²⁷ B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," in IEEE Transactions on Computers, vol. 45, no. 9, pp. 993-1002, Sept. 1996, doi: 10.1109/12.537122.

²⁸ Kurin, Vitaly & Godil, Saad & Whiteson, Shimon & Catanzaro, Bryan. (2019). Improving SAT

Solver Heuristics with Graph Networks and Reinforcement Learning.
10.48550/arXiv.1909.11830.

²⁹ Mirco Giacobbe, Daniel Kroening, Abhinandan Pal, and Michael Tautschnig. 2025. Neural model checking. In Proceedings of the 38th International Conference on Neural Information Processing Systems (NIPS '24), Vol. 37. Curran Associates Inc., Red Hook, NY, USA, Article 2742, 86375–86398.

³⁰ H. Saafan, M. W. El-Kharashi and A. Salem, "SoC connectivity specification extraction using incomplete RTL design: An approach for Formal connectivity Verification," 2016 11th International Design & Test Symposium (IDT), Hammamet, Tunisia, 2016, pp. 110-114, doi: 10.1109/IDT.2016.7843024. keywords: {Formal verification;Documentation;Pins;Standards;Libraries;Hardware design languages;Data mining;Bus interface;Connectivity;Formal Verification;Integration;IP Reuse;IP-XACT;System-on-Chip},

³¹ <https://github.com/berkeley-abc/abc>