

---

# GA to GA

以基因演算法最佳化基因演算法參數-以TSP為例

期末報告 第二組

0711239李勝維 0713218蔡沛瑤 H092638廖洧舜

---

# 目錄

---

**1** 研究背景與動機

**2** 研究目的與貢獻

**3** 研究方法

**4** 實驗設定與成果

**5** 結論

**6** 參考資料

**7** Q&A

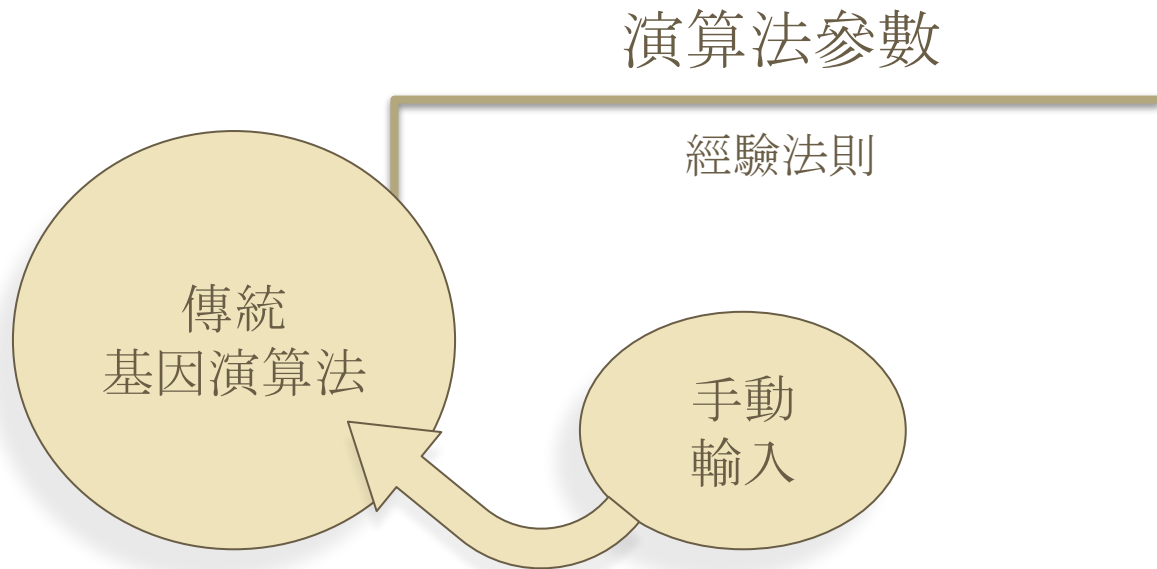
# 研究背景 & 研究動機

## PART 01



# 研究背景

---



# 研究背景

---

TSP

VRP

Job shop

.....

調整參數獲得好的解較費時

# 研究動機

---

人工試驗  
參數



電腦挑選  
參數

# 研究動機

---

尋找文獻



GA-based  
reinforcement learning



缺少調整GA參數的  
自動方法

# PART 02

## 研究目的與研究貢獻

---

- 由程式給定的參數效能好於專家設定
- 完成的程式可用於其他問題(不限於本次專題), 並置於公開 Github repository

Github repo link:





# 研究方法

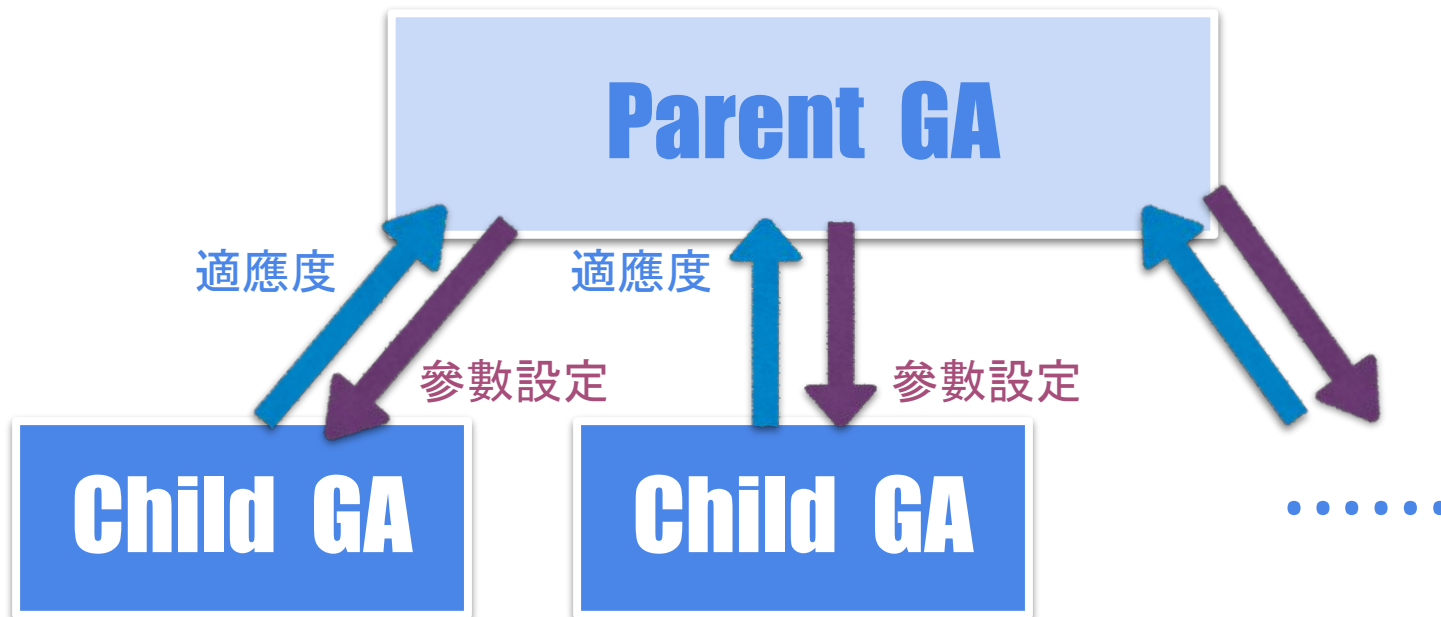
## PART 03

- 編碼
- 解碼、fitness
- 其他重要的運算過程



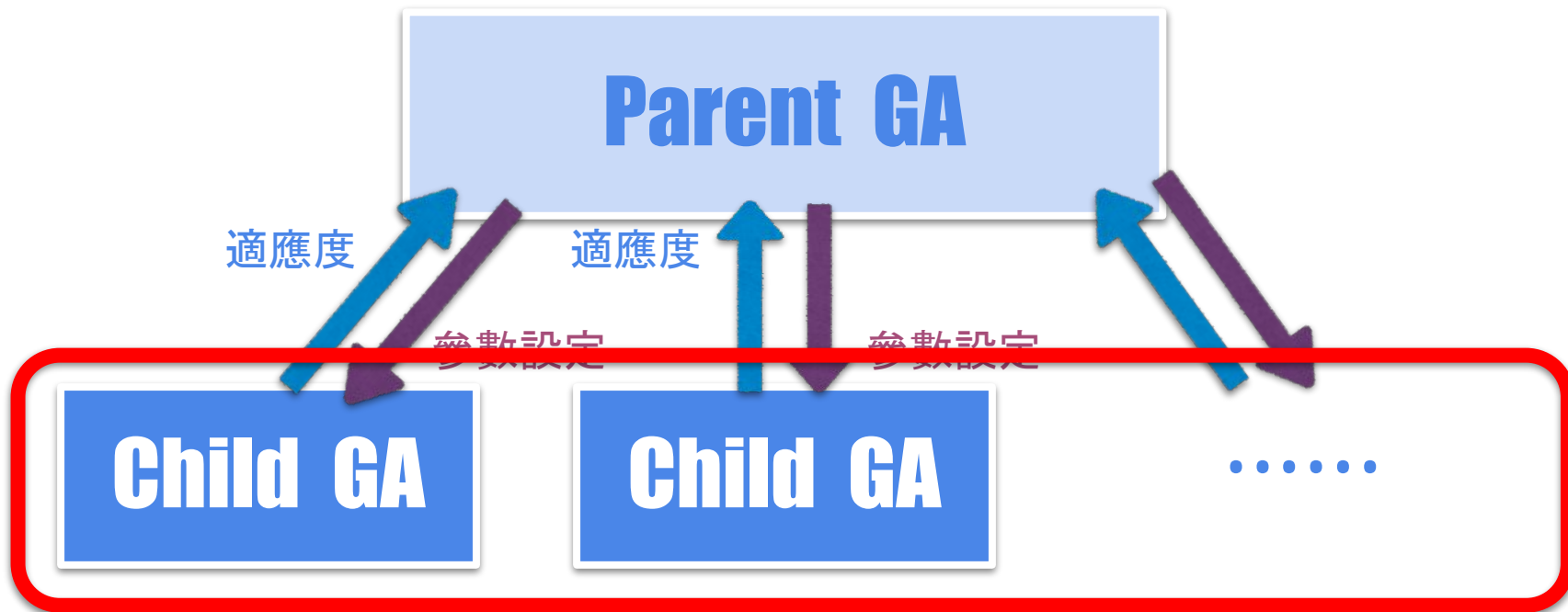
# 程式架構

---



# 程式架構

---



# Child GA--我們是怎麼設定參數的

```
def Child_GA(Pc, Pm, NUM_CHROME, TSP_graph):  
    NUM_BIT = len(TSP_graph) - 1  
    NUM_PARENT = NUM_CHROME  
    NUM_CROSSOVER = max(int(Pc * NUM_CHROME / 2), 1)  
    NUM_CROSSOVER_2 = NUM_CROSSOVER*2  
    MAX_NUM_ITERATION = 30000
```

把Child GA整體包裝成一個函數

# Child GA——編碼

染色體編碼: (Permutation encoding)



## Child GA--解碼及適應度計算

---

```
def fitFunc(x):  
    cost = TSP_graph[0][x[0]]  
    for i in range(NUM_BIT-1):  
        cost += TSP_graph[x[i]][x[i+1]]  
    cost += TSP_graph[x[NUM_BIT-1]][0]  
    return -cost
```

# Child GA--新的交配方式

## Cycle Crossover Operator

$$P_1 = \overset{1}{(1)} 2 3 \overset{4}{(4)} 5 6 \overset{3}{(7)} \overset{2}{(8)},$$

$$P_2 = (8 \text{ 5 2 1 \text{ 3 } 6 4 7).$$

重複時停止

$$O_1 = (1 \times \times 4 \times \times 7 \times 8).$$

剩下的以P2中的順序補上

$$O_1 = (1 \text{ 5 2 4 \text{ 3 } 6 7 8).$$

## Child GA——新的


$$P_1 = \overset{1}{(1)} \overset{4}{2} \overset{3}{3} \times \overset{8}{8}).$$

从P2中的顺序补上

$$P_2 = (8 \quad 2 \quad 4 \quad \underline{3} \quad \underline{6} \quad 7 \quad 8).$$



# Child GA--真香

---

Cycle Crossover  
Operator



Uniform  
crossover

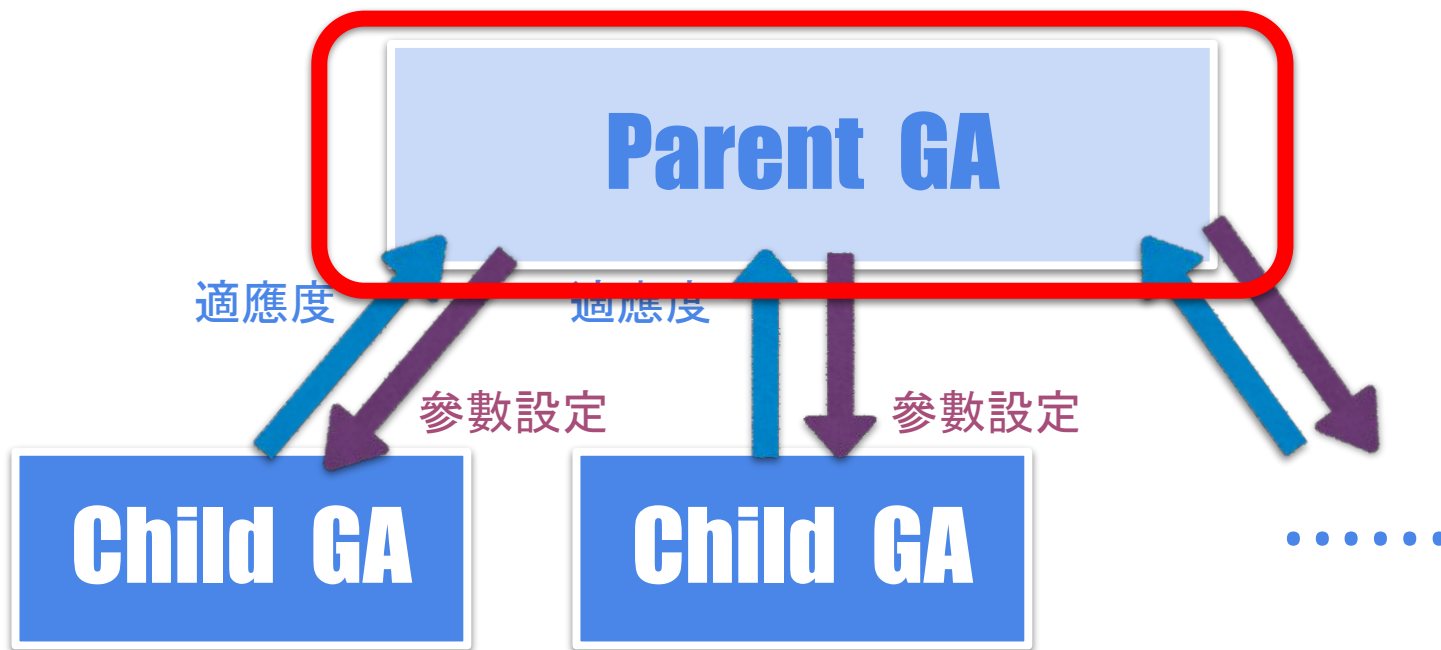
# Child GA--怎麼判斷這次做的好不好？

整個Child GA函數回傳：直到收斂所需要的迴圈數

```
memory = MEMORY(capacity=len(TSP_graph)**2)
for i in range(1, MAX_NUM_ITERATION+1):
    parent = selection(pop, pop_fit)
    offspring = crossover_uniform(parent)
    mutation(offspring)
    offspring_fit = evaluatePop(offspring)
    pop, pop_fit = replace(pop, pop_fit, offspring, offspring_fit)
    mean = -1 * np.average(pop_fit)
    if mean == memory.get():
        return i
    memory.add(mean)
```

# 程式架構

---



## Parent GA——編碼

---

[

交配率

突變率

染色體數目

]

染色體編碼: (Value Encoding)

# 參數範圍設定

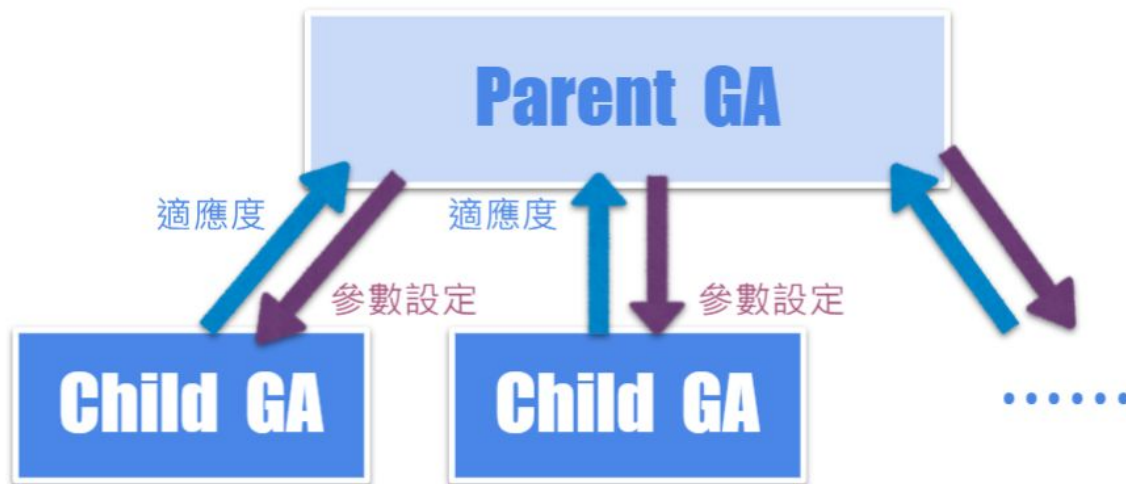
參數範圍限制	區間
交配率	[0.6, 1]
突變率	[0, 0.01]
染色體數目	[30, 50]

# Parent GA—解碼及適應度計算

```
def fitFunc(self, x):  
    '''  
    Definition of fitness function:  
    average of iterations for each map  
    '''  
  
    Pc, Pm, chrome = x  
    sum_ = 0  
    for graph in self.TSP_maps:  
        sum_ += float(Child_GA(Pc=Pc, Pm=Pm,  
                                NUM_CHROME=chrome, TSP_graph=graph))  
    fit_value = sum_ / len(self.TSP_maps)  
    return fit_value
```

# Parent GA——平行化運算適應度函數

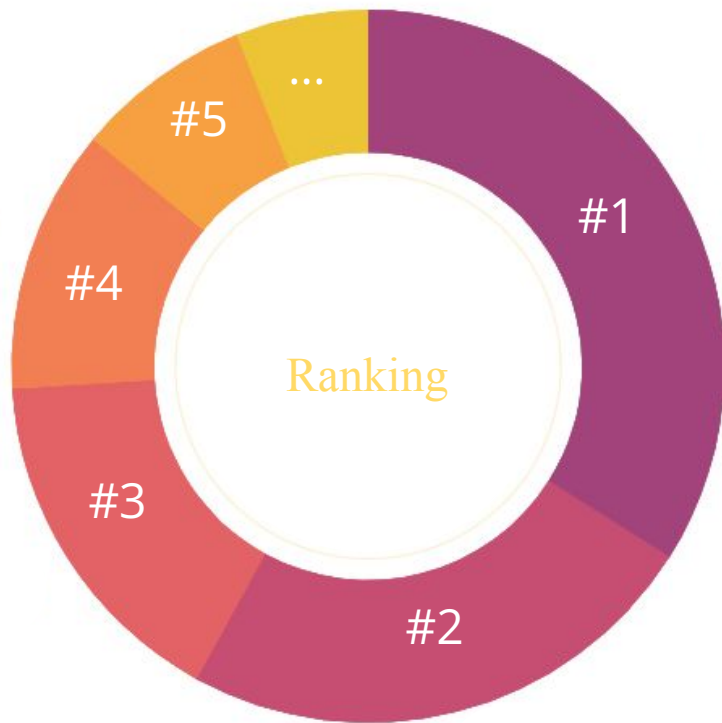
```
def evaluatePop(self, P):  
    with mp.Pool(mp.cpu_count()) as pool:  
        return pool.map(self.fitFunc, P)
```



## Parent GA——擇偶方式

Rank based wheel selection:

用適應度排名，名次越前，權重越大





# Parent GA--擇偶方式

---

```
def selection(self, p, p_fit):  
    '''  
    Rank selection  
    '''  
    a = []  
    sorted_p = [x for _, x in sorted(zip(p_fit, p), reverse=True)]  
    weights = list(range(1, len(sorted_p)+1))  
    for _ in range(self.NUM_PARENT):  
        parent = random.choices(sorted_p, weights=weights)[0]  
        a.append(parent)  
    return a
```

# Parent GA--交配方式

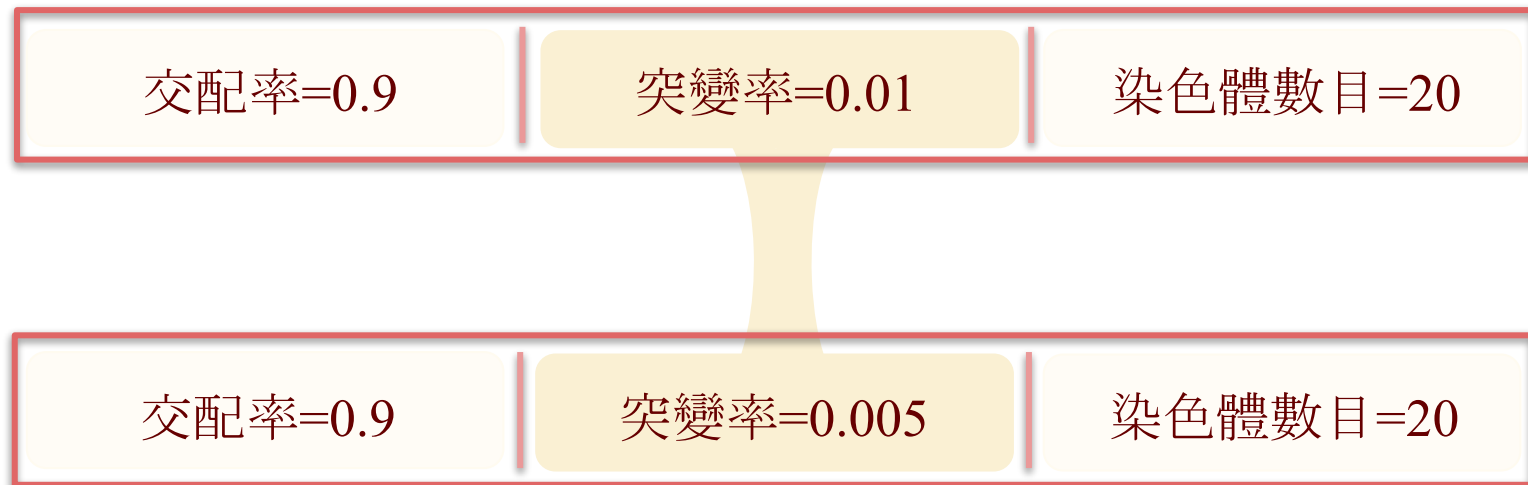
```
def crossover(self, P):  
    a = []  
    for p in P:  
        if self.TF("Pc"):  
            c = random.randrange(1, self.NUM_BIT)  
            another = random.choice(P)  
            child1 = p[:c]+another[c:]  
            child2 = another[:c]+p[c:]  
            a.append(child1)  
            a.append(child2)  
    return a
```

# Parent GA--突變方式

---

Uniform mutation: 隨機選擇一個基因組 (genome), 並初始化它

EX:



# Parent GA--突變方式

```
def mutation(self, p):
    for ch in p:
        if self.TF("Pm"):
            k = random.randrange(self.NUM_BIT)
            if k == 0: # Initialize Pc
                ch[k] = random.uniform(*self.Pc_range)
            elif k == 1: # Initialize Pm
                ch[k] = random.uniform(*self.Pm_range)
            elif k == 2: # Initialize NUM_CHROME
                ch[k] = random.randint(*self.Chrome_range)
            else:
                raise IndexError("Wrong mutation!")
```

# 實驗設定 & 成果

## PART 04



# Child GA--參數設定

為了確定本次研究利用Parent GA是否調整一組近似最佳的參數，我們透過手動調整三個參數找出最好的搭配



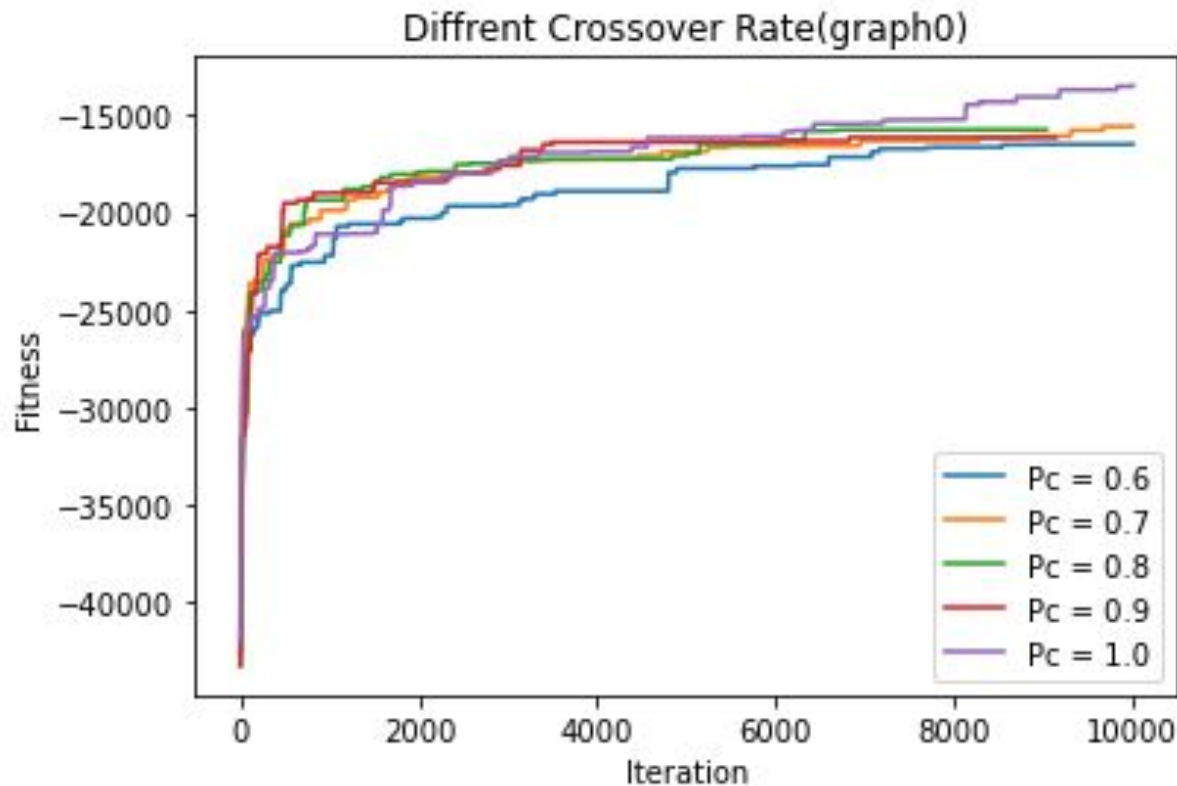
# Child GA--手動參數設定(Case 1)

---

Case 1 : **變動交配率**, 固定突變率、染色體數目

參數	Case 1 - 交配率差異
交配率	<b>[0.6, 0.7, 0.8, 0.9, 1.0]</b>
突變率	0.01
染色體數目	30

# Child GA--手動參數設定(Case 1)



當交配率越高, 收斂的結果較佳!



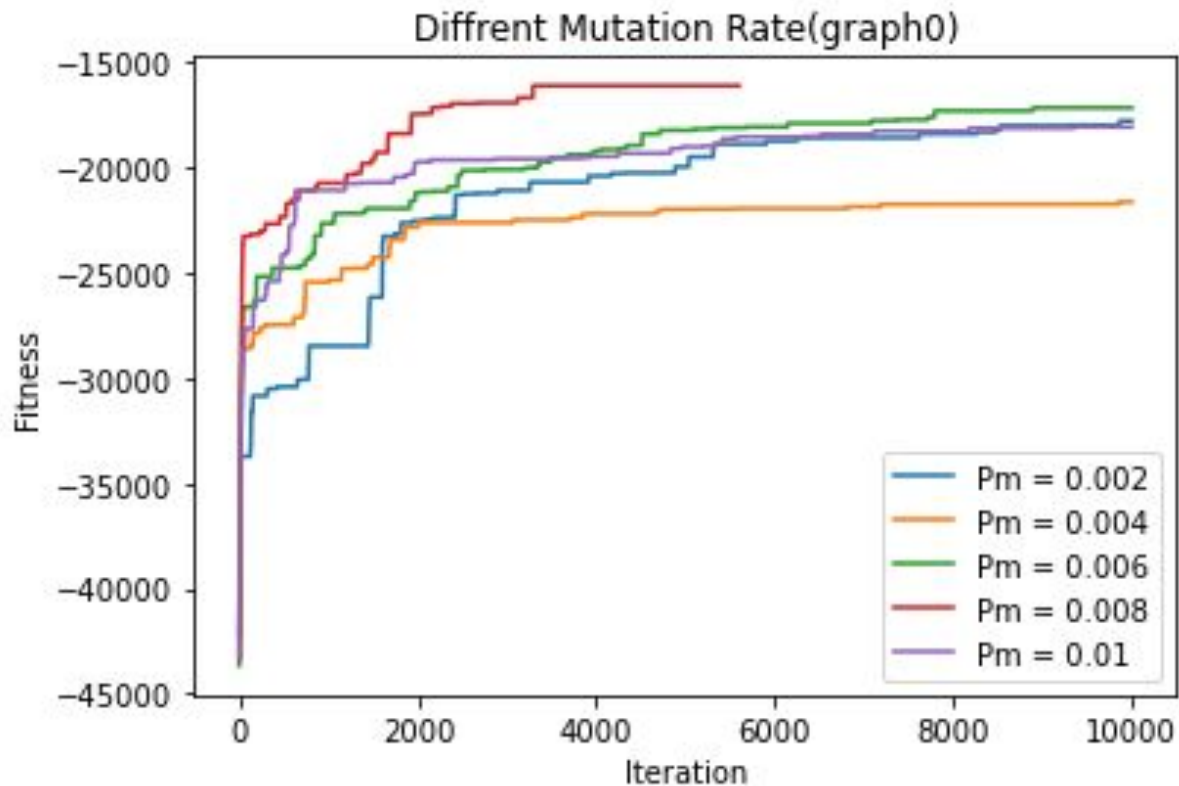
# Child GA--手動參數設定(Case 2)

---

Case 2 : **變動突變率**, 固定交配率、染色體數目

參數	Case 2 - 突變率差異
交配率	1.0
突變率	<b>[0.002, 0.004, 0.006, 0.008, 0.01]</b>
染色體數目	30

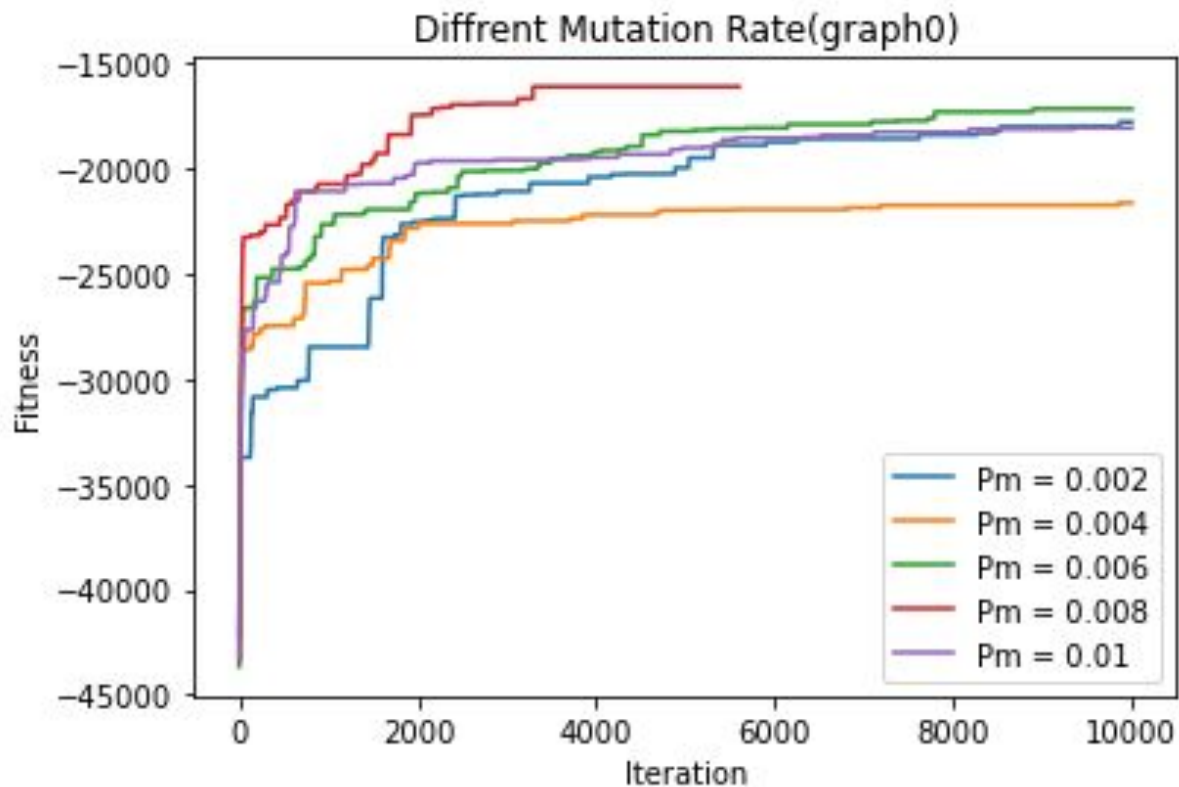
## Child GA--手動參數設定(Case 2)



多次數據平均過後  
當突變率為0.008時  
不僅收斂結果較佳  
且收斂速度快



## Child GA--手動參數設定(Case 2)



不同instance下  
結果卻不同？



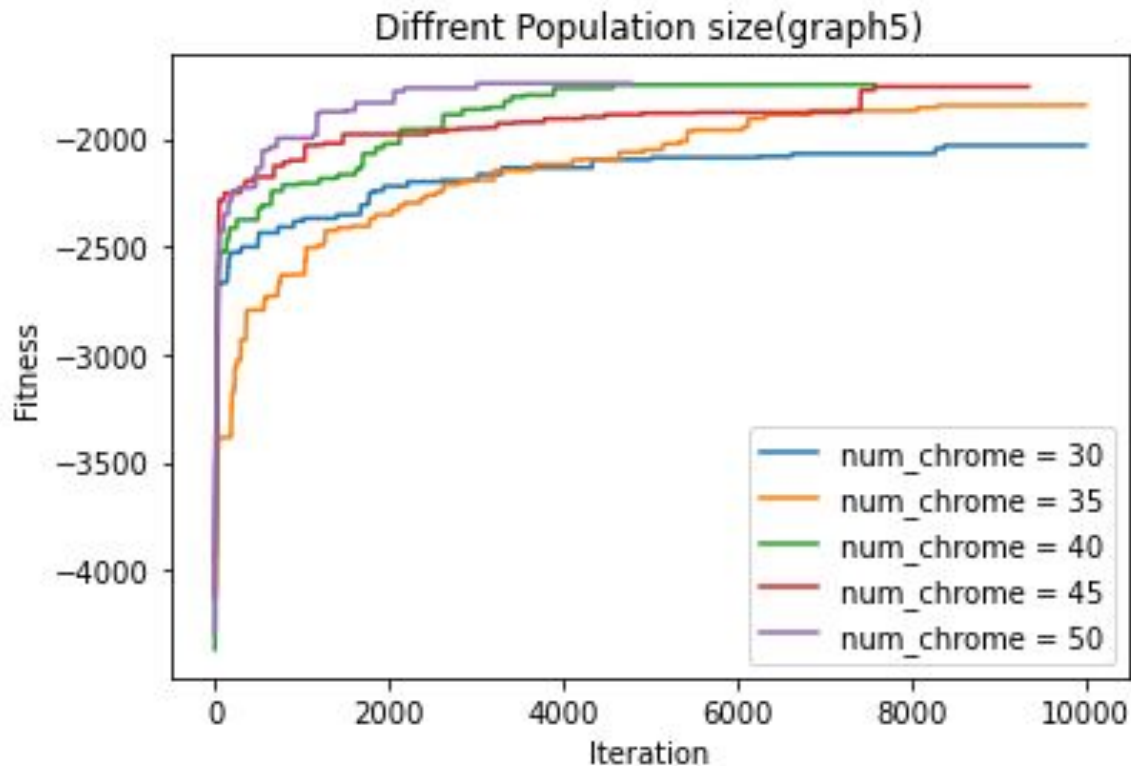
# Child GA--手動參數設定(Case 3)

---

Case 3 : **變動染色體數目**, 固定交配率、突變率

參數	Case 3 - 染色體數目差異
交配率	1.0
突變率	0.01
染色體數目	<b>[30, 35, 40, 45, 50]</b>

## Child GA--手動參數設定(Case 3)



與突變率情形類似  
在一樣的參數下  
不同instance的表現  
不一

# Child GA--手動參數設定

利用傳統方法測試多組參數，在分別校調交配率、突變率及染色體數目後，選出三組最佳的組合：

參數組1

交配率	突變率	染色體數目
0.7	0.008	40
4373		

參數組2

交配率	突變率	染色體數目
0.7	0.006	40
4959		

參數組3

交配率	突變率	染色體數目
0.8	0.008	35
4561		

收斂所需迭代數

# Child GA—參數設定 by Parent GA

$P_c = 0.8$

$P_m = 0.01$

$\text{Num\_chrome} = 5$

$\text{Num\_iteration} = 20$

*iteration 1*

Pc	Pm	Num_chrome	Best fit_value
0.8019	0.002818	33	3791.286
0.8019	0.002818	33	4102.571
0.7620	0.007838	39	4138
0.8019	0.002818	33	4154.429
0.9378	0.007580	43	4218.429

...

*iteration 10*

Pc	Pm	Num_chrome	Best fit_value
0.7620	0.002818	33	3460.143
0.7620	0.002818	33	3491.714
0.7620	0.002818	33	3756.429
0.8019	0.002818	33	3791.286
0.8019	0.002818	33	3801


...

*iteration 20*

Pc	Pm	Num_chrome	Best fit_value
0.7620	0.002818	33	2802.286
0.7620	0.002818	33	3460.143
0.7620	0.002818	33	3462.286
0.7620	0.002818	33	3491.714
0.7620	0.002818	33	3650.714

# Child GA--參數比較

將Parent GA所調的參數與三組最佳的手動設定組合做比較



Parent GA設定	
chrome #	收斂所需 迭代數
1	2802
2	3460
3	3462

手動設定	
組合	收斂所需 迭代數
1	4373
2	4959
3	4561



# 結論

## PART 05



# 結論與延伸

- 手動設定的參數表現不如預期結果很差, 可能是當初做實驗時並沒有考慮到各個參數的關聯性
- **Parent GA**所求得之參數組合表現非常好, 雖然在搜尋的過程中稍微耗時, 但相比起手動設定及分析花的時間還少許多
- 本研究只有做三個參數的設計, 未來可再加入其他參數更加優化整個演算法(如競爭、交配的方法等), 並可運用至其他的演算法或者機器學習的參數優化
- 基因演算法雖然在參數的設計上會影響收斂的快慢與好壞, 但可以多加著墨在染色體的Encoding及Decoding、Fit function的設計

# 參考資料

---

1. Abid Hussain, Yousaf Shad Muhammad, M. Nauman Sajid, Ijaz Hussain, Alaa Mohamd Shoukry and Showkat Gani. (2017, October 25). Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Computational Intelligence and Neuroscience*, Article ID 7430125.
2. Avni Rexhepi, Adnan Maxhuni and Agni Dika. (2013, January). Analysis of the impact of parameters values on the Genetic Algorithm for TSP
3. **TSPLIB**: <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/>
4. 國立陽明交通大學 109學年度 1491號課程「基因演算法與管理科學應用」上課教材  
"GA07 排列解的編碼與 Traveling Salesman Problem (TSP)"

# Q&A時間

