Solve Job Assignment Problem

with Brute Force Algorithm and Genetic Algorithm

張偉治

1. Brute Force

觀察暴力解的時間複雜度，當有 n 個工作時，第一次可選擇的任務數量為 n，下一次迭代時因為已經有一個工作被選走，因此第二次選擇時可選擇的任務數量為 n-1，最終得到暴力解的時間複雜度為 O(n!)*O(n)，其中 O(n)為計算路徑分數的時間複雜度。

1.1 路徑分數計算

1.1.1 說明

根據輸入路徑，循序從資料輸入 input 中檢索對應位置的值並相加，得到之總和即為該路徑所對應的路徑分數。

1.1.2 程式碼

```
def score(path):
    temp = 0
    for i in range(len(path)):
        temp = temp + input[i][path[i]-1]
    return temp
```

1.2 解碼

在工作指派問題中，最多會有 n!種路徑存在，將路徑編碼後可得一對一且映成，值為 0 ~ n!-1 的密碼，加密最大的作用是密碼相對於路徑較容易進行操作與指派，並且密碼只要經過 n 次的雜湊後即可解碼出相對應的路徑，可在時間複雜度與操作性上取得平衡。同時加密過的路徑在剪枝時只要透過雜湊將不要的分支排除即可，但國慶連假每個老師都出了大量的作業，剪枝就沒在報告中實作。

1.2.1 說明

在解密時需要參照根據 input 大小所生成的階乘表，密碼會根據階乘表進行操作，以密碼 11 舉例，因為 11=1*(3!)+2*(2!)+1*(1!)，因此可解碼為[1, 2, 1]的路徑。使用階乘表最主要是因為階乘的計算複雜度成長十分快速，透過階乘表可大幅減少階乘計算的次數。

1.2.2 程式碼

```
def decoder(value, times):
    temp = value
    decode = []
    for i in range(1,len(input)):
        temp_2 = temp//times[i]
        temp_3 = temp%times[i]
        decode.append(temp_2)
        temp = temp%times[i]
    return decode
```

1.3 二次解碼與路徑生成

一次解碼 D 在工作指派問題中的實際意義是：m 次迭代後，在剩餘 n-m 個工作中，選擇第 D[m]+1 筆工作。可以從中得到 D 的數學特性 D[m] <= n-m，程式碼的實現就是基於這個特性。

1.3.1 說明

再次以密碼 11 舉例，11 可一次解碼為[1, 2, 1]，再將[1, 2, 1]帶入進行二次解碼可得到真實路徑[2, 4, 3, 1]。因此二次解碼的過程同時就是從 D 生成路徑的方法。

1.3.2 程式碼

```
def path_gen(decoder, length):
    path = []
    stack = list(range(1,length+1))
    for i in decoder[:]:
        temp = i
        count = 0
        while( temp != -1 and count<len(stack) ):
            if(stack[count]>=0):
                if( temp==0 ):
                    path.append(stack[count])
                    stack[count] = -1
                temp = temp - 1
            count = count + 1
    for i in stack:
        if(i!=-1):  path.append(i)

    return path
```

2. Genetic Algorithm

2.1 基因生成

同 section 1 說明，基因 G 被定義為在 n 個工作中，第 m 次任務選擇剩餘的第 d 個任務的集合 D，因此基因 G 的第一個部分可能為 1 < G[1] < n，最後一個部分因為只剩一個任務可以指派，因此 G[n] 必為 1。而最大可能基因數量被限制在 n! 內。

2.1.1 說明

基於一開始密碼與解碼的設計，可確保基因在交配後不會有不合法的基因產生，因為基因被定義為在 n 個工作中，第 m 次任務選擇剩餘的第 d 個任務的集合 D，而不是第 m 次任務選擇任務 a 的集合 A。後者在實作上，容易出現大量不合法基因，若要排除掉不合法基因則需要大量且複雜的判斷式。

2.1.2 程式碼

```
def chromosomes(length):
    chromo = []
    while(length):
        chromo.append(random.randint(1,length))
        length = length - 1
    return chromo
```

2.2 演化

演化主要分成四的部分：基因池基因分數計算、分數排名、強勢交配以及突變，其中基因池分數計算的原理同 section 1，排名的部分則使用合併排序。

2.2.1 強勢交配

在演化設計中，定義分數最好的基因為強勢基因，若基因池數量為 2k，則定義強勢交配為分數前 k+1 的基因分別與強勢基因交配產出兩組後代，因此經過強勢交配後維持基因池大小為 2k，這樣設計不僅可簡化程式的撰寫，同時可表留剛好數量的優良基因，收斂則由演化次數常數決定

2.2.2 突變

遍歷交配後產生的新基因池的每一個節點，每一個節點皆有相同機率產生突變，突變的值被限制 1 <= G[m] <= n-m 內。

2.2.3 程式碼

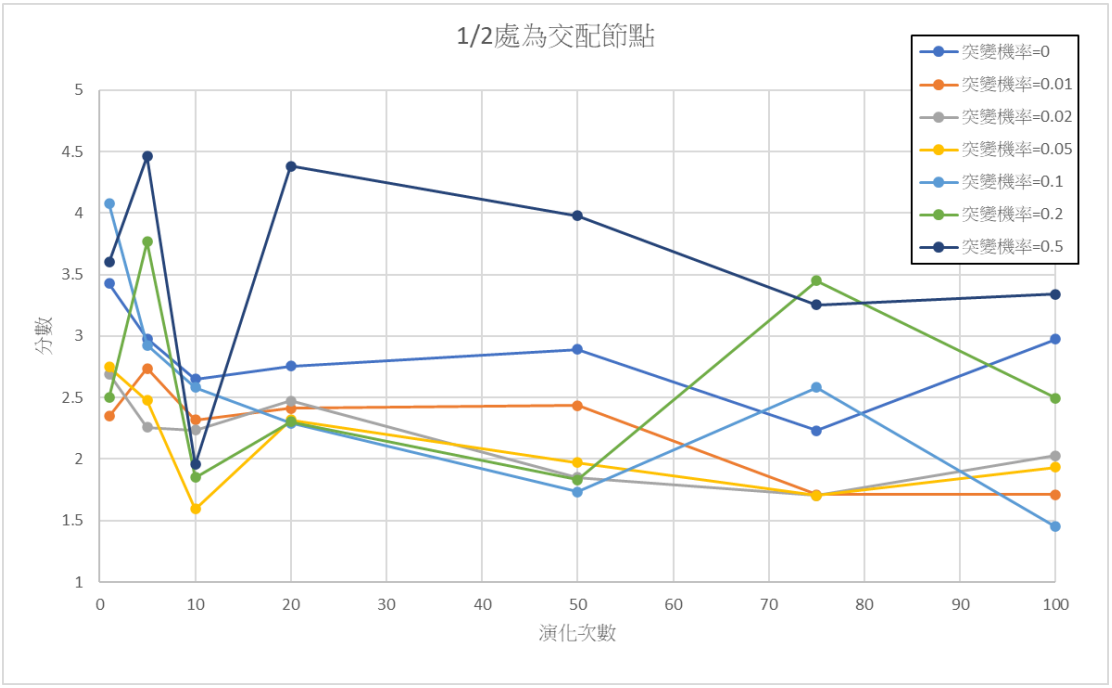
```
def evolution(pool, cut, prob):
    #變數設定
    evolu = []
    rank = []
    temp_path = []
```

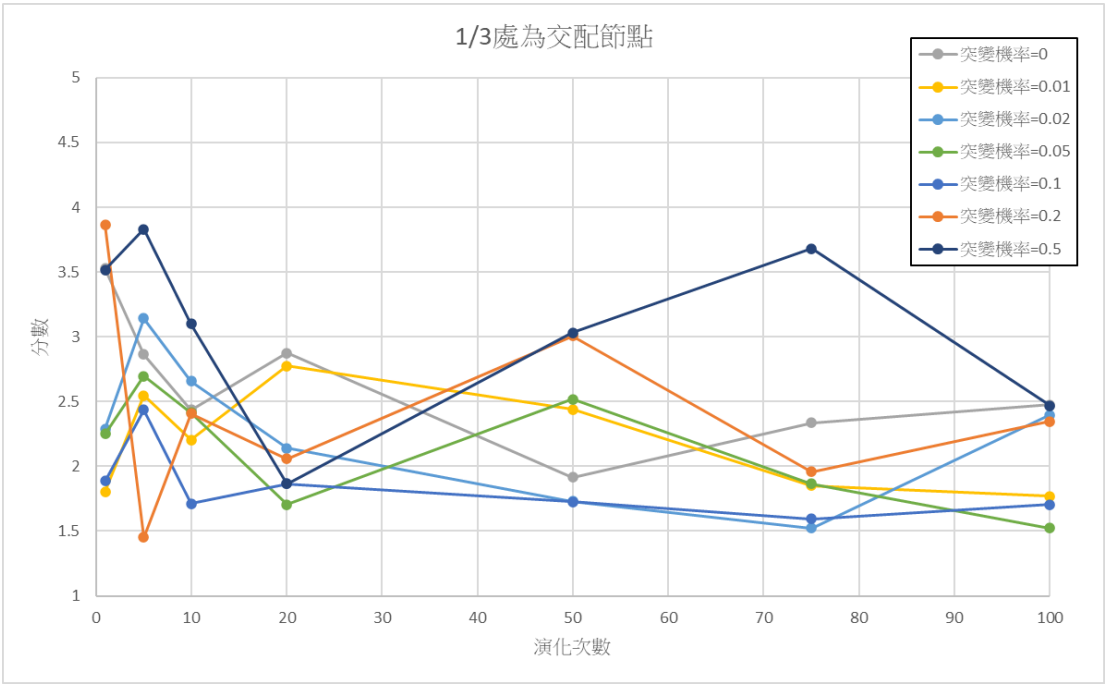
```
    #生成所有分數
    for i in range( len(genetic_pool) ):
        temp_path = path_gen(pool[i][:],len(pool[i][:]))
        rank.append( [score(temp_path),i] )

    #排序
    rank = mergeSort(rank)

    #pr前50時進行強勢交配
    chromo1 = pool[ rank[i][1] ][:]                    #強勢基因
    for i in range(len(genetic_pool)//2):
        chromo2 = pool[ rank[i+1][1] ][:]
        evolu.append(crossover(chromo1, chromo2, cut))
        evolu.append(crossover(chromo2, chromo1, cut))

    #突變
    for i in range(len(evolu)):
        for j in range(len(evolu[0][:])):
            if(probability(prob)):
                evolu[i][j] = random.randint(1,len(evolu[0][:])-j)

    return evolu
```

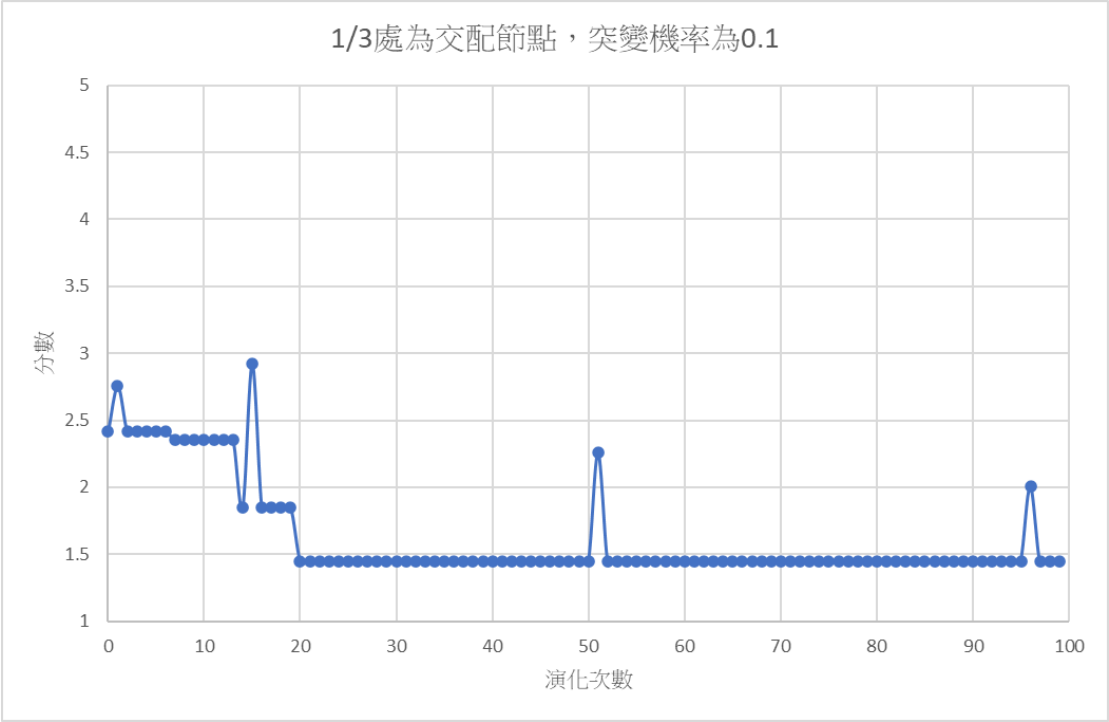
3. Observation in Genetic Algorithm

分別以交配節點、突變機率和演化次數為變量進行單次實驗並繪製成圖表，基因池大小固定為 8。由於程式設計之初沒有考慮到如果強勢基因比其子代更優秀，因此沒有設計強勢基因直接進入下一輪基因池的機制，因此雖然該演算法能在極短的嘗試後找到最佳解，但演算法無法順利收斂，不過因為時間上不允許，該機制就留給後進加入。

從圖表可以看出，從三分之一處進行基因交換的效果比從中間來的好，合理推測是因為從中間交換基因無法保留較多強勢基因的特徵，而從三分之一處則可保留較多優秀的特徵，而不是單純取平均。基本上，過大的突變機率或是過小的突變機率皆不利於最佳解的求得，前者使基因池的基因不容易穩定，後者則使基因容易陷入區域解中。


1/2處為交配節點
突變機率=0
突變機率=0.01
突變機率=0.02
突變機率=0.05
突變機率=0.1
突變機率=0.2
突變機率=0.5
分數
演化次數
5
4.5
4
3.5
3
2.5
2
1.5
1
0
10
20
30
40
50
60
70
80
90
100


1/3處為交配節點
突變機率=0
突變機率=0.01
突變機率=0.02
突變機率=0.05
突變機率=0.1
突變機率=0.2
突變機率=0.5
分數
演化次數
5
4.5
4
3.5
3
2.5
2
1.5
1
0
10
20
30
40
50
60
70
80
90
100



再來因為沒有考慮到強勢基因的保留，且圖表上每個點皆為獨立實驗，因此不容易看出演化次數和分數之間的關係，但在單次實驗中可以觀察到分數和演化次數呈現相關，圖表中的圖波是突變發生但分數變差的結果。

4. Difference of two algorithm and opinion about homework

以 n =7 的 input 為例，因為兩演算法皆使用相同的加密與解碼方式，在比較時間複雜度時可以排除加密與解碼的影響，使用 Brute Force Algorithm 的時間複雜度為 O(7!)，而 Genetic Algorithm 若設定演化次數為 30 次，則時間複雜度為 O(30*(8+8*log8))，明顯 Genetic Algorithm 的時間複雜度較 Brute Force Algorithm 小，且 Genetic Algorithm 的時間複雜度受到演化次數影響，同時也大程度受到基因池大小影響，基因池設計的越小，則計算需要的時間越少，但容易陷入區域最佳解當中。

這次作業最難的部分在於時間複雜度的控制，其中最重要的部分則是加密與解碼的演算法的設計，如果演算法設計不好，不僅程式無法順利求解，甚至時間複雜度會遠高於 O(n!)，但若不需要考慮時間複雜度與空間複雜度，這次的作業在繳交時間以及難度上皆設計得宜，我的作業在「強勢基因的保留」以及「基因池大小對基因演算法的影響」因為時間關係並沒有太多著墨，不過從程式設計的過程中以及作業中的分析，皆可知道這兩個因素都對基因演算法有著相當大的影響。希望我的作業能給未來修課的同學一個參考以及未來同學能接續我未完成的部分。