人工智慧導論 HW2 報告

資訊 113 F74094017 李昆翰

P1. Brute Force 的實作方法：

我的 Brute Force 的實現主要是：先得出除了起點和終點的必經節點的所有排列（permutation），例如說本次的演唱會要到 5 個地方，我會先將要成為出發地和終點站的 A 點先做排除，將中間的 B、C、D、E 四個區域的路徑做排列組合，得到 24 條路徑後。具體作法為如下兩截圖之程式碼。

```
37    # list the node without start and end points
38    unvisited_nodes = list(range(1, len(mat)))
39
40    # Generate every possible permutations between regions
41    # 參考：permutations = list(itertools.permutations(unvisited_nodes))
42    permutations = generate_permutations(unvisited_nodes)
```

圖一：排列函式的進入點

```
15  def generate_permutations(arr):
16      if len(arr) == 0:
17          return [[]]
18
19      permutations = []
20      for i in range(len(arr)):
21          first_elem = arr[i]
22          rest = arr[:i] + arr[i+1:]
23          for p in generate_permutations(rest):
24              permutations.append([first_elem] + p)
25
26      return permutations
```

圖二：排列函式

（此作法的借鑒對象為 python itertools library 中的 permutation 函式）

之後再將所有得到的組合嵌入起點和終點的中間去形成一條一條的路徑，最後得出路徑長度去做比較。具體作法為如下截圖之程式碼。

```
    # init. properties in shortest path
    shortest_path = [0] + unvisited_nodes + [0]
    shortest_distance = total_distance(mat, shortest_path)

    for perm in permutations:
        path = [0] + perm + [0]
        distance = total_distance(mat, path)
        if distance < shortest_distance:
            shortest_path = path
            shortest_distance = distance

    return shortest_path, shortest_distance
```

圖三：迭代式的比較誰最小

P2. simulated annealing 的實作方法：

我的 simulated annealing 的實現主要用到 python 的 random 和 math 函式庫，至於實作方式則和講義中的 pseudocode 很像，就是定義一個溫度（init_temp）和冷卻比例（cooling_rate），在經過每一次迭代後，當溫度不大於 1 時，就結束 simulated annealing。

而在每一次迭代中，我會先做一次的隨機變換。我的隨機變換的作法是使用 random 函式隨機出兩個數字表示要互相交換的元素，之後將那兩個元素做交換。具體作法為以下程式碼：

```
def random_move(curr_path):

    # copy a new path to swap elements
    new_path = curr_path.copy()

    # random two elements to swap
    i = random.randint(1, len(curr_path) - 2)
    j = random.randint(1, len(curr_path) - 2)
    while i == j:
        i = random.randint(1, len(curr_path) - 2)
        j = random.randint(1, len(curr_path) - 2)

    # swap
    new_path[i], new_path[j] = new_path[j], new_path[i]

    return new_path
```

圖四：隨機變換函式

在隨機出新的路徑後，我將現存迭代的路徑（curr_solution）和新的路徑（new_solution）的長度去做相減，得到一個 delta 值，並在之後用一個 if 條件式，判斷若 delta 值大於零，或 delta 值跟現有溫度的相除的自然對數大於我們隨機出來的一個數字，就將新的路徑取代現存迭代的路徑。最後，若現有迭代路徑被更新了，就將其和目前的最佳路徑做長度對比，如果表現更好就取代掉。具體如下。

```
58      while temperature > 1:
59          # random swap
60          new_solution = random_move(curr_solution)
61          new_energy = total_distance(mat, new_solution)
62          # getting energy delta
63          energy_delta = curr_energy - new_energy
64
65          # to know if we need to substitute current solution
66          if energy_delta > 0 or math.exp(energy_delta / temperature) > random.random():
67              curr_solution = new_solution
68              curr_energy = new_energy
69
70              # check if the new solution is better than the current best ones
71              if curr_energy > best_energy:
72                  best_solution = curr_solution
73                  best_energy = curr_energy
74
75          # cool down
76          temperature *= cooling_rate
```

圖五：simulated annealing 的全部過程

P3. 從 P2 的實作中發現的事：

在接下來，我將針對 P2 的方法做實驗。本次的實驗中，我將輸入的 distance matrix 和 initial path 固定為以下：

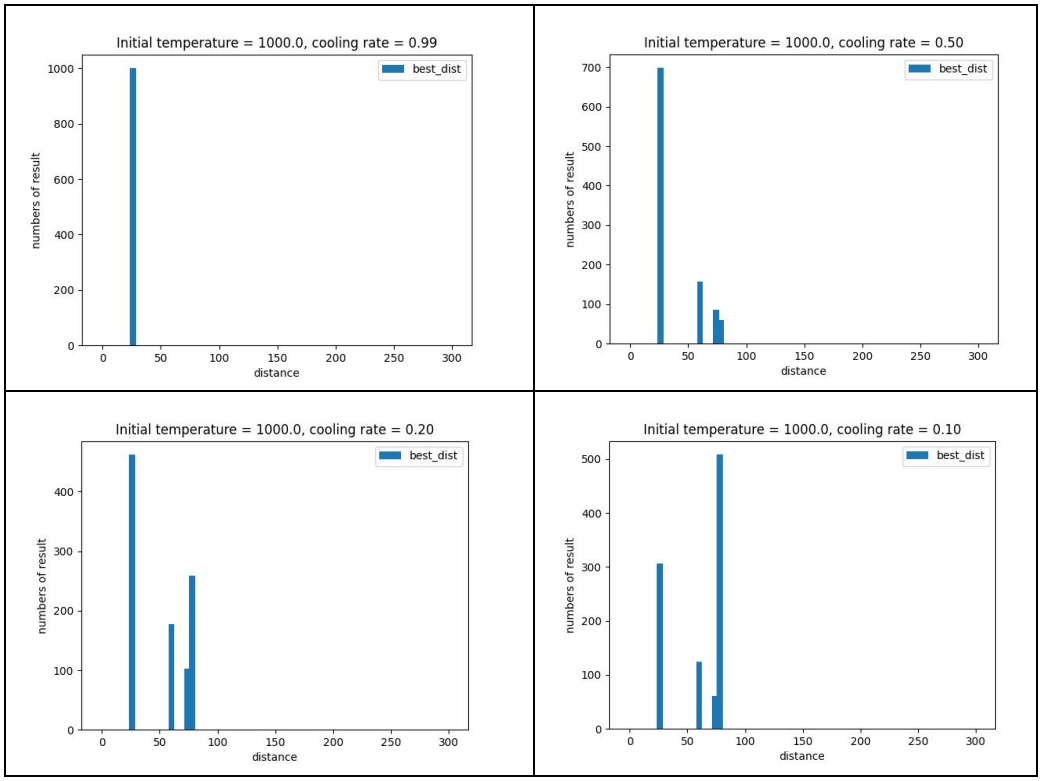
```
108  mat =  [# A, B, C, D, E
109             [ 0, 1, 9, 8, 40], # A
110             [ 1, 0, 2, 35, 50], # B
111             [ 9, 2, 0, 30, 10], # C
112             [ 8, 35, 30, 0, 5], # D
113             [40, 50, 10, 5, 0], # E
114             ] # be seen as an adjacency matrix
115  init_path = [0, 4, 3, 2, 1, 0]
```

圖六：實驗的控制變因

換句話說，我的實驗只會著重在變換溫度以及 cooling rate 而已。主要的原因是覺得 simulated annealing 的重點在於溫度和 cooling rate 的參數變動帶動隨機次數的差別，而非 distance matrix 和 initial path。

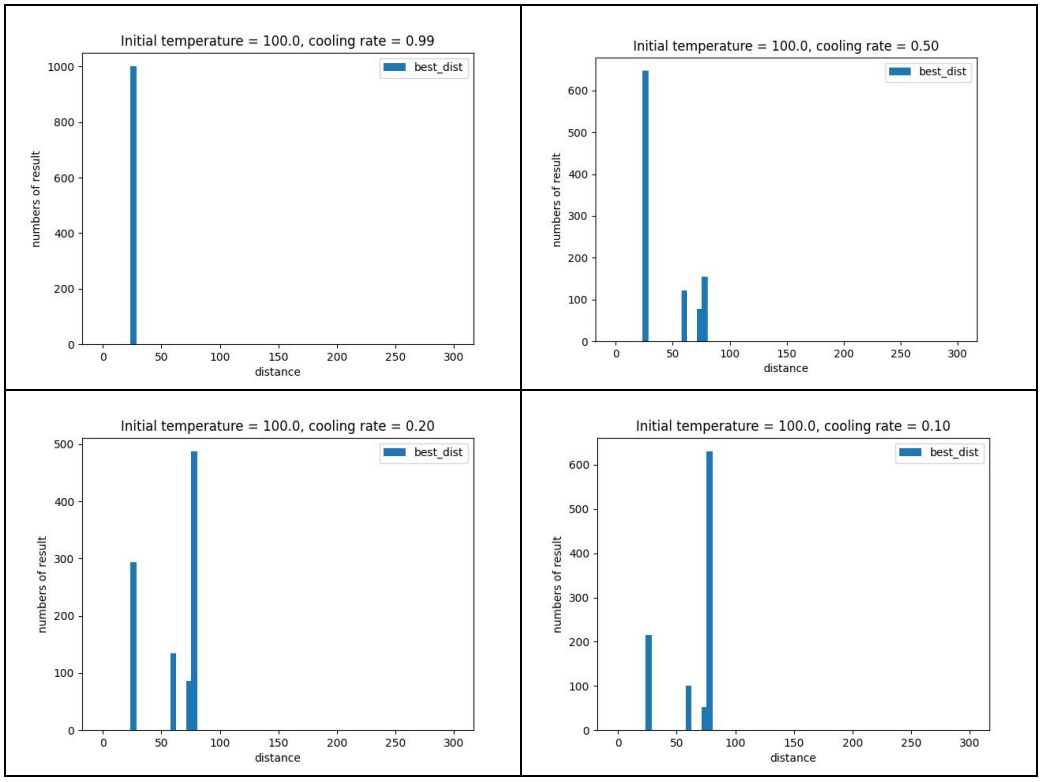
在本次的實驗中，我總共做了兩組實驗。這兩組實驗的溫度分別是 1000.0 和 100.0 度，且各自會和指定的 cooling rate = 0.99、0.50、0.20、0.10 各做 1000 次迭代。最後蒐集來的資料我使用 python matplotlib 將其視覺化。下面的部分，我將針對我使用 python matplotlib 所 plot 出來的兩組 bar chart 圖組做分析和解釋。

首先是第一組實驗，使用的參數是溫度 1000.0 度、cooling rate = 0.99、0.50、0.20、0.10。結果為以下圖組：



從這組圖組可以發現到，當 cooling rate 很大的時候，在 1000 次的採樣下，基本上得到的 best_path 長度是 26。但是，在 cooling rate 逐漸變小的時候，simulated annealing 得到的結果就開始變得不固定。到 cooling rate = 0.2 的時候，採樣到的錯誤樣本已經比正確的樣本要來的多，甚至到 cooling rate = 0.1 時，有一組錯誤樣本數已經比超越正確樣本的數量。

接下來，是第二組實驗，使用的參數是溫度 100.0 度、cooling rate = 0.99、0.50、0.20、0.10。結果為以下圖組：



從第二份圖組可以觀察到，當 cooling rate 很大的時候，採樣到的數據依舊都是正確的數據。不過，當 cooling rate 變小時，如果我們和溫度 1000.0 度的實驗圖組做比較的話，溫度 100.0 度的正確樣本數少了不少。除此之外，在 cooling rate = 0.20 時，溫度 100.0 度圖組中的一組錯誤樣本數已經超越了正確樣本數，比起溫度 1000.0 度圖組來說要快的非常多。

P4. 比較兩者優缺點和實作的心得：

從 P3 的結果，和我於 P1 實作中，我們可以說：由於 simulated annealing 具有隨機的成分，因此，當我們未給與其足夠多的時間做 cool down 的話，那此方法會根據隨機的多寡，而呈現出各式不同的不穩定樣本組。雖然說若我們將 cool down 時間縮短，它可以比 Brute Force 來的快。然而，若我們只專注在我實驗中如此大小 distance matrix 的話，可能使用 Brute Force 會比 simulated annealing 來的穩定且好。但相反的，若 distance matrix 很大，使得我的 Brute Force 方案要求得的 permutation 的次數大於 920 或以上的話，那若我們用溫度 1000.0 度和 cooling rate = 0.99 的 simulated annealing 則會比 Brute Force 來的快，且因為 cooling rate 很大的關係，錯誤樣本數也不至於到太多，或運氣好點可能也不太會遇到。

從本次的作業中，藉由實作我記住了 simulated annealing 的大致操作方式，也從實驗中得知光是將 cooling rate 砍了快一半，所得到的樣本就開始有不少的錯誤樣本數，凸顯出隨機的不穩定。不過這讓我有隱約在其中感覺到一點機器學習中收斂的概念，藉由演算法不斷的推進下，在每一次的迭代中從錯誤樣本和正確樣本的比較，調整演算的方式，使模型不斷地向著標的前行。在這其中，模型的迭代次數也和最終的成果的品質有所關連，某種程度而言就和 simulated annealing 一樣，當迭代（cool down）的次數很少，就比較沒辦法得到我們要的最佳路徑。