

## 程式介面說明

在ubuntu中執行執行檔，會印出執行結果。

E-C-F-B-A-I-D-G-H-J Path Length: 106.0

每一次執行結果皆不同。

## 程式碼說明

利用python進行實作，所有程式都在SA\_TSP.py。

`def readfile():`

讀取TSP.txt檔案，並將檔案轉成array，讓程式比較好處理。

`def generate_new_state(S):`

產生新的路徑，方法為隨機交換兩個城市。

`def calculate_distance(S):`

計算路徑所要花的距離，使用for迴圈迭代計算。

`__main__`

實作退火演算法，主要架構下面會說明。

## Simulated Annealing演算法說明



Procedure TSPSA:

begin

init-of-T; { T為初始溫度}

S={1, ....., n}; {S為初始值}

termination=false;

while termination=false

begin

for i=1 to L do

begin

generate(S' form S); { 從當前迴路S產生新迴路S'}

$\Delta t := f(S') - f(S)$ ; {f(S)為路徑總長}

IF ( $\Delta t < 0$ ) OR ( $\text{EXP}(-\Delta t/T) > \text{Random-of-}[0, 1]$ )

S=S';

IF the-halt-condition-is-TRUE THEN

termination=true;

End;

T\_lower;

End;

End

模擬退火演算法新解的產生和接受可分為如下四個步驟：

- 第一步是由一個產生函數從當前解產生一個位於解空間的新解；為便於後續的計算和接受，減少演算法耗時，通常選擇由當前新解經過簡單地變換即可產生新解的方法，如對構成新解的全部或部分元素進行置換、互換等，注意到產生新解的變換方法決定了當前新解的鄰域結構，因而對冷卻速度表的選取有一定的影響。
- 第二步是計算與新解所對應的目標函數差，因為目標函數差僅在變換部分產生，所以目標函數差的計算最好按增量計算。事實表明，對大多數應用而言，這是計算目標函數差的最佳方法。
- 第三步是判斷新解是否被接受，判斷的依據是一個接受準則。最常用的接受準則是Metropolis準則：若 $\Delta f' < 0$ 則接受 $S'$ 作為新的當前解 $S$ ，否則以概率 $\exp(-\Delta f'/T)$ 接受 $S'$ 作為新的當前解 $S$ 。
- 第四步是當新解被確定接受時，用新解代替當前解，這隻需將當前解中對應於產生新解時的變換部分予以實現，同時修正目標函數值即可。此時，當前解實現了一次迭代，可在此基礎上開始下一輪試驗，而當新解被判定為接受時，則在原當前解的基礎上繼續下一輪試驗。

## 執行結果

執行的路徑結果大部分的數值都在100~200之間。

**A-H-E-J-B-D-G-I-C-F Path Length: 85.0**

作業四因為網路上有比較多的資源、不會有太多的複雜計算，所以實作起來比之前作業簡單。

## 參考資料

<http://wiki.mbalib.com/zh-tw/>

<http://wiki.mbalib.com/zh-tw/%E6%A8%A1%E6%8B%9F%E9%80%80%E7%81%AB%E7%AE%97%E6%B3%95>

<https://ithelp.ithome.com.tw/articles/10186172>

<https://www.cnblogs.com/heaad/archive/2010/12/20/1911614.html>

<http://wiki.mbalib.com/zh-tw/>

<http://wiki.mbalib.com/zh-tw/%E6%A8%A1%E6%8B%9F%E9%80%80%E7%81%AB%E7%AE%97%E6%B3%95>