Megan Tan, A20527707 <- replace with your name

**CS 581 Spring 2024 Programming Assignment #01**

Due: **Sunday, March 3, 2024, 11:59 PM CST**

Points: **100**

**Instructions:**

1. Place **all your deliverables (as described below) into a single ZIP** file named:

LastName\_FirstName\_CS581\_Programming01.zip

1. Submit it to Blackboard Assignments section before the due date. **No late submissions will be accepted**.

**Objectives:**

1. (100 points) Implement and evaluate two search algorithms.

**Input data file:**

Your input file will be a CSV (comma separated values) file (see Programming Assignment #01 folder in Blackboard – campus.csv).

You **CANNOT** modify nor rename input data files.

Each row in that input file will correspond to STATE information: STATE\_LABEL, and state 2D Cartesian space coordinates: X and Y. You can assume X and Y to be positive integers.

Input (four states in this case, but there will be more) file (text file) format:

A,XA,YA

B,XB,YB

C,XC,YC

D,XD,YD

Where:

* + A, B, C, D are state LABELS,
  + XI, YI are state coordinates.

**Deliverables:**

Your submission should include:

* Python code file(s). Your py file should be named:

cs581\_P01\_AXXXXXXXX.py

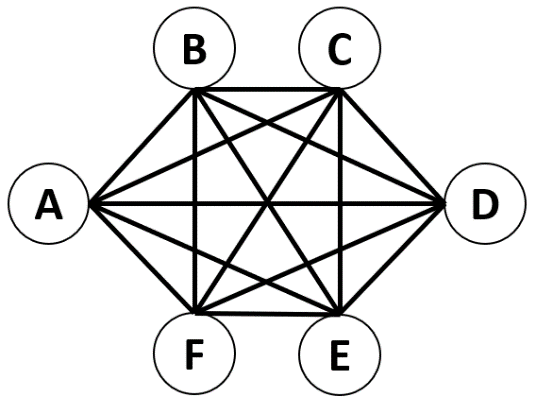
where AXXXXXXXX is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.

* this document with your results and conclusions. You should rename it to:

LastName\_FirstName\_CS581\_Programming01.doc or .pdf

**Problem description:**

You are given a weighted complete graph G (example shown on Fig. 1). Your task is to solve a Traveling Salesman Problem [TSP] (find minimum cost/weight Hamiltonian Cycle) on G using algorithms specified below.



*Figure 1: Sample complete (fully connected, weights NOT shown) graph G.*

Assume that edge weights represent **straight line distances between states**.

Your task is to implement two search algorithms in Python:

* Simulated Annealing, and
* Genetic Algorithm,

and apply them to solve the TSP (**starting at Stuart Building [SB] node**) problem using provided data.

Your program should:

* Accept four (4) command line arguments corresponding to two states / state capitals (initial and goal states) so your code could be executed with

python cs581\_P01\_AXXXXXXXX.py FILENAME ALGO P1 P2

where:

* + cs581\_P01\_AXXXXXXXX.py is your python code file name,
  + FILENAME is the input CSV file name (graph G data),
  + ALGO is mode in which your program should operate
    - 1 – Simulated Annealing,
    - 2 – Genetic Algorithm,
  + P1 is a value for a specific algorithm parameter:
    - Simulated Annealing: P1 is the initial temperature T value,
    - Genetic Algorithm: P1 is the number of iterations K,
  + P2 is a value for a specific algorithm parameter:
    - Simulated Annealing: P2 is the a parameter for the temperature cooling schedule,
    - Genetic Algorithm: P2 is the mutation probability Pm value,

Example:

python cs581\_P01\_A11111111.py DATA.CSV 2 1000 0.01

If the number of arguments provided is NOT four your program should display the following error message:

ERROR: Not enough or too many input arguments.

and exit.

* Load and process input data file provided (assume that input data file is ALWAYS in the same folder as your code - this is REQUIRED! DO NOT HARDCODE YOUR LOCAL FILE PATH). Make sure your program is **flexible enough to accommodate different input data set** (with a different [size, nodes, edges, etc.] graph, but structurally the same). Your submission will be tested using a different file!
* Run Simulated Annealing and Genetic Algorithm searches to find the Traveling Salesman Problem solution starting at INITIAL (first state/line in the input CSV file) state and measure execution time (in seconds) for both methods.
* Report results on screen in the following format:

Last Name, First Name, AXXXXXXXX solution:

Initial state: INITIAL

Simulated Annealing:

Command Line Parameters: <list your parameters here>

Initial solution: LABEL1, LABEL2, LABEL3, …, LABELN-1, LABELN

Final solution: LABEL1, LABEL2, LABEL3, …, LABELN-1, LABELN

Number of iterations: AAAA

Execution time: T1 seconds

Complete path cost: Y1

Genetic Algorithm:

Command Line Parameters: <list parameters here>

Initial solution: LABEL1, LABEL2, LABEL3, …, LABELN-1, LABELN

Final solution: LABEL1, LABEL2, LABEL3, …, LABELN-1, LABELN

Number of iterations: AAAA

Execution time: T2 seconds

Complete path cost: Y2

where:

* + AXXXXXXXX is your IIT A number,
  + START\_NODE is the label/name of the initial graph node,
  + AAAA is the number of iterations completed before termination,
  + LABEL1, LABEL2, LABEL3, …, LABELN-1, LABELN is a solution represented as a list of visited states (with LABEL1 = START\_NODE and LABELN = INITIAL),

* Save your solutions to a file named:
* Simulated Annealing: INPUTFILENAME\_SOLUTION\_SA.csv file.
* Genetic Algorithm: INPUTFILENAME\_SOLUTION\_GA.csv file.

Where: INPUTFILENAME is the input file name WITHOUT extension (for example for input file DATA2.CSV, the GA solution file should be named DATA2\_SOLUTION\_GA.csv.

* Solution file (text file) format:

XXXX

STATE1

STATE2

STATE3

STATEN

Where:

* + XXXX will be the final path / solution cost
  + STATE1, STATE2, …, STATEN state LABELS (one per line, names as in the input file) represent solution path (the one displayed on screen; here assumed to be of length 4) in order.

**Algorithm Parameters [SPECIFY WHAT IS NOT GIVEN BELOW]:**

Simulated Annealing parameters are:

* Initial state: pick one at random
* Initial temperature: T = P1
* What is a move? 2-edge swap
* Termination condition: Temperature T > 0.
* Temperature cooling schedule: exponential (P2 = a provided as command line argument)

Ti = TINITIAL \* e-i \* a

* Cost / objective function:

Total cost of path (should be minimized)

………………………………………………………………………………………………………………………………….

Genetic Algorithm parameters are:

* Individual representation (it does not need to be binary):

Array denoting states. The size of the array will be the number of states, with numbers from 0 to the number of states denoting the path of the city. The number represents that state by the index in the given file.

………………………………………………………………………………………………………………………………….

* Population size N = 10 (Note: Population size is relatively small due to really bad performance at large p1 and p2 values; elitism is used to carry the best fitness individual to each new generation)
* Fitness function:

Total cost of path (should be minimized)

………………………………………………………………………………………………………………………………….

* Selection mechanism: Roulette Wheel, One elite is carried over to new generation every generation
* Probability of crossover: Pc = 1,
* Crossover mechanism: 2-point crossover with crossover points 0.25 \* size of array (rounded down) and 0.5 \* size of array (rounded up). Eg array of size 10 will have cross over points 3 and 7.
* Probability of mutation: Pm = P2,
* Termination condition: number of iterations > P1 = K.

**Results and Conclusions**

Assume that the INITIAL state is the **first state in the CSV file**. Run both algorithms:

* with three different values of P2 for both algorithms,
* with number of iterations K / T: 100, 1000, 10000 [repeat each 5 times and find average min, max, and average path cost and search times] for every P2 value,
* and for every ALGO-P1-P2 value combination provide ONE min/average/max fitness function plot (one plot with three traces; add labels and legend). Pick a run that was interesting or best and explain your choice.

Report your findings in Table A below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TABLE A: Results comparison** | | | | | |
| Algo | P1 | P2 | Min/Max/Average Path cost | Min/Max/Average search time in seconds | Notes/comments |
| SA | 100 | 0.1 |  |  |  |
| SA | 1000 | 0.5 |  |  |  |
| SA | 10000 | 0.9 |  |  |  |
| SA | 100 | 0.1 |  |  |  |
| SA | 1000 | 0.5 |  |  |  |
| SA | 10000 | 0.9 |  |  |  |
| SA | 100 | 0.1 |  |  |  |
| SA | 1000 | 0.5 |  |  |  |
| SA | 10000 | 0.9 |  |  |  |
| GA | 100 | 0.1 |  |  |  |
| GA | 1000 | 0.5 |  |  |  |
| GA | 10000 | 0.9 |  |  |  |
| GA | 100 | 0.1 |  |  |  |
| GA | 1000 | 0.5 |  |  |  |
| GA | 10000 | 0.9 |  |  |  |
| GA | 100 | 0.1 |  |  |  |
| GA | 1000 | 0.5 |  |  |  |
| GA | 10000 | 0.9 |  |  |  |

Plots:

What are your conclusions? What have you observed? Which algorithm/parameter set performed better? Was the optimal path found? Write a summary below.

|  |
| --- |
| **Conclusions** |
|  |