Matthew Muranaka

Objectives:

1. Implement and evaluate two search algorithms (genetic algorithm and simulated annealing).

Input data file:

The input file is a CSV (comma separated values) file (`campus.csv`).

Each row in the input file will correspond to STATE information: `STATE_LABEL`, and state 2D Cartesian space coordinates: `X` and `Y`. I 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:

- Python code file(s) named:

`tester.py`

- This document with observations and conclusions. You should rename it to:

`Travelling_Salesman_Problem_Write_Up.pdf`

Problem description:

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

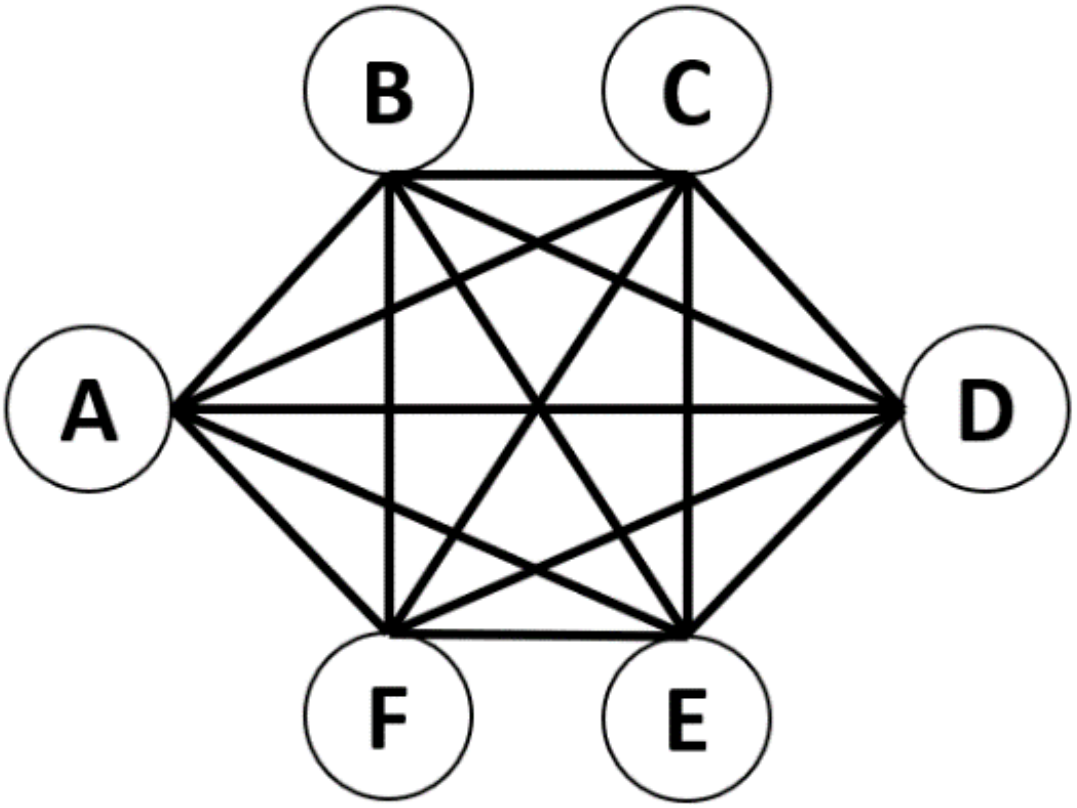


Figure 1: Sample complete (fully connected, weights NOT shown) graph G.

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

My task is to implement two search algorithms in Python:

- Simulated Annealing
- Genetic Algorithm

and apply them to solve the TSP (**[SB] node**) problem using provided data.
The program should:

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

  ```
  tester.py FILENAME ALGO P1 P2
  ```

  where:
  - `tester.py` is your python code file name,
  - `FILENAME` is the input CSV file name (graph G data),
  - `ALGO` is mode in which the 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 α parameter for the temperature cooling schedule,
  - Genetic Algorithm: `P2` is the mutation probability P_m value,

Example:

```
tester.py DATA.CSV 2 1000 0.01
```

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

```
ERROR: Not enough or too many input arguments.
```

and exit.

- Load and process input data file provided. Ensure the program is flexible enough to accommodate different input data sets (with a different [size, nodes, edges, etc.] graph, but structurally the same).
- 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:

```
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:

- 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),

- Saved 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:

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 = α provided as command line argument)

$$T_i = T_{INITIAL} * e^{-i * \alpha}$$

- Cost / objective function:

the cost function is simply the distance traveled

Genetic Algorithm parameters are:

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

My representation is just the states in an array (the chromosome is the route, the genes are the states)

Also note that I implemented a form of elitism where 2/10 of the population are elitists and half of that undergoes only mutation (not up for selection). I implemented this unique form of elitism because I wanted to combat the many horrible solutions that would be probably given by the crossover mechanism.

- Population size N = **population size of 100**,
- Fitness function:

fitness function is the total cost of the route (distance traveled)

- Selection mechanism: Roulette Wheel,
- Probability of crossover: $P_c = 1$,
- Crossover mechanism: 2-point crossover with crossover points
    - **the points are selected at random, invalid routes are squared to make them less likely to be selected**
- Probability of mutation: P_m = P1,
- Termination condition: number of iterations > P2 = 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.

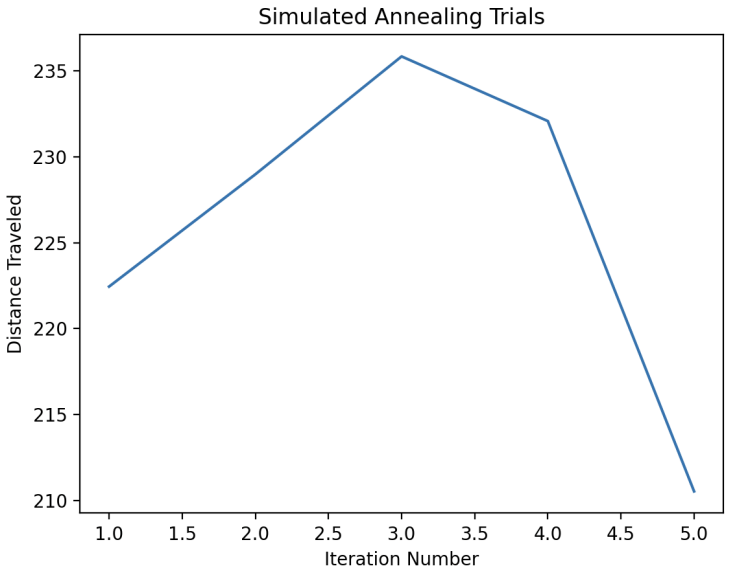
Report your findings in Table A below.

TABLE A: Results comparison

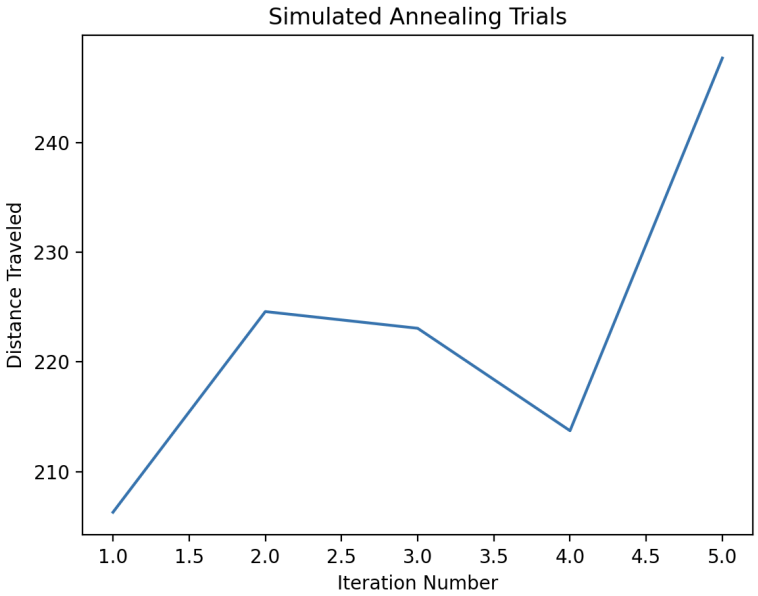
Algo	P1	P2	Min Path cost	Max Path cost	Average Path cost	Min search time in seconds	Max search time in seconds	Average search time in seconds	Notes/comments
SA	100	0.1	211	235	235	0.0124	0.0126	0.0125	
SA	1000	0.1	206	248	248	0.0131	0.0299	0.019	
SA	10000	0.1	203	235	235	0.0131	0.0305	0.0192	
SA	100	0.01	208	245	245	0.0389	0.0715	0.0461	
SA	1000	0.01	214	246	246	0.0388	0.0701	0.0458	
SA	10000	0.01	204	234	234	0.0391	0.0708	0.0461	
SA	100	0.001	210	239	239	0.1225	0.1583	0.1299	
SA	1000	0.001	191	253	253	0.1228	0.1586	0.1303	
SA	10000	0.001	192	231	231	0.1231	0.1583	0.1303	
GA	100	0.1	180	202	202	0.282	0.29	0.2852	
GA	1000	0.1	182	202	202	4.3398	4.3916	4.3666	
GA	10000	0.1	189	202	202	197.5562	198.2863	197.7666	
GA	100	0.01	198	201	201	0.2762	0.3136	0.2879	
GA	1000	0.01	194	204	204	4.3602	4.4129	4.3942	
GA	10000	0.01	195	201	201	197.6759	197.884	197.7911	
GA	100	0.001	193	202	202	0.2788	0.3191	0.2887	
GA	1000	0.001	197	201	201	4.3292	4.3996	4.3657	
GA	10000	0.001	188	202	202	196.104	197.7022	197.0764	

Plots:

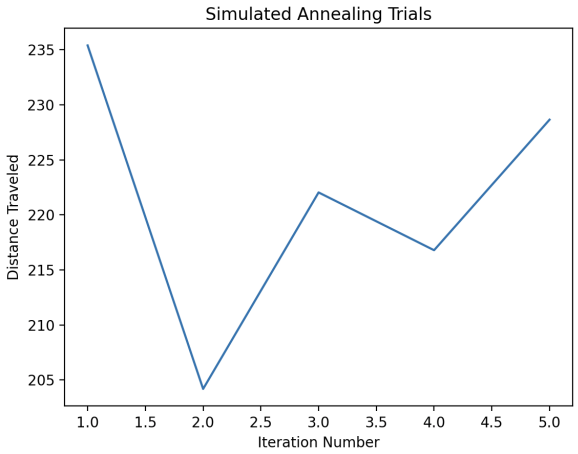
SA, 100, 0.1



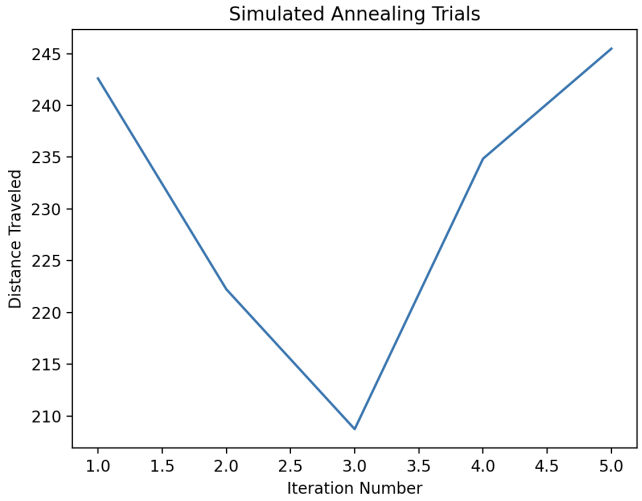
SA, 1000, 0.1



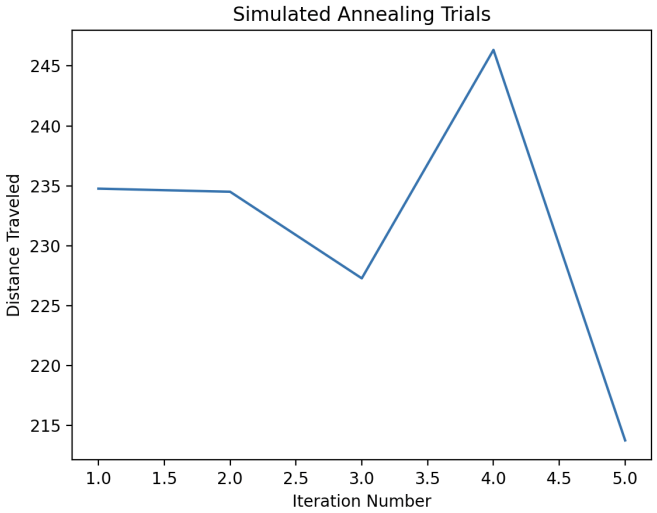
SA, 10000, 0.1



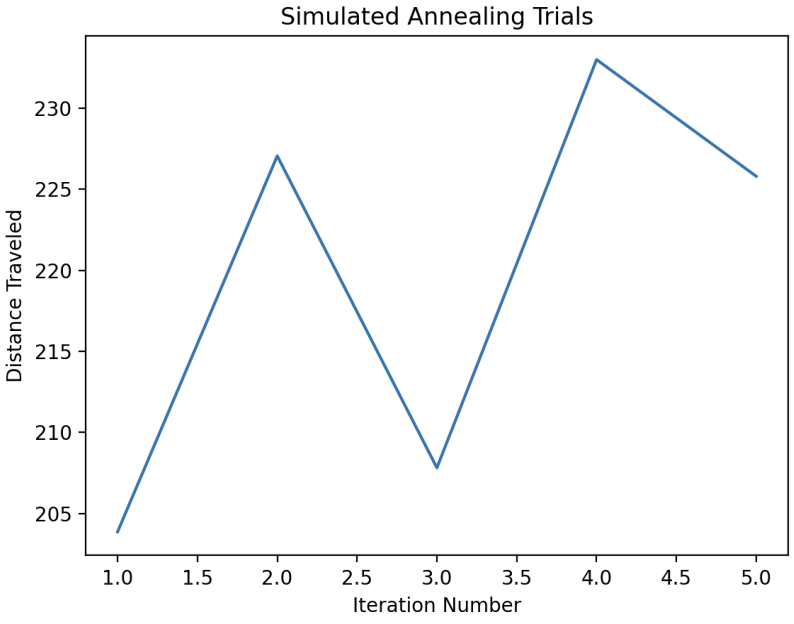
SA, 100, 0.01



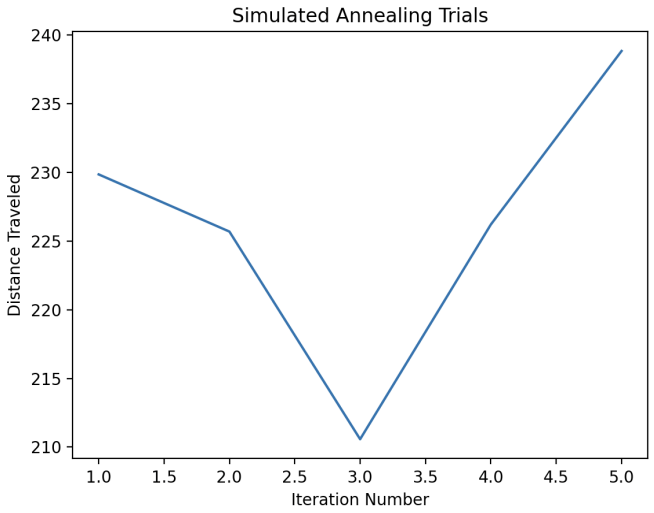
SA, 1000, 0.01



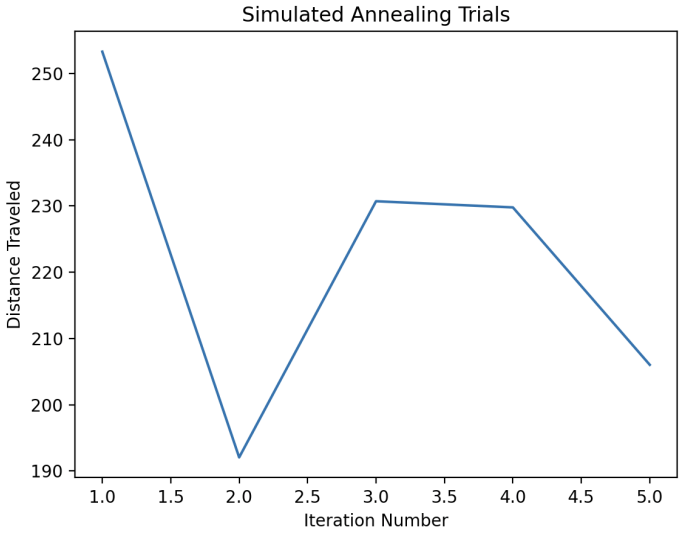
SA, 10000, 0.01



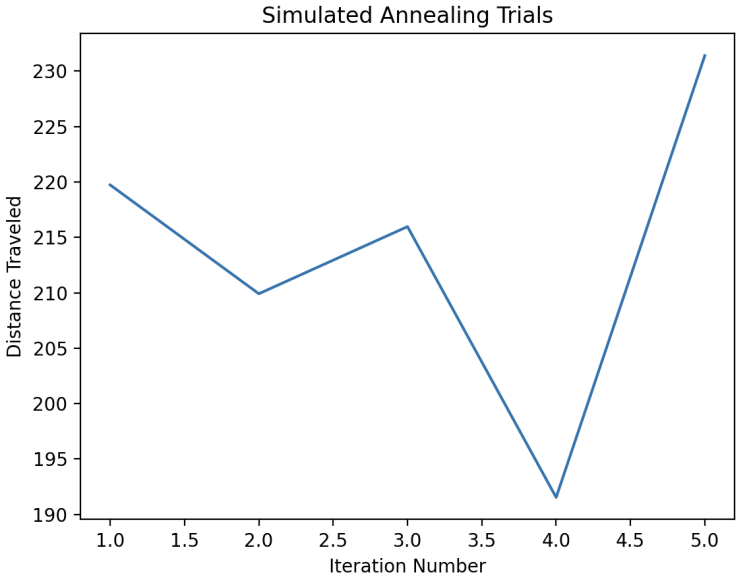
SA, 100, 0.001



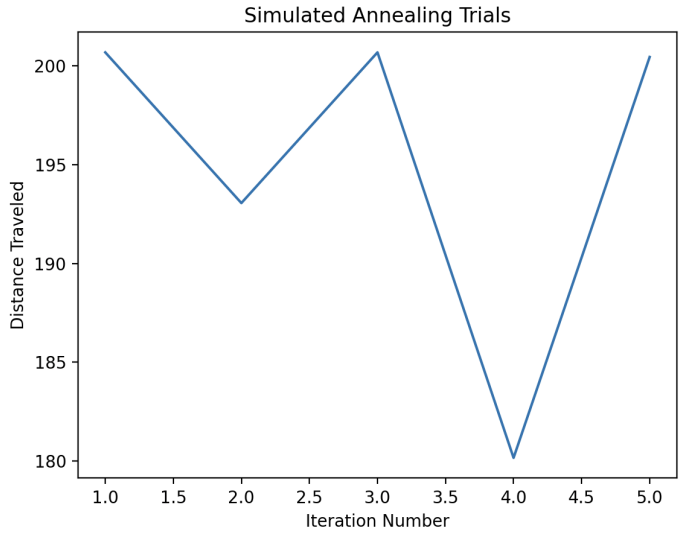
SA, 10000, 0.001



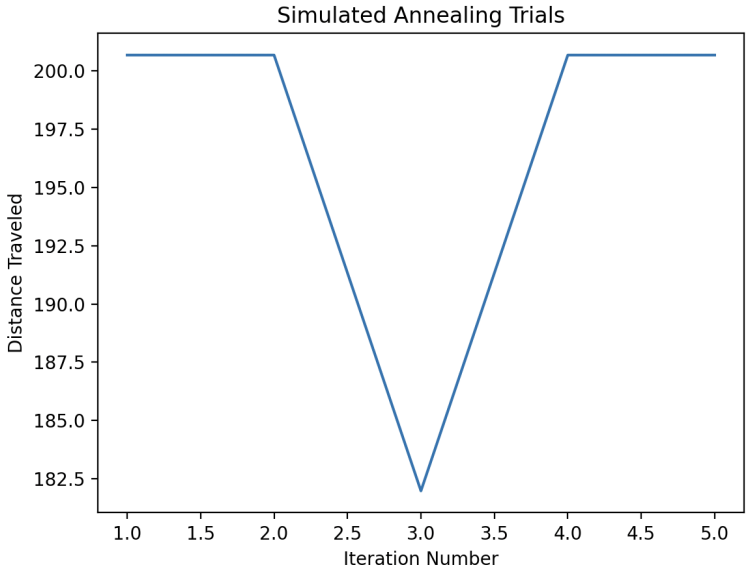
SA, 10000, 0.001



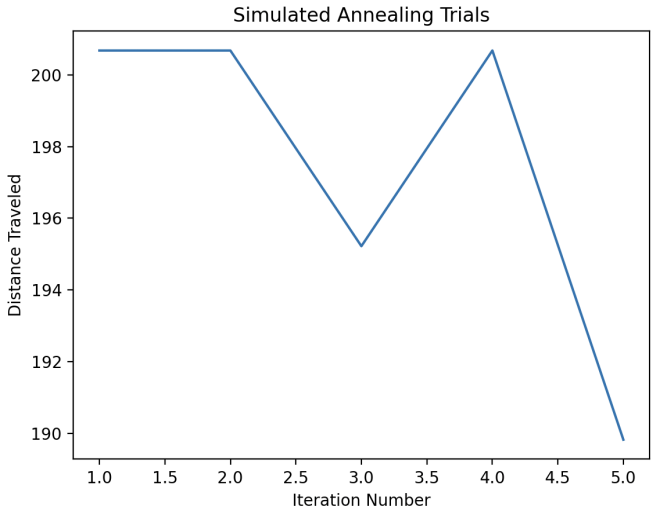
GA,100,0.1



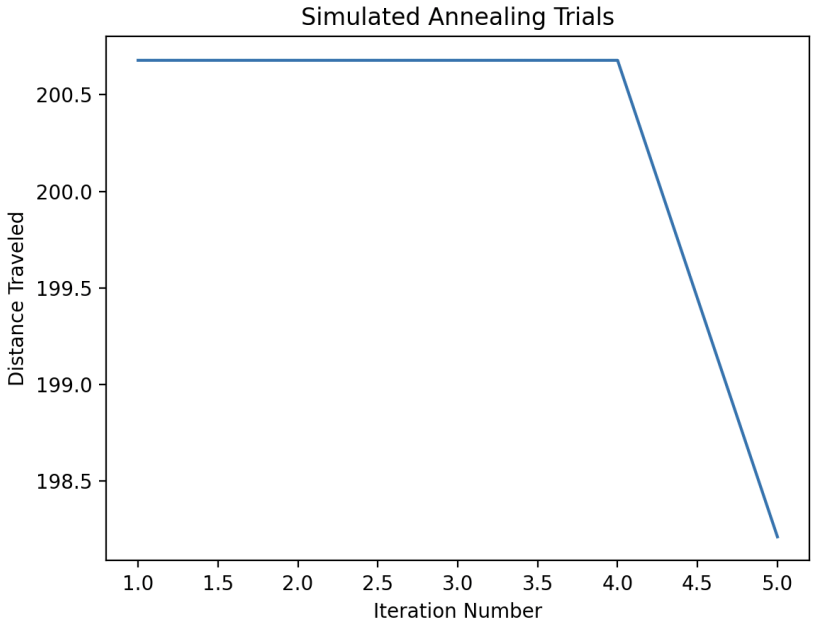
GA, 1000, 0.1



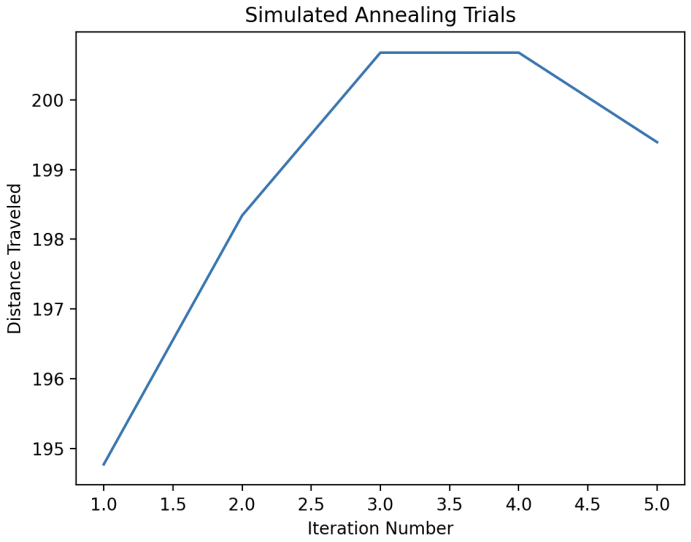
GA, 10000, 0.1



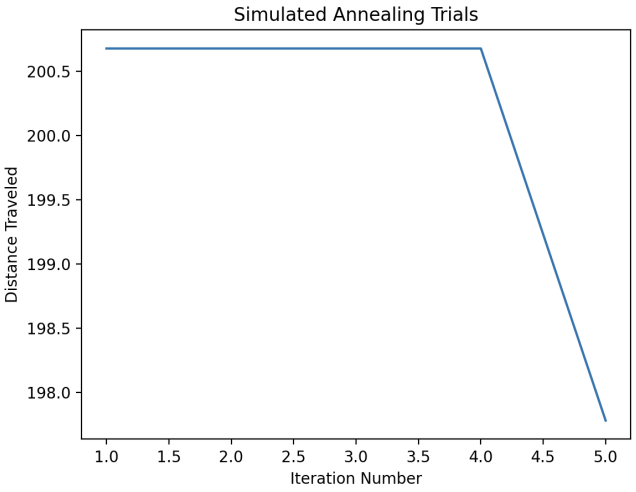
GA, 100, 0.01



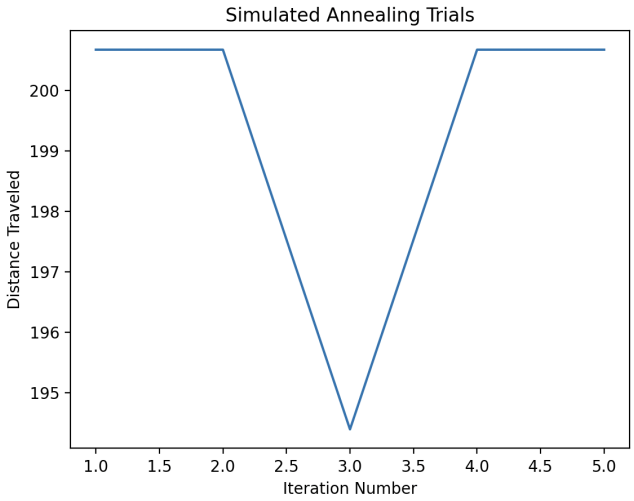
GA, 1000, 0.01



GA, 10000, 0.01

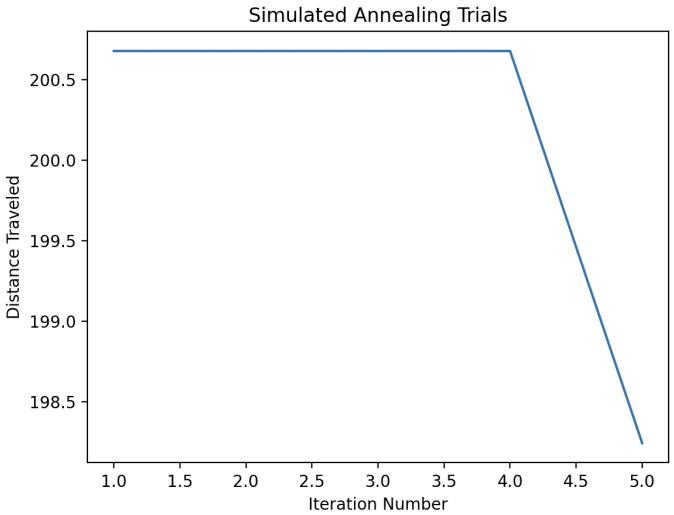


GA, 100, 0.001

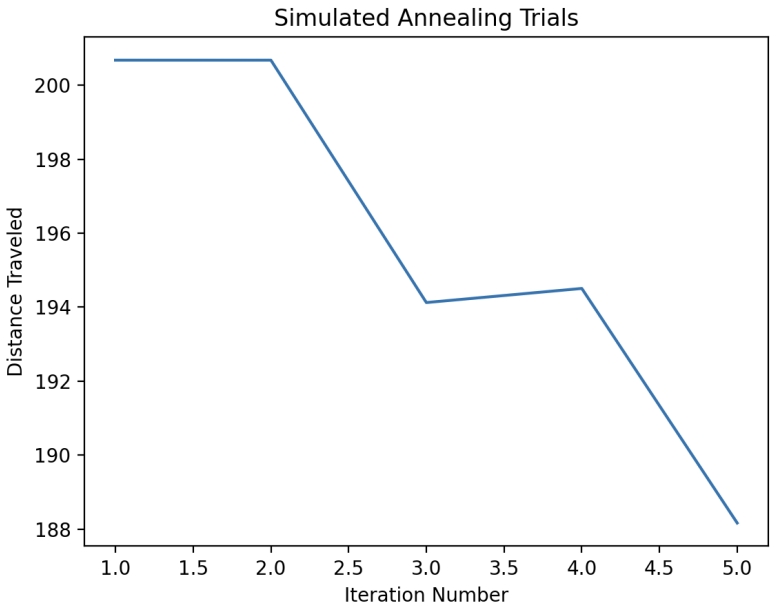


not sure why these numbers got messed up but you can see them in the table

GA, 1000, 0.001



GA, 10000, 0.001



Honestly, probably the coolest was the last genetic algorithm plot (10000, 0.001). I think it's cool because it improves with each iteration. I find this strange because the graph should not have any relation.

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

Conclusions

I think the biggest surprise was that the genetic algorithm produced better results than the simulated annealing algorithm despite the crossover mechanism. When discussing with friends concerning my simulated annealing algorithm, I am unsure of my problem because my implementation has the same affect but produces much worse results. Possibly a grader could look over my code and inform me of my error.

One thing I observed, given my genetic algorithm implementation is that oftentimes, the max total cost will be around the same amount. I assume this is the initial cost and this is a result of the crossover mechanism not producing any valid children and the elitist staying in the population. I suppose a fix to this would involve changing the elitist mechanism (if I couldn't change crossover).

The optimal path was not found. For neither SA nor GA, my solutions were nowhere near the optimal path. I have looked extensively over my code and am a bit bummed because I believe I have my desired structure.

Another conclusion I have, made when plotting a fitness vs iterations graph for the simulated annealing algorithm, is that often a global minima (mistaken as a local minima) gets passed over. The plot would have lows early in the graph with the solution resulting in a higher (but local minima) fitness than other solutions. I think a way to fix this is implementing some form of random-restart simulated annealing.