Project 3

Cap 4630

Dr. Marques

Andrew Donate

**Traveling Salesman Problem: Baseline Solution using Ant Colony Optimization**

Prologue

The following document describes of Project 3 for the Intro to Artificial Intelligence course (CAP

4630). This project was done entirely by Andrew Donate as the “architect”, “developer”, and

“reporter”. This project does take inspiration from another project but is sourced down below in the references section.

Introduction

In this Python program, the objective was to develop an AI solution using ACO (Ant Colony Optimization) to efficiently solve the TSP (Traveling Salesman Problem). By simulating the behavior of ants and the pheromone trails they create, the program aims to iteratively construct and refine tours until an optimal or near-optimal route is obtained. The ants select nodes based on certain criteria and deposit pheromone on the edges, which influences the decision-making process of subsequent ants. Through the process of pheromone updates, the ACO algorithm explores the solution space and converges towards an optimal solution to the TSP.

Functions

1. \_\_init\_\_ (Edge Class): represents a connection between two cities in the TSP. It stores the indices of the cities, the weight (distance) of the edge, and the amount of pheromone on the edge.
2. \_\_init\_\_(Ant Class): represents an ant agent in the ACO algorithm. It stores the parameters alpha and beta, which control the importance of pheromone and heuristic information in the ant's decision-making process. It also keeps track of the number of nodes, the edges connecting them, the ant's tour, and the distance traveled.
3. \_select\_node: used by an “ant” to choose the next node to visit in the TSP. It calculates a roulette wheel selection based on the pheromone levels and heuristic information of the unvisited nodes. The alpha and beta parameters influence the probabilities of node selection, favoring higher pheromone levels and lower distances.
4. find\_tour: used by an “ant” to construct a tour by iteratively selecting the next node based on the \_select\_node function until all nodes are visited. It returns the completed tour.
5. get\_distance: calculates the total distance traveled by an ant in its tour by summing the weights (distances) of the edges between consecutive nodes. It returns the calculated distance.
6. \_\_init\_\_(SolveTSPUsingACO Class): sets up various parameters and data structures, including the generation\_distances list to store the best distance at each generation, the colony size, the scaling factor for pheromone, the evaporation rate, the weight of pheromone deposit, the number of steps (generations), the number of nodes, the nodes' coordinates, and the labels for the nodes. It also creates the edges between nodes, initializes the ants with the specified parameters, and sets initial values for the global best tour, global best distance, and initial tour.
7. \_add\_pheromone: “deposits” pheromones on the edges of the tour based on the distance traveled by the “ant”. It calculates the amount of pheromone to add and updates the pheromone levels accordingly.
8. \_max\_min: implements the Max-Min Ant System (MMAS) strategy for ACO. It performs iterations for the specified number of steps and updates the pheromone trails accordingly. It tracks the best tour found in each iteration and updates the global best tour and distance if necessary. The function also maintains a record of the best distance at each generation and plots the best tour, the generational change in distance, and the initial path.
9. plot: generates a visual representation of the ACO algorithm's results. It creates a figure with three subplots, each displaying different information such as the best tour found, generational change in distance, and the initial path. The function utilizes Matplotlib to plot the tour, distances, and paths with customizable line widths, point radii, and annotation sizes.
10. userSettings: prompts the user to enter parameters for the ACO algorithm, such as the number of cities, population size, and number of generations. It then creates an instance of the SolveTSPUsingACO class with the specified parameters, runs the ACO algorithm using the \_max\_min method, and plots the results using the plot method. It also includes input validation functions to ensure the user provides valid numerical inputs.
11. checkUserInputInt: takes input from user and ensures the value is an integer, if not returns false.
12. checkUserInputFloat: takes input from user and ensures the value is a float, if not returns false.
13. \_\_name\_\_ (Main): serves as the entry point of the program.

Initial/Revised Implementation Issues

Originally, I was trying to only convert the Genetic Algorithm functions in my previous project thinking that all I needed was to change what algorithm to use and I would have no further issues. Halfway through development I reached a roadblock as most of the code added onto project two because more cluttered and unreadable. Due to this issue, I had decided to start from scratch and only imported parts of my previous project code when needed instead of directly modifying a copy of the program code. Other issues the began to arise were my implementation of a graphing system, my previous code began to show more of its flaws as I began testing and the amount of processing time became atrocious. So I decided to reuse smaller chunks of the previous plotting function and rewrote the function containing only a handful of lines from it.

Limitations and Future Improvements

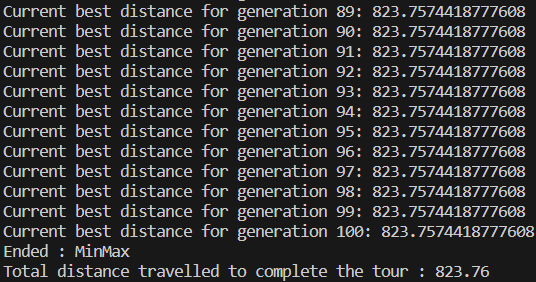
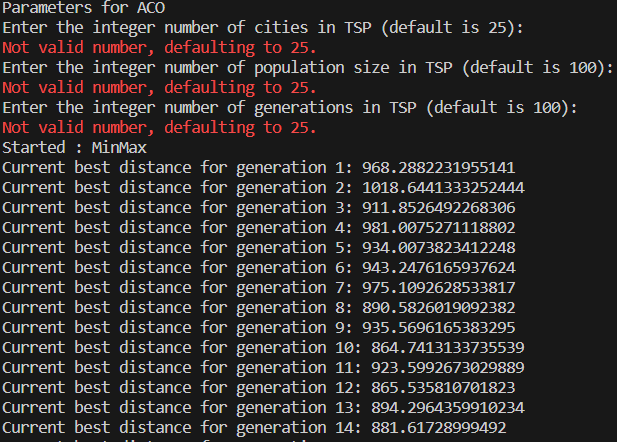
Like the previous project, due to time constraints I was unable to optimize the program to the fullest, there are still some slow downs which could have been addressed and potentially some bugs I was not able to find. In the future once the class ends and I have more time to work on personal projects, I would like to redo the projects I have taken in this class and to improve upon them.

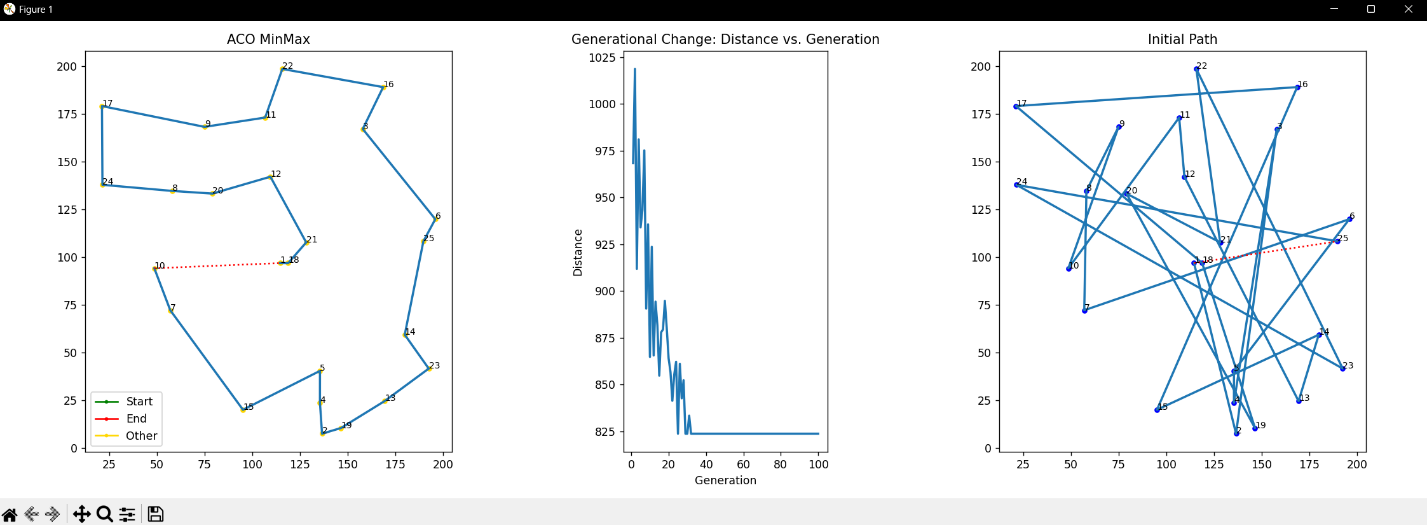
Questions

1. How were the cities and distances represented (as a data structure)?
   1. The cities and distances are represented using a class called "edges." The "Edge" class represents a connection between two cities. It contains attributes such as the city indices (a and b), the distance between them, and the amount of pheromone on that edge.
2. How did you encode the solution space?
   1. The solution space is encoded using a list of ants. Each “ant” represents a potential solution to the traveling salesman problem (TSP). The ants construct tours by selecting nodes based on certain criteria, and the quality of the solution is evaluated based on the distance traveled.
3. How did you handle the creation of the initial ant population?
   1. The creation of the initial ant population is handled in the constructor of the SolveTSPUsingACO class. The specified number of ants is created, each with its own alpha and beta parameters.
4. How did you handle the updating of the pheromone trails?
   1. The updating of the pheromone trails is done in the "\_add\_pheromone" method. After each iteration, the pheromone on the edges traversed by the best ant tour and the global best tour is increased by a certain amount. The pheromone on other edges is reduced by a factor (rho).
5. Which strategy(ies) did you use to compute the best solution?
   1. The Max-Min Ant System (MMAS) was used to compute the best solution. During the first approximate 75% of the iterations, the pheromone is updated based on the best tour found so far. After that, the pheromone is updated based on the global best tour.
6. Which stopping condition did you use? Why?
   1. The stopping condition is based on the number of iterations (steps) specified by the user input. The algorithm iterates for the given number of steps, updating the pheromone trails and evaluating the best tour at each step. It was chosen as to mimic the previous project, giving the user more input on how to control the outcome.
7. What other parameters, design choices, initialization and configuration steps are relevant to your design and implementation?
   1. Other relevant parameters, design choices, initialization, and configuration steps in the implementation include:
   * Alpha and beta parameters
   * Rho
   * Pheromone deposit weight
   * Initial pheromone
   * Min scaling factor
   * Plotting of initial, generational, and final outcomes.
8. Which (simple) experiments have you run to observe the impact of different design decisions and parameter values? Post their results and your comments.
   1. The experiments done were based on some initial inputs from project 2, and others were just the default inputs multiplied by an arbitrary amount.

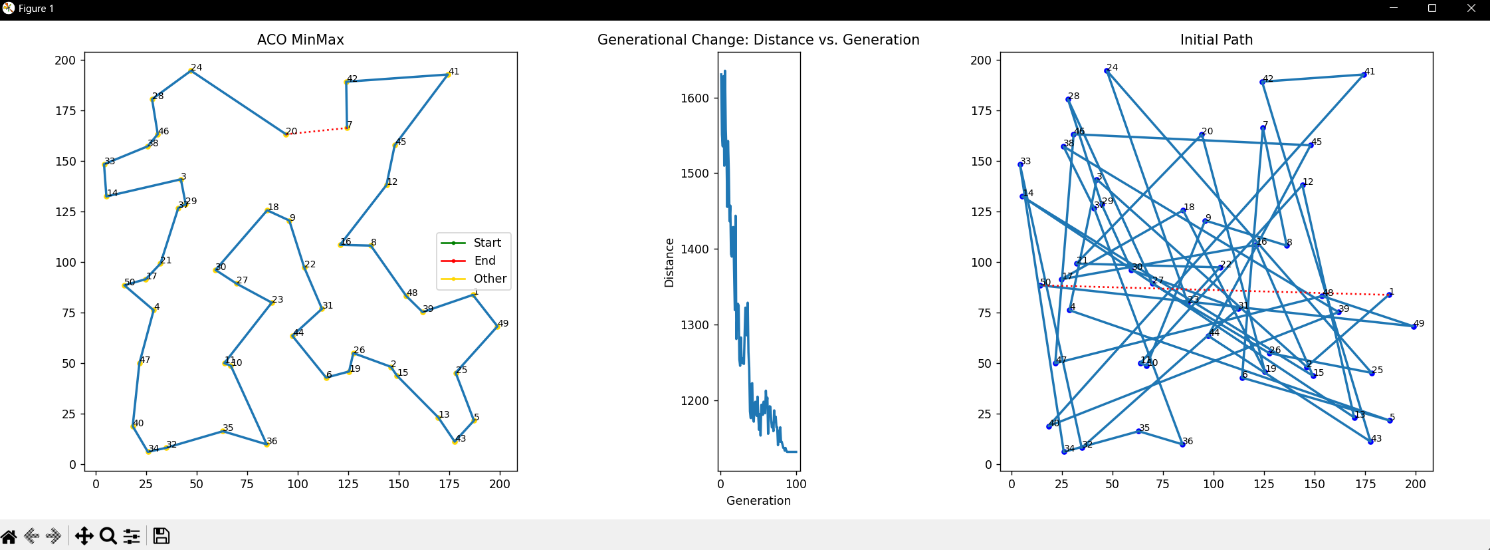
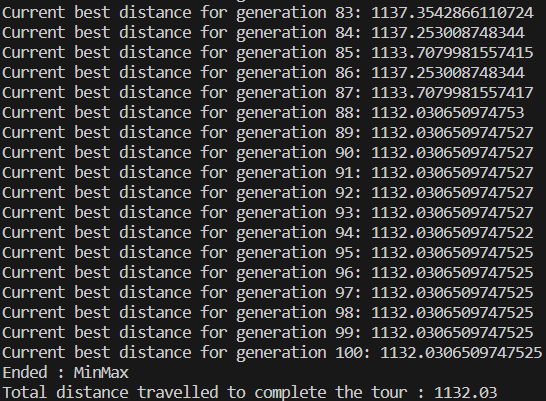
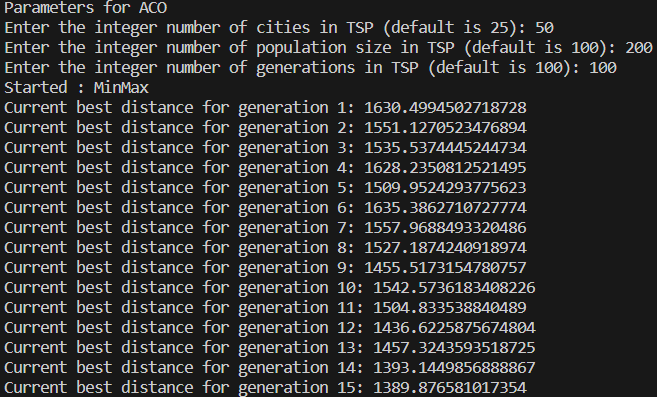
Demos

**Demo1:**

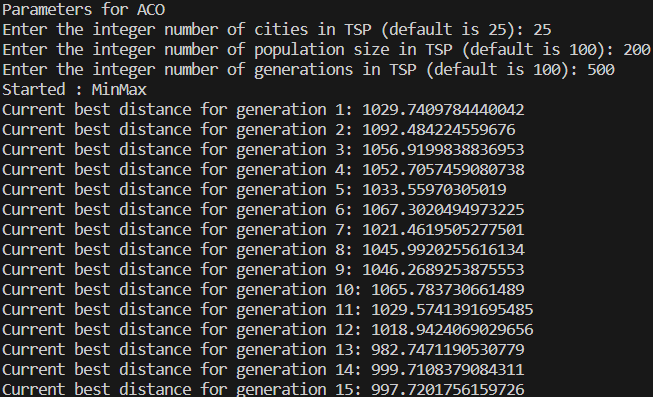


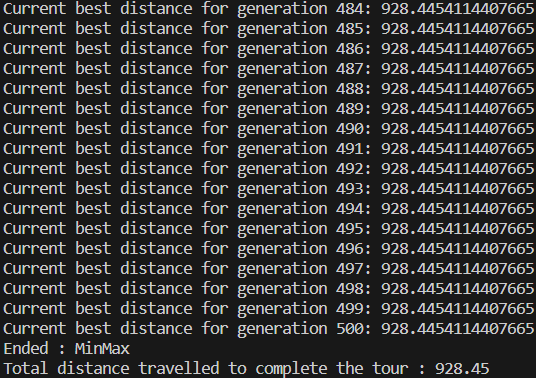


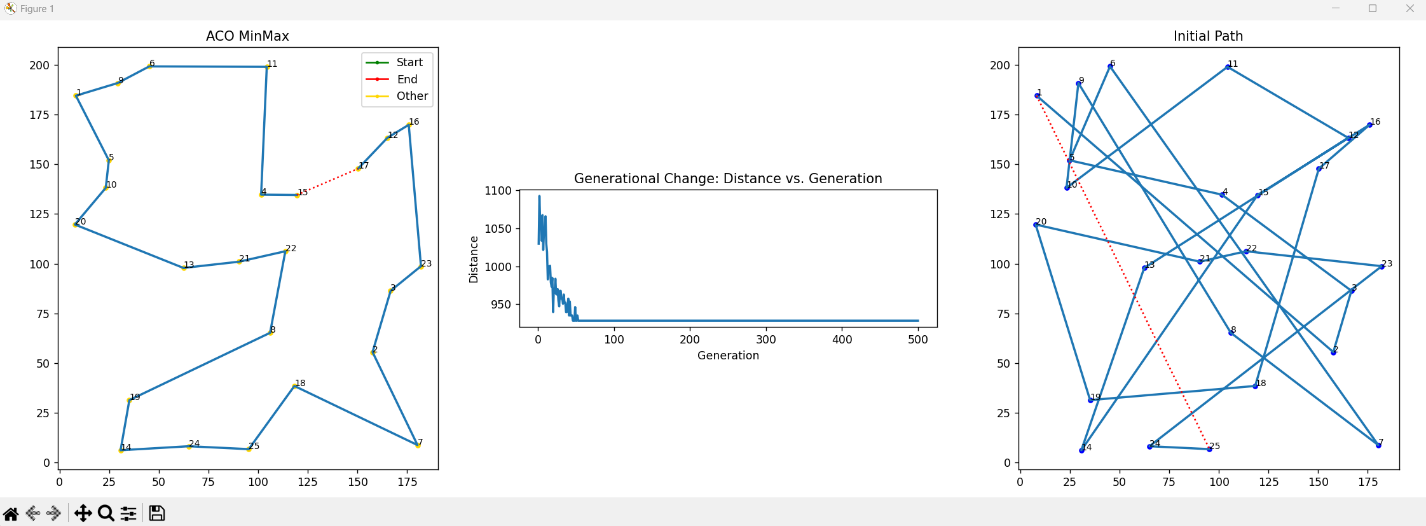
**Demo2:**

****

**Demo3:**

****

****

****

References

T. Stutzle and H. Hoos, "MAX-MIN Ant System and local search for the traveling salesman problem," Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97), Indianapolis, IN, USA, 1997, pp. 309-314, doi: 10.1109/ICEC.1997.592327.

Rodrigues, N. (2019, April 4). *Traveling Salesman Problem using Ant Colony Optimization*. GitHub. https://github.com/nishnash54/TSP\_ACO/blob/master/aco\_tsp.py