Project 2

Cap 4630

Dr. Marques

Andrew Donate

Traveling Salesman Problem: Baseline Solution (Genetic Algorithms)

Prologue

The following document describes of Project 2 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 genetic algorithms to efficiently solve the Traveling Salesman Problem (TSP). By employing genetic operators such as selection, crossover, and mutation, the program aims to evolve a population of candidate solutions iteratively, gradually improving their fitness until an optimal or near-optimal route is obtained.

Functions

1. distance: calculates and returns the distance between the current city and the input city. It uses the difference in x and y coordinates to compute the distance, applying the Pythagorean theorem and returning the value.
2. routeDistance: calculates and returns the total distance of a given route. It checks if the distance was already calculated, if not, it then calculates the distance by summing up the distances between consecutive cities in the route.
3. routeFitness: calculates the fitness of an individual route. If not calculated already it will do so by taking the reciprocal of the route distance (using the function routeDistance).

4. createRoute: generates a random route by shuffling the given cities and returns the route.
5. initialPopulation: creates an initial population of routes by repeatedly calling createRoute(4) and appending the generated routes to the population list until the population size is reached.
6. sortRoutes: calculates the fitness of each route in the population using the Fitness class and returns a sorted list of tuples, where each one contains the index of a route and its fitness value.
7. selectParents: selects a subset of routes as "parents" for the next generation based on their ranking in the population. It first adds the top routes (amountOfElite) based on ranking to the selection results list. Then adds the remaining by randomly selecting routes with a higher cumulative percentage fitness, favoring those with higher values.
8. matingPool: creates a mating pool by selecting routes from the population based on the indices provides in the selectionResults list. It goes over selectionResults and moves the corresponding routes to the mating pool.
9. breedOffSpring: takes two parents, randomly selects a gene range from one parent and creates a child by combining the selected gene range from the first to the remaining in the second.
10. breedEntirePopulation: breeds the entire population by selecting a certain number of "elites," randomly selecting the remaining, and creating children by breeding pairs. The new children are then part of the next generation.
11. changeCity: mutates an individual by randomly swapping cities in a route based on mutation rate.
12. changeCityWholePopulation: calls changeCity(11) to apply mutation to the entire population.
13. nextGeneration: generates the next generation of the population by calling the following, selectParents(7), matingPool(8), breedEntirePopulation(10), and changeCityWholePopulation(12). The results of the previous functions create the next generation of the population.

14. geneticAlgorithm: calls initialPopulation(5), nextGeneration(13), and sortRoutes(6) to run the genetic algorithm based on the number of generations and “evolves” the population and tracks the best route found.
15. geneticAlgorihtmPlot: has the same functionality as geneticAlgorithm(14) but also calls plotRoute(16).
16. plotRoute: plots out the progression chart (distance/generations), the initial route, and the best route.
17. userSettings: prompts the user to enter various parameters such as number of cities, population size, mutation rate, number of “elites”, enable/disable stagnation, plotting preferences, generation amount, and calls checkUserInputInt(18) and checkUserInputFloat(19) to ensure invalid inputs are addressed.
18. checkUserInputInt: takes input from user and ensures the value is an integer, if not returns false.
19. checkUserInputFloat: takes input from user and ensures the value is a float, if not returns false.

Initial Implementation Issues

1. In the userSettings() function initially when implementing the function of stagnation I forgot to ensure that the number of generations was set properly. As a result, if the user chose no to stop on stagnation, the code would default to 500 generations, but if they chose to stop on stagnation, the number of generations was set at the initial value of 0 instead of the default 500 (an incorrect indent caused the issue on line 346), causing the code to create only two new generations instead of the default 500 (Lines 188 – 194 show where I had to debug the code to make sure what inputs were going into the genericAlgorithm() function).

Other issues that are not mentioned in this document were too minor to address as they mainly consisted of spelling errors or indentation.

Revised Implementation

1. userSettings() after some debugging was fixed and full functional once the errors were corrected. Multiple different types of inputs were entered in (referenced from the assignment output demos) and their results were on par with what was referenced.
2. Updated routeDistance() function: Instead of using an explicit for loop to calculate the path distance, NumPy's library had some functions pre-made that was used to improve readability and performance (See lines 37-50).
3. Updated sortRoutes() function: Utilized NumPy's functions once again to improve the functionality and readability of sortRoutes (See lines 74-82).

Previous code that was written is still in the program but commented out to show those who would like to see the source code how it was improved upon.

Limitations and Future Improvements

Although the revised implementation does fix some of the issues in the initial, there are still limitations and improvements that could be done in the future.

1. Lack of multi-threaded performance: in the future if I were to improve upon this program, I would find a way to allow computers with many cores to utilize the hardware and speed up the calculations in the program. When running the demos below the time it took to run through all the generations was horrendous, especially since with Windows Task Manager pulled up you could see that on the system I tested on, only 10% of processing was being used. If multi-threading was implemented, I am sure the time to run the program would decrease leading to faster results.
2. Unnecessary duplicate functions: although geneticAlgorithm and geneticAlgorithmPlot functions are slightly different they contain majority the same calculation code. In the future I would like to combine the two functions so the program takes up less space, increase readability, and reuse already written code.

Questions

1. How were the cities and distances represented (as a data structure)?
    a. Cities and distances were represented as a Dynamic Data Structure as when creating, deleting, and modifying the list the cities and distances were stored in the list was dynamically adjusted for the number of cities.
2. How did you encode the solution space?
    a. Using the functions listed prior and with the instances of the City and Fitness class, the solution space was encoded as a population of routes, and the genetic algorithm evolves the population to find the best solution to the TSP.
3. How did you handle the creation of the initial population?
    a. By calling the initialPopulation function, it repeatedly calls createRoute function which appends the generated routes to the population list until the given population size has been reached.
4. How did you compute the fitness score?
    a. By calling the routeFitness function, it calculates the fitness of an individual route by taking the reciprocal of the route distance. The closer to one the distance was the better the fitness score.
5. Which parent selection strategy did you use? Why?
    a. A combination of elitism and roulette wheel section was chosen. Just like real life survival of the fittest wins, but there will always be some unknown reason why something evolves which is why roulette wheel was also used.
6. Which crossover strategy(ies) did you try? Which one worked out best?
    a. The crossover strategy used was Partially Mapped Crossover as it picks random subset of genes from one parent and fills in the remaining from the second parent. I believe this was the best option as it took random number of genes from the first parent and used those genes not used in parent two to fill in the rest leading to somewhat accurate gene crossover like in the real world.
7. Which mutation strategy(ies) did you try? Which one worked out best?
    a. Randomly swapping cities in a route, also known as swap mutation. Other mutation methods were not tested due to time constraints, but I believe the one used is not that effective compared to others due to only swapping around two cities instead of more as mutation can affect more than just two cities out of the many inputed.
8. Which strategy did you use for populating the next generation? Why?
    a. A combination of parent selection, mating pool creation, breeding, and mutation.
9. Which stopping condition did you use? Why?
    a. When choosing to enable stagnation the limit would be the number of generations a user inputs, when disabled the limit is 500 generations as from the test results down below when stagnation threshold is inputted the value of

generations was never reached (This is a bad practice and should have been changed.)

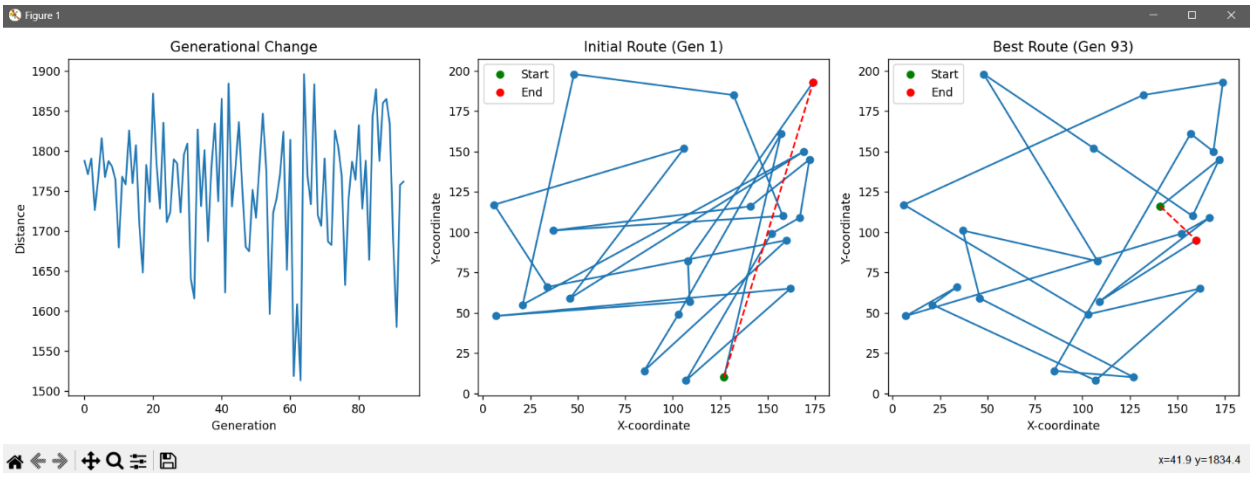
10. What other parameters, design choices, initialization and configuration steps are relevant to your design and implementation?
    a. User settings and plotting were crucial in my design as without user settings you would have to manually change values in the code (hard coding) which could lead to issues if non-modifiable sections were to be changed. Without plotting it would be hard to visualize what the path looks like. Just being given the distance between all the plots connected does not do it justice to see what the progress was made from the initial to the final distances.
11. Which (simple) experiments have you run to observe the impact of different design decisions and parameter values? Post their results and your comments.
    a. Most of the experiments done were based on the “Exemplary Assignment 2 Run Screenshots.pdf” given to us by the professor. Most of the inputs and options listed were implemented besides the plotting function which was added to help visualize and improve on the examples best path output.

Demos

Demo1:

```
Parameters for Genetic Algorithm
Enter the integer number of cities in travelling salesman problem (default is 25): 25
Enter the integer population size (default is 100): 1000
Enter the float mutation rate (default is 0.01): 0.2
Enter the proportion of new children in each generation (default is 0.2): 0.25
Do you want to stop on stagnation? (default is N) (Y/N): y
Enter number of consecutive generations with no improvement to use as a stopping condition (default is 40): 30
Would you like the output to be graphed? (default is N) (Y/N): y
Initial distance: 1787.909
Gen 1 | Minimum Total Distance: 1771.311
Gen 2 | Minimum Total Distance: 1790.903
Gen 3 | Minimum Total Distance: 1726.333
Gen 4 | Minimum Total Distance: 1765.817
Gen 5 | Minimum Total Distance: 1816.218
Gen 6 | Minimum Total Distance: 1767.734
Gen 7 | Minimum Total Distance: 1787.762
Gen 8 | Minimum Total Distance: 1781.12
```

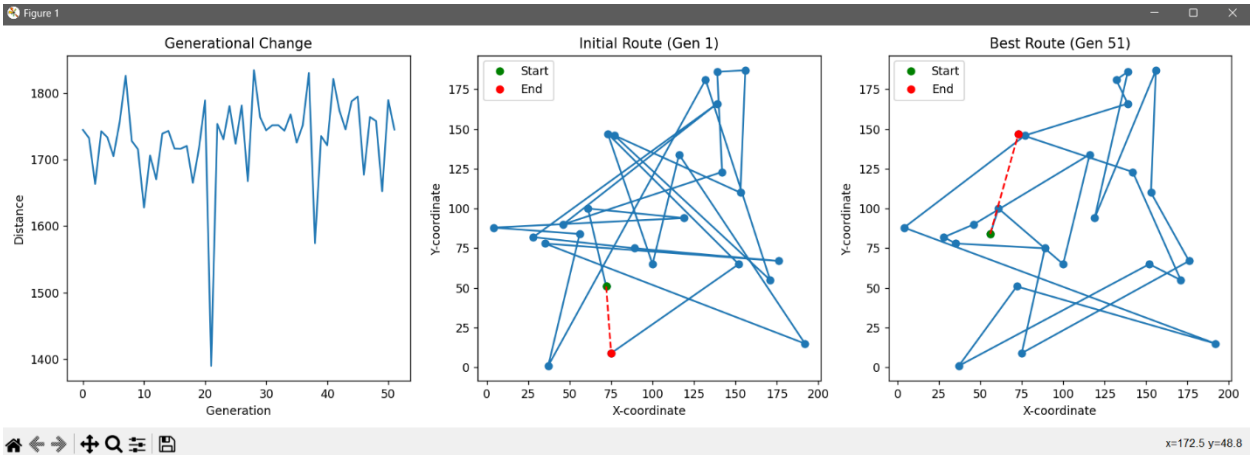
```
Gen 84 | Minimum Total Distance: 1843.788
Gen 85 | Minimum Total Distance: 1877.591
Gen 86 | Minimum Total Distance: 1787.737
Gen 87 | Minimum Total Distance: 1860.343
Gen 88 | Minimum Total Distance: 1865.394
Gen 89 | Minimum Total Distance: 1834.119
Gen 90 | Minimum Total Distance: 1687.189
Gen 91 | Minimum Total Distance: 1580.17
Gen 92 | Minimum Total Distance: 1757.843
Gen 93 | Minimum Total Distance: 1761.95
Final distance: 1761.949625129279
```



Demo2:

```
Parameters for Genetic Algorithm
Enter the integer number of cities in travelling salesman problem (default is 25): 25
Enter the integer population size (default is 100): 500
Enter the float mutation rate (default is 0.01): 0.2
Enter the proportion of new children in each generation (default is 0.2): 0.25
Do you want to stop on stagnation? (default is N) (Y/N): y
Enter number of consecutive generations with no improvement to use as a stopping condition (default is 40): 30
Would you like the output to be graphed? (default is N) (Y/N): y
Initial distance: 1744.849
Gen 1 | Minimum Total Distance: 1732.619
Gen 2 | Minimum Total Distance: 1663.395
Gen 3 | Minimum Total Distance: 1742.884
Gen 4 | Minimum Total Distance: 1733.663
Gen 5 | Minimum Total Distance: 1705.422
Gen 6 | Minimum Total Distance: 1755.912
Gen 7 | Minimum Total Distance: 1826.332
Gen 8 | Minimum Total Distance: 1728.135
```

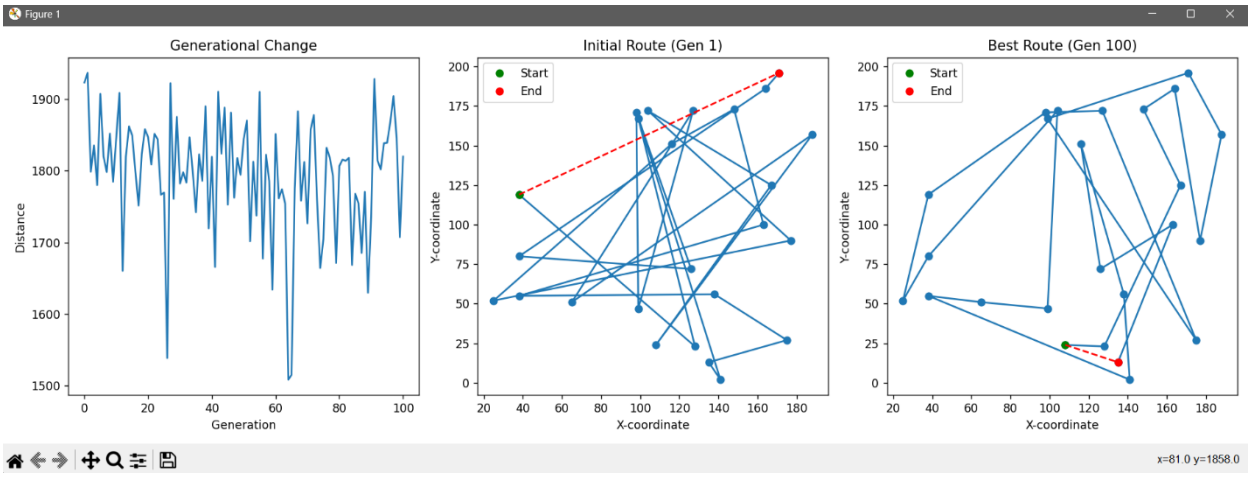
```
Gen 42 | Minimum Total Distance: 1773.866
Gen 43 | Minimum Total Distance: 1745.572
Gen 44 | Minimum Total Distance: 1787.938
Gen 45 | Minimum Total Distance: 1795.019
Gen 46 | Minimum Total Distance: 1677.31
Gen 47 | Minimum Total Distance: 1764.286
Gen 48 | Minimum Total Distance: 1758.142
Gen 49 | Minimum Total Distance: 1652.592
Gen 50 | Minimum Total Distance: 1789.855
Gen 51 | Minimum Total Distance: 1745.362
Final distance: 1745.36221071039
```



Demo3:

```
Parameters for Genetic Algorithm
Enter the integer number of cities in travelling salesman problem (default is 25): 25
Enter the integer population size (default is 100): 1000
Enter the float mutation rate (default is 0.01): 0.5
Enter the proportion of new children in each generation (default is 0.2): 0.5
Do you want to stop on stagnation? (default is N) (Y/N): n
Enter the integer number of generations (default 500): 100
Would you like the output to be graphed? (default is N) (Y/N): y
Initial distance: 1923.045
Gen 1 | Minimum Total Distance: 1936.625
Gen 2 | Minimum Total Distance: 1798.673
Gen 3 | Minimum Total Distance: 1835.348
Gen 4 | Minimum Total Distance: 1780.029
Gen 5 | Minimum Total Distance: 1907.407
Gen 6 | Minimum Total Distance: 1820.149
Gen 7 | Minimum Total Distance: 1798.257
Gen 8 | Minimum Total Distance: 1851.941
```

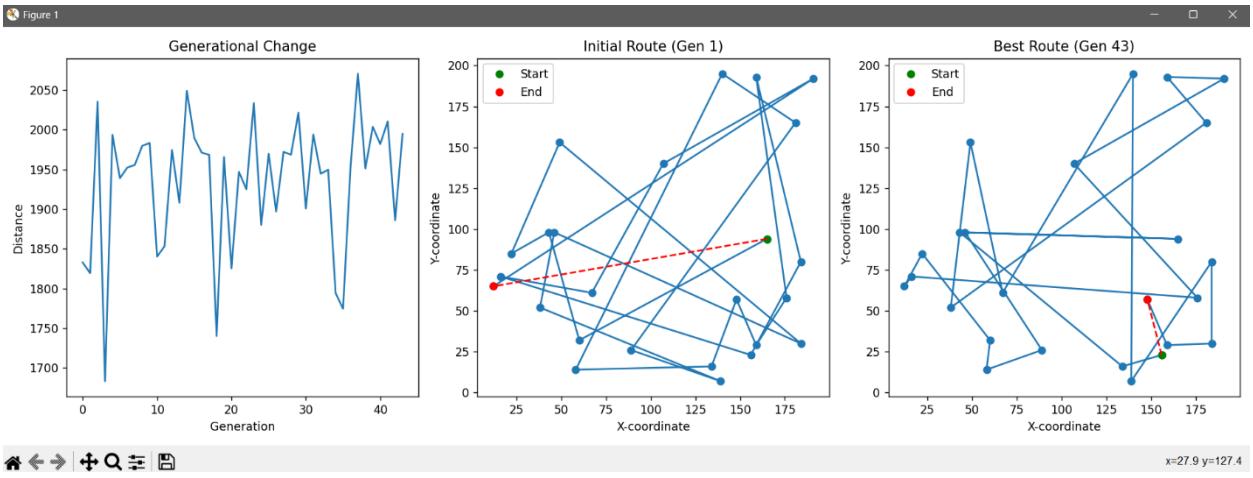
```
Gen 91 | Minimum Total Distance: 1928.176
Gen 92 | Minimum Total Distance: 1814.02
Gen 93 | Minimum Total Distance: 1801.856
Gen 94 | Minimum Total Distance: 1838.428
Gen 95 | Minimum Total Distance: 1839.078
Gen 96 | Minimum Total Distance: 1868.151
Gen 97 | Minimum Total Distance: 1904.28
Gen 98 | Minimum Total Distance: 1844.526
Gen 99 | Minimum Total Distance: 1707.232
Gen 100 | Minimum Total Distance: 1819.844
Final distance: 1819.8443365792864
```



Demo4:

```
Parameters for Genetic Algorithm
Enter the integer number of cities in travelling salesman problem (default is 25): 25
Enter the integer population size (default is 100): 1000
Enter the float mutation rate (default is 0.01): 0.4
Enter the proportion of new children in each generation (default is 0.2): 0.1
Do you want to stop on stagnation? (default is N) (Y/N): Y
Enter number of consecutive generations with no improvement to use as a stopping condition (default is 40): 40
Would you like the output to be graphed? (default is N) (Y/N): y
Initial distance: 1832.561
Gen 1 | Minimum Total Distance: 1819.111
Gen 2 | Minimum Total Distance: 2035.45
Gen 3 | Minimum Total Distance: 1683.027
Gen 4 | Minimum Total Distance: 1993.641
Gen 5 | Minimum Total Distance: 1938.939
Gen 6 | Minimum Total Distance: 1952.318
Gen 7 | Minimum Total Distance: 1955.626
Gen 8 | Minimum Total Distance: 1979.689
```

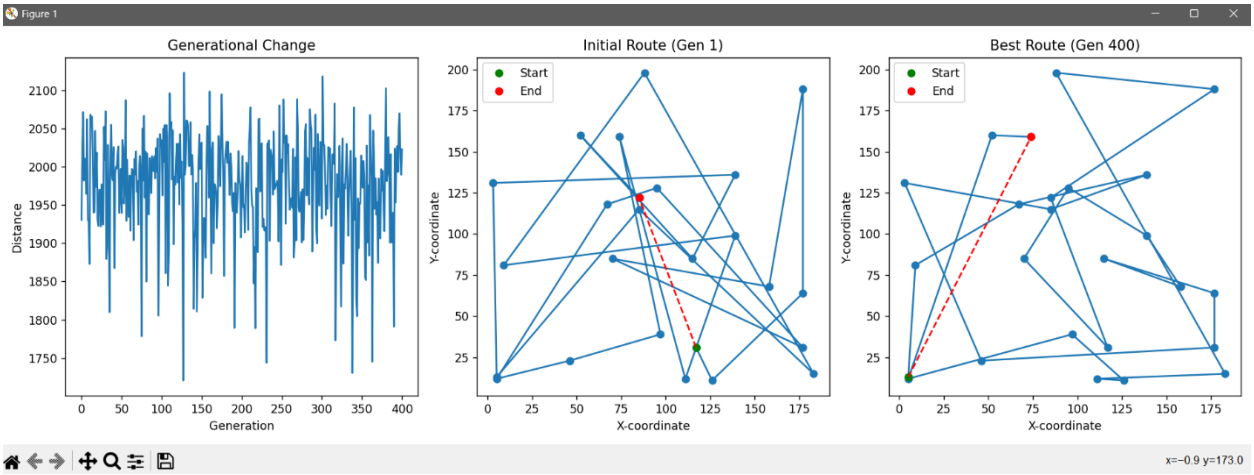
```
Gen 34 | Minimum Total Distance: 1794.052
Gen 35 | Minimum Total Distance: 1774.322
Gen 36 | Minimum Total Distance: 1951.914
Gen 37 | Minimum Total Distance: 2070.691
Gen 38 | Minimum Total Distance: 1951.162
Gen 39 | Minimum Total Distance: 2003.589
Gen 40 | Minimum Total Distance: 1981.955
Gen 41 | Minimum Total Distance: 2010.386
Gen 42 | Minimum Total Distance: 1885.847
Gen 43 | Minimum Total Distance: 1994.813
Final distance: 1994.8129329870217
```



Demo5:

```
Parameters for Genetic Algorithm
Enter the integer number of cities in travelling salesman problem (default is 25): 25
Enter the integer population size (default is 100): 750
Enter the float mutation rate (default is 0.01): 0.7
Enter the proportion of new children in each generation (default is 0.2): 0.7
Do you want to stop on stagnation? (default is N) (Y/N): n
Enter the integer number of generations (default 500): 400
Would you like the output to be graphed? (default is N) (Y/N): y
Initial distance: 1930.39
Gen 1 | Minimum Total Distance: 2027.72
Gen 2 | Minimum Total Distance: 2071.3
Gen 3 | Minimum Total Distance: 1982.052
Gen 4 | Minimum Total Distance: 2010.809
Gen 5 | Minimum Total Distance: 1994.432
Gen 6 | Minimum Total Distance: 1964.665
Gen 7 | Minimum Total Distance: 2062.053
Gen 8 | Minimum Total Distance: 1930.311
```

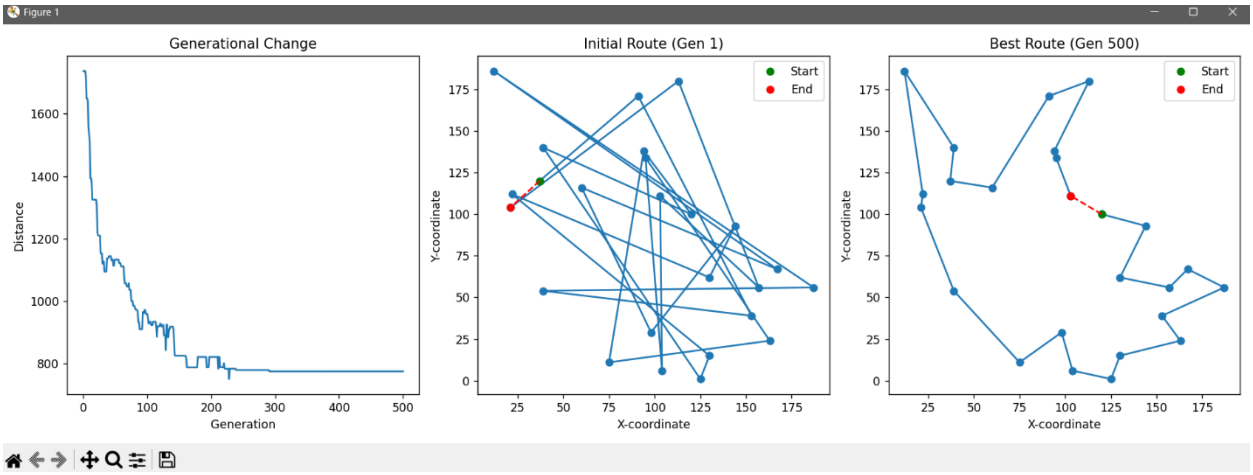
```
Gen 386 | Minimum Total Distance: 2016.208
Gen 387 | Minimum Total Distance: 1901.054
Gen 388 | Minimum Total Distance: 1935.581
Gen 389 | Minimum Total Distance: 1939.856
Gen 390 | Minimum Total Distance: 1791.122
Gen 391 | Minimum Total Distance: 2023.55
Gen 392 | Minimum Total Distance: 1953.29
Gen 393 | Minimum Total Distance: 2011.712
Gen 394 | Minimum Total Distance: 2026.994
Gen 395 | Minimum Total Distance: 1993.103
Gen 396 | Minimum Total Distance: 2050.825
Gen 397 | Minimum Total Distance: 2069.762
Gen 398 | Minimum Total Distance: 2011.793
Gen 399 | Minimum Total Distance: 1989.428
Gen 400 | Minimum Total Distance: 2022.704
Final distance: 2022.703560045916
```



Demo6:

```
Parameters for Genetic Algorithm
Enter the integer number of cities in travelling salesman problem (default is 25): l
Not valid number, defaulting to 25.
Enter the integer population size (default is 100): l
Not valid number, defaulting to 100
Enter the float mutation rate (default is 0.01): .
Not valid number, defaulting to 0.01
Enter the proportion of new children in each generation (default is 0.2): /
Not valid number, defaulting to 0.2
Do you want to stop on stagnation? (default is N) (Y/N): ;
Enter the integer number of generations (default 500): l
Not valid number, defaulting to 500
Would you like the output to be graphed? (default is N) (Y/N): y
Initial distance: 1737.077
Gen 1 | Minimum Total Distance: 1737.077
Gen 2 | Minimum Total Distance: 1737.077
Gen 3 | Minimum Total Distance: 1737.077
```

```
Gen 495 | Minimum Total Distance: 774.505
Gen 496 | Minimum Total Distance: 774.505
Gen 497 | Minimum Total Distance: 774.505
Gen 498 | Minimum Total Distance: 774.505
Gen 499 | Minimum Total Distance: 774.505
Gen 500 | Minimum Total Distance: 774.505
Final distance: 774.5054042172533
```



References

Stoltz, E. (2021, March 18). *Evolution of a salesman: A complete genetic algorithm tutorial for python*. Medium. https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35

Vishal. (2021, April 24). *Check user input is a number or string in Python*. PYnative. https://pynative.com/python-check-user-input-is-number-or-string/