**Project 5 – Wisdom of Crowds TSP**

Stone Barrett

Computer Science Engineering

Speed School of Engineering

University of Louisville, USA

[scbarr04@louisville.edu](mailto:scbarr04@louisville.edu)

1. **Introduction**

In this project, I was asked to adapt the idea of a wisdom of artificial crowds algorithm to be implemented in solving the Traveling Salesperson Problem. The Traveling Salesperson Problem (TSP), as we’ve come to learn, is a variation of an NP-complete computer science problem. The problem entails a salesperson – or in this case, a program – being given a list of cities that need to be visited on his or her business route. The route they must take will complete a Hamiltonian cycle of the graph, meaning that every city will be visited once and only once, sans the starting city which must also be the ending city. The goal of the problem is to find the shortest possible route to take through all the cities. Thus far in class, our projects have been slowly tipping the scales between optimization and speed, where higher optimization requires lower speed and vice versa. For example, brute forcing a solution by generating every single possible permutation of the list of cities and comparing every total cost gives a certainly best solution but takes a very long time to compute; meanwhile, variations of the greedy algorithm can return a solution somewhat quickly, but by nature of the algorithm they may not be very good cost-wise. This time around, the goal was to implement what is known as a wisdom of artificial crowds algorithm and find a very good solution in very good time. This project builds upon the previous one – genetic algorithm – because it uses the previous project in its implementation. For there to be an artificial crowd to gain wisdom from, there must first be a genetic algorithm that produces generations of individuals to draw crowds from.

1. **Approach**

The wisdom of crowds algorithm is an idea meant to contrast the standard genetic algorithm approach. The standard genetic algorithm will use fitness functions, crossover and mutation methods, and other tools to simulate natural selection and produce many generations of solutions to a problem – each getting better as evolution continues – and taking the best individual solution from the final generation produced. Wisdom of artificial crowds contrasts this by, instead of taking the best single solution from a population of members all produced by the same lineage, taking multiple individuals from different evolutionary lineages and discretely comparing their solutions’ parts to create one solution better than the sum of its parts.

In this case, the genetic algorithm I described in Project 4 runs to completion four times. Each of these four runs returns a population that is then passed into a separate wisdom of crowds method. This new method then extracts the shortest cost route individual from each of the populations. These four individuals make up the artificial crowd. The program then randomly chooses two of these experts from the crowd to compare opinions (orders of cities to visit). Random subsections of an expert’s opinion (subsections of lists of cities) are extracted and added to an aggregate solution. The other expert’s opinion is then compared to what is in the aggregate solution, and the remaining city slots are filled in. What gets returned is a solution made up of the crossing of the very best of two populations.

1. **Results** 
   1. **Data**

Such as before, the data provided with the project assignment folder will serve as the sample data when running the program. This time, the sample data includes randomly generated lists of cities in the form of two-dimensional geographical coordinates. There are lists of size 11, 22, 44, 77, 97, and 222. The list of 222 is over double the maximum sample data provided in this course thus far. These files will be read from the source code’s directory to create the list data structures used throughout the runtime. The genetic algorithm from Project 4 will be ran with a population size of 200, generation count of 2500, genetic elite count of 25, and mutation chance of .1%.

* 1. **Results** (Numerical results and any figures or tables.)

For each dataset, there will be essential figures provided in this order: the genetic algorithm’s improvement curve (specifically the fourth run of each dataset); the route produced by the genetic algorithm; the route produced by the wisdom of crowds modifying function; and finally, the terminal output showing the genetic algorithm’s initial generation’s route cost, progress indicator (per 500 generations), the genetic algorithm’s final generation’s route cost, and the wisdom of crowds algorithm’s route cost.

11 Cities

Figure 3.1

Chart

Description automatically generated with medium confidence

Figure 3.2

Chart, line chart

Description automatically generated

Figure 3.3

Chart, line chart

Description automatically generated

Figure 3.4

Text

Description automatically generated

22 Cities

Figure 3.5

Chart

Description automatically generated

Figure 3.6

Chart, line chart

Description automatically generated

Figure 3.7

Chart, line chart

Description automatically generated

Figure 3.8

Text

Description automatically generated

44 Cities

Figure 3.9

Chart

Description automatically generated

Figure 3.10

Chart, line chart

Description automatically generated

Figure 3.11

Chart, radar chart, line chart

Description automatically generated

Figure 3.12

Graphical user interface, text

Description automatically generated

77 Cities

Figure 3.13

Chart

Description automatically generated

Figure 3.14

Chart, radar chart

Description automatically generated

Figure 3.15

Chart, line chart

Description automatically generated

Figure 3.16

Text

Description automatically generated

97 Cities

Figure 3.17

Chart

Description automatically generated

Figure 3.18

Chart, radar chart

Description automatically generated

Figure 3.19

Chart

Description automatically generated

Figure 3.20

Text

Description automatically generated

222 Cities

Figure 3.21

Chart, histogram

Description automatically generated

Figure 3.22

Chart, radar chart

Description automatically generated

Figure 3.23

Diagram

Description automatically generated

Figure 3.24

Text

Description automatically generated

Final Table

Graphical user interface, application

Description automatically generatedFigure 3.25

1. **Discussion**

Time

To begin discussion, I’d like to mention that the completion time values displayed in the terminal output figures are not entirely accurate. The timer within the program was set stop when all processes had finished execution. This included the GUI windows produced by *MatPlotLib*, which require manual attention before closing*.* This meant that when I let the program run on large datasets and stepped away from my workstation, sometimes I didn’t notice that it had completed until sometime later, inflating the timer’s count. However, it should be noted that this inflation was not so much that it threw off the general trend of the times recorded. Surprisingly, the larger datasets did not take nearly as long as expected, with expectations based on previous projects. This goes to show, yet again, how powerful the genetic algorithm approach is – the datasets can increase in size significantly, but the processing time increases at a manageable rate instead of exponential.

Another surprise relating to time came in the form of the wisdom of crowds method that accompanied the genetic algorithm. The genetic algorithm may have taken some time to complete, but once it did, creating an aggregate solution using wisdom of crows was always instantaneous. This is a very quick modification to the results of a genetic algorithm implementation.

Evolution Curve

As is well demonstrated by comparing Figure 3.1 to Figure 3.21, improvement curves for the graph of generation count versus route cost take shape and plateau much faster on smaller datasets. Comparing these two also suggests that allowing the larger datasets to produce even more generations would probably have yielded further evolution and shorter route costs. However, in the interest of time and control variables, I opted to keep it to 2500 generations across the board.

The Importance of Mutation

An interesting occurrence is visualized clearly in Figure 3.5. The improvement curve for this run seems to be plateauing until something happens to decrease costs almost instantaneously. What would have to have happened here is a mutation. There was some unforeseen improvement that could not happen over previous generations because crossover wouldn’t allow for it, but when the random mutation chance was hit, two edges were swapped and allowed for the improvement to come through in a cascade of new generations. This shows just how important random mutation is when modeling genetic algorithms. Too high of a chance can prevent crossover from doing its job, but too low of a chance can allow for tunnel vision and early stagnation.

Wisdom of Artificial Crowds

The wisdom of crowds algorithm showed very fast improvement to an already effective algorithm – on some datasets. Figure 3.25 shows that overall, running the wisdom of crowds algorithm the way it is implemented in my program results in slightly worse outcomes than in the genetic algorithm by itself. This is because my implementation uses randomness to select which parts of a crowd member’s opinion will be considered for the aggregate opinion. This is not a strict enough heuristic to improve upon a genetic algorithm all the time. It will improve sometimes and usually on smaller datasets, but not all the time and probably not on larger datasets. The solution to this – given more time – is to implement a more intelligent heuristic that considers a larger crowd of experts and keeps track of how many times a specific edge appears in the crowd members’ solutions so that the most common edges (occurrence above a certain line of demarcation or percentile) can be added to the aggregate solution. This is probably most easily done with an occurrence matrix data structure. This, however, would not complete a full route, given that only edges that appear a lot are being used. So, completing the route using branch and bound or greedy algorithms should work fine. This would most likely result in an improved route cost.

1. **References**

NumPy. (n.d.). Retrieved October 22, 2021, from <https://numpy.org/>

*Operator - standard operators as functions¶*. operator - Standard operators as functions - Python 3.10.0 documentation. (n.d.). Retrieved October 22, 2021, from <https://docs.python.org/3/library/operator.html>

*Pandas*. pandas. (n.d.). Retrieved October 22, 2021, from <https://pandas.pydata.org/>

*Pathlib - object-oriented filesystem paths¶*. pathlib - Object-oriented filesystem paths - Python 3.10.0 documentation. (n.d.). Retrieved October 22, 2021, from <https://docs.python.org/3/library/pathlib.html>

*Random - generate pseudo-random numbers¶*. random - Generate pseudo-random numbers - Python 3.10.0 documentation. (n.d.). Retrieved October 22, 2021, from <https://docs.python.org/3/library/random.html>

Stoltz, E. (2021, March 18). *Evolution of a salesman: A complete genetic algorithm tutorial for python*. Medium. Retrieved October 22, 2021, from <https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35>

*Time - Time Access and conversions¶*. time - Time access and conversions - Python 3.10.0 documentation. (n.d.). Retrieved October 22, 2021, from <https://docs.python.org/3/library/time.html>

*Visualization with python¶*. Matplotlib. (n.d.). Retrieved October 22, 2021, from <https://matplotlib.org/>