**Vehicle Routing Problem: Wisdom of Crowds Using Genetic Algorithms**

Jacob Taylor Cassady

Computer Engineering & Computer Science

Speed School of Engineering

University of Louisville, USA

[jtcass01@louisville.edu](mailto:NameID@louisville.edu)

# Introduction

# Approach

## Genetic Algorithm

The genetic algorithm implemented is inspired by sexual reproduction of gametes in biology. This algorithm retains a constant population of “chromosomes” which are representations of possible solutions/agents for/within the given problem. These chromosomes are a set of alleles that describe its performance. The algorithm makes use of two functions to evolve the population overtime to weed out the poor performers and mate the good performers.

### Crossover Methods

The implemented crossover methods have been shown to improve performance in a genetic algorithmic approach to TSP (ABDOUN & ABOUCHABAKA, 2011). Each algorithm was run with an 80% crossover probability, meaning that with each generation the top 20% of the population was used to generate replacements for the bottom 80%.

#### Uniform Crossover

The uniform crossover forms a child by randomly alternating between the two parents. For reference to the implementation of this method please see **Figure 6** in the appendix.

#### Ordered Crossover

The ordered crossover breaks each parent into three sequences, S1, S2, and S3 with matching indices for both parents. The child is then produced by taking S2 from one parent and filling S1 and S3 with alleles from the other parents starting at S1 and leaping genes already included. For reference to the algorithm as formalized in the literature, please refer to **Figure 1** below. For refence to the implementation of this method please see **Figure 8** in the appendix.

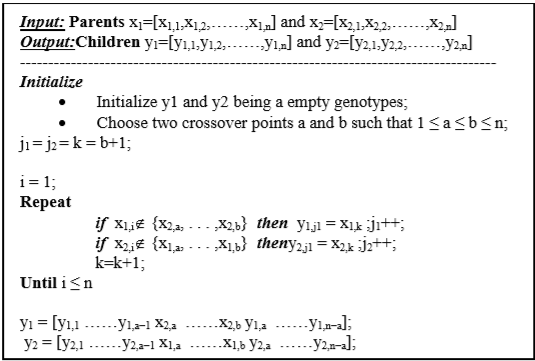


Figure 1 : Ordered Crossover Algorithm (ABDOUN & ABOUCHABAKA, 2011)

#### Partially Mapped (PM)

The partially mapped crossover breaks each parent into three sequences, S1, S2, and S3 with matching indices for both parents. The child is then produced by taking S1 and S3 from one parent and filling in S2 with alleles from the other parent starting at S2 and leaping genes already included. For reference to the algorithm as formalized in the literature, please refer to **Figure 2**. For reference to the implementation of this method please see **Figure 7** in the appendix.

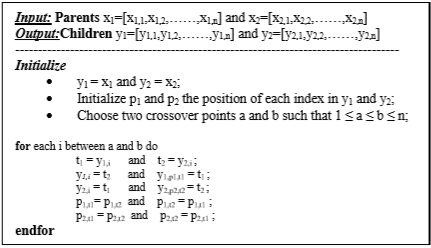


Figure 2 : Partially Mapped Algorithm (ABDOUN & ABOUCHABAKA, 2011)

### Mutation Methods

The implemented mutation methods have been shown to improve performance in a genetic algorithmic approach to solving TSP (ABDOUN & ABOUCHABAKA, 2011). All genetic algorithms were run with a mutation rate of 2%, meaning that each generation had 2% of its chromosomes undergo mutation.

#### TWORS

The TWORS mutation method randomly swaps two alleles’ locations within the chromosome. For reference to the implementation of this method, please see **Figure 9** in the appendix of this document.

#### Reverse Sequence (RSM)

The Reverse Sequence mutation method reverses the sequence of the chromosome. For reference to the implementation of this method, please see **Figure 10** in the appendix of this document.

## GUI

A GUI was developed to visualize different stages of the algorithm. A heat map was developed to understand the crowd’s edge frequency. Additionally, a route solution representation was generated to ensure proper connection of the final path.

### Heat Map

Heat maps were generated to help understand the crowd’s edge frequency. The edges were plotted with their RGB values denoting its frequency within the crowd. The most red colored edges are those that occur least frequent, while the most blue colored edges are those that occur most frequent. Please refer to **Figure 3** below for an example of a heat map generated from the edges with an occurrence rate in the top 80% for a set of 44 cities.

A close up of a map

Description automatically generated

Figure 3 : Heat Map Random44.tsp 20% Superiority Threshold

### Route Solution

A graphical representation of the final solution was generated to ensure it is reasonable. Each edge is colored using its vertices’ IDs to quantize its red and blue color magnitudes while the green magnitude is calculated from the modulus of the starting vertex id with respect to the ending vertex id. For reference to the implementation of this plotting method, please refer to **Figure 4** in the appendix.

A picture containing indoor, object, sky

Description automatically generated

Figure 4 : Route Solution Random44.tsp

## Greedy Heuristic

As noted above, to combine the solutions of the crowd of genetic algorithms the algorithm creates a dictionary of edges across the entire crowd and keeps track of the frequency of each edge. Edges that meet some predetermined “superiority threshold” are kept to develop a fragmented graph. For edges that contain the same vertex, the edge with the highest edge count and lowest distance traveled is retained. The relevant code for creating the fragmented graph is shown in **Figure 12** in the appendix of this document.

Once a recombination route is generated, there could still be some stray vertices. The remaining vertices are iterated over and the nearest vertex to an existing route segment is chosen and “lassoed” into the route segment. For reference to the route’s lasso function please refer to **Figure 13** in the appendix of this document. For reference to the method for choosing the next vertex to insert into the group of route segments, please refer to **Figure 14** in the appendix of this document.

Lastly, once all vertices are part of a route segment, a recombine method is used to connect the route segments into a contiguous route. This is done by iterating over the unvisited starting indices and connecting them in order as they appear in the list of edges generated from the lassoing of vertices. For reference to the implementation of this segment recombination method, please refer to **Figure 15** in the appendix of this document.

# Results

## Data

The algorithms were tested on different datasets ranging from 6 cities to 222. The dataset files were generated randomly. Within the test file, cities are enumerated, and x and y coordinates are provided. The input data was formatted like the example shown in **Figure 5** below.

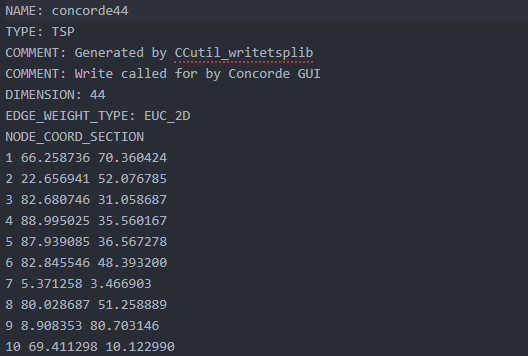


Figure 5: Random44.tsp Input File Format

## Results

# Discussion

# References

ABDOUN, O., & ABOUCHABAKA, J. (2011, October). A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. *International Journal of Computer Applications, 31*(11). Retrieved from https://arxiv.org/ftp/arxiv/papers/1203/1203.3097.pdf

Baraglia, R., Hidalgo, J. I., & Perego, R. (2001, December). A Hybrid Heuristic for the Traveling Salesman Problem. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 5*(6), 613-622. doi:10.1109/4235.974843

Yi , S. M., Steyvers , M., Lee, M. D., & Dry , M. J. (2011). Wisdom of the Crowds in Traveling Salesman Problems.

Wikipedia, Traveling Salesman Problem - <https://en.wikipedia.org/wiki/Travelling_salesman_problem#History>

NumPy Documentation - <https://docs.scipy.org/doc/>

Pandas Documentation - <https://pandas.pydata.org/pandas-docs/stable/>

Matplotlib Documentation - <https://matplotlib.org/3.1.1/contents.html>

# Appendix