OPR 992 Project: Finding Waldo

Cassidy K. Buhler

1 Introduction

Where's Waldo, a book series authored by Martin Handford, tests the reader's ability to identify a red-and-white striped shirt wearing character in a crowded scene. There are seven books in the series, which gives a total of 68 different opportunities to spot Waldo.

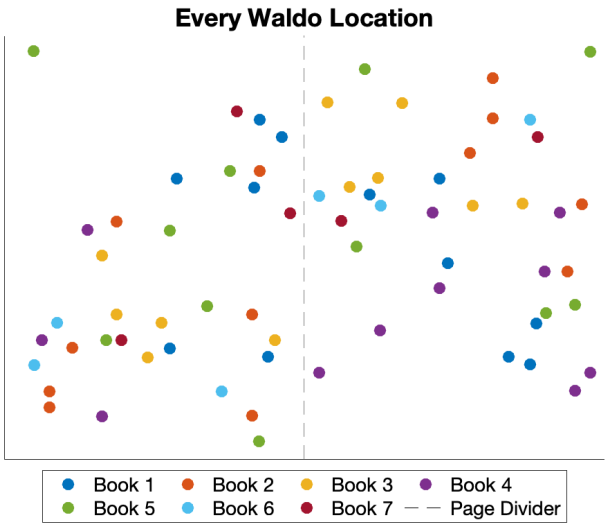

Figure 1: Graphing the Waldo locations in each book from the Waldo dataset.

Finding Waldo is often accomplished by carefully scanning the page. The audience for these books are children, so a brute force search is expected. The first popular attempt to establish a strategy of finding Waldo was posted by online magazine, Slate[3]. They claimed that Waldo is 53% likely to be in two 1.5 inch regions so readers would have a higher chance of finding Waldo if they looked in these regions first. The method was tested on 12 images in the Waldo books by comparing the duration for two people to find Waldo, one who used the strategy and one who did not. The improvement was minimal, on average the strategy took the user 14 seconds to find Waldo, compared to 15 seconds without the strategy. However, the method only claimed that there would be a 53% chance that Waldo could be found in these areas, which matches the results since it was faster in 7 out of the 12 trials.

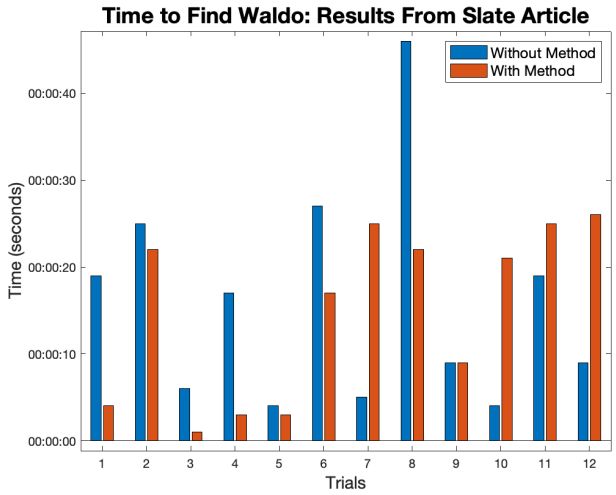

Figure 2: The results from the Slate article.

A couple years later, a researcher Randal Olsen decided to improve this method and wanted to obtain a more foolproof method of finding Waldo[16]. Olsen approaches the problem as a Traveling Salesman Problem (TSP), finding the shortest path that connects all known locations of Waldo; such a path is denoted the "optimal" search path. A reader is then able to scan a page using this path and they are guarantee to finding Waldo. Note that Olsen does not test the method by considering the time it takes to find Waldo, as the Slate article did, his objective was to find a path that would guarantee finding Waldo.

1.1 Traveling Salesman Problem (TSP)

Given a collection of cities, a salespersons wishes to determine the shortest path that travels through all the cities exactly once.

Let x_{ij} be a binary variable defined as

$$x_{ij} = \begin{cases} 1 & \text{city i connects to city j} \\ 0 & \text{otherwise} \end{cases}$$

Let the distance to travel from city i to city j be denoted as c_{ij}.

The Dantzig-Fulkerson-Johnson formulation of TSP [5] is the following

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{i \neq j, i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n \\ & \sum_{j \neq i, j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n \\ & \sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \geq 1, \qquad \forall Q \in \{1, \ldots, n\}, |Q| > 2 \end{aligned}$$

This optimization problem is known as a TSP, and is a classic NP hard problem. In other words, it returns a path, not a cycle. An exact solution can only be found with a very small dataset, using brute force $O(n!)$ or dynamic programming $O(2^n n^2)$. Therefore, unless the dataset is sufficiently small, an optimal solution must be approximated.

The origin of TSP is unknown, but records date back to the 1800s where William Rowan Hamilton [8] and Thomas Kirkman [11] initially outlining the difficulty of finding a Hamiltonian cycle. In the 1900s, Karl Menger popularized TSP and discusses brute force and nearest neighbors approach to this combitorial problem [13].

2 Methodology

In this paper, we use explore methods to solving TSP and applying it to finding Waldo. Note that we will be using the "open" variation of TSP where the solution is a tour that contains all the points but does not return to the starting point. We approach this problem with the following methods:

1. Reducing the dataset to obtain an exact solution
    - Cluster data with k-means and compute exact TSP solution for clusters centroids with dynamic programming
2. Reducing the dataset for faster solution
    - Cluster data with k-means, but visit the centroids based on the size of clusters. The largest cluster is visited first, therefore improving the probability of encountering Waldo early on the path.
3. Using heuristics on entire dataset to approximate solution
    - Genetic algorithm (GA)
    - Ant colony optimization (ACO)
    - Nearest Neighbors (NN)

While both the first and second approach cluster the data, the second approach considers the time it would take to find Waldo. In other words, since Waldo is more likely to appear early on in the path, the reader would have a higher chance of finding Waldo before completing the path and wouldn't need to search the entire path to find Waldo.

2.1 Exact Solution with Reduced Data

In this section, we use the Bellman–Held–Karp [2, 9] algorithm, a dynamic programming algorithm, to solve TSP. This algorithm only works for small datasets, so we used k-means clustering to reduce the data. Then, we used the Bellman-Held-Karp algorithm to find the shortest path through the k centroids.

2.1.1 K-Means Clustering

K-means clustering is an unsupervised machine learning method. Given k clusters, each point is assigned to a cluster. Each cluster then has a centroid, which is computed from the mean of the data points in that cluster. The goal is to assign points into k clusters such that the total distance from each point to its corresponding centroid is minimized.

Let C_j be the centroid vector of clsuter j where $j = 1, \ldots, k$, p_i is the vector for point i where $i = 1, \ldots, n$, and x_{ij} is a binary variable where it is 1 if p_i is assigned to cluster j, and 0 otherwise.

The k-means clustering problem is the formulated as the following.

$$\begin{aligned} &\min x, c \sum_{ij} ||p_i - c_j||_2 x_{ij} \\ \text{s.t. } &\sum_j x_{ij} = 1, i = 1, \ldots, n \\ &x \in 0, 1 \end{aligned}$$

The number of clusters, k, should be carefully picked. There are many methods to doing so, and I will be using the elbow graph method. Once a k is selected, we run the dynamic programming algorithm to obtain the optimal path.

2.2 Faster Solution with Reduced Data

In this section, we approach the Waldo problem slightly differently. As in the first method, we cluster the data with k-means. However, instead of running the Bellman-Held-Karp algorithm, we use a greedy approach and visit the centroids with the highest density first.

In the instance where there are clusters of the same size, we will take a brute force approach to find the shortest path among these feasible paths. While greedy approaches tend to not do so well in practice, as mentioned in section 2.2.3, the idea is that Waldo will be encountered early on in the path, thus shortening the expected length of the path.

2.3 Approximate Solution with Heuristic Methods

In this section, we explore heuristic methods for solving the problem using the entire dataset.

2.3.1 Ant Colony Optimization

ACO is a methodology based off of the behavior of ants searching for food. When ants finds food, they secrete a chemical called pheromones which attract other ants in the colony. Scientists have observed that ants will find the shortest path from their colony to their food, due to this instinct. In the literature, ACO has been applied to TSP and has found to yield favorable results [6].

ACO algorithms simulate ants exploring random walks on the data and ants choose their path based on the distance next city and the pheromone levels at that location. As ants

discover shorter paths, they deposit more pheromones, thus influencing the routes of nearby ants.

The probability of ant k moving from city x to city y is mathematically defined as

$$p_{xy}^{k} = \frac{(\tau_{xy})^{\alpha} \left(\frac{1}{d_{xy}}\right)^{\beta}}{\sum_{z \in \hat{x}} (\tau_{xz})^{\alpha} \left(\frac{1}{d_{xz}}\right)^{\beta}}$$

where τ_{xy} is the pheromone level between x and y, d_{xy} is the euclidean distance between x and y, $\hat{x}$ is the set of possible cities that x can visit, and α and β are parameters.

2.3.2 Genetic Algorithms

GA is a type of evolution algorithm that is based upon the idea of natural solution. The algorithm uses two feasible paths as candidates, and combines them to produce an offspring path. This process is repeated among the offspring paths, with the goal is that each generation will continue to produce shorter and shorter paths. Similar to nature, an offspring is not an exact copy of their ancestors as there are genetic variations due to mutation and crossover.

GA is a popular heuristic method, and due to its ability to be parallelizable, it has been commonly used to solve combinatorial problems such as TSP [19, 15, 18, 4].

2.3.3 Nearest Neighbors

NN is a simple greedy algorithm that generates a path by selecting the next city based on closest available city to the current city. It is used to solve TSP by setting each city as the initial starting city. For n cities, NN yields n paths and the shortest path among these is our solution. In the past, NN was one of the initial approaches to solving TSP. However, it has been documented that the solutions are less favorable and due to its greedy nature, there is no guarantee it will find a feasible path [1, 7].

3 Software & Hardware

To generate solutions, MATLAB and Python were used. More specifically MATLAB 2020b and Python 3.8.5. The computer processor is 3.6 GHz 10-Core Intel Core i9.

Clustering the data with K-means was accomplished with a Python package [17]. The MATLAB file exchange is where the code was obtained for Bellman-Held-Karp algorithm [12], ACO [14], GA [10], and NN [10]. The code for GA and NN were apart of the same folder, and I used "tspo_ga.m" for GA and "tspo_nn.m" for NN. The modifications I made to the code was minimial, as I changed the style of the graphed shortest path to be consistent among each method. In addition, the output of the Bellman-Held-Karp code is a cycle so I dropped the heaviest edge from the solution to make a path.

4 Results

In this section, the solutions from each method is presented.

4.1 Exact Solution with Clustered Data

First, I needed to pick an appropriate number of clusters k. To do so, computed a score for each k. Let x_i for $i = 1 \ldots, n$ be a Waldo location. C_k is a cluster center where $k = 1, \ldots, K$ for K total clusters. The score is computed as the following

$$\text{score } = \sum_{k \in K} \left(\sum_{i \in C_k} ||x_i - C_k||_2^2 \right)$$

A score is computed for each k, and we picked k based on the elbow method heuristic. For this dataset, $k = 10$ was selected.

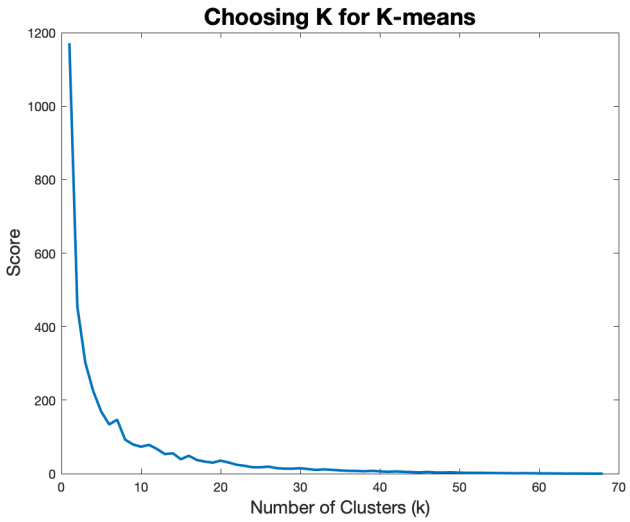

Figure 3: This graph shows the scores for every k. A lower score implies a lower error. I picked $K = 10$ since it was in the elbow of the graph.

Then, I ran the Bellman-Held-Karp algorithm on the coordinates of the 10 centroids. This gave me a path of length 27.6047.

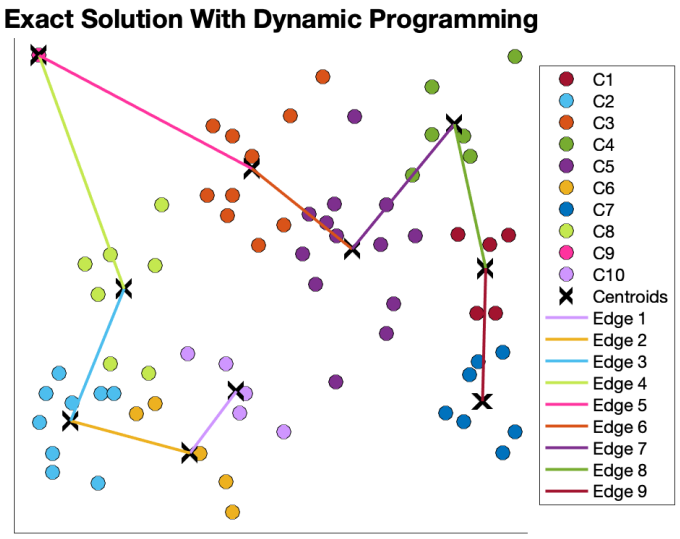

Figure 4: This graph is the Waldo data clustered with $k = 10$ clusters with each centroid is marked. The optimal path is displayed with length 27.6047.

We see that this clustering method doesn't guarantee that Waldo is located on the path, however, it would be helpful if the reader has a broad view while searching for Waldo on the path. In addition, the outlier in the top left of the page became its own cluster, so it could be faster to visit this cluster last since there is such a small probability ($\frac{1}{68} \approx 1\%$) that this cluster would contain Waldo. This result is what motivates the approach in the next section.

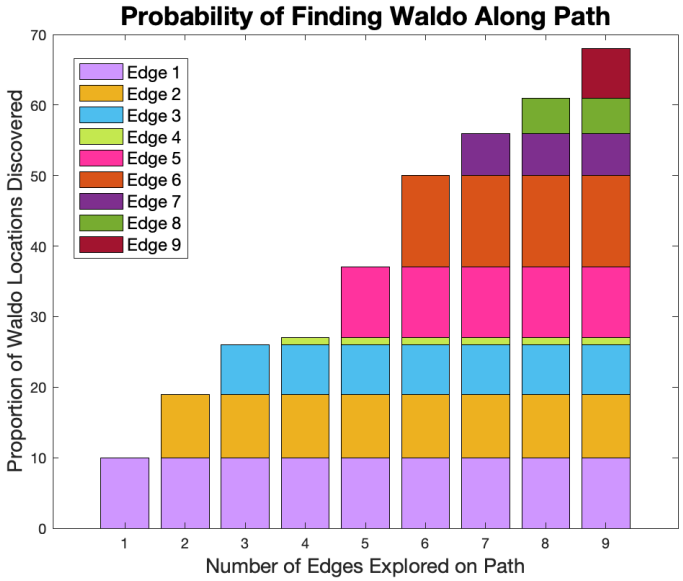

Figure 5: The size of clusters visited along the optimal path. Edge 4 provides very little information since there is only 1 point in this cluster.

4.2 Approximate Solution with Clustered Data

If we take in consideration the number of points in each cluster, we can take a greedy approach to picking the shortest path such that we encounter the higher density of points early in the path. In this instance, a user wouldn't need to search the whole path since there would be a higher probability of spotting Waldo at the start of the path.

We see in the optimal solution that edge 4 provides minimal exposure, so we would want to visit the smallest cluster last. In other words, we visit the centroids with the higher density clusters first and decrease until all centroids are visited.

Cluster Label	Size
1	5
2	9
3	10
4	6
5	13
6	5
7	7
8	7
9	1
10	5

Path	Order of Clusters Visited	Length of Path
1	5-3-2-7-8-4-1-6-10-9	55.9928
2	5-3-2-8-7-4-1-6-10-9	44.1939
3	5-3-2-7-8-4-10-1-6-9	65.5610
4	5-3-2-8-7-4-10-1-6-9	53.7620
5	5-3-2-7-8-4-6-10-1-9	64.8537
6	5-3-2-8-7-4-6-10-1-9	53.0547
7	5-3-2-7-8-4-1-10-6-9	54.6979
8	**5-3-2-8-7-4-1-10-6-9**	**42.8989**
9	5-3-2-7-8-4-10-6-1-9	64.7547
10	5-3-2-8-7-4-10-6-1-9	52.9558
11	5-3-2-7-8-4-6-1-10-9	66.9549
12	5-3-2-8-7-4-6-1-10-9	55.1559

Table 1: The table on the left contains the number of points in each cluster. On the right, there are 12 feasible paths and the shortest path is bolded.

We take the size of the clusters and rank them from largest to smallest. We see that these sizes are not unique, so we will have multiple feasible solutions that we will consider. The path with the shortest distance is then the optimal path. In this instance, there are 12 feasible paths, but path 8 has the shortest length.

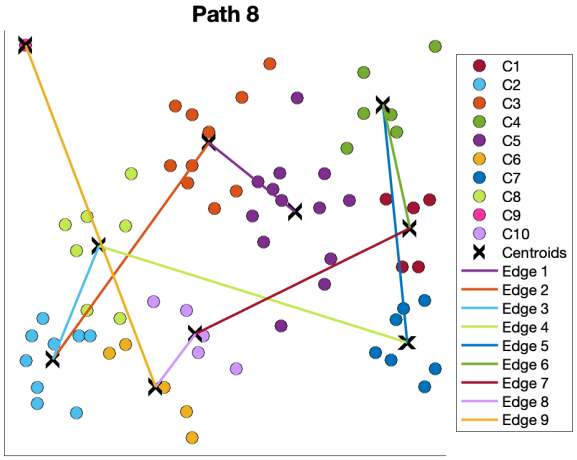

Figure 6: Displaying path 8, the shortest feasible path. It has length 42.8989.

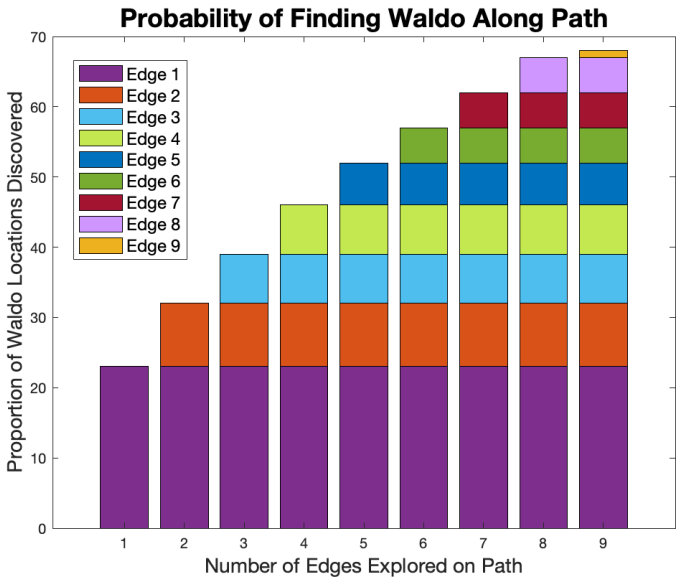

Figure 7: This bar graph displays the number of points we have been exposed to based on which edge we have explored on the path.

While the length of the path 8 is longer than the path from section 4.1, figure 7 shows that we have been exposed to over half of the locations of Waldo in just the first 3 edges. As expected, the last edge has the contains the outlier cluster and the size of the clusters we visit is non-increasing.

4.3 Approximate Solutions on Entire Data

In this section, we see how the heuristic approaches perform on the entire dataset.

4.3.1 Ant Colony Optimization

The parameters for ACO are outlined in Table 2. These parameters are the default setting for the package.

Parameter	Value
Iterations	250
Number of ants	50
Evaporation rate	0.5
α	1
β	1

Table 2: ACO parameters.

After running for 250 iterations, we see that the shortest path is 62.302. The path and pheromones are displayed in figure 8.

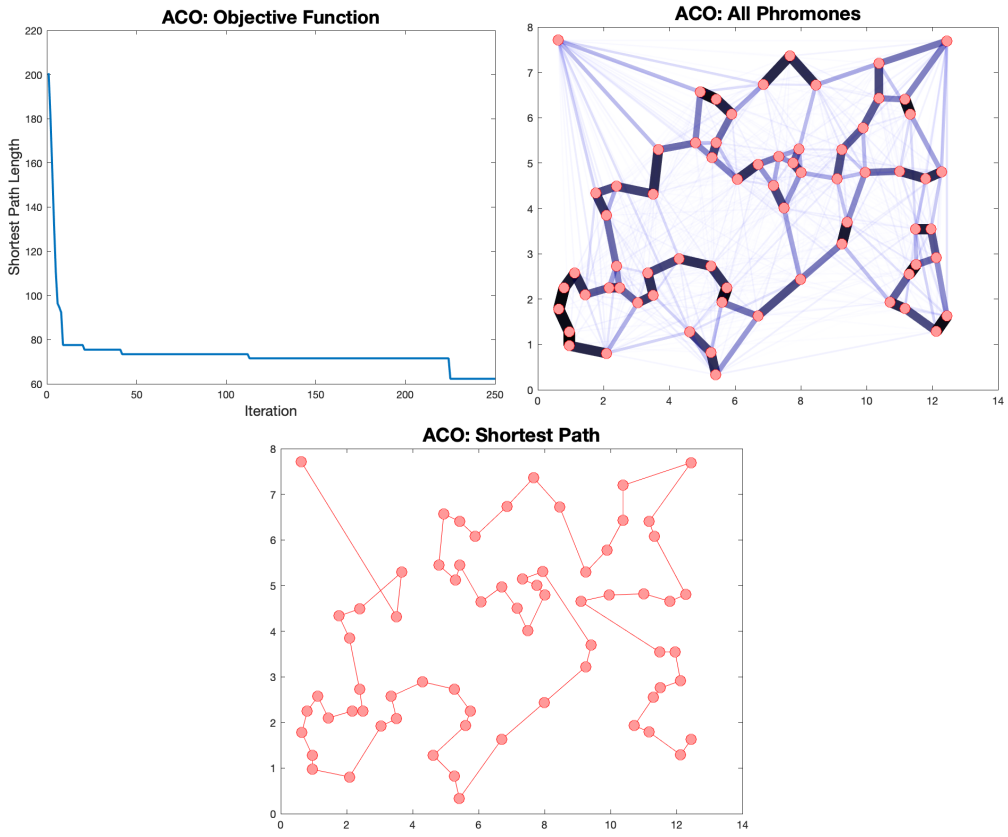

Figure 8: Results from ACO. The shortest path is displayed of length 62.3002.

4.3.2 Genetic Algorithm

The genetic algorithm did not require an input of iterations and ran until it hit a convergence criteria which occurred at 6485 iterations. This yielded a path of length 58.7690. This is an improvement from Olsen's GA result, which was 60.324768.

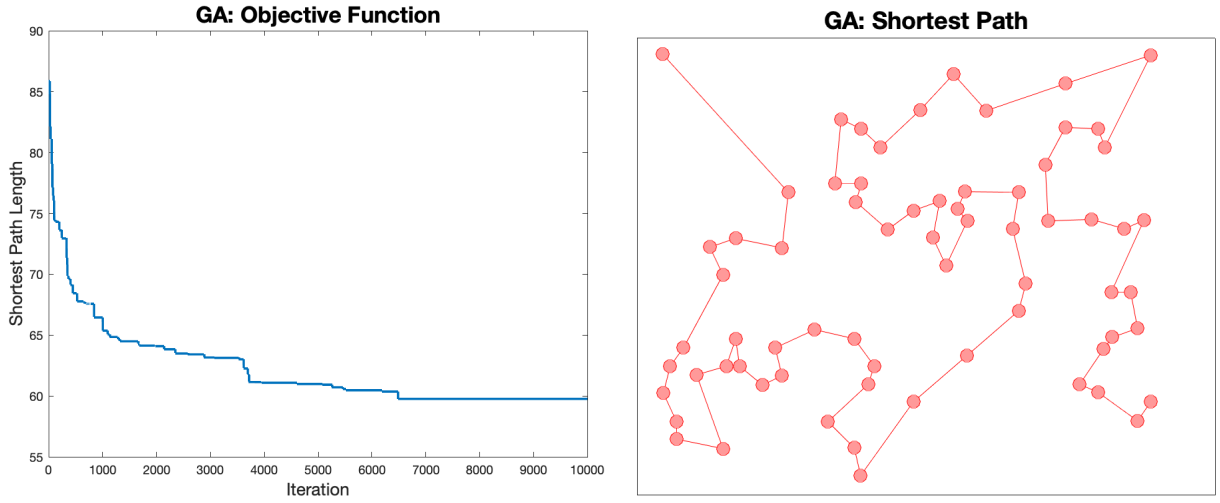

Figure 9: Results from GA.

4.3.3 Nearest Neighbors

The nearest neighbors approach gave 68 path lengths, and the shortest being of length 67.8534.

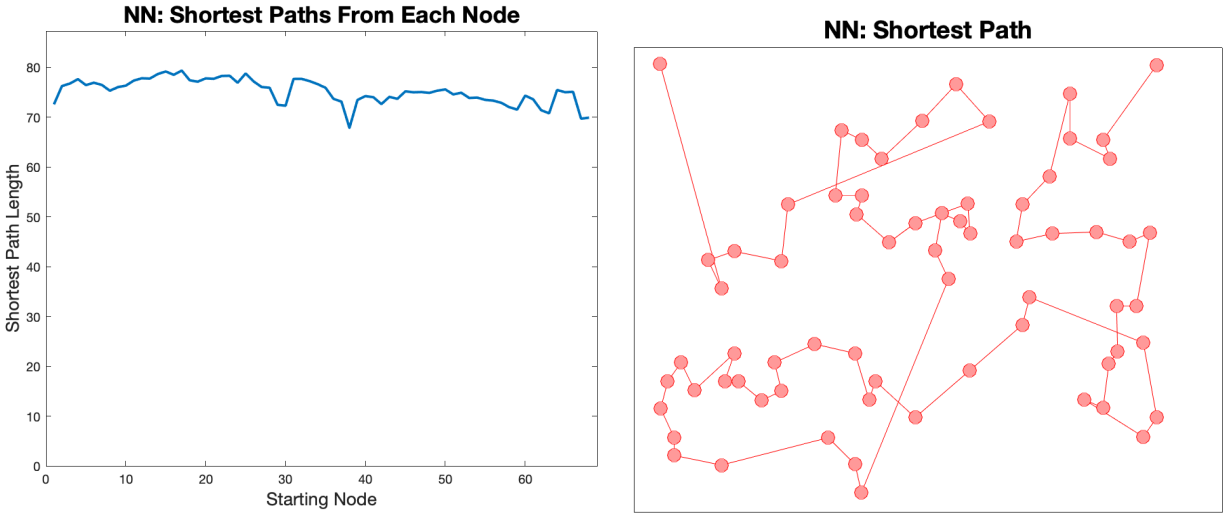

Figure 10: On the left is displayed the length of the paths starting at given node. The shortest path is displayed on the right.

The final results from the methods we explored are in the tables below.

Method	Path Length
Exact	27.6047
Greedy	42.8989

Table 3: Paths with clustered data

Method	Path Length
GA	59.7690
ACO	62.302
NN	67.8634

Table 4: Paths with full data

5 Discussion

From these results, we see that GA found the best path among the entire dataset. This implementation of GA was even a slight improvement from Randal Olsen's results. ACO had a comparable short path, and given more time I would have tinkered with the parameters to try to improve the solution further. This could be considered in future work. Lastly, we saw that the NN approach gave the longest path. This was expected, since the literature has shown that greedy is not always best [1].

With respect to the clustered data, it makes sense that the exact solution provided the shortest path since it was using an exact algorithm. However, it is not clear if the shortest path would be the fastest path. In this application, it is only necessary to search the entire path when Waldo is in the last cluster. Therefore, our results from section 4.2 would suggest that a longer length path might be worth implementing, since it is not likely a user would need to search the entire path.

In future work, it would be interesting to consider a "bang for buck" greedy approach, by vising clusters based on the size divided by distance. In addition, the greedy path had quite a few intersecting edges which may not be desirable since it scans the same region more than once, so there could be additional analysis on how to address this. Lastly, there are other methods of clustering data, so another extension could explore this further.

References

[1] Bang-Jensen, J., Gutin, G., and Yeo, A. When the greedy algorithm fails. *Discrete optimization 1*, 2 (2004), 121–127.

[2] Bellman, R. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM) 9*, 1 (1962), 61–63.

[3] Blatt, B. Here's waldo. `https://slate.com/culture/2013/11/wheres-waldo-a-new-strategy-for-locating-the-missing-man-in-martin-hanfords-beloved-series.html`, November 2013. Online; accessed 3-March-2021.

[4] Braun, H. On solving travelling salesman problems by genetic algorithms. In *International Conference on Parallel Problem Solving from Nature* (1990), Springer, pp. 129–133.

[5] Dantzig, G., Fulkerson, R., and Johnson, S. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America 2*, 4 (1954), 393–410.

[6] Dorigo, M., and Gambardella, L. M. Ant colonies for the travelling salesman problem. *biosystems 43*, 2 (1997), 73–81.

[7] Gutin, G., Yeo, A., and Zverovich, A. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. *Discrete Applied Mathematics 117*, 1-3 (2002), 81–86.

[8] Hamilton, W. R. *The Mathematical Papers of Sir William Rowan Hamilton*, vol. 3. CUP Archive, 1931.

[9] Held, M., and Karp, R. M. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics 10*, 1 (1962), 196–210.

[10] Kirk, J. Traveling salesman problem (tsp) genetic algorithm toolbox. `https://www.mathworks.com/matlabcentral/fileexchange/75525-traveling-salesman-problem-tsp-genetic-algorithm-toolbox?s_tid=srchtitle`, June 2020. Online; accessed 3-March-2021.

[11] Kirkman, T. P. On the representation of polyhedra. *Philosophical Transactions of the Royal Society of London Series*, A 146 (1856), 413–418.

[12] Kivelevitch, E. Dynamic programming solution to the tsp. `https://www.mathworks.com/matlabcentral/fileexchange/31454-dynamic-programming-solution-to-the-tsp`, May 2011. Online; accessed 3-March-2021.

[13] Menger, K. Bericht über ein mathematisches kolloquium. *Monats-hefte für Mathematik und Physik*, 38 (1931), 17–38.

[14] Mirjalili, S. Ant colony optimiztion (aco). https://www.mathworks.com/matlabcentral/fileexchange/69028-ant-colony-optimiztion-aco?s_tid=srchtitle, October 2017. Online; accessed 3-March-2021.

[15] Mühlenbein, H., Gorges-Schleuter, M., and Krämer, O. Evolution algorithms in combinatorial optimization. *Parallel computing 7*, 1 (1988), 65–85.

[16] Olson, R. Here's waldo: Computing the optimal search strategy for finding waldo. http://www.randalolson.com/2015/02/03/heres-waldo-computing-the-optimal-search-strategy-for-finding-waldo/, February 2015. Online; accessed 3-March-2021.

[17] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[18] Pfeifer, R., Schreter, Z., Fogelman-Soulie, F., and Steels, L. The dynamics of evolution and learningtoward genetic neural networks. *Connectionism in perspective* (1989), 173–198.

[19] Potvin, J.-Y. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research 63*, 3 (1996), 337–370.