Delivery Route Optimization

Team 2

Danny Suradja | dsuradja@uci.edu| 40429022

Hyun Woo (Harry) Choi | hwchoi4@uci.edu |23738187

Kenny Gao| kennyg2@uci.edu| 90542960

I. Introduction

On average, an Amazon delivery driver is tasked with stopping at 200 addresses each day. Without effective route planning or optimization, the driver is likely to pass through the same neighborhoods multiple times, retracing routes and failing to meet the daily delivery quota within the designated shift. This inefficiency leads to unnecessary time, labor, and fuel expenses, which in turn negatively affect both operational performance and the customer experience.

For this project, our team intends to implement an optimization algorithm designed to minimize the delivery cost for an Amazon driver. While a fully developed route-optimization system would consider real-world factors such as traffic conditions, weather forecasts, and road construction, we have simplified the problem for the purposes of this project. Our focus is on optimizing a single warehouse and 200 nearby addresses to create a more manageable scenario.

Initially, we planned to generate multiple sets of 200 addresses, but due to the limitations of the free quota on the Google Distance Matrix API, we chose to simplify further. Instead, we applied three different search algorithms to one set of 200 randomly selected addresses near the Irvine Amazon warehouse.

For our dataset, we initialized 2006 McGaw Ave, Irvine, CA (longitude: -117.8417, latitude: 33.6420) as "UCI Warehouse," the starting and end point of our route. The set of addresses is taken from Address Point OCPW/OCFA dataset made available on the Orange County Public Works Open Data Portal. From over one million entries in the dataset, we randomly sampled 200 unique addresses and saved them as one dataset of interest for our delivery optimization problem. We also utilized Google's Distance Matrix API on our dataset of interest to generate a 201x201 matrix showing the driving distance between each address in our dataset. The matrix is used to calculate the actual driving distance based on the best route provided by our algorithms.

The first algorithm we implemented is the nearest neighbor algorithm. In this method, the algorithm chooses the best path by choosing the address with the shortest euclidean distance to the current address. The algorithm uses longitude and latitude of the nodes and calculate the euclidean distance using the following formula:

$$distance = \sqrt{(longitude_{frontier} - longitude_x)^2 + (longitude_{frontier} - longitude_x)^2},$$

where $longitude_{frontier}$ and $longitude_{frontier}$ correspond to longitude and latitude from the frontier node while $longitude_x$ and $longitude_x$ correspond to the longitude and latitude of each unvisited node. The algorithm iterates this calculation over all addresses and returns to UCI Warehouse once all 200 addresses have been visited.

The second method is using the Simulated Annealing method. In this approach, we start with an initial route that begins and ends at the warehouse and includes all 200 addresses in a random order. At each iteration, the algorithm seeks to improve the current route by making small random modifications. Specifically, we select two addresses in the route at random and swap their positions to create a new neighboring route.

We calculate the total distance of the new route using the euclidean distance between consecutive addresses, utilizing the longitude and latitude of each node. If the new route has a shorter total distance than the current route, the algorithm accepts it as the new current route. If the new route is longer, it may still be accepted with a probability that decreases over time, determined by the temperature parameter T:

$$P(accept) = e^{-\Delta E/T},$$

where ΔE is the increase in total distance $\Delta E = distance_{new} - distance_{current}$

The temperature T starts at a high value and decreases according to a schedule, and this gradual reduction in temperature reduces the probability of accepting worse solutions over time. By occasionally accepting worse solutions, the algorithm avoids getting trapped in local minima and explores a wider range of possible routes. The process repeats until a stopping criterion is met, such as reaching a minimum temperature or completing a maximum number of iterations.

In addition to the Nearest Neighbor and Simulated Annealing methods, we implemented the A* search algorithm. This algorithm evaluates each node using two components: the actual cost of reaching the node and a heuristic estimate of the remaining cost, expressed as $f(n) = g(n) + h(n)$. In our implementation, $g(n)$ represents the driving distance between two addresses, while $h(n)$ is the Geodesic distance, which adjusts the Euclidean distance to account for the Earth's curvature. The algorithm leverages a priority queue to prioritize nodes with the lowest $f - score$, ensuring the most promising paths are explored first. By using an admissible heuristic, the A* algorithm guarantees an optimal solution.

To evaluate the effectiveness of our implementations, we established a benchmark using large-scale random sampling. Specifically, we generate 10,000 random and unique routes by randomly ordering the sequence of all 200 addresses. Each route begins and ends at the warehouse and each node can only be visited once. The shortest route out of all iterations is retained and serves as the benchmark to assess performance of our algorithms. To compare our approach with the non-optimized baseline, we use the following formula to calculate how much improvement we have:

$$\frac{Random\ Sampling's\ Shortest\ Distance - Distance\ Traveled\ from\ Algorithms}{Random\ Sampling's\ Shortest\ Distance} \times 100\%$$

We expect the formula to output a positive value, indicating a successful optimization compared to the baseline. Negative values indicate our approach does not fare better than the baseline.

We will also compare our algorithm's solutions with Google's Optimization API for Travelling Salesperson Problem as the optimized benchmark. To compare our results, we will use the following deviation formula:

$$\frac{Distance\ Traveled\ from\ Algorithms - Google's\ Optimization\ API\ Distance}{Google's\ Optimization\ API\ Distance} \times 100\%$$

Negative number from the calculation indicates our algorithm performs better than Google's Optimization API, while positive number indicates otherwise.

II. Methods and Results

1. Random Sampling

As a baseline to compare the performance and results of our more advanced algorithms, we implemented a random sampling approach as a blind search algorithm to solve the delivery route optimization problem. This algorithm does not consider the overall problem structure nor use any specific heuristic to find the solution, purely relying on randomness. Using our dataset of 200 addresses, the algorithm generated 10,000 unique permutations of routes to visit all addresses once. For every permutation, UCI Warehouse was added as the starting point and ending point, resulting in 202 addresses with 201 routes.

For each route, we calculated the total driving distance using a precomputed distance matrix by Google's Distance Matrix API. As this is an optimization problem, we are interested in finding the route corresponding to the shortest distance among all 10,000 iterations. During the random sampling process, if the total distance of a route was shorter than the previously recorded best distance, that route was stored as the best route generated by the random sampling algorithm. After completing 10,000 iterations, the algorithm found the total driving distance of the shortest route to be 5080.962 km.

Figure 1 illustrates the shortest route generated by random sampling algorithm with blue dots representing the addresses, the starred marker as the starting point and end point (UCI Warehouse), and red lines connecting each address in the order of the best route. The figure highlights the lack of any noticeable patterns with significant overlapping paths showing inefficiencies of this approach and the coherent nature of using random sampling to solve the problem. However, it serves as the benchmark when we are evaluating more advanced approaches to solving the problem.

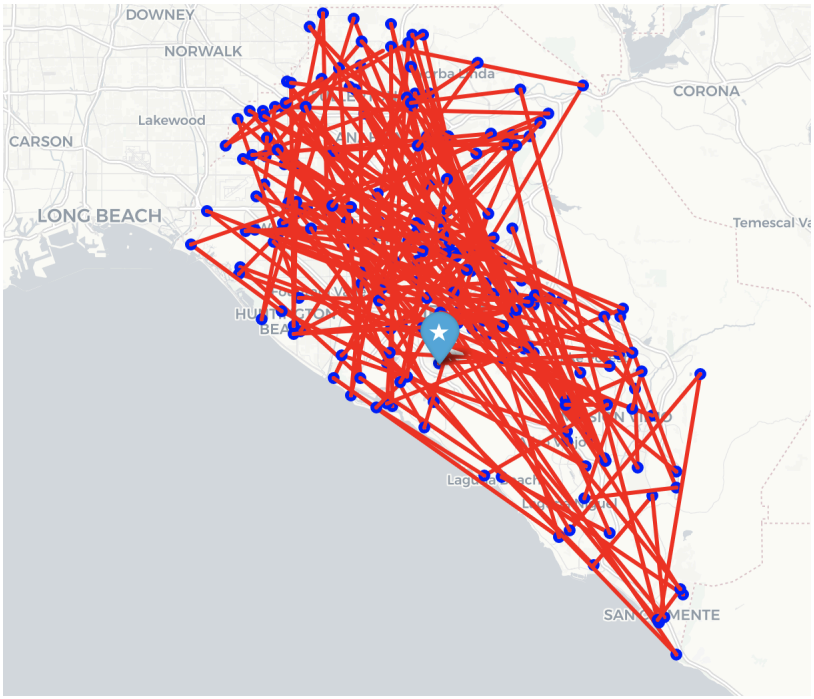

Figure 1. Best Route Generated by Random Sampling

2. Nearest-Neighbor Algorithm

The first algorithm we implemented to solve this delivery route optimization problem is the nearest-neighbor algorithm. Nearest-neighbor algorithm is a greedy search algorithm that always returns the locally optimal solution for each step of problem solving without considering the overall structure of the problem. This algorithm utilizes euclidean distance between two addresses as the heuristic approach and selects the address with the smallest euclidean distance from the current explored address as the next path. It is important to note that the heuristic used is based solely on euclidean distance between addresses and not the actual driving distance.

The algorithm begins with initializing UCI Warehouse (2006 McGaw Ave, Irvine, CA) as the initial node, and placing all 200 addresses into a group of potential next paths. The algorithm then computes euclidean distance between the initial node and each of the 200 addresses, using the longitude and latitude coordinates. The address with the smallest distance is chosen to be visited next. This corresponding address is then removed from the pool of potential paths and added into a list of visited addresses. The same methodology is repeated iteratively until there are no more potential paths (i.e. all 200 addresses have been visited). Once no addresses remain in the pool, the algorithm concludes by returning to UCI Warehouse as the end point. Throughout the process, the algorithm compiles the sequence of visited addresses and produces the route as the solution to the given dataset.

The total driving distance generated by the nearest-neighbour algorithm comes up to be 952.123 km. When compared to our baseline distance from the random sampling algorithm, there is a significant improvement of approximately 81.3%.

In Figure 2.1, there is a more structured and organized pattern when compared to the result from random search, although some overlapping routes are still apparent. Specifically, the algorithm generates a better sequence of addresses that are in close proximity to each other or clustered together.

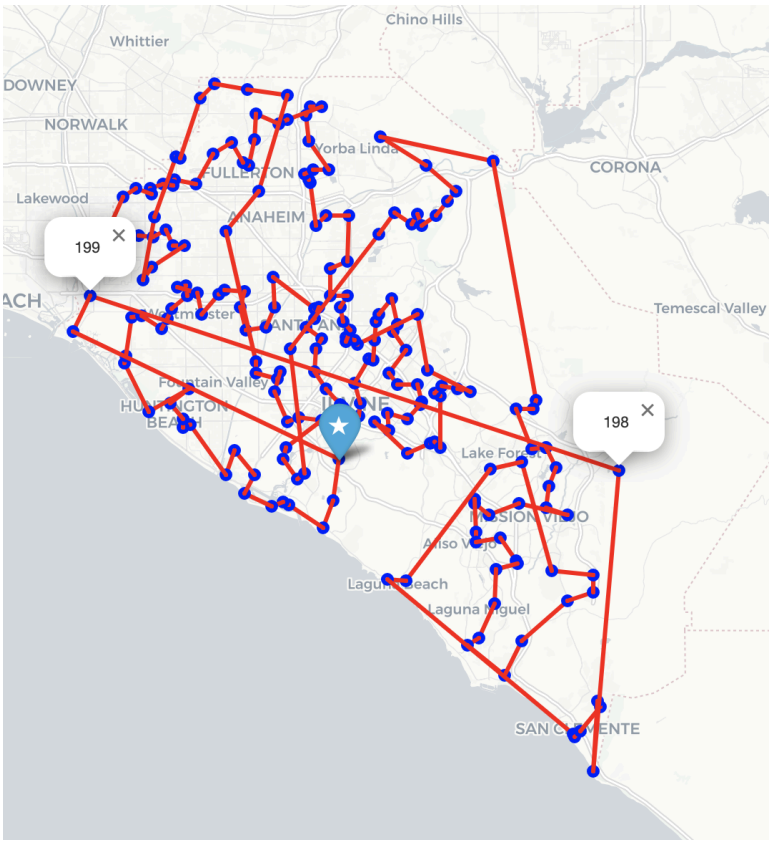

Figure 2.1. Route Generated by Nearest-Neighbor Algorithm

However, looking deeper into figure 2.1, there is a long and inefficient path extending northwest between node 198 and 199. This is a notable feature because there appears to be other addresses in closer proximity to nodes 198 and 199. Zooming in around node 199 (shown in Figure 2.2), the algorithm appears to skip node 199, as well as node 200, during its initial search around the area as the algorithm tries to find the locally best solutions for other nodes in proximity (node 17, 18, 19). This behaviour highlights the limitation of nearest-neighbor algorithm and its greedy nature which only selects the locally optimal solution (shortest distance), without looking at the overall problem scenario. For instance, node 199 and 200 were

not selected as they were not the immediate closest neighbors to node 17,18, or 19 despite being geographically close to these nodes.

The implemented nearest-neighbor algorithm does not have capabilities to revise backtrack and improve earlier decisions and route generated which leads to route inefficiencies. This is shown by nodes 198, 199, 200 being generated towards the tail end of the path as the algorithm sequentially picks up nodes they did not select earlier.

Despite its limitations, the nearest neighbor method is highly desirable for its speed and simplicity. With a time complexity of $O(n^2)$ and minimal computational overhead, the algorithm is efficient and capable of producing results quickly.

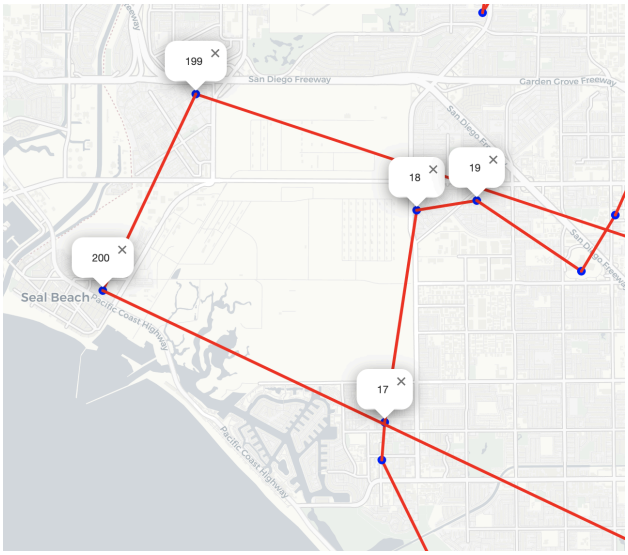

Figure 2.2. Zoomed-in View of West End Around Node 199

3. Simulated Annealing Algorithm

The simulated annealing algorithm for route optimization draws inspiration from the physical annealing process in metallurgy, where metals are heated and slowly cooled to reduce defects. In our implementation, the algorithm starts with a random initial route and iteratively improves it through a temperature-controlled process. At each iteration, it considers a modified route created by swapping two random delivery locations and decides whether to accept this new route based on both its distance and the current temperature. Higher temperatures early in the process allow the algorithm to accept worse solutions occasionally, helping it escape local optima, while the gradual cooling makes the algorithm increasingly selective about accepting only better solutions.

The simulated annealing algorithm implemented for this delivery route optimization problem demonstrates distinct characteristics when compared to both random sampling and

nearest neighbor approaches. While random sampling produces entirely chaotic paths, our simulated annealing solution shows partial optimization with more structured route segments, achieving a total distance of 1367.3 km. The visualization reveals the algorithm's attempt to balance between exploration and exploitation, as evidenced by the formation of some localized route patterns while still maintaining longer connections between different regions of Orange County.

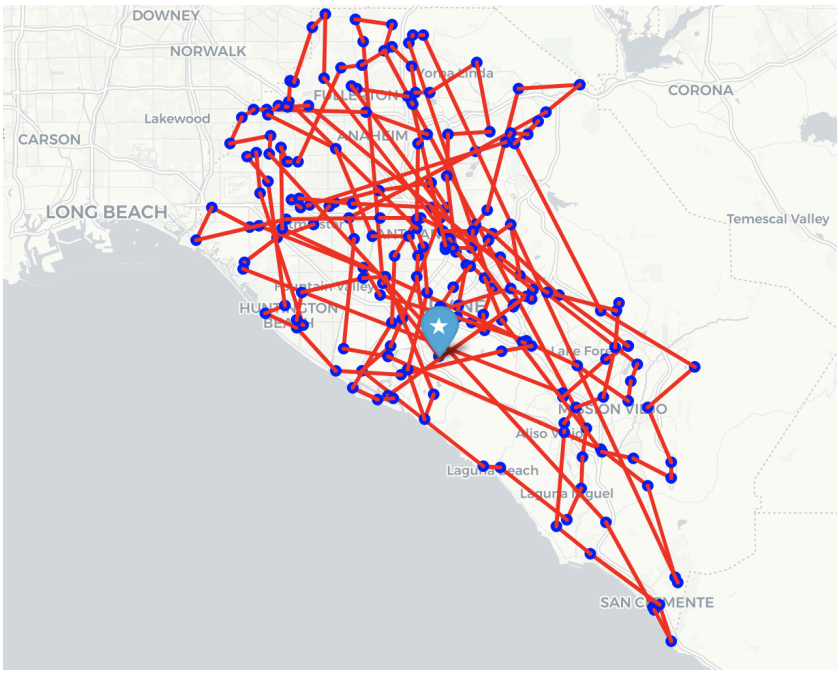

Figure 3. Result from Simulated Annealing Algorithm

However, the algorithm's performance falls notably short of the nearest neighbor approach (952.1 km), particularly in handling clustered delivery locations. The visualization shows several suboptimal characteristics, including significant crossovers in dense areas and long-distance jumps between different regions of the county. These inefficiencies are especially apparent in the coastal and inland region transitions, where the algorithm sometimes fails to optimize connections between neighboring clusters effectively. The route structure suggests that while the algorithm successfully avoids getting completely trapped in local optima, it may be over-exploring the solution space at the expense of exploiting obvious local efficiencies.

The relatively poorer performance of simulated annealing compared to nearest neighbor in our context can be attributed to the specific geographical characteristics of our problem space. Orange County's compact geography and the clustered nature of delivery locations naturally favor the greedy, locally-optimal decisions made by the nearest neighbor algorithm. Simulated annealing, with its time complexity of $O(k \cdot n)$ (where n is the number of nodes and k is the number of iterations), requires a trade-off between computational resources and solution quality. In such scenarios, where many delivery points are in close proximity to each other, the

sophisticated global optimization capabilities of simulated annealing become less advantageous than the straightforward local optimization approach. This reveals an important insight about algorithm selection: the effectiveness of an algorithm can be highly dependent on the specific characteristics of the problem space, and sometimes simpler approaches may be more practical for well-structured, geographically concentrated problems.

4. A* Algorithm

The third approach we implemented was the A* search algorithm. As we learned in class, A* uses a priority queue to explore nodes based on the sum of the actual cost to reach a node and the heuristic value estimating the cost from that node to the goal (f(n) = g(n) + h(n)). By selecting the node with the smallest f-score, A* efficiently explores the most promising paths. The algorithm updates the cost-to-reach for neighboring nodes whenever a better path is found and terminates when the goal is reached.

For our implementation, we constructed a driving distance matrix using Google's Distance Matrix API, which has dimensions of 201x201. Each index, ranging from 0 to 200, corresponds to a randomly sampled address. The rows and columns of this matrix represent the distances between pairs of these addresses. Additionally, we created a separate Geodesic distance matrix, which we used as the heuristic function. Geodesic distance, the straight-line (Euclidean) distance adjusted for the Earth's spherical shape, serves as an effective heuristic because it represents the shortest possible distance between two points on the Earth's surface. This makes it admissible, meaning it will never overestimate the true cost, ensuring the optimality of the algorithm.

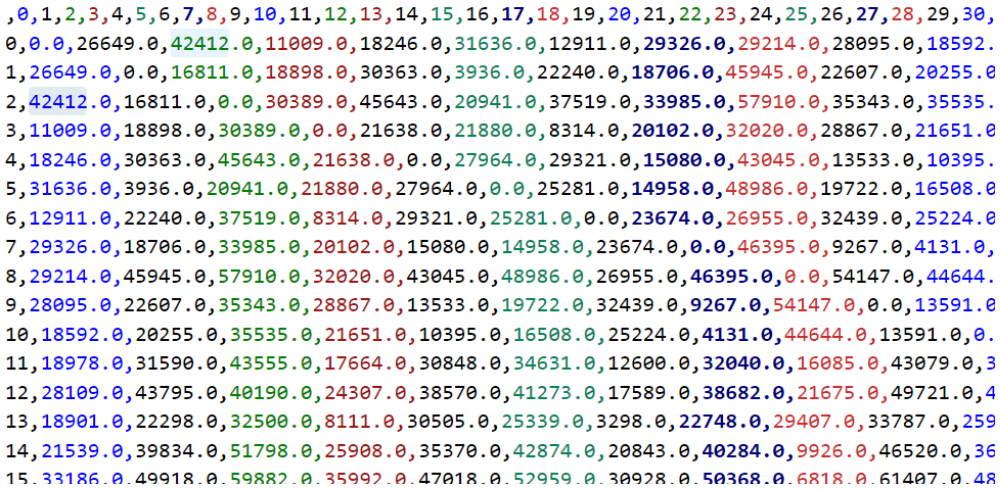

Figure 4.1. Distance Matrix Used in A* Algorithm

Here is a summary table on the implementation of A* search.

Function	a_star_search	tsp_a_star
Description	finds the shortest path between a start and goal.	tsp_a_star uses a_star_search iteratively to find the best next node at each step, eventually forming a complete round trip.
Inputs	1. start: Starting node index 2. goal: Goal node index 3. travel_distance_matrix: 2D array for actual distances (g-score) 4. geodesic_distance_matrix: 2D array for heuristic (h-score)	1. travel_distance_matrix: 2D array of travel distances 2. geodesic_distance_matrix: 2D array of heuristic (straight-line) distances
Initialization	1. Initialize priority queue: (f_score, current_node) 2. Initialize g_score for all nodes to infinity except start (0) 3. Initialize came_from: Empty dictionary to reconstruct the path	Initialize starting node, visited set, path list, total_cost
Main Loop	While there are nodes in the queue: ● Extract the node with the smallest f_score ● If the current node is the goal, reconstruct and return the path using came_from For all neighboring nodes: ● Calculate the tentative g_score (actual distance to reach neighbor) ● If the new path is better (tentative_g_score < g_score[neighbor]), update came_from, g_score, and f_score ● Push the neighbor into the queue with the updated f_score	While there are unvisited nodes: ● Use a_star_search to find the shortest path to the next unvisited node ● Update the path and total_cost, move to the next node
Termination	If goal is unreachable, return None and	After visiting all nodes, return to

	infinite cost	the starting node and update total cost
Output	Return the reconstructed path and total_cost (the total travel distance)	Return the complete route (path) and total travel distance (total_cost)

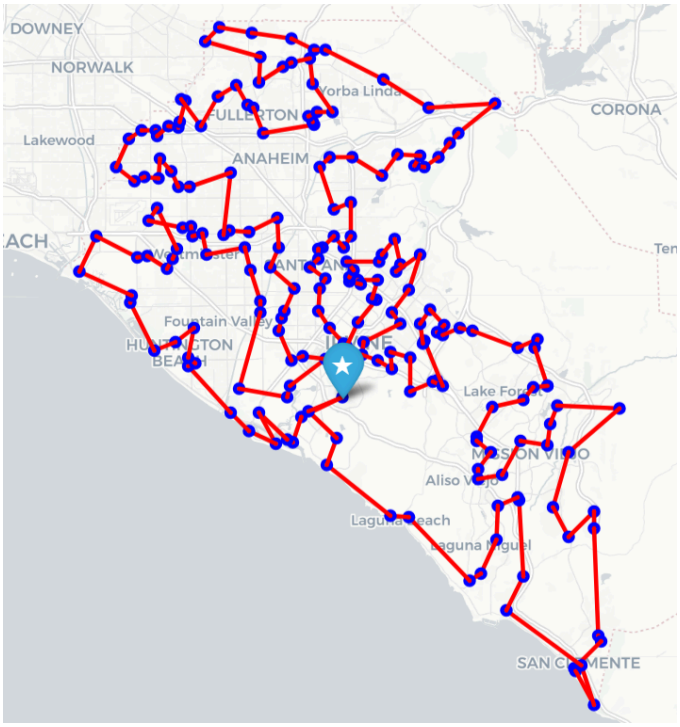

Figure 4.2. Optimal Route Generated by A* Algorithm

The route generated by A* is the cleanest and most efficient among the three approaches, with the shortest total distance of about 834 km, compared to 1381.13 km from Simulated Annealing and 952 km from Nearest Neighbor. This superior performance of A* can be attributed to a combination of its cost and heuristic functions, along with its exhaustive exploration process paired with efficient pruning. The heuristic function, in this case the geodesic distance, plays a crucial role by guiding the search towards more promising paths and avoiding irrelevant or costly ones. Since the heuristic is admissible, it significantly enhances the algorithm's ability to find an optimal path. The iteration process within A* allows for the implicit evaluation of all potential paths, exploring them one node at a time and always selecting the node with the lowest f-score. When a node is visited, A* evaluates all possible neighbors, updating the path and adding the neighbor to the queue if it results in a more favorable route.

Unlike Simulated Annealing and Nearest Neighbor, A* does not rely on random exploration of the solution space, which may not guarantee an optimal solution in a limited timeframe, nor does it greedily choose the nearest city, a strategy that can often lead to suboptimal routes. Instead, A* systematically evaluates all paths to ensure the most efficient one is found. This methodology contributes to its key properties: optimality (guaranteeing the shortest path given admissible heuristics) and completeness (ensuring a solution will be found if one exists). However, these strengths come at a cost. In the worst case, the time complexity of A* can be exponential, often O(n!), depending on the quality of the heuristic and the size of the problem. Additionally, A* can be memory-intensive, as the priority queue must store all the explored nodes during the search process.

5. Optimized Solution through Google's OR-Tools

The Google Optimization API for Travelling Salesperson Problem, part of Google's OR-Tools, provides a powerful framework for solving the delivery route optimization problem. The algorithm employs advanced optimization techniques such as Constraint Programming and Mixed-Integer Programming to compute globally optimal solutions efficiently.

For our delivery route optimization problem, the solution generated by Google's Optimization API yields a total distance of 692.334 km. As shown in Figure 5, the optimized route indicates minimal overlapping, highlighting the efficiency of this methodology in finding a highly optimized path.

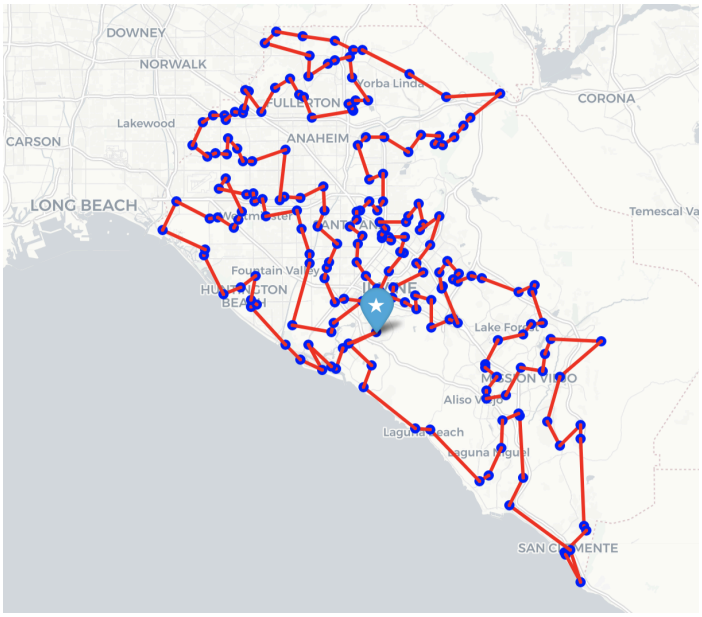

Figure 5. Optimal Route Generated by Google's OR-Tools API

III. Summary

The solutions derived from the three algorithms implemented to solve the delivery optimization problem demonstrate significant improvement over the baseline approach using random sampling. A* approach achieves the shortest driving distance with 834.351 km, showing a 83.6% improvement compared to random sampling. Nearest-neighbor algorithm results in a driving distance of 952.123 km, which is a 81.3% improvement compared to random sampling. Simulated annealing algorithm finds 1602.263 km as the solution, showing a 68.5% improvement. When compared to the optimal solution of 692.334 km derived from Google's Optimization API for Travelling Salesperson Problem, A* algorithm performs the best out of all three approaches followed by nearest-neighbor algorithm with each solution deviating by 20.53% and 37.52% respectively, while simulated annealing algorithm lags behind, deviating by 131.43%.

The dataset used in this study is relatively compact, with Orange County as the area constraint, and a considerable number of addresses located in close proximity. Under such circumstances, A* and nearest neighbor algorithms perform well due to their characteristics in finding locally optimal solutions efficiently. However, it would be valuable to explore these methods with a dataset that has a broader area boundary and more geographically dispersed. In such a case, utilizing simulated annealing may fare better due to its ability to escape local optima and explore a broader solution space. Future study could also involve testing algorithms on additional datasets with varying size and geographic distributions to provide a more robust comparison and evaluation of their performance.

IV. References

"Address Point OCPW / OCFA." Arcgis.com, 2020, data-ocpw.opendata.arcgis.com/datasets/28dca2ac9290452f836906e583313eb7_0/explore. Accessed 19 Nov. 2024.

"Distance Matrix API." Google Developers, developers.google.com/maps/documentation/distance-matrix/overview

"Traveling Salesman Problem." Google Developers, 2019, developers.google.com/optimization/routing/tsp

The algorithms for solving the delivery optimization problem, including all associated files, can be found in the following GitHub repository: https://github.com/Hyunwoo36/uci_tsp.git