

Problem Solving - A Solution for GrocerQuick - Sean Young - 2026

by Sean Young

Submission date: 16-Jan-2026 12:32AM (UTC+0000)

Submission ID: 273467820

File name: 66120_Sean_Young_Problem_Solving_-_A_Solution_for_GrocerQuick_-_Sean_Young_-_2026_692035_1476967867.docx (1.47M)

Word count: 1852

Character count: 9996

Problem Solving
A Solution for GrocerQuick

Sean Young
2026

1 CONTENTS

2	Introduction	2
3	Defining the Problem	2
3.1	Structural Decomposition	2
3.2	The MoSCoW Method	3
3.3	The Travelling Salesman Problem	4
4	Proposing a Solution	5
4.1	Process 1. Delivery Overview	5
4.2	Process 2. Vehicle Preparation	7
4.3	Process 3. Route Generation	9
4.4	Process 4. Delivery Time Estimation	11
5	Reflection	13
5.1	Problem Solving Methods	13
5.2	The Solution	13
6	References	15

2 INTRODUCTION

GrocerQuick are an online grocery service who are looking for solutions to improve the efficiency of their delivery routes, reduce fuel consumption and minimise the frequency of delayed deliveries. This paper will explore some problem-solving methods that could be applied to this problem, along with proposing a possible solution to the problem and an assessment of how problem-solving methods were applied and to what degree the proposed solution may suite GrocerQuick's needs.

3 DEFINING THE PROBLEM

3.1 STRUCTURAL DECOMPOSITION

When approaching a complex problem, it can be useful to start by breaking it down into smaller parts. [1] This is a form of structural decomposition, where problems are broken apart into smaller sub-problems. [2] This can make the problem more manageable because the sub-problems are less complex than the larger whole. [2] When each of the sub-problems have been solved, they can be pieced together to form a complete solution to the original problem. [1]

Through structural decomposition, GrocerQuick's problem could be broken down into sub-problems as following.

- Calculate the number of vehicles required for GrocerQuick's orders.
- Assign a set of deliveries to each delivery vehicle.
- Calculate time and fuel-efficient routes that cover all delivery points.
- Generate an accurate delivery prediction for the customer.
- Calculate how much fuel will be consumed by each vehicle.

3.2 THE MoSCoW METHOD

When planning a project within a limited timeframe, it is very important to clearly define the project scope. [3] The MoSCoW method can help to prioritise which are the most critical goals of a project and which goals are less critical or outside the scope of the project. [3] This is achieved by separating the possible outcomes into categories, Must have, Should have, Could have and Won't have. [3] This method is superior to simple high or low priority categorisation because the MoSCoW categories clearly define what will and won't be included in the end product. [3]

Here, the MoSCoW method has been used to identify which goals will be prioritised in the solution for GrocerQuick.

Must Have	Should Have
Generate delivery routes that include all delivery points	Calculate the number of vehicles required to deliver all orders
Estimate a delivery time, accounting for travel distance to each delivery point	Generate efficient delivery routes to minimise fuel use and travel time
Calculate the amount of fuel required for each route	Estimate a delivery time, accounting for the time to unload each delivery
Could Have	Won't Have
Generate efficient delivery routes while keeping calculation time reasonable	Calculate number of vehicles required with vehicles of varying capacity
Estimate a delivery time, accounting for 15 minutes of possible delay	Generate absolutely optimal delivery routes, regardless of calculation time
	Allow customers to choose delivery time

3.3 THE TRAVELLING SALESMAN PROBLEM

When considering the task of optimising GrocerQuick's delivery routes, it may be useful to consider the Traveling Salesman Problem, or TSP.

Calculated in the 18th century, the TSP describes the problem of finding the shortest possible route around a set of cities, covering each one before returning to the starting point. [4]

Perhaps the simplest solution for the TSP is to measure the distance of every possible route sequentially, eventually producing the absolute optimal route through brute force. [5] However, as the number of cities increases, the brute force method quickly requires an impractical number of calculations and becomes computationally prohibitive. [5] If n is the number of cities, the total number of possible routes between them all can be expressed as $(n-1)!/2$. [4] The result is that, for a set of just 10 cities, there are already more than 3.6 million possible routes. [5]

An alternative solution is to use a greedy algorithm, such as Nearest-Neighbour, where just one route is generated by plotting a path between each city in turn and selecting the closest city every time. [5] The result is a route that may not be the most efficient one possible but will be much more efficient than a random route and can be generated quickly and consistently, with a small number of calculations. [5]

4 PROPOSING A SOLUTION

The proposed delivery solution for GrocerQuick is comprised of one main process and three sub-processes. The following is a step-by-step breakdown of how these processes operate, with flow charts and diagrams for illustration.

4.1 PROCESS 1. DELIVERY OVERVIEW

1. The number of orders is divided by the order capacity of the delivery vehicles. This calculates the number of vehicles required.
2. An array is generated, with a corresponding value to represent each vehicle.
3. The delivery points are divided by the total number of vehicles. For example, if there are 4 vehicles, 40 delivery points would be divided into 4 quadrants.
4. A delivery vehicle is assigned to each set of delivery points.
5. A loop operation begins, whereby an if statement is used to query whether all of the vehicles in the array are prepared to depart.
6. If the answer is no, the next unprepared vehicle is selected and a sub-process is called to prepare the vehicle. See Process 2.
7. Once the if statement is satisfied, all vehicle are able to depart and the process terminates.

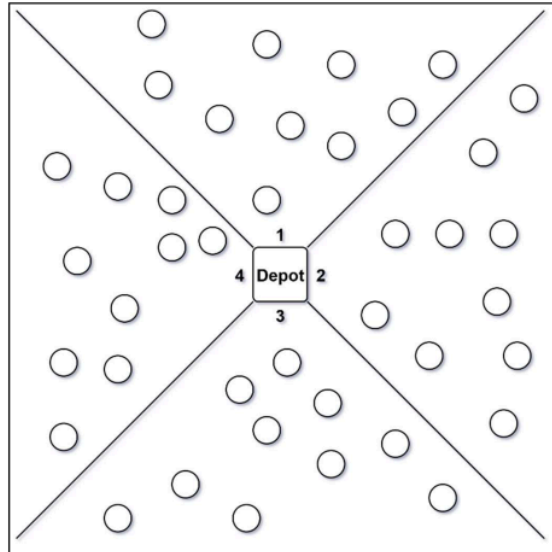
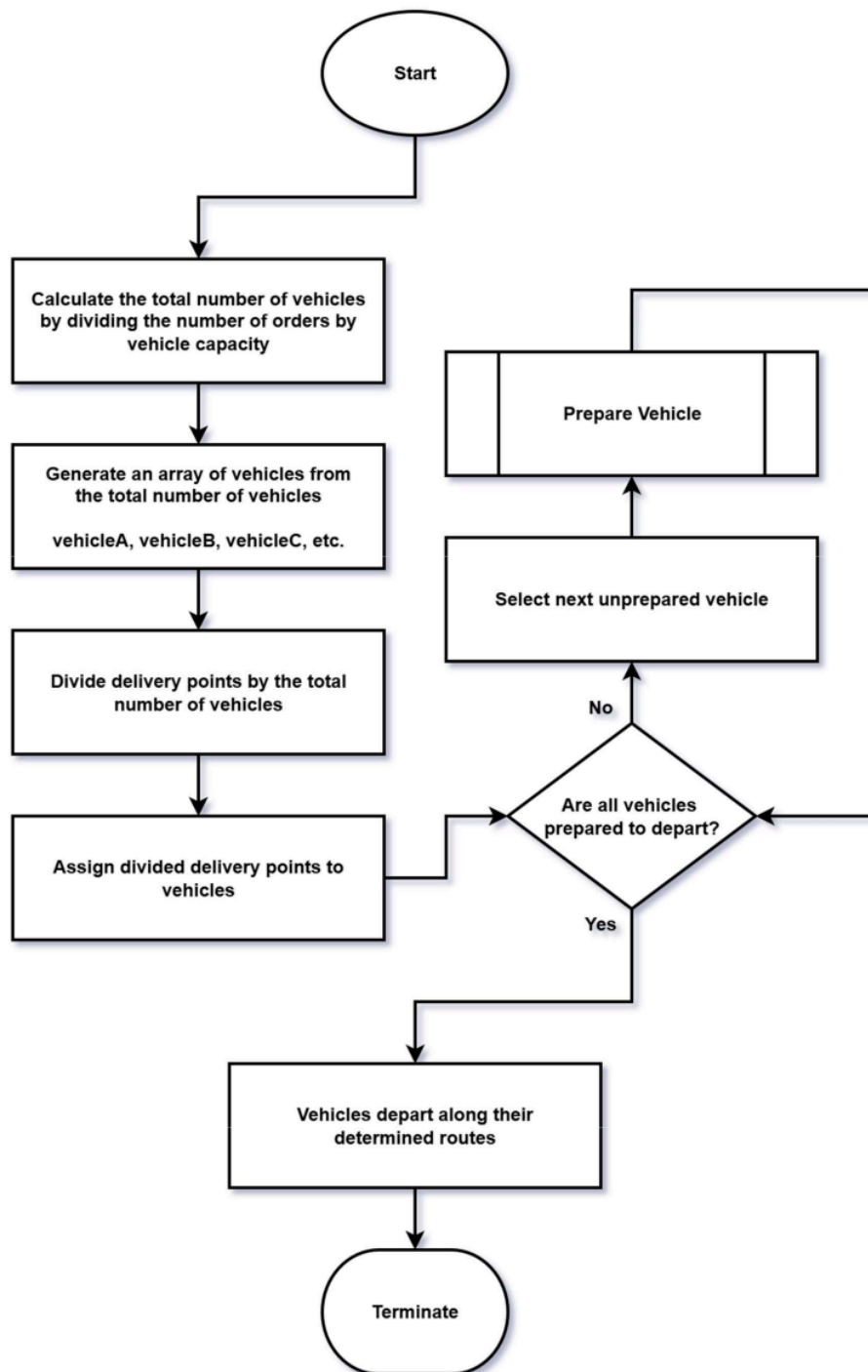
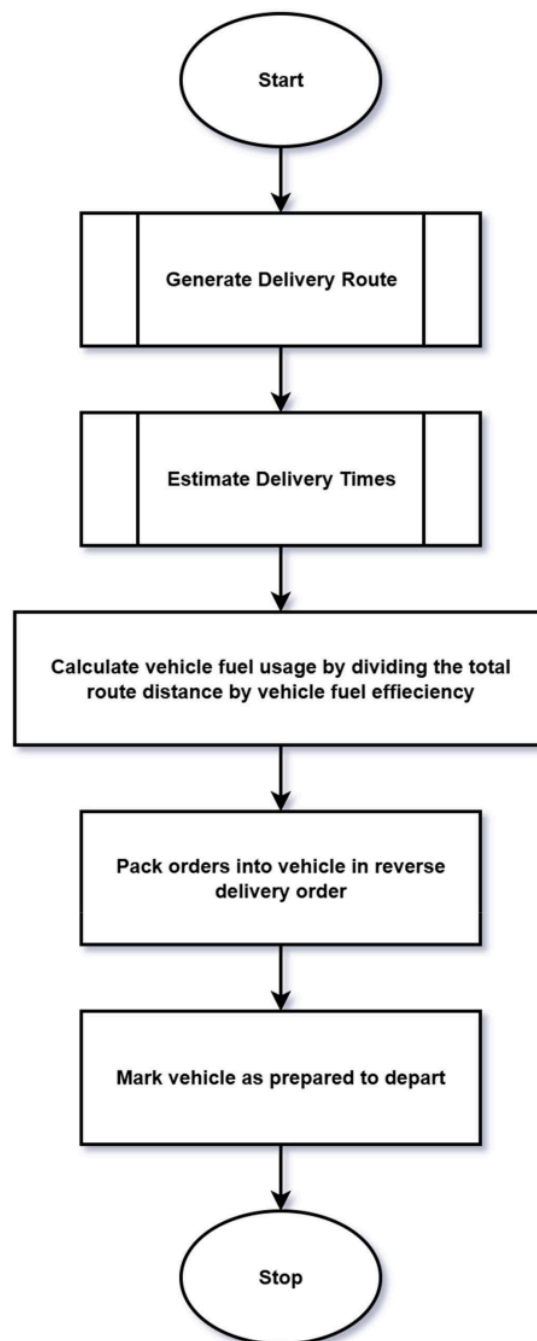


Figure 1 ~ 40 delivery points divided between 4 vehicles



4.2 PROCESS 2. VEHICLE PREPARATION

1. To start, a sub-process is called to generate an efficient route between the assigned delivery points. This process uses a variation of the nearest neighbour algorithm from the TSP, discussed previously. See Process 3.
2. Once the route generation is completed, a second sub-process uses data from the route to estimate a delivery time for each order. See Process 4.
3. The amount of fuel used by the vehicle is calculated by dividing the total travel distance of the route with the fuel efficiency of the delivery vehicle.
4. The assigned orders are packed into the delivery vehicle in reverse order, so that the order currently being delivered is never packed behind other orders.
5. The vehicle is marked as prepared to depart and the process ends, returning operation to Process 1.



4.3 PROCESS 3. ROUTE GENERATION

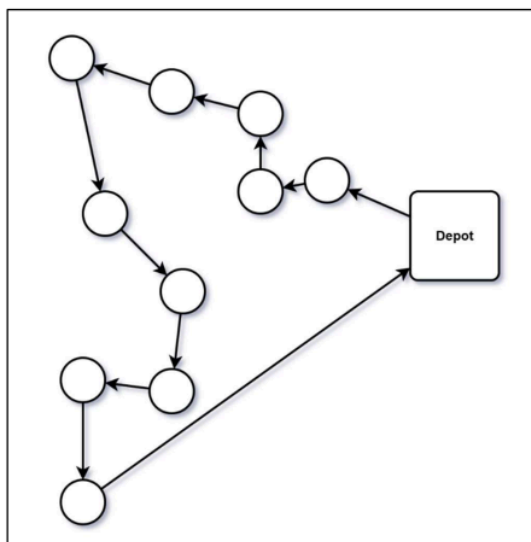
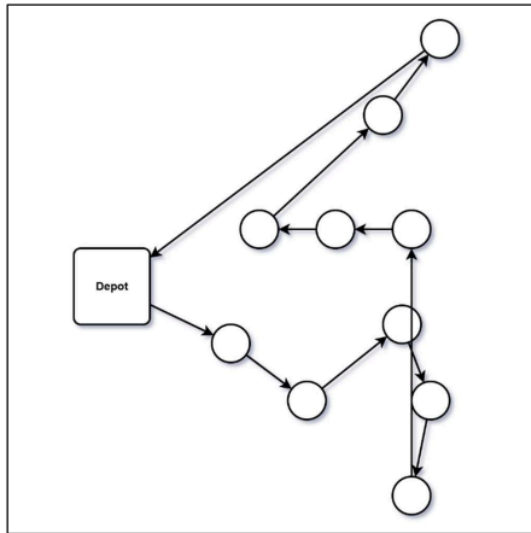
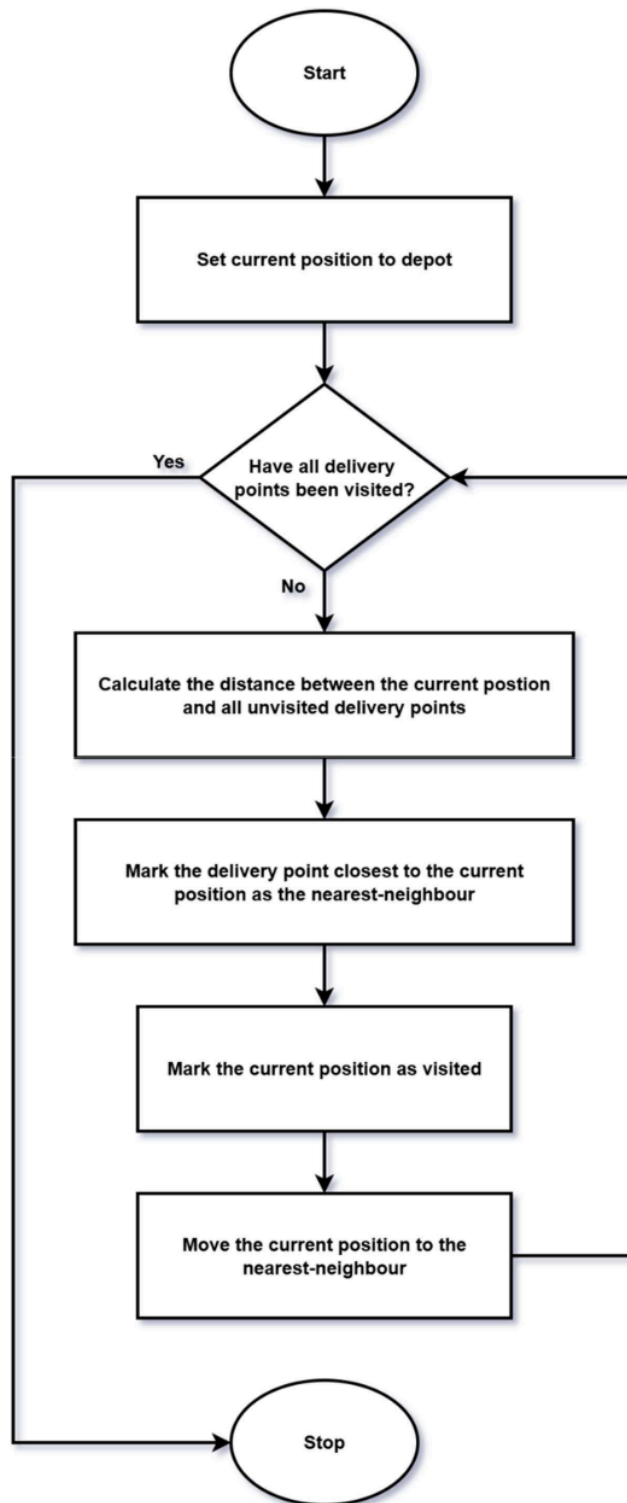


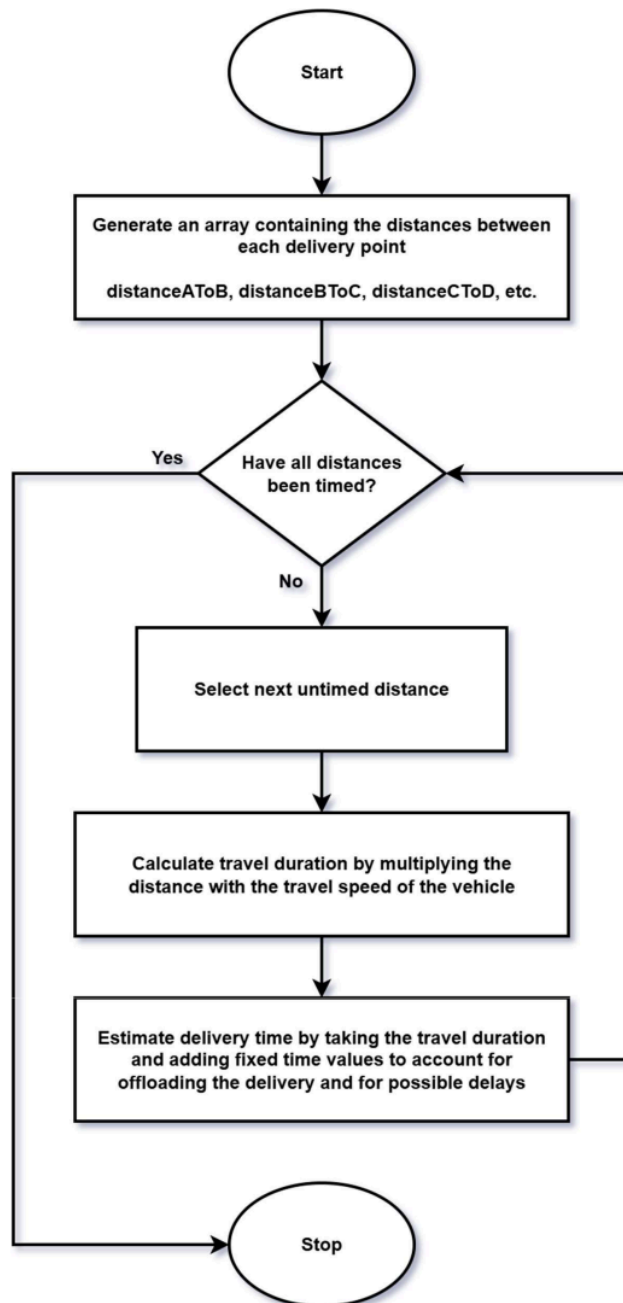
Figure 2 ~ Examples of two routes generated by this process

1. The current position is set to the depot.
2. A loop operation begins, whereby an if statement queries whether all delivery points have been visited or not.
3. If the answer is no, the distance between the current position and all unvisited delivery points is calculated.
4. The delivery point closest to the current position is selected as the nearest-neighbour.
5. The current position is marked off as visited.
6. The current position is moved to the position of the selected nearest-neighbour and the process loops back to the if statement.
7. Once the if statement is satisfied and all delivery points have been visited, the route has been successfully generated and the process stops and returns operation to Process 2.



4.4 PROCESS 4. DELIVERY TIME ESTIMATION

1. An array is generated, containing the distance between each delivery point as a route segment.
2. A loop operation begins, whereby an if statement queries if each route segment has been timed.
3. If the answer is no, the next untimed route segment is selected.
4. The travel duration of the segment is calculated by multiplying the distance with the travel speed of the vehicle.
5. The estimated delivery time is calculated by taking the travel duration from the previous calculation and adding fixed time values to account for offloading the delivery and for possible delays.
6. The process loops back to the if statement and, once it is satisfied, stops and returns operation to Process 2.



5 REFLECTION

5.1 PROBLEM SOLVING METHODS

When first approaching this problem, structural decomposition was used to break it down into smaller sub-problems. From there, it became possible to begin conceptualising the different components and how they might come together to form a working solution for the whole problem.

The MoSCoW method was used to select which goals would be prioritised and de-prioritised when constructing the solution. By sorting the project goals into clearly defined categories, it made it clear where focus should be directed. Delineating the goals that would not be met set firm boundaries for the scope of the solution.

The Traveling Salesman Problem was key to understanding some of the complexities of route optimisation. Looking at some of the algorithms that have been used to create solutions for the TSP proved extremely valuable and greatly informed the solution for GrocerQuick.

5.2 THE SOLUTION

The number of vehicles required for the deliveries was calculated by taking the number of orders and dividing it with the capacity of the vehicles. This method is simple and scalable but relies on the assumption that all vehicles have the same capacity.

Deliveries were assigned to vehicles by evenly segmenting the delivery area by the vehicle capacity. This ensures each vehicle receives an appropriate number of deliveries but does not account for the proximity of delivery points.

Routes generation was based on the Nearest-Neighbour algorithm. This requires little calculation time and generates reasonably efficient routes. However, more sophisticated algorithms, such as Ant Colony Optimisation, could yield better results.

Estimated delivery time was calculated by taking the distance between delivery points and dividing it with the vehicle's travel speed. Fixed values were added to account for unloading deliveries and possible delays. This results in reasonably accurate delivery estimates but does not allow for customers to choose their own delivery time.

Finally, fuel usage was calculated by taking the total travel distance and dividing it with the fuel efficiency of the delivery vehicle. This allows GrocerQuick to monitor fuel usage and assess whether delivery routes are cost effective.

It is the opinion of the author that the end solution meets all of the requirements laid out in the original problem but that there are multiple areas where room for more comprehensive solutions can be identified. If implemented, these would add increased functionality and flexibility that would likely be valued by GrocerQuick, however they would also greatly increase the complexity and scope of the solution.

6 REFERENCES

- [1] Ada Computer Science, "Decomposition," Ada Computer Science, [Online]. Available: https://adacomputerscience.org/concepts/compthink_decomposition. [Accessed 12 January 2026].
- [2] . P. J. Rich, G. Egan and J. Ellsworth, "A Framework for Decomposition in Computational Thinking," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, Aberdeen, Scotland, 2019.
- [3] Agile Business Consortium Limited, "Chapter 10: MoSCoW Prioritisation," Agile Business Consortium Limited, 2026. [Online]. Available: <https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html>. [Accessed 6 January 2026].
- [4] R. Matai, S. Singh and M. L. Mittal, "Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches," in *Traveling Salesman Problem, Theory and Applications*, InTech, 2010, pp. 1-13.
- [5] W3School, "DSA The Traveling Salesman Problem," W3School, [Online]. Available: https://www.w3schools.com/dsa/dsa_ref_traveling_salesman.php. [Accessed 9 January 2026].