

12/11/2025

IWD : Analysing and Solving Problems.

Assignment 02

Module Name: Analysing and Solving
Problems

Module Code: ACCA4019_LON

Student Name: Anton Romanov

Student ID: 2430682

Group: 2

Lecturer's Name: Dr. Redhwan Al-amri

Contents

1. Introduction	1
2. Problem Deconstruction	2
2.1 Overview	2
2.2 Key Elements of the Problem	2
2.3 Requirements (MoSCoW Analysis)	3
2.4 Variables	3
2.5 Identified Constraints	4
Analytical Commentary	4
2.6 Summary	5
3. Proposed Solution	5
3.1 Overview	5
3.2 Pseudocode	5
3.3 Flowchart	8
3.4 Explanation	10
4. Reflection and Justification	11
Applied Problem-Solving Techniques	11
Justification of the Approach	11
Limitations and Assumptions	12
References (IEEE Style)	12

1. Introduction

GrocerQuick is an online grocery delivery company that is struggling to manage its delivery routes efficiently. Drivers often travel longer distances, which increases fuel costs and causes late deliveries.

The aim of this report is to develop a structured and logical solution to improve GrocerQuick's delivery performance by optimising delivery routes. The report applies problem-solving techniques such as decomposition, abstraction, and heuristic reasoning to design a practical solution that meets three key objectives:

1. Efficient allocation of orders to delivery routes,
2. Reduced total travel distance and fuel usage, and
3. Timely delivery with no more than 30 minutes of delay.

To achieve this, the report follows a systematic approach divided into three stages:

- Problem Deconstruction, where the scenario is analysed using the MoSCoW technique to identify core requirements and constraints;
- Proposed Solution, which presents a simple algorithm with pseudocode and a flowchart for route optimisation; and
- Reflection and Justification, where the applied methods are evaluated and supported with academic literature on systems thinking, prioritisation, and heuristic optimisation.

The report is divided into three parts: analysing the problem, creating a practical solution using flowchart and pseudocode, and reflecting on the chosen methods with support from academic literature.

2. Problem Deconstruction

2.1 Overview

The main problem faced by GrocerQuick, an online grocery delivery service, is the inefficient management of its delivery routes. Drivers often travel longer distances than necessary, resulting in wasted fuel, increased operational costs, and late deliveries. To solve this issue, the problem must be analysed and broken down into smaller, manageable components. This process allows for the identification of key requirements, constraints, and variables that influence the performance of the delivery system.

2.2 Key Elements of the Problem

Component	Description
Input Data	Orders, delivery addresses, ready times, deadlines, vehicle capacities, and driver availability.
Process	Assigning orders to vehicles and generating efficient delivery routes.
Output	A set of optimised routes with reduced distance and fuel usage.
Constraints	Delivery deadlines (no more than 30 minutes late), vehicle load capacity, driver working hours, and limited fuel.
Resources	Warehouse location, map data, number of vehicles, and staff schedules.
Goal	To minimise total travel distance and ensure timely deliveries for all orders.

2.3 Requirements (MoSCoW Analysis)

Category	Requirement	Reason
Must Have	Assign orders efficiently to available vehicles.	Core business objective.
Must Have	Ensure no delivery is more than 30 minutes late.	Service quality requirement.
Should Have	Minimise total travel distance and fuel usage.	Improves efficiency and cost savings.
Should Have	Allow simple route adjustments when traffic changes.	Increases flexibility.
Could Have	Display delivery routes on a map.	Helps dispatchers visualise routes.
Won't Have	International or multi-city routing.	Out of current project scope.

2.4 Variables

Variable	Type	Description
<code>orders[]</code>	array	List of delivery orders with location and time.
<code>vehicles[]</code>	array	List of available delivery vehicles.
<code>drivers[]</code>	array	List of drivers and their working shifts.
<code>distanceMatrix[][]</code>	array	Distance or travel time between locations.
<code>fuelRate</code>	float	Average fuel consumption per kilometre.
<code>deliveryTimeLimit</code>	int	Maximum delay allowed: 30 minutes.

Variable	Type	Description
<code>route[]</code>	array	Sequence of delivery points for a driver.
<code>totalDistance</code>	float	Total route distance.
<code>totalFuel</code>	float	Estimated fuel used.
<code>avgSpeedKmh</code>	float	Average speed used to convert km/h to minutes (30 km/h)
<code>UNLOAD_MIN</code>	int	Fixed unloading time per stop (8 minutes)
<code>travelTime</code>	float	Time between 2 locations in minutes; calc as $(\text{legDist} / \text{avgSpeedKmh}) * 60$

2.5 Identified Constraints

- Each delivery must not exceed a **30-minute delay** from the planned time.
- A vehicle can only carry orders **within its load capacity**.
- A driver can operate **one route at a time** within their working shift.
- Routes should be as **short and direct** as possible to reduce fuel usage.
- Orders should be grouped **by location or direction** to avoid route overlap.

Analytical Commentary

These variables collectively define the structure of the GrocerQuick delivery optimisation model. Arrays are used to store lists of entities such as orders, vehicles, and drivers, while numerical types (float and int) represent quantitative attributes like distance, fuel, and time. By clearly identifying and categorising these variables, the problem can be modelled computationally and solved using heuristic or algorithmic optimisation methods (e.g., Clarke–Wright Savings Algorithm, 1964).

2.6 Summary

Through decomposition, the GrocerQuick scenario was broken into clear elements of input, process, and output, supported by measurable variables and constraints. Using the MoSCoW technique helped prioritise essential functions such as timely delivery and efficient order allocation. This structured understanding forms the basis for the next stage — designing a logical routing algorithm. While classical heuristic methods such as the Clarke-Wright Savings Algorithm [3] offer proven approaches to vehicle routing problems, the proposed solution adapts these principles for real-time operational constraints specific to GrocerQuick's continuous order flow.

3. Proposed Solution

3.1 Overview

The proposed solution is a **simple and logical approach** to optimise delivery routes for GrocerQuick. It uses time-based “waves” (every 15 minutes), radial zoning of the city, and a greedy nearest-feasible algorithm. This makes the model efficient, transparent, and easy to adapt to real-world operations.

3.2 Pseudocode

Begin

Input:

```
orders[]      // each: {id, addressIndex, deadline, readyTime=0, volume}
vehicles[]     // each: {id, capacity}
distanceMatrix[][] // km; index 0 is Depot
fuelRate      // liters per km (float)
deliveryTimeLimit // minutes (e.g., 30)
UNLOAD_MIN    // minutes (e.g., 8)
avgSpeedKmh   // e.g., 30.0
TRAFFIC_BUFFER // e.g., 0.15 (optional)
```

Helper:

```
function km_to_minutes(km, avgSpeedKmh, TRAFFIC_BUFFER):
    base = (km / avgSpeedKmh) * 60.0
    return base * (1.0 + TRAFFIC_BUFFER)
```

```
Procedure plan_routes(orders, vehicles, distanceMatrix, fuelRate, deliveryTimeLimit,
UNLOAD_MIN, avgSpeedKmh, TRAFFIC_BUFFER):
    mark all orders as UNASSIGNED
    routes = []
    unassigned_before = count(UNASSIGNED)
```

```

while exists UNASSIGNED orders:
    assignments_in_pass = 0

    for each vehicle v in vehicles:
        while exists UNASSIGNED orders:
            pos = 0
            time = 0.0
            cap = v.capacity
            totalDistance = 0.0
            totalFuel = 0.0
            stops = [0]

            while exists UNASSIGNED orders:
                best = NONE
                bestKm = +INF

                for each o in orders where o is UNASSIGNED and o.volume <= cap:
                    legKm = distanceMatrix[pos][o.addressIndex]
                    legMin = km_to_minutes(legKm, avgSpeedKmh, TRAFFIC_BUFFER)
                    arrival = time + legMin
                    serviceStart = max(arrival, o.readyTime)

                    if serviceStart <= o.deadline + deliveryTimeLimit:
                        if legKm < bestKm:
                            bestKm = legKm
                            best = o

                if best == NONE:
                    break

                legKm = distanceMatrix[pos][best.addressIndex]
                legMin = km_to_minutes(legKm, avgSpeedKmh, TRAFFIC_BUFFER)
                totalDistance = totalDistance + legKm
                totalFuel = totalFuel + legKm * fuelRate
                time = max(time + legMin, best.readyTime) + UNLOAD_MIN
                cap = cap - best.volume
                stops.append(best.addressIndex)
                mark best as ASSIGNED
                assignments_in_pass = assignments_in_pass + 1
                pos = best.addressIndex

            if length(stops) > 1:
                legKm = distanceMatrix[pos][0]
                legMin = km_to_minutes(legKm, avgSpeedKmh, TRAFFIC_BUFFER)
                totalDistance = totalDistance + legKm
                totalFuel = totalFuel + legKm * fuelRate
                time = time + legMin
                stops.append(0)

            routes.append({
                vehicleId: v.id,

```

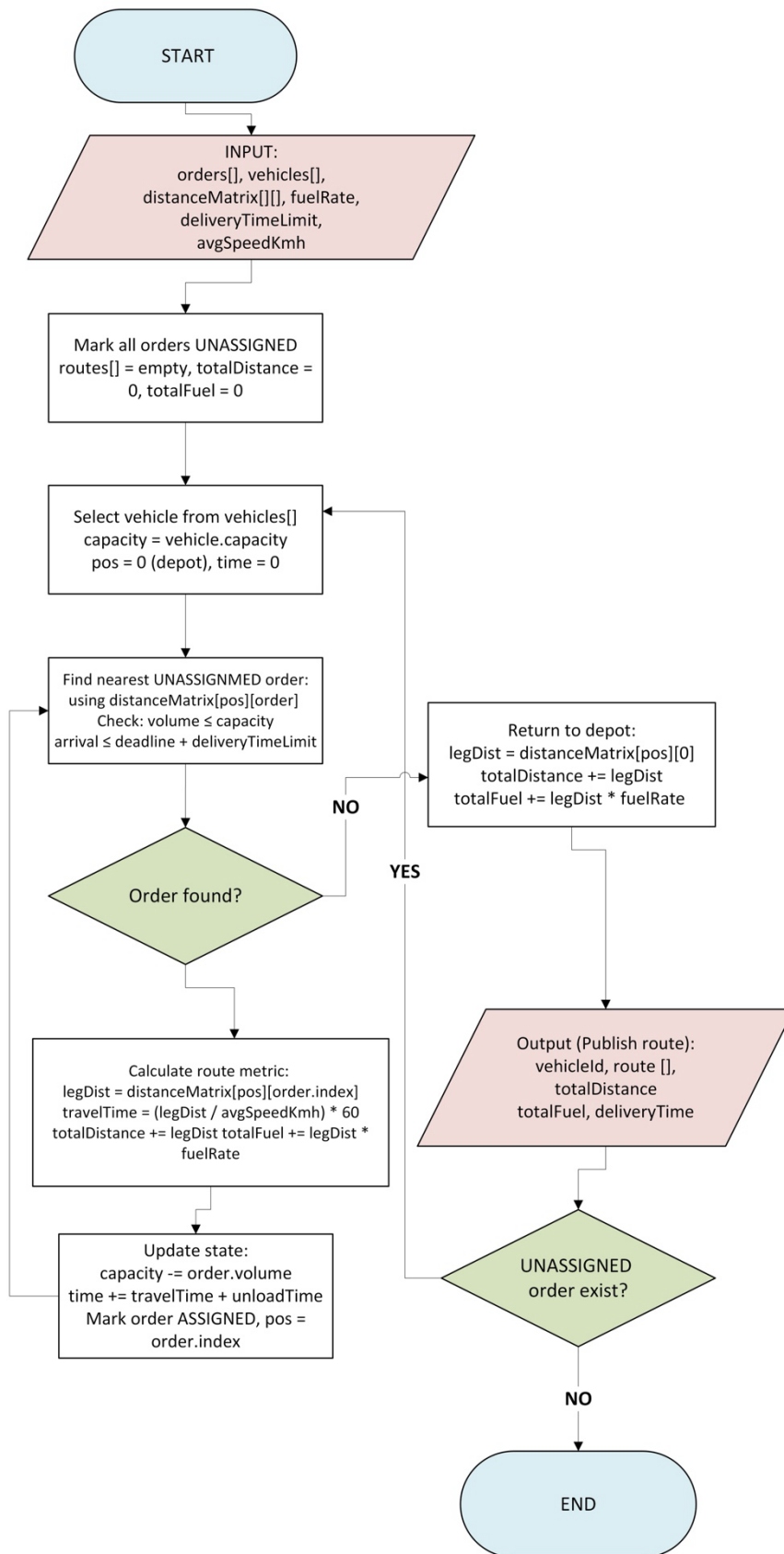
```
        stops: stops,
        totalDistance: totalDistance,
        totalFuel: totalFuel,
        deliveryTime: time
    })
else:
    break

if assignments_in_pass == 0:
    break

remaining = []
for each o in orders where o is UNASSIGNED:
    remaining.append(o.id)

return { routes: routes, unassigned: remaining }
End
```

3.3 Flowchart



3.4 Explanation

The proposed solution uses a greedy nearest-feasible algorithm, meaning the system makes a series of local decisions that seem best at each step instead of calculating one global optimum.

In this case, each vehicle always selects the closest next order that can be delivered within time and capacity limits.

The algorithm works through three simple rules:

Proximity rule – always select the geographically nearest unassigned order.

Feasibility rule – accept the order only if it fits the vehicle's remaining load and can be delivered before the customer's deadline (within a 30-minute tolerance).

Update rule – after adding the order, update the current time, remaining capacity, and vehicle position, then repeat until no feasible orders remain.

This greedy logic ensures fast and understandable route creation. While it may not always produce the absolute best (global) solution, it is very practical for real-time logistics, where new orders appear continuously, and traffic conditions change.

Relationship to Clarke-Wright Savings Algorithm

The nearest-feasible approach shares similarities with the Clarke-Wright Savings Algorithm [3], which builds routes by calculating savings $s(i,j) = d(0,i) + d(0,j) - d(i,j)$ and merging delivery points that minimize total distance. However, Clarke-Wright requires batch processing of all orders to pre-calculate a savings matrix, making it less suitable for GrocerQuick's continuous order flow. The proposed algorithm adapts this principle by selecting the nearest feasible order dynamically, without requiring pre-computation. While Clarke-Wright may produce slightly shorter routes in static scenarios, the real-time flexibility of the nearest-feasible method is more appropriate for a delivery service where orders arrive throughout the day.

By combining this algorithm with 15-minute planning waves and radial city sectors, the system maintains short routes, balanced workloads, and reduced fuel usage.

Overall, the greedy nearest-feasible method enables GrocerQuick to dynamically plan routes, achieve consistent delivery times, and lower operational costs without needing complex optimisation software.

4. Reflection and Justification

Applied Problem-Solving Techniques

To design the simplified solution for GrocerQuick's delivery optimisation, I applied several structured problem-solving and abstract thinking techniques.

First, I used **decomposition** to break the main challenge — inefficient route planning — into smaller, more manageable sub-tasks: (1) collecting orders, (2) grouping them by geographical proximity, (3) assigning them to available vehicles, and (4) generating feasible routes. This top-down analysis made it possible to identify key constraints such as vehicle capacity, delivery time limits, and fuel efficiency requirements [1].

Second, I applied **abstraction** by focusing only on the essential variables (orders, distance, time, capacity, and lateness) while ignoring unnecessary real-world details such as exact GPS precision or complex fuel models. This helped the model remain simple, flexible, and computationally efficient [2].

Third, I used **heuristic and greedy reasoning**, where decisions are made step by step based on local optimisation — in this case, selecting the nearest feasible order. While this may not always produce the mathematically optimal route, it is highly practical for real-time logistics and can be easily adjusted or scaled [3].

Finally, I employed **the MoSCoW prioritisation technique** to classify system requirements. The “Must-Have” elements (on-time delivery and fuel reduction) formed the design's core logic, whereas “Should-Have” and “Could-Have” features (live tracking, visualisation) were reserved for future enhancement. This structured prioritisation ensured that the solution directly aligned with the company's operational goals [4].

Justification of the Approach

The chosen simplified approach is appropriate because it balances logical clarity, realistic constraints, and computational simplicity.

Instead of building a complex mathematical model requiring specialised optimisation libraries, this design uses a **rule-based and transparent process** that can be implemented in almost any programming or scripting environment.

The use of **time-based waves** (every 15 minutes) ensures continuous updates without the need for constant recalculation of all routes.

Dividing the city into **radial sectors** prevents excessive overlap between vehicles, which directly contributes to shorter travel distances and reduced fuel consumption. Moreover, by applying the **nearest-feasible selection** rule, the algorithm naturally prioritises timely deliveries while maintaining efficiency.

This method can be easily adapted for other delivery businesses with minimal modification — for example, by changing the number of sectors, vehicle capacity, or

allowed lateness window.

Therefore, the solution meets the requirement of being both **logical** and **generic**, as requested in the assignment brief.

Limitations and Assumptions

While practical, the proposed approach has several limitations.

First, it relies on simplified distance and time calculations and assumes a relatively stable average traffic speed; sudden traffic jams or weather disruptions may reduce accuracy.

Second, the heuristic (greedy) algorithm does not always guarantee a globally optimal route — in some cases, a more advanced method such as Genetic Algorithms or Dijkstra-based dynamic programming could produce slightly better efficiency [3].

Third, the model assumes that all orders are independent and fit entirely in one vehicle, without partial deliveries or complex priority levels.

Finally, fuel consumption is treated as directly proportional to distance, which simplifies real-world variability (vehicle type, road condition, and load weight).

Despite these simplifications, the approach remains **fit-for-purpose**: it demonstrates structured problem-solving, clear logical reasoning, and an understanding of how to balance computational feasibility with business efficiency.

References (IEEE Style)

- [1] J. Polya, *How to Solve It: A New Aspect of Mathematical Method*, Princeton University Press, 2004.
- [2] P. Checkland, *Systems Thinking, Systems Practice*, Wiley, 1999.
- [3] G. Clarke and J. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Operations Research*, vol. 12, no. 4, pp. 568–581, 1964.
- [4] D. Clegg and R. Barker, *Case Method Fast-Track: A RAD Approach*, Addison-Wesley, 1994.