1 Introduction

In the competitive landscape of global supply chains, minimizing transportation costs is of paramount importance for logistics companies. Among these costs, fuel expenses stand out due to market volatility and the sheer volume of goods that must be transported across considerable distances. Efficiently determining where, when, and how much to refuel can yield significant savings, while also reducing the overall environmental and operational footprint associated with excessive fuel consumption or inefficient travel routes.

Although various studies have examined routing problems under refueling constraints, most focus on either limiting the number of stops or ensuring vehicles remain within feasible travel distances. Russell [6] provided one of the earliest analyses of the interplay between route selection and fuel constraints, while Bartolini et al. [2] explored a Traveling Salesman Problem (TSP) variant designed to account for refueling stops. Funke et al. [4] similarly showed the value of well-chosen stops in reducing overall costs.

Although recent work has addressed electric and alternative-fuel vehicles [1, 5], our focus is on diesel trucks with a capacity of 250 liters or more. Many real-world logistics operations still rely heavily on diesel fleets; thus, the urgent need remains to optimize refueling schedules, particularly when trucks can accommodate partial refills along their routes. By carefully selecting fueling points and volumes, operators can capitalize on varying fuel prices and mitigate the impact of price fluctuations and supply uncertainties.

In this paper, we present a novel refueling strategy designed to minimize diesel costs for trucks with tanks of at least 250 liters. Our algorithm integrates route planning with fuel decisions, ensuring that vehicles can complete deliveries efficiently while reducing total fuel expenditures. By allowing partial refills, logistics companies can fine-tune refueling decisions and adapt to dynamic market conditions, thereby achieving further cost savings over simpler full-refill approaches.

2 Algorithm

This chapter presents the design and implementation of an algorithm to compute an optimal partial-refueling strategy for a vehicle traversing a network of fuel stations. The primary objective is to minimize the total expenditure on fuel while accounting for varying station prices and a limited fuel tank capacity.

2.1 Problem Description

The underlying problem can be framed as a shortest-path search in a graph, augmented by state variables that account for fuel availability. Specifically, each fuel station is modeled as a node in the graph and the edges denote road segments with associated distances between stations. Since fuel prices differ from one station to another, a decision must be made regarding the quantity of fuel to purchase at each node.

- **Nodes (Stations):** Each station is characterized by a unique identifier, geographic coordinates, and a fuel price.
- **Edges (Distances):** Road segments connect pairs of stations, each with a known travel distance.
- **State Variables:** A state is defined by the pair $(\text{station\_id}, \text{fuel\_in\_tank})$, indicating the current station and the quantity of fuel available to travel onward.

Unlike conventional shortest-path problems, the algorithm does not merely accumulate distances but incorporates fuel purchases and consumption. This necessitates dynamic decisions on fueling strategies to minimize cost.

2.2 Algorithmic Structure

2.2.1 Data Acquisition and Preprocessing

The function `optimal_partial_refueling` initiates by establishing a connection to a PostgreSQL database to retrieve the following:

1. **Stations List:** Each entry provides a station's ID, geographic coordinates, and price per liter. A price of zero is used to denote special nodes such as the start and end stations, where fuel cannot be purchased.
2. **Station Relations:** Each record indicates the distance between two stations.

These data are used to construct:

- An adjacency list, denoted as `distance_graph`, which maps each station to all other stations, with the corresponding distances. This results in a fully connected (all-to-all) graph where every station is considered a neighbor of every other station.
- A dictionary, `station_data`, capturing pertinent attributes of each station, namely coordinates and price.

2.2.2 State-Space Representation

The core idea is to expand a search over states of the form (s, f), where s represents the station identifier and f is the current amount of fuel in the tank. At each state, a *cost* is recorded, representing the cumulative amount of money spent on fuel up to that point. By incorporating the fuel quantity as a second component in the state, the algorithm can evaluate different refueling strategies at intermediate nodes.

2.2.3 Priority Queue Exploration

To determine which state to expand next, a min-heap is employed (`heapq` in Python). Each entry in the priority queue takes the form

$$(\,\text{cost\_so\_far},\ \text{station\_id},\ \text{fuel\_in\_tank}\,),$$

thereby facilitating a best-first search analogous to Dijkstra's algorithm [3]. By always choosing the lowest-cost state for expansion, the method aims to converge rapidly on the optimal solution.

2.2.4 Refueling Policies

From the current station (s, f), the algorithm explores a set of possible *actions*, each yielding a next state and potentially modifying both the fuel quantity and the cost:

- **Travel without Purchasing Fuel:** If f is sufficient to reach a neighboring station, no new cost is incurred; only the fuel quantity f is reduced according to the travel distance.
- **Full Refueling:** If the station has a nonzero price and is neither the start nor the end station, the algorithm may opt to top off the fuel tank to its capacity. This increases the state cost by the volume of fuel purchased times the station's price.
- **Partial Refueling:** In some cases, it can be cheaper to buy only the exact amount of fuel necessary to reach a neighboring station, especially if the neighbor has lower fuel prices. The algorithm thus considers purchasing precisely enough liters to travel to the next station.

These discrete choices (no purchase, fill to capacity, or buy just enough) reduce computational overhead compared to evaluating every possible fractional amount of fuel.

2.2.5 Optimality of Discrete Refueling Actions

We argue that it is sufficient to consider only three discrete actions at each station: no refueling, refueling just enough to reach a specific neighbor, or refueling to full capacity. This is formalized in the following lemma:

Lemma 1 (Sufficiency of Discrete Refueling Actions)**.** *Let (s, f) be a state at station s with fuel level f, and suppose the algorithm chooses to purchase a fractional amount δ of fuel such that the resulting fuel level $f' = f + \delta$ is neither exactly the amount needed to reach a neighbor nor the full tank capacity. Then there exists another state (s, f'') resulting from one of the three discrete actions—(i) no refueling, (ii) partial refueling to a neighbor, or (iii) full refueling—that dominates (s, f') in cost and reachable options. Consequently, (s, f') need not be expanded in the search.*

Proof. We use mathematical induction on the number of stations to prove that an optimal refueling strategy consists exclusively of either:

1. Purchasing the minimal fuel required to reach the next station, or
2. Filling the tank to full capacity.

Base Case: For the final station s_n (the destination), no refueling is necessary. The claim holds vacuously.
Inductive Step: Assume the claim holds for all stations after s_k. We prove it holds for s_k.

Let f be the current fuel level at s_k. The driver must either:

- **Skip refueling** if $f \geq d_{k,k+1}$ (sufficient fuel to reach s_{k+1}), or
- **Refuel** by purchasing δ liters where $\delta \geq d_{k,k+1} - f$.

Consider any refueling amount δ where:

$$d_{k,k+1} - f < \delta < C - f$$

We demonstrate that such an amount is suboptimal via the following cases:

1. **Case 1: Excess fuel is not used before s_{k+1}.** If the extra fuel $\delta - (d_{k,k+1} - f)$ is not used by the next station, then it contributes no benefit. A smaller purchase $\delta_{\text{min}} = d_{k,k+1} - f$ would yield a strictly lower cost. The surplus fuel is wasted unless p_k is strictly the cheapest price among all future stations— in which case a full refill is preferable.
2. **Case 2: Excess fuel is used after s_{k+1}.** If the excess fuel is meant to reduce future purchases:
    - If p_k is not the cheapest among reachable stations, it's suboptimal to over-purchase here. The inductive hypothesis guarantees discrete actions suffice later.
    - If p_k is the cheapest, then a full tank is strictly better than any partial fill.
3. **Case 3: Intermediate purchases as a hedge.** Hedging between current and future costs by choosing an arbitrary intermediate δ is unnecessary: the inductive hypothesis ensures that optimal paths beyond s_{k+1} also use only discrete actions. Thus, either buying just enough or fully refueling always yields an equal or better total cost.

Dominance Justification. Suppose a non-discrete refueling amount δ leads to a state (s_k, f'). In each of the above cases, we showed that either $f'' = f + \delta_{\text{min}}$ or $f'' = C$ leads to a state (s_k, f'') such that:

$$f'' \geq f', \quad \text{and} \quad \text{cost}(s_k, f'') \leq \text{cost}(s_k, f')$$

Since transitions from a state depend only on location and available fuel, and all costs are non-negative, any path reachable from (s_k, f') is also reachable from (s_k, f'') at equal or lower cost. Hence, (s_k, f'') dominates (s_k, f'), and the latter need not be expanded.
Therefore, at s_k, the optimal strategy always involves either:

- Purchasing δ_{min} (if cheaper fuel is expected ahead), or
- Filling the tank fully (if current price is minimal).

By induction, the claim holds for all stations. Thus, non-discrete refueling amounts are never part of a strictly optimal solution and can be safely pruned. □

2.3 Dominance Pruning

A substantial efficiency gain is realized through *dominance pruning*. Let (s, f) be a state with cost C. This state is dominated if there exists another state (s, f') at the same station s with cost $C' \leq C$ and fuel $f' \geq f$. Because such a state (s, f') is strictly superior or equal in both cost and fuel, (s, f) need not be explored further. Consequently, dominated states are pruned to mitigate the explosion in the search space commonly associated with multi-dimensional shortest-path problems.

This is a standard and conservative form of dominance. However, a broader definition can also be considered: let (s, f) be a state with cost C, and suppose there exists a state (s, f') with cost C' such that

$$f' + \Delta f = f \quad \text{and} \quad C' + \Delta f \cdot \text{Price}(s) \leq C,$$

where Δf is an amount of fuel that can be purchased at station s at unit price $\text{Price}(s)$. In this case, the state (s, f) is also dominated by (s, f') because we could reach the same fuel level more cheaply by starting from f' and buying Δf fuel at s.

While this extended dominance rule can lead to more aggressive pruning and may reduce the number of state expansions, especially in densely connected graphs, it involves slightly more arithmetic and comparison operations. Although we did not observe or measure a significant impact on runtime, the simpler definition is often preferred due to its conceptual clarity and lower implementation complexity. Nevertheless, the extended form can still offer advantages in scenarios with large neighborhoods or high fuel price variability.

2.4 Reconstruction of the Optimal Solution

As soon as the algorithm extracts from the priority queue a state corresponding to the target station, it concludes that an optimal path has been found. Internally, a `parent` map is maintained:

$$\text{parent}[(s, f)] \;=\; (\text{prev\_station},\, \text{prev\_fuel},\, \text{liters\_bought})\,.$$

By tracing from the final state backward through this map, the algorithm reconstructs the unique sequence of refueling decisions leading to the optimal solution. The list of purchase events is then returned in correct chronological order, together with the total cost incurred.

2.5 Performance Considerations

Several factors contribute to the practical efficiency of this approach:

1. **Priority-Based Expansion:** As in Dijkstra's algorithm, a priority queue confines expansions to the most promising partial solutions, enabling a direct pursuit of cost-optimal states.
2. **Dominance Pruning:** By discarding dominated states, the algorithm effectively suppresses redundant paths and reduces the complexity of search.
3. **Discrete Refueling Actions:** Restricting the fueling actions to key increments (none, just enough, or full capacity) curtails the branching factor, making the problem tractable for large networks.
4. **Early Stop Condition:** Once the goal station is reached with the lowest cost, further expansions are unnecessary, preventing exhaustive exploration.

2.6 Algorithmic Structure

2.6.1 Pseudo-Code

To provide a structured view of the algorithm, the following pseudo-code outlines the main process:

Algorithm 1 Optimal Partial Refueling Algorithm

Require: Start station S_{start}, End station S_{end}, Initial fuel F_{init}, Tank capacity F_{max}
Ensure: Optimal refueling plan with minimal cost

Initialize priority queue PQ with $(0, S_{\text{start}}, F_{\text{init}})$
Initialize cost dictionary $cost\_so\_far$
Initialize parent dictionary $parent$
while PQ is not empty **do**
    Pop $(\text{cost}, \text{station}, \text{fuel})$ from PQ
    if station $== S_{\text{end}}$ **then**
        Return total cost and refueling plan
    end if
    for each neighbor n in distance_graph[station] **do**
        $\delta \leftarrow$ fuel required to reach n
        if fuel $\geq \delta$ **then**
            new_fuel $\leftarrow$ fuel $- \delta$
            new_cost $\leftarrow$ cost
            if new state $(n, \text{new\_fuel})$ is not dominated **then**
                Update $cost\_so\_far$ and $parent$
                Push $(\text{new\_cost}, n, \text{new\_fuel})$ to PQ
            end if
        end if
        for each discrete refueling option $\Delta \in \{\delta - \text{fuel}, F_{\text{max}} - \text{fuel}\}$ where $\Delta > 0$ **do**
            refuel_cost $\leftarrow \Delta \cdot p_{\text{station}}$
            new_fuel $\leftarrow$ fuel $+ \Delta - \delta$
            if new_fuel ≥ 0 **then**
                new_cost $\leftarrow$ cost $+$ refuel_cost
                if new state $(n, \text{new\_fuel})$ is not dominated **then**
                    Update $cost\_so\_far$ and $parent$
                    Push $(\text{new\_cost}, n, \text{new\_fuel})$ to PQ
                end if
            end if
        end for
    end for
end while
Raise error “No feasible route” if end station is not reached

2.7 Time Complexity Analysis

The performance of the algorithm is primarily dictated by the following operations:

- **Graph Construction:** The adjacency list representation requires iterating over all station relations, leading to an $O(|E|)$ complexity, where E is the number of edges.
- **Priority Queue Operations:** The algorithm maintains a priority queue for state expansion, exhibiting Dijkstra-like behavior. However, since multiple non-dominated fuel levels can exist per station, the number of distinct states $|V|$ may exceed the number of gas stations. Although the action space is limited to three discrete fueling choices, the total number of states can still grow with the number of neighbors per station. Due to this and the effects of dominance pruning, providing a tight worst-case complexity bound is non-trivial and omitted here for clarity.
- **State Expansions:** Each station may have multiple fuel states associated with it, which increases the number of states considered but is mitigated by dominance pruning.
- **Dominance Pruning:** In the worst case, pruning requires checking all stored fuel states for a given station, which can lead to $O(|V|)$ complexity. However, using efficient data structures such as balanced trees can reduce this to $O(\log |V|)$ on average.

Overall, the worst-case complexity approximates $O(|V| \log |V| + |E|)$, which is similar to Dijkstra's algorithm but with additional overhead for fuel state management. However, dominance pruning significantly reduces redundant state expansions, ensuring practical efficiency.

2.8 Illustrations

2.8.1 Graph Representation of the Problem

To visualize the fuel station network, we use a graph representation, where:

- **Nodes** represent fuel stations.
- **Edges** represent roads connecting stations.
- **Labels** indicate fuel prices and distances between stations.

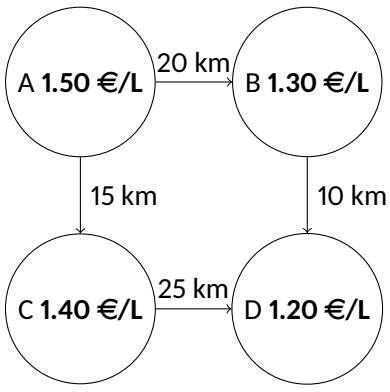


Figure 2.1: Graph representation of the fuel station network

Illustration of Discrete Actions

At each station, the vehicle has three fueling choices:

- **No refueling:** Fuel level decreases according to travel cost.
- **Partial refueling:** Just enough fuel is purchased to reach the next station.
- **Full refueling:** The fuel tank is filled completely.

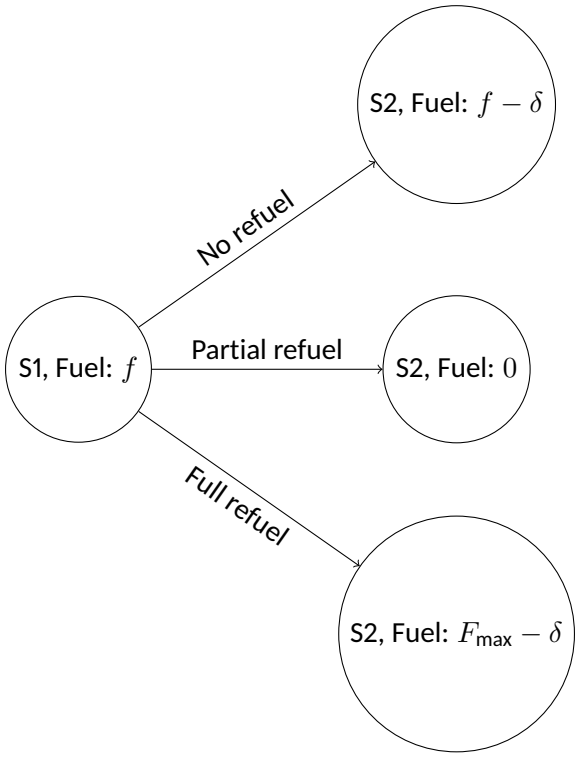


Figure 2.2: Fueling decisions at station S1, leading to different fuel states at station S2. Here, δ denotes the fuel required to travel from S1 to S2.

2.8.2 Illustration of Dominance Pruning

If two paths lead to the same station but one has a higher fuel level and/or lower cost, the dominated path is discarded.

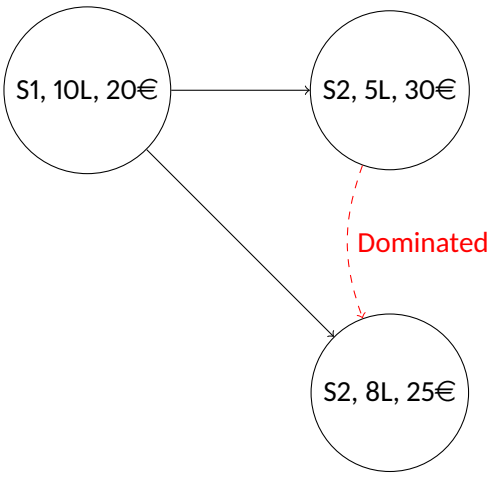


Figure 2.3: Illustration of dominance pruning

2.8.3 Conclusion

These diagrams illustrate the key concepts of the algorithm:

- **Graph representation** of the problem.
- **Decision choices** at each station.
- **Dominance pruning** to optimize the search space.

These visualizations help in understanding how the algorithm efficiently finds the most cost-effective refueling strategy.

3 Implementation

This chapter details the architectural and technical choices made to realize the partial-refueling system described in Chapter 2. The solution integrates four primary components: a relational database (PostgreSQL), a user-facing front end (Tkinter), the core computational logic (Python), and a visualization service (GraphHopper). Each subsection discusses the rationale for its respective technology and the specific manner in which it is utilized within the overall system.

3.1 PostgreSQL Database

A PostgreSQL database underpins the application by storing essential information on gas stations and their interconnections. It consists of two primary tables:

- **Gas Station Table:** Stores attributes such as the station's unique ID, latitude, longitude, name, address, and price per liter. This design allows rapid lookups of geographical and pricing data.
- **Station Relation Table:** Captures the distance between pairs of stations, reflecting their travel cost in terms of road distance. Each record contains references (foreign keys) to two station IDs and the distance metric linking them.

Figure 3.1 provides a schematic view of these tables and their relationship. By defining relevant indexes on station IDs, queries remain efficient and scalable, even for large station networks. The system is designed to query each table only a few times (three queries in total) before caching retrieved data in memory to minimize database overhead.

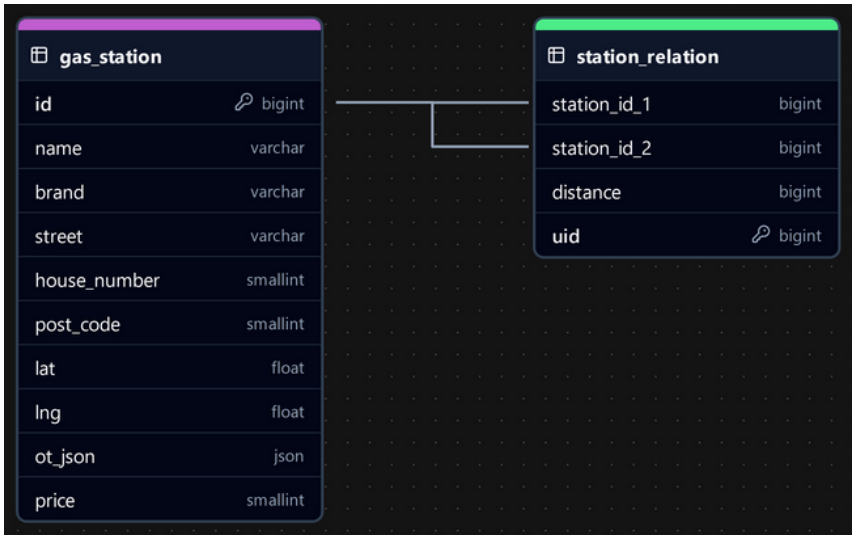


Figure 3.1: This figure shows the schema of the PostgreSQL database with two central tables: the Gas Station Table and the Station Relation Table.

3.2 Tkinter User Interface

User interactions and data entry are facilitated by a graphical interface developed with Tkinter. This framework is selected for its native integration with Python, straightforward widget abstractions, and relatively low overhead. Notable UI components include:

- **Input Fields and Dropdowns:** These widgets prompt the user to find the starting station, the destination station, the fuel tank capacity (in liters) and the current range of the vehicle (in kilometers).
- **Error Checking:** Input validation ensures that station identifiers are recognized and numeric values fall within practical limits. If an inconsistency arises (e.g., a station not found in the database), an error dialog is displayed, allowing for a direct correction without crashing the application.
- **Event-Driven Layout:** Tkinter frames and button callbacks facilitate an event-based design: upon clicking the *Calculate* button, the algorithm is invoked, and any resulting route is subsequently displayed to the user.

This design secures robust error handling and organizes user-driven processes into a comprehensible workflow and is shown in Figure 3.2.

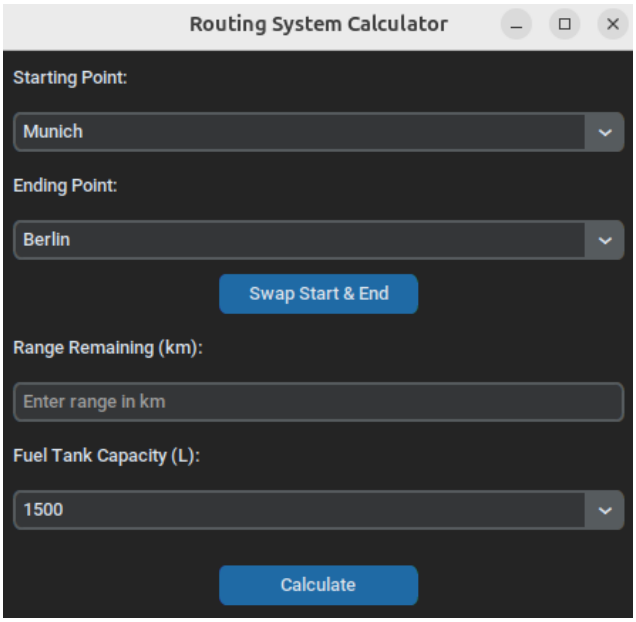


Figure 3.2: This figure shows the user interface with input fields and buttons.

3.3 Core Computation in Python

The system's business logic is implemented in Python for seamless synergy with both Tkinter and PostgreSQL. The algorithmic workflow can be summarized as follows:

1. **Data Retrieval:**On receiving valid input, the application executes a minimal set of queries to extract the relevant gas station metadata (latitude, longitude, and price) and the station-to-station distances.
2. **Partial-Refueling Algorithm:**The application invokes a dedicated function (imported from a separate Python file) to determine an optimal route under the user's constraints (capacity, current fuel range, etc.). This function applies the principles detailed in Chapter 2:

- Checking whether direct travel (i.e., no refueling) is possible.
- Otherwise, identifying one or more refueling stops such that total fuel cost is minimized.
- Raising exceptions if no valid path exists, which are subsequently caught to inform the user via the UI.

3. **Runtime Storage:**The Python process maintains in-memory structures to avoid repeating database queries. All relevant station data and intermediate results are preserved until the calculation completes.

This component-centric architecture simplifies debugging and performance optimization, as each step (query, algorithm, presentation) can be refined in isolation.

3.4 Route Visualization via GraphHopper

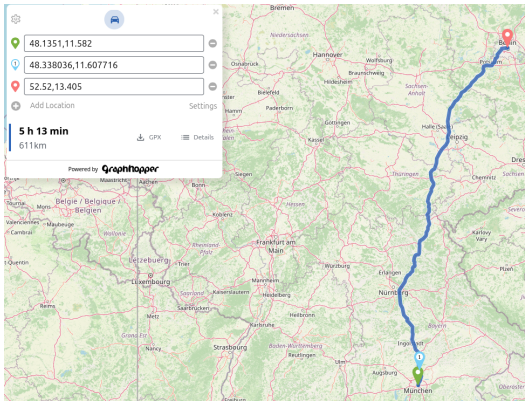
Once the final station path (and any refueling stops) are calculated, a web browser is invoked to visualize the recommended route through a locally hosted GraphHopper service. The salient details are as follows:

1. **Coordinate Assembly:**The algorithm returns a list of `(latitude, longitude)` pairs representing the starting station, intermediate waypoints (i.e., refueling stops) and the final station.

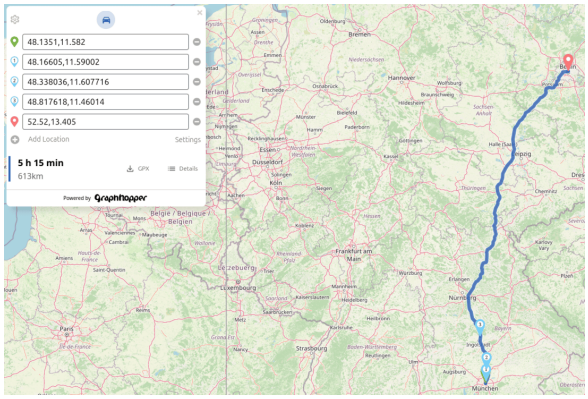
2. **Parameter Encoding:**These coordinates are concatenated into URL parameters, including a *profile* (e.g., `car`), which directs GraphHopper to compute and display a drivable path.

3. **Map Display:**The system executes the `webbrowser.open` command (in Python), passing the constructed URL to GraphHopper. Intermediate stations appear as waypoints; cost metrics are not overlaid but can be viewed in the textual output from the console.

The Python script implementing this functionality is included in Appendix 4.2.



(a) This figure shows a computed route with one refueling stop.



(b) This figure shows a computed route with three refueling stops.

Figure 3.3: This figure compares a computed route with one and three refueling stops.

3.4.1 Illustrative Use Cas

The user begins by opening the application interface, which presents a form requiring input for:

- Starting location
- Destination
- Remaining fuel range (in kilometers)
- Fuel tank capacity (in liters)

Once the user enters the required information and clicks on the "Calculate" button, the system performs an optimized database query to determine the most efficient route with necessary refueling stops. The calculation considers fuel consumption, station availability, and cost optimization.

3.4.2 Computation and Display

After the optimized route is computed, a pop-up window appears displaying:

- Number of refueling stops
- Estimated total cost of the trip

The user acknowledges this information by clicking "OK," which then triggers the system to open the route visualization in GraphHopper's web UI.

3.4.3 Interactive Map Features

The GraphHopper UI displays the computed route, including refueling stops as waypoints. The user can:

- Zoom and pan to explore the route.
- Modify the path by adding or removing stops.
- View additional route information such as distances and estimated travel times.

This approach ensures that users receive an optimized travel plan while retaining flexibility to make adjustments if needed.

Results

3.5 Introduction

This document presents a comprehensive analysis of refueling costs in the context of different algorithms for optimizing the refueling process. The goal is to evaluate the efficiency and cost-effectiveness of three algorithms – "Random", "Partial Optimization", and "Full Refill" – based on metrics such as average costs, fuel consumption, refueling efficiency, and cost variability.

The following sections provide a detailed analysis of fueling costs across different algorithms designed to optimize the refueling process. The objective is to assess the cost-effectiveness of three distinct algorithms – "Random," "Partial Optimization," and "Full Refill" – by evaluating key performance metrics, including average fueling costs, fuel consumption, refueling efficiency, and cost variability. This analysis aims to highlight the strengths and weaknesses of each algorithm, offering insights into their practical implications for cost optimization in refueling strategies.

3.6 Description of Algorithms

3.6.1 Optimal Partial-Refueling Strategy (My algorithm)

The core challenge of optimizing fuel costs during a journey through a network of fuel stations involves making strategic decisions at each station to minimize the total expenditure on fuel, given the variability in fuel prices and the vehicle's limited tank capacity. To address this, an algorithm can be designed that frames the problem as a shortest-path search in a graph, enhanced with state variables to track fuel availability.

In my approach, each fuel station is represented as a node, and the road segments connecting them are represented as edges, with the distances between stations known. The state of the journey at any point is defined by the current station and the amount of fuel remaining in the tank. The algorithm's task is to navigate this graph, making decisions at each station whether to refuel and, if so, how much fuel to purchase, to minimize the overall cost of fuel.

The algorithm uses a priority queue to efficiently explore the most cost-effective paths and employs dominance pruning to avoid redundant calculations. At each station, it evaluates several options: traveling to the next station without refueling (if possible), refueling the tank completely, or refueling partially (buying only the amount of fuel needed to reach the next station). Once the algorithm identifies the target station, it reconstructs the optimal sequence of refueling decisions to provide a step-by-step guide for the journey.

3.6.2 Full Refill Algorithm

The "Full Refill" algorithm simplifies the refueling decision-making process. Whenever a fuel station is encountered that allows the vehicle to reach the subsequent station, this algorithm decided the tank is filled to its maximum capacity. While straightforward, this strategy may overlook potential cost savings from variations in fuel prices at different stations.

3.6.3 Random Algorithm

In contrast to the strategic approaches of the other algorithms, the Random algorithm follows a deliberately arbitrary strategy. At each step, it checks how far the vehicle can travel with the current fuel level and randomly selects one of the reachable fuel stations within that range. Upon arrival at the chosen station, it refuels the tank completely. This process is repeated until the destination is reachable.

The algorithm does not consider fuel prices or optimal refueling amounts and serves primarily as a baseline to highlight the advantages of more sophisticated optimization methods.

3.7 Description of the data

To comprehensively evaluate the algorithms, their performance was assessed across all possible routes connecting the cities of Berlin, Munich, Cologne, and Hamburg. This evaluation includes key metrics, such as the number of refuels, liters purchased, and total costs, while systematically varying the initial remaining fuel (RangeRemaining) ranging from 5 to 500 kilometers and the vehicle's fuel tank capacities ranging from 50 to 1250 liters. These ranges and capacities were selected to simulate diverse real-world scenarios, covering small passenger vehicles to large commercial trucks.

The resulting dataset comprises 6,600 entries, with each entry characterized by the following columns:
Start Name: The city where the route originates.
End Name: The city where the route terminates.
RangeRemaining (km): The vehicle's initial fuel range, expressed in kilometers.
FuelCapacity (liters): The vehicle's fuel tank capacity, expressed in liters.
NumberOfRefuels: The total number of refueling stops required for the route.
LitersBought: The total volume of fuel purchased over the route, expressed in liters.
Total Cost (EUR): The total cost incurred for fuel over the route, expressed in euros.

3.8 Evaluating Refueling Strategies Over One-Leg Routes

In the following section, the performance of the three algorithms – "Random," "Partial Optimization," and "Full Refill" – is evaluated based on key metrics related to fuel cost and efficiency. The analysis focuses exclusively on one-leg routes between two cities.

Algorithm	AvgCostperRoute	AvgRefuels	AvgLitersBought	AvgLitersRemaining
Random	1186.56	1.02	530.49	467.83
Partial Optimization	100.48	1.44	66.24	4.32
Full Refill	783.63	1.03	511.57	449.54

Table 3.1: Metrics comparing three algorithms: Random, Partial Optimization, and Full Refill. Partial Optimization shows the lowest cost, with slightly more average refuels.

Algorithm	CostperLiter	RefuelEfficiency	CostStandardDeviation	CostMedian
Random	2.24	522.64	924.14	1084.5
Partial Optimization	1.52	45.85	77.58	95
Full Refill	1.53	498.2	597.62	748

Table 3.2: Continuation of metrics comparison.

3.8.1 Cost Comparison and Distribution Analysis

As shown in Table 3.1, the Partial Optimization algorithm demonstrates the most cost-effective performance, achieving the lowest average and median fueling costs per route among all strategies. While it involves a slightly higher frequency of refueling stops compared to the Full Refill approach, this trade-off appears beneficial in terms of overall cost savings. The Random algorithm, by contrast, results in the highest costs, both in average and median terms, underscoring the inefficiency of a non-strategic, arbitrary refueling pattern.

These trends are further illustrated in Figure 3.4, which presents a box plot of cost distributions across all evaluated routes. The Full Refill algorithm shows a significantly higher median fueling cost and a wider distribution, highlighting its variability and unpredictability in fuel expenditure. The Partial Optimization algorithm, on the other hand, exhibits a much narrower cost range with a consistently lower median, suggesting greater stability and cost control across different route configurations and vehicle types.

This observation is supported by Figure 3.5, a histogram of the two fueling cost densities. Here, the Partial Optimization algorithm's costs are tightly clustered in the lower price range, with a sharp peak indicating high frequency of minimal-cost routes. The Full Refill algorithm shows a broader, flatter distribution, reflecting both greater cost dispersion and a pronounced tail of high-cost scenarios. This spread confirms that Full Refill is more susceptible to cost inflation, particularly in cases where unnecessary full refills are made, e.g. near the destination.

Further supporting these findings, the Cost Standard Deviation, shown in Table 3.2, quantifies the variability in the costs of the route. The random algorithm exhibits the highest cost variability, indicating significant unpredictability in its results. In contrast, Partial Optimization demonstrates the lowest standard deviation, signifying its capacity for consistent and predictable cost control. This statistical insight aligns closely with the distributional patterns observed in Figures 3.4 and 3.5, reinforcing the conclusion that Partial Optimization not only reduces average costs but also minimizes variability between scenarios.

The consistently superior performance of Partial Optimization can be attributed to its adaptive refueling logic, which selectively purchases only the fuel required to reach the next favorable station rather than refilling indiscriminately. This targeted behavior allows it to avoid overfilling and better align the refueling quantities with the actual travel demand, thus reducing overall costs.

However, Table 3.1 also shows that the Full Refill algorithm tends to leave a larger amount of fuel in the tank at the destination of the route, compared to the Partial Optimization strategy. Although this has little effect on single-leg routes, it can be a drawback on chained routes. Full Refill often leaves more fuel at the destination, potentially delaying the need to refuel. In contrast, Partial Optimization typically ends with an almost empty tank, which may require an immediate refill at the start of the next route, regardless of fuel price or station quality. This limitation is explored further in the next section on multi-leg route performance.

In terms of fuel consumption, the data further reinforce these findings. The Random algorithm results in the highest average fuel usage, likely due to inefficiencies and non-optimal detours. Partial Optimization, in contrast, yields the lowest total liters bought, again demonstrating its alignment with real-time fuel needs. The Full Refill strategy falls in between, consuming more fuel than necessary due to its systematic tendency to overfill, even when only partial amounts are needed.

In summary, the analysis confirms that Partial Optimization provides consistently lower and more predictable costs, driven by efficient, adaptive behavior. Although it requires slightly more frequent stops, it effectively balances fuel quantity with pricing dynamics, making it the most economically favorable strategy for route-based fueling optimization.

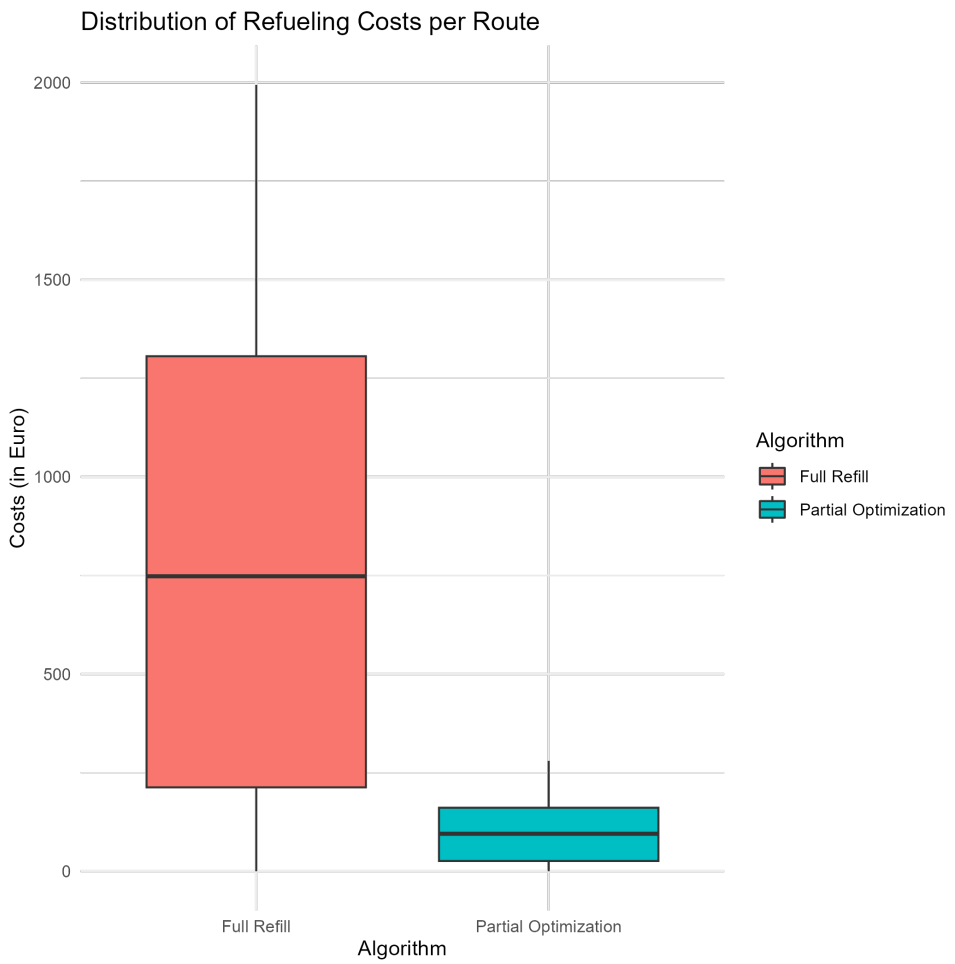
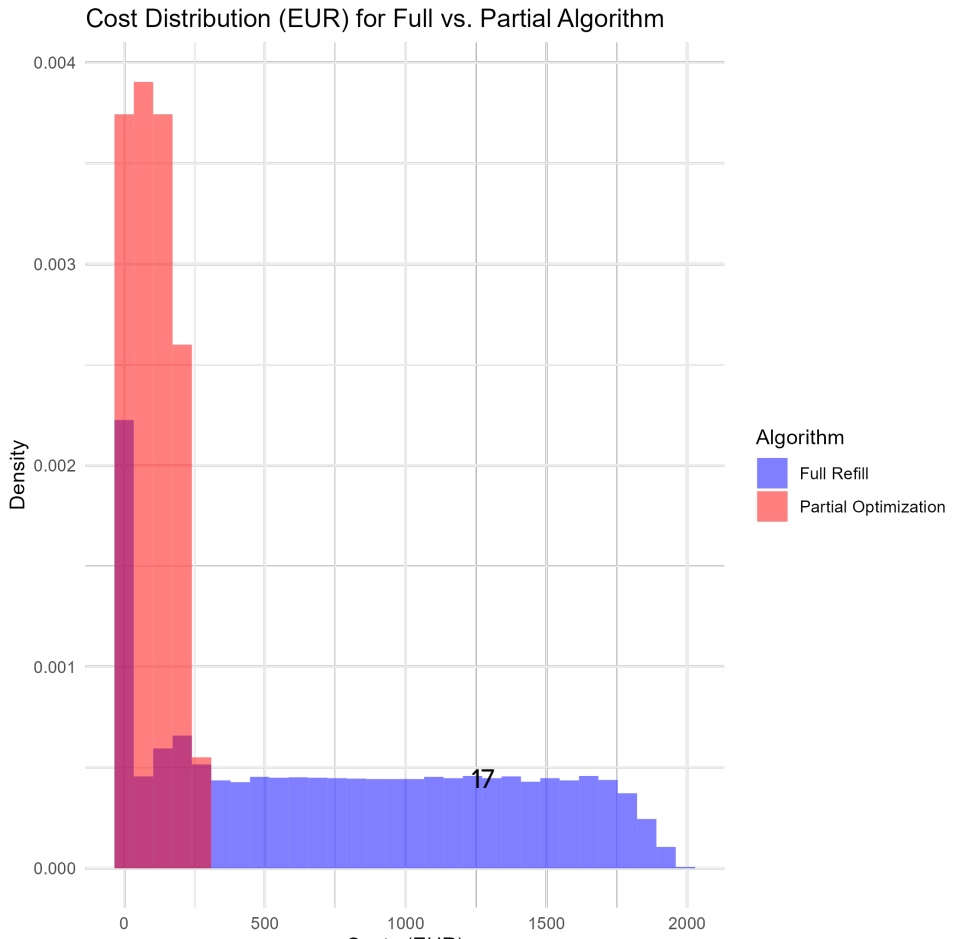


Figure 3.4: Distribution of fueling costs per route for 6600 different combinations of Routes, Fuel Capacities and Ranges Remaining: The average costs of the partial optimization algorithm are lower on average with a narrower cost distribution.



3.8.2 Refueling Efficiency, Frequency, and Operational Strategy

An important dimension in evaluating refueling strategies lies in the efficiency and frequency of fueling operations, which directly affect both cost structures and route behavior. A key metric in this context is Refuel Efficiency, calculated as:

$$\left[\frac{\sum \text{LitersBought}}{\sum \text{NumberOfRefuels}}\right]$$

This metric represents the average volume of fuel purchased per stop and provides insight into whether a strategy favors large, infrequent refuels or smaller, more frequent ones. A higher value suggests that the algorithm tends to fill the tank more fully at each stop, while a lower value indicates more granular refueling behavior.

As shown in Table 3.2, the Full Refill algorithm exhibits the highest refuel efficiency, consistently maximizing the volume added at each stop. This strategy aligns with its core principle of minimizing the number of stops by refueling fully whenever possible. In contrast, the Partial Optimization algorithm shows the lowest refuel efficiency, reflecting its targeted approach of adding only the amount of fuel required to reach the next strategically chosen station. While this leads to more frequent refueling, it enables finer control over fuel purchases and can capitalize on price variations across stations.

This trade-off is further illustrated in Figure 3.6, which compares the average number of refueling stops per route. The Full Refill strategy averages around one stop, whereas the Partial Optimization strategy averages between 1.4 and 1.5 stops. This difference highlights a fundamental difference of the algorithm designs: Full Refill prioritizes convenience and fewer stops, while Partial Optimization pursues cost efficiency through flexible, demand-matched fueling.

The increased frequency of refueling in Partial Optimization should not be misunderstood as inefficiency. On the contrary, the algorithm's smaller, targeted refuels are intentionally designed to optimize cost rather than minimize stops. Together with prior findings, particularly its lower total costs and reduced cost variability, this suggests that frequent but strategic fueling operations can yield better financial outcomes.

Ultimately, these observations demonstrate that refueling frequency and efficiency must be considered in the broader context of cost control. While Full Refill achieves higher per-stop efficiency, it may lead to overfilling at unfavorable prices. Conversely, Partial Optimization, despite more stops, leverages a more adaptive and price-sensitive approach, leading to greater overall economic efficiency.

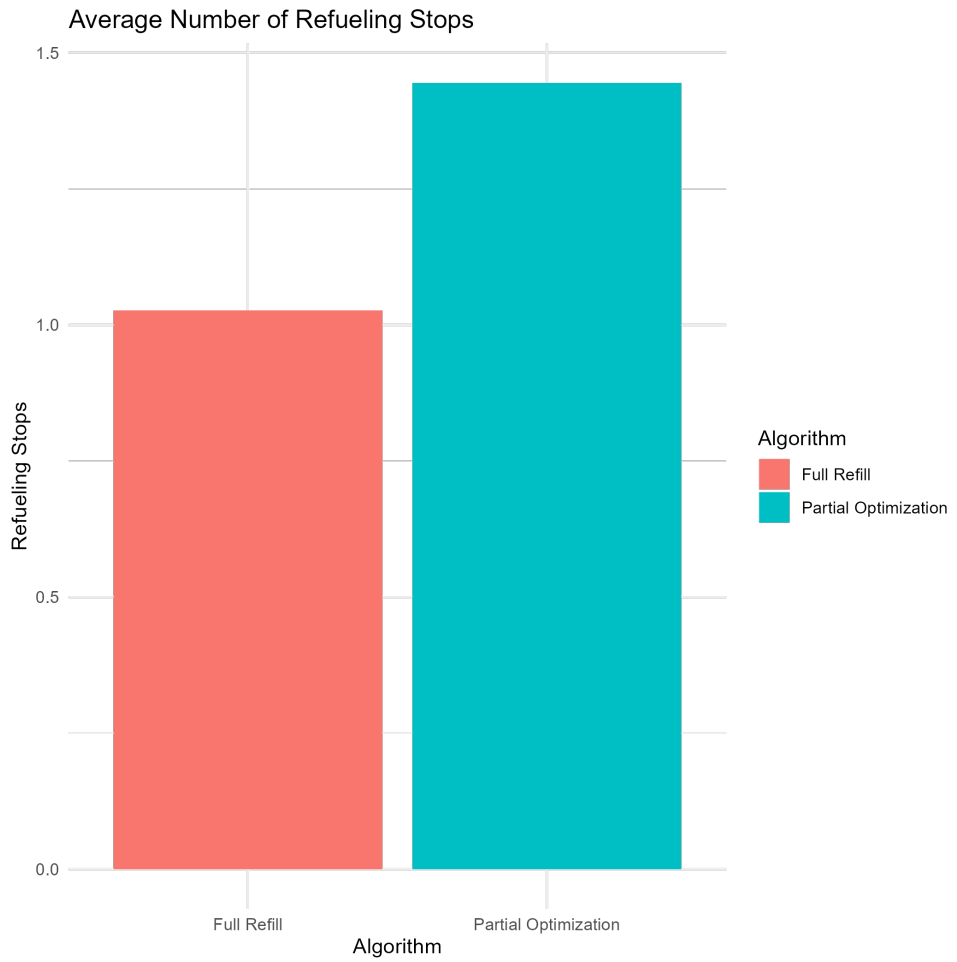


Figure 3.6: Average number of refueling stops: Partial Optimization requires more frequent stops (1.4–1.5) than Full Refill (1), reflecting a strategy of smaller, cost-efficient refuels.

3.8.3 Robustness Across Various Scenarios

To assess the stability of the fueling strategies, we examine their performance across two critical scenario parameters: Tank Capacity and initial fuel range (RangeRemaining). Figures 3.7 and 3.8 illustrate how fueling costs vary as a function of these variables.

Figure 3.7 presents fueling costs across a spectrum of tank capacities. The Full Refill algorithm exhibits a sharply increasing cost trend as tank capacity grows. Distinct peaks appear at specific capacities, indicating inefficient overfilling or exposure to high-cost stations due to indiscriminate refueling behavior. This volatility reflects the algorithm's inability to optimize costs for one-leg routes for higher tank capacities. In contrast, the Partial Optimization algorithm maintains a consistently low cost profile across all tank sizes, with minimal fluctuations, demonstrating a strategy that scales efficiently regardless of vehicle type.

Similarly, Figure 3.8 shows fueling costs in relation to initial fuel range, the distance a vehicle can travel with its existing fuel at the start of the route. The Full Refill approach again reveals a broad cost distribution across all range values, signaling high sensitivity to the starting fuel level. This variability suggests that the algorithm does not adjust effectively based on how much fuel is already in the tank. Cost spikes occur at certain ranges because carrying more fuel can eliminate the need for an additional refueling stop. In the case of a full refuel, this can lead to a significant cost difference compared to neighboring ranges. By comparison, the Partial Optimization algorithm delivers more stable and predictable costs regardless of the initial range, highlighting its flexibility in adapting to various starting conditions without incurring penalties.

Taken together, it confirms that Partial Optimization offers a robust cost advantage across diverse scenarios and initial conditions. Whether dealing with a small tank and minimal fuel or a large tank with full range, its targeted refueling logic consistently identifies cost-effective decisions. This adaptability contrasts sharply with the Full Refill strategy, which suffers from cost spikes and unpredictability as vehicle parameters vary.

These findings reinforce the strength of Partial Optimization as a scalable and resilient fueling strategy, capable of maintaining efficiency not only in ideal conditions but also across the full range of realistic, real-world vehicle configurations.

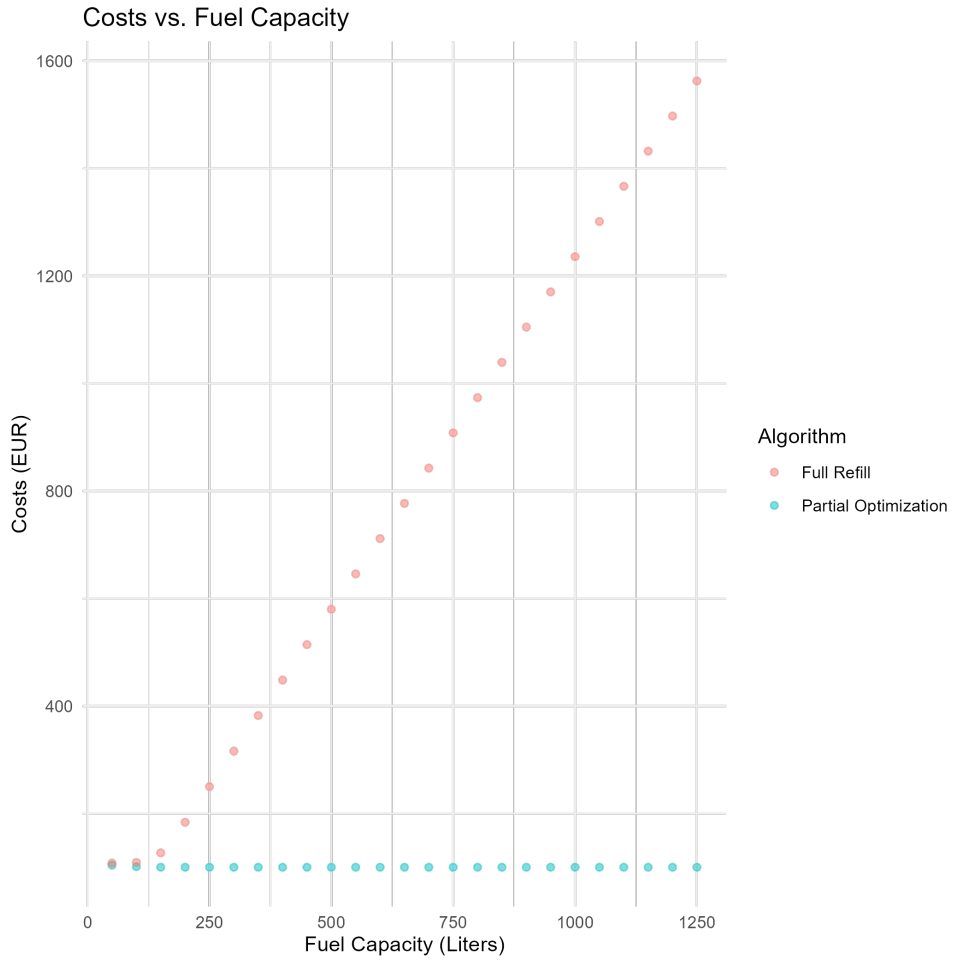


Figure 3.7: Average fueling costs by tank capacity: Partial Optimization stays consistently low across capacities, while Full Refill shows rising and fluctuating costs, especially at higher capacities.

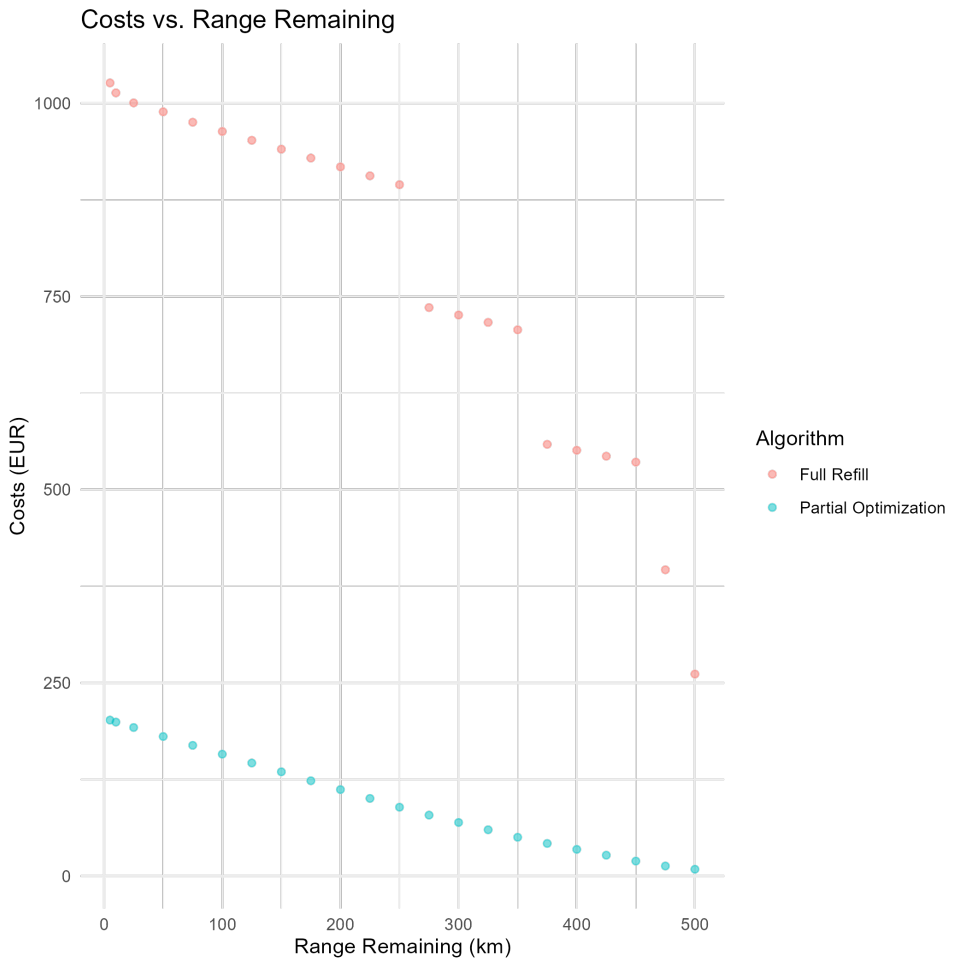


Figure 3.8: Average fueling costs by initial range: Partial Optimization remains consistently low, while Full Refill shows wide cost variation across all ranges.

3.9 Extension: Chained Routes - Evaluating Refueling Strategies Over Consecutive Routes

While previous analyses focused on isolated one-leg routes, real-world logistics operations often involve multi-leg journeys, where refueling decisions made in one segment may affect the next. To evaluate algorithm performance in these more realistic scenarios, we introduced chained routes, composed of three consecutive sub-routes. This setup allows us to explore whether the cost advantage observed for the Partial Optimization algorithm in single routes persists over extended distances and more complex decision chains.

A key consideration in this context is fuel remaining at the end of each leg. The Full Refill strategy, by design, tends to leave vehicles with more fuel at the destination, which can postpone the need for an immediate refill at the start of the next leg, potentially leading to higher costs. Conversely, the Partial Optimization algorithm often completes a route with minimal fuel remaining, maximizing savings on a per-leg basis but raising the risk of needing to refuel under less favorable conditions at the beginning of the next leg.

To test these trade-offs, we analyzed all possible combinations of chained routes among the four cities in the dataset, using the same range of tank capacities and initial fuel levels. The summary results are presented in Tables 3.3 and 3.9.

Despite initial concerns that the Partial Optimization strategy might struggle under chained condi-

Algorithm	AvgCostperRoute	AvgRefuels	AvgLitersBought	AvgLitersRemaining
Random	2388.74	2.5	1077.6	467.86
Partial Optimization	490.77	7.06	323.02	4.32
Full Refill	1575.86	2.63	1028.49	449.54

Table 3.3: Metrics comparing three algorithms for chained routes: Random, Partial Optimization, and Full Refill. Partial Optimization shows the lowest cost, with slightly more average refuels.

Algorithm	CostperLiter	RefuelEfficiency	CostStandardDeviation	CostMedian
Random	2.22	430.97	1663.55	1886
Partial Optimization	1.52	45.73	130.83	498
Full Refill	1.53	390.68	1085.06	1285

Table 3.4: Continuation of metrics comparison.

tions due to low fuel remaining at the destination, the data reveals the opposite. It continues to outperform both alternatives with the lowest average cost per full journey and the lowest cost per liter, as well as the smallest cost standard deviation. This consistent performance underscores the algorithm's ability to identify favorable refueling opportunities across legs, even when starting with a nearly empty tank.

This finding is confirmed in Figure 3.9, which presents the cost distribution for chained routes. The Partial Optimization algorithm maintains a lower and narrower cost range, while Full Refill remains prone to cost spikes. Although the gap between the two strategies narrows slightly compared to single-leg routes, the Partial Optimization approach still clearly dominates in both mean and variability.

In Figure 3.10, a histogram comparison of the two algorithms shows some increased overlap in cost ranges for chained routes, reflecting the added complexity. Nevertheless, the Partial Optimization algorithm maintains a significant peak in the low-cost region, while Full Refill stretches across a broader, more expensive spectrum.

These results confirm the robustness of the Partial Optimization algorithm, even in the case of more complex scenarios. The potential drawback of starting each leg with low fuel is effectively offset by the algorithm's ability to continue selecting cost-efficient stations across the extended route. As a result, the strategy not only achieves lower overall costs but also ensures greater predictability, making it highly suitable for applications where cost control and consistency are critical.

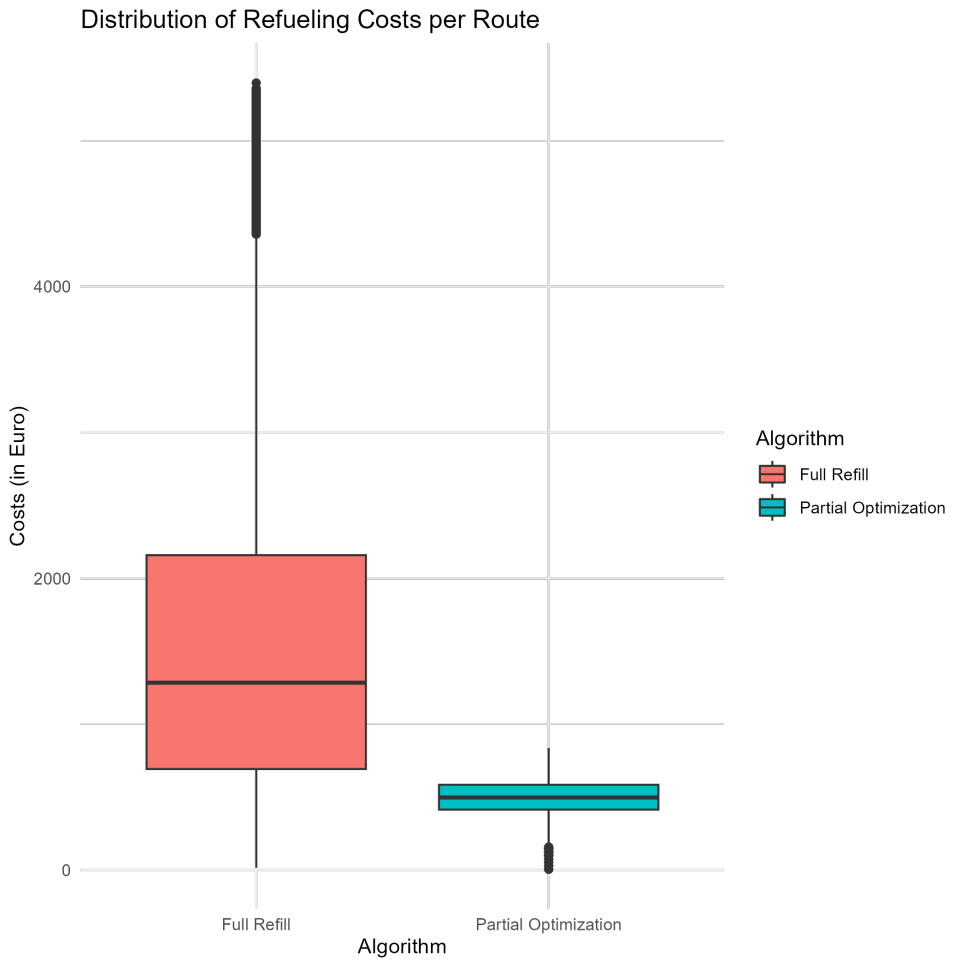


Figure 3.9: Distribution of fueling costs, considering chain routes: The average costs of the partial optimization algorithm are lower on average with a narrower cost distribution.

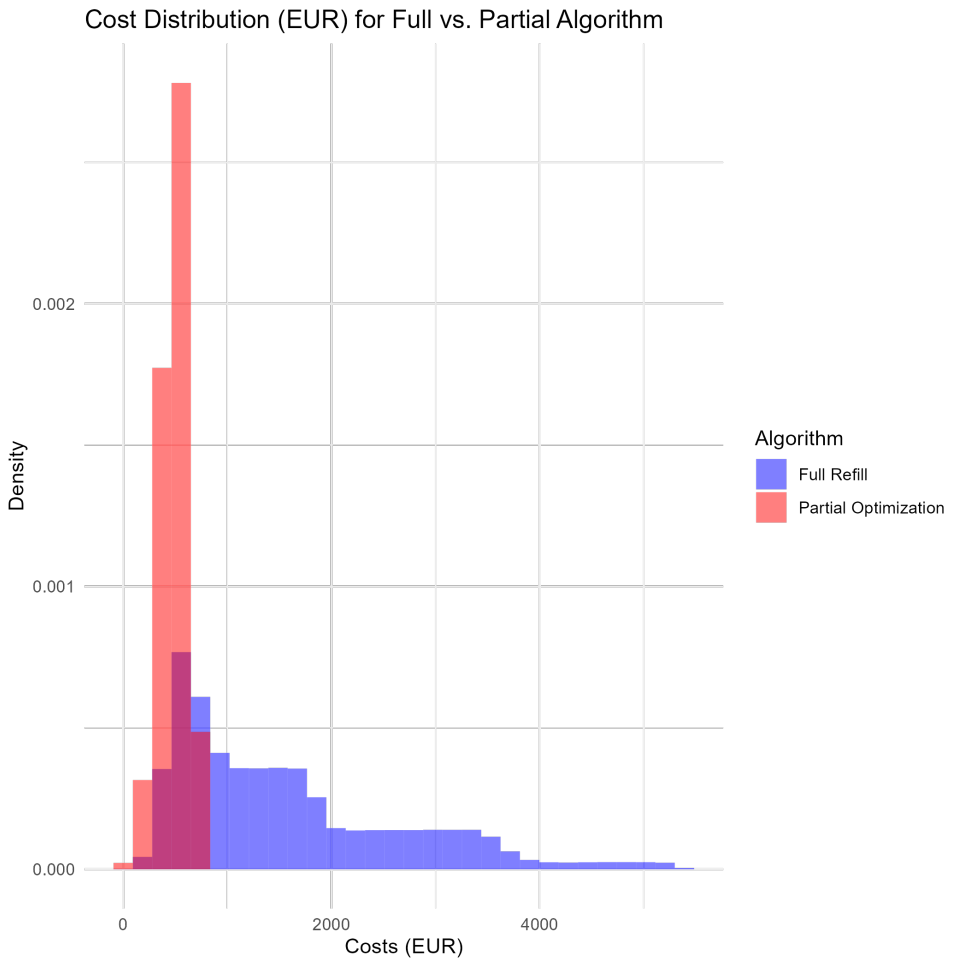


Figure 3.10: Histogram of cost distributions (EUR) for Full vs. Partial Algorithm, considering chain routes: Partial Optimization shows a sharp peak at low costs, while Full Refill has a wider spread with higher cost variability.

4 Conclusion and Outlook

4.1 Conclusion

In this thesis, we have presented a data-driven approach to fuel-saving optimization, focusing on minimizing gasoline expenses through strategic refueling decisions. By integrating algorithmic techniques with real-world fuel price data, we developed an efficient method that balances cost savings and route constraints. The study demonstrated that a well-structured optimization algorithm, leveraging dominance pruning and state-space exploration, significantly reduces refueling expenses compared to naive or full-refill strategies.

Our results highlight that the *Partial Optimization* strategy achieves the lowest refueling costs while maintaining flexibility in route planning. The method successfully adapts to fluctuating fuel prices, making it a practical solution for logistics companies aiming to optimize their operational expenses. Furthermore, we derive complexity bounds that indicate our approach remains computationally feasible, even for large-scale networks with multiple fuel stations.

While the *Full Refill* strategy ensures minimal refueling stops, it leads to higher overall costs, whereas the *Random* strategy proves highly inefficient. The findings reinforce the importance of data-driven decision-making in optimizing transportation-related expenditures.

4.2 Outlook

Although this study provides an effective framework for fuel cost optimization, several avenues for future research and improvement exist. One potential extension is the integration of real-time fuel price updates, allowing the system to dynamically adjust refueling strategies based on live data. This would enable even greater adaptability to market fluctuations.

Another promising direction involves incorporating additional constraints, such as time windows for refueling stops, driver rest periods, and fuel station availability. Expanding the model to support alternative fuel vehicles, such as electric and hydrogen-powered trucks, would also be a valuable contribution given the ongoing transition to sustainable transportation.

Moreover, enhancing the system with machine learning techniques could refine predictive modeling for fuel consumption and price trends, further improving decision-making accuracy. Finally, deploying the developed algorithm in real-world logistics operations and conducting empirical validation through field tests would provide critical insights into its practical effectiveness and potential refinements.

By continuing to refine and extend this research, we can further contribute to cost-efficient and environmentally sustainable transportation strategies, benefiting both businesses and the broader logistics industry.

Appendix

This appendix provides the Python script used to open a GraphHopper route in the web browser. The script constructs a URL based on the selected routing points and profile settings and then opens it in the default browser.

```python
#!/usr/bin/env python3

import argparse
import webbrowser
from urllib.parse import urlencode

BASE_URL = "http://localhost:8989/maps/"

def open_graphhopper_route(routing_points, profile="car"):
    if len(routing_points) < 2:
        raise ValueError("At least a start and an end point must be provided.")

    params = {
        "point": routing_points,
        "profile": profile,
        "layer": "OpenStreetMap"
    }

    full_url = f"{BASE_URL}?{urlencode(params, doseq=True)}"
    print("Opening URL:", full_url)
    webbrowser.open(full_url)
```

Listing 1: Python script to open a GraphHopper route in the web browser.

Bibliography

[1] J. D. Adler, P. B. Mirchandani, G. Xue, M. Xia, and M. Zhao. The electric vehicle shortest-walk problem with battery exchanges. *Networks and Spatial Economics*, 16(1):155–173, 2016.

[2] C. Bartolini, C. Gambella, L. Liberti, and F. Malucelli. The traveling salesman problem with refueling. In *15th Meeting of the EURO Working Group on Locational Analysis (EWGLA)*, Italy, 2011.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[4] S. Funke, A. Nusser, and S. Storandt. On minimizing fuel stops for vehicle routing. In *Proceedings of the 12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 103–114, Ljubljana, Slovenia, 2012. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[5] A. Montoya, C. Guéret, J. E. Mendoza, and J. G. Villegas. The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, 103:87–110, 2017.

[6] F. P. Russell. An analysis of minimum travel cost with fuel constraints. *The Journal of the Operational Research Society*, 28(12):1163–1170, 1977.