Solving the Pickup and Delivery Problem with Time Windows Using an Adaptive Large Neighborhood Search Heuristic

Yusuf Sami

January 7, 2026

1 Introduction

In this assignment, a vehicle routing problem with pickup and delivery requests and strict time windows is studied. Each customer requires a shipment to be collected from one location and delivered to another within a specific time period, while all routes start and end at a central depot. This problem is known as the Pickup and Delivery Problem with Time Windows (PDPTW) and is highly relevant in logistics and transportation planning. To solve it, an Adaptive Large Neighbourhood Search (ALNS) algorithm is developed. The method repeatedly removes and reinserts requests using several destroy and repair operators, guided by a simulated annealing acceptance mechanism. After parameter tuning, the algorithm produces efficient and high-quality routing solutions within a limited computation time.

2 Methodology

In the following section, the general approach, theoretical background, and implementation details for solving the Pickup and Delivery Problem with Time Windows (PDPTW) are discussed in greater depth. By applying specific algorithms and heuristic methods, the objective is to develop an effective strategy for obtaining high-quality solutions to the PDPTW using the so-called Adaptive Large Neighbourhood Search algorithm.

2.1 Adaptive Large Neighbourhood Search

The Large-Neighbourhood Search (LNS) algorithm is able to explore a relatively complex neighbourhood. First, it creates an initial solution to the given problem. Second, it alternatively rebuilds the solution by means of using a destroy and repair operator. The destroy operator will remove $\alpha\%$ of the customers in the current solution. How these customers are removed depends on the algorithm or heuristic implemented. Similarly, the repair operator will insert the same $\alpha\%$ customers back into the partially destroyed solution. This way, a new solution is found in the neighbourhood of the original solution.

The Adaptive Large-Neighbourhood Search (ALNS) algorithm is a generalization of the LNS algorithm. While the main steps are similar to LNS, such as repetitive destroying and repairing a solution, ALNS has the option to have multiple destroy and repair operators. Each of these operators are each assigned a weight ρ_i^+ and ρ_i^-, respectively, for the repair and destroy operators. These weights determine the probability that a specific operator can be chosen, and are initialised at the same value at the start of the program. The probability that a specific operator is chosen, is shown in Equation 1.

$$p_j^- = \frac{\rho_j^-}{\Sigma_{i=1}^{|\Omega^-|}\rho_i^-} \qquad p_j^+ = \frac{\rho_j^+}{\Sigma_{i=1}^{|\Omega^+|}\rho_i^+} \tag{1}$$

After each iteration, these weights are updated based on whether a specific operator is chosen and how the new solution compares to the original solution. First, the weights or the selected destroy and repair operator are updated, according to the equations shown below.

$$\rho_i^+ = \lambda \cdot \rho_i^+ + (1-\lambda)\Psi \tag{2}$$

$$\rho_i^- = \lambda \cdot \rho_i^- + (1-\lambda)\Psi \tag{3}$$

Here, p_j^- and p_j^+ represent the probabilities of choosing a specific destroy and repair operator, respectively, with $p_j \in [0, 1]$. The parameter λ represents the decay factor, which controls how sensitive the weights are to changes in the performance of each operator. The term Ψ is the score function, which depends on the result of the comparison between the initial solution x and the newly generated solution x'. This score function is defined as follows:

$$\Psi = max \begin{cases} w_1 & \text{if the new solution is a global best} \\ w_2 & \text{if the new solution is better than the current one} \\ w_3 & \text{if the new solution is accepted} \\ w_4 & \text{if the new solution is rejected} \end{cases} \tag{4}$$

The values of the weights typically satisfy the condition $w_1 \geq w_2 \geq w_3 \geq w_4 \geq 0$. In this report, the parameters are set to $[w_1, w_2, w_3, w_4] = [1.0, 0.5, 0.3, 0.1]$. The values are chosen such that the weights should not exceed a value between 0 and 1.
A solution is always accepted if it represents a new global best or improves upon the current solution. However, in cases where the new solution is worse, it may still be accepted with a certain probability, which is one of the characteristics of the ALNS heuristic. The acceptance criterion governing this process is described in detail in the following section.
Through this adaptive algorithm, operators that often result in better solutions receive higher weights, therefore increasing their probability of being selected in the following iterations. The opposite holds as well: operators that often result in worse solutions receive lower weights, therefore reducing their probability of being selected in the following iterations.
On the contrary, if an operator provides a worse solution, the weight and probability of that operator being chosen again decrease. These operators are updated each step and normalised to sum to a value of 1, in order to decrease the chance of one value blowing up, and thus taking too significant a weight to be selected.

2.2 Simulated Annealing

Simulated annealing is a metaheuristic algorithms, inspired by the annealing process in metallurgy. The algorithm "explores" the solution space by improving the solution over time, with a probability to make the solution worse to escape local minima. The probability with which a potentially worse solution is accepted decreases over time, according to a predefined "cooling schedule".

The probability that a worse solution is accepted is dependent on two different variables. First, the difference in objective cost between the current solution x, and a proposed worse solution x', which is noted as $\Delta(x, x')$. Second, the temperature of the simulated annealing process T. The equation with which the acceptance probability p is determined is shown in Equation 5.

$$p = \exp(-\Delta(x, x')/T) \tag{5}$$

It can be seen that a larger difference in objective cost decreases the acceptance probability. However, a higher temperature increases the acceptance probability. This means that by applying the cooling schedule and decreasing T over time, the acceptance probability will most likely decrease over time as well, reducing the chance of accepting a worse solution.
The cooling schedule for T is linear, reducing by a constant value each iteration: $T = \alpha T$, with $\alpha \in [0, 1)$. In some cases, the starting temperature T_{start} is chosen beforehand, together with an end temperature T_{end} to define a temperature range. Together with the given cooling rate, this results in a given number of iterations before T_{end} is reached.

However, the choice was made to fix T_{start} and the number of iterations, making T_{end} more flexible, while limiting the number of iterations. This gives more control over the exact number of iterations, rather than an expected number by defining a cooling scheme.

2.3 Destroy operators

2.3.1 Random removal

The Random Removal Operator is one of the simplest destroy methods in the ALNS framework. Its objective is to diversify the search by partially disrupting the current solution.

In this operator, a predefined number of customer requests are randomly selected from those currently served and removed from their routes. Each removal consists of identifying the request within its route, deleting both the pickup and delivery nodes, and transferring the request to the set of unserved requests. This process continues until the specified number of removals is reached or no served requests remain.

The operator does not use any cost or time-window information during selection; all requests have the same probability of being removed. This purely random process introduces exploration by allowing the algorithm to escape local optima and generate new routing patterns when the removed requests are later reinserted during the repair phase.

The degree of destruction is controlled by the removal rate, which determines how many requests are eliminated in one iteration. Choosing this rate carefully ensures that the algorithm maintains a balance between solution stability and diversity.

2.3.2 Worst removal

The Worst Removal Operator focuses on eliminating the requests that contribute the most to the total travel distance. Its purpose is to intensify the search by directly targeting inefficient parts of the current solution.

In this operator, each served request is temporarily removed one by one in a copied version of the current solution. The total distance is recomputed after each removal to measure the saving achieved by excluding that request. These savings represent how much the overall distance would decrease if a given request were removed. Once all savings are calculated, the requests with the highest savings—those that cause the greatest inefficiency—are removed from the actual solution.

By removing the most costly requests, the operator creates an opportunity for the subsequent repair phase to reinsert them in more efficient positions or to combine them with other routes. The number of requests removed is controlled by the same destruction parameter as used in other operators.

This operator introduces a more directed form of destruction than random removal, as it systematically identifies and eliminates the worst-performing requests, contributing to the intensification of the search process.

2.3.3 Related (Shaw) removal

The Related (Shaw) Removal Operator aims to remove groups of requests that are similar to each other in terms of spatial or temporal characteristics. Unlike random removal, which selects requests independently, this operator focuses on removing a cluster of related requests to allow meaningful route restructuring during the repair phase.

The process begins by selecting a random "seed" request from those currently served. For every other request, a relatedness score is calculated based on the distance between pickup locations and the difference in their time windows. The relatedness between two requests i and j is expressed as:

$$R(i,j) = w_1 \cdot d_{ij} + w_2 \cdot |TW_i - TW_j| \tag{6}$$

where d_ij is the Euclidean distance between the pickup locations of the two requests, TW_i and TW_j represent their pickup time-window start times, and w_1 and w_2 are weighting parameters controlling the relative importance of distance and time similarity.

After computing these scores, the operator sorts all requests according to increasing relatedness to the seed. The seed request and the $n-1$ most related requests are then removed from the solution and added to the unserved list.

By removing highly related requests together, the operator enables the subsequent repair phase to explore new combinations of routes that would not emerge through purely random destruction. This mechanism therefore enhances intensification, allowing the algorithm to reconfigure spatially or temporally similar requests into more efficient route structures.

2.3.4 Time-oriented removal

The Time-Oriented Removal Operator focuses on removing requests with the most restrictive time windows in order to relax temporal constraints within the solution. Requests that have very narrow pickup windows tend to reduce scheduling flexibility and increase the difficulty of finding feasible route combinations.

In this operator, the width of each request's pickup time window is determined by calculating the difference between its ending and starting times. All served requests are then ranked according to their time-window width, and those with the smallest values—indicating the tightest time constraints—are removed from the solution and added to the set of unserved requests.

By selectively eliminating time-critical requests, the operator allows the algorithm to reconstruct routes with greater temporal flexibility during the repair phase. The number of requests removed is defined by the destruction parameter, as in the other removal operators. This method is particularly effective in instances where narrow time windows significantly restrict the feasible search space.

2.4 Repair operators

2.4.1 Random insertion

The Random Insertion Operator is the simplest repair mechanism used in the ALNS framework. Its objective is to reintroduce previously removed requests into the solution in a random manner, promoting diversification and maintaining computational efficiency.

During this process, one unserved request is randomly selected and inserted into a randomly chosen route. If the insertion is not feasible due to capacity or time-window constraints, another route is selected and tested. When all existing routes are infeasible, a new route is created containing only that request. This procedure continues until all unserved requests have been successfully reinserted.

Since the selection and placement of requests are entirely random, this operator does not consider any cost or distance information during reconstruction. Although it may not always lead to high-quality solutions, it provides valuable exploration capability and serves as a baseline repair method that can be alternated with more informed insertion operators during the search process.

2.4.2 Greedy insertion

The Greedy Insertion Operator is a repair method that reinserts unserved requests by always selecting the option that results in the smallest possible increase in total travel distance. Its objective is to intensify the search by constructing a locally optimal solution at each iteration.

In this operator, for every unserved request, all possible insertion positions across the existing routes are evaluated. For each potential position, the additional travel distance that would result from inserting the request is computed. The request–position combination that produces the minimum distance increase is then selected and applied to the solution. If a request cannot be feasibly inserted into any existing route, a new route is created to serve it.

This greedy approach ensures that every insertion step provides the most cost-effective local improvement based on the current state of the solution. Although the operator may lead to local optima and does not guarantee global improvement, it plays a key role in the intensification of the search, often producing high-quality intermediate solutions that guide the ALNS towards better overall performance.

2.4.3 Regret-k insertion

The Regret-k Insertion Operator is a more advanced repair strategy that extends the greedy insertion concept by introducing a limited look-ahead mechanism. Instead of selecting the request that provides the smallest immediate cost increase, this operator prioritizes requests whose deferred insertion would result in the largest potential loss in solution quality.

For each unserved request, all feasible insertion positions across the current routes are evaluated, and the corresponding cost increases are recorded. These insertion costs are then sorted from best to worst.

The regret value of a request is defined as the difference between the cost of its k^{th}-best insertion and its best insertion:

$$R_k(i) = c_k(i) - c_1(i) \tag{7}$$

where $R_k(i)$ denotes the regret value of request i; $c_1(i)$ represents the smallest (best) feasible insertion cost for request i; $c_k(i)$ refers to the cost of the k^{th}-best insertion.

Requests with the highest regret values are inserted first, at the position corresponding to their best insertion cost. This ensures that requests with few good insertion opportunities are placed early, preventing situations where all feasible positions become unavailable later in the process.

The Regret-k Insertion Operator therefore balances exploration and intensification by combining the local efficiency of the greedy approach with a broader anticipation of future insertion options. In practice, using $k = 2$ and $k = 3$ often provides an effective trade-off between computational time and solution quality.

2.5 General improvements and additions

One part of the assignment is to improve the given code template outside of the given scope. One of the given suggestions was to improve the code by means of adding the option to include Electri Vehicle analysis to the feasibility algorithm. This addition is implemented and explained in more detail in the following section.

2.5.1 Electric vehicles

A major improvement that is made to the given template code, is integration of functionality for electric vehicles (EV). The instance data already includes relevant EV parameters such as battery capacity, energy consumption rate, and recharge rate, along with the locations of available charging stations.
The first modification involves the option to read and process EV data from the instance files, which has been included in "Problem.py". The second modification is implementing the logic to verify the feasibility of a route. This verification process is being done in multiple steps, each with its own relevance. However, a couple of assumptions are made regarding the use of EVs:

1. The vehicle will try to drive as far as possible with the available battery capacity.
2. When the vehicle needs to recharge, the battery will be charged until full capacity.
3. All previous requirements, such as vehicle capacity and time windows, remain to be respected.

For the first assumption, this will cover a couple of scenarios. If the vehicle is able to drive the entire route without recharging, starting and stopping at the depot, it will do so. Each depot is simultaneously a recharging station, which is manually verified for each instance. This means that the vehicles are able to recharge when arriving back at the depot. Additionally, if the EV cannot drive the full route without recharging, it will first drive the given route as far as possible until it has reached a "critical location".This critical location is the last stop in the route that the EV can still reach without needing to recharge, while simultaneously being able to reach the nearest recharge station.
The second assumption is done in order to simplify the problem. When the EV starts charging, it will only stop charging and depart once the battery is back to full capacity.
The first constraint refers to the battery level. Feasibility requires that the battery level always remains between 0 and full capacity Q:

$$0 \leq y_j \leq Q \qquad \text{for all visited nodes } j. \tag{8}$$

When a vehicle travels from node i to node j, traversing an arc of length d_{ij}, the battery is drained by a certain amount depending on the battery consumption rate r_c. The new value y_j can be determined as follows:

$$0 \leq y_j \leq y_i - d_{ij} \cdot r_c \leq Q \tag{9}$$

The second point is the time window constraint. When a vehicle decides to recharge somewhere during the route, the time windows for further pickup and delivery should still be accepted. This can be done in the following way, shown in Equation 10.

$$\tau_i + v \cdot (d_{ir} + d_{rj}) + (Q_{bat} - y_i) \cdot r_r \leq \tau_j \tag{10}$$

Here, τ_i is the time at which the EV leaves node i, t_{ir} and t_{rj} the time it takes to drive from node i to the recharge station and from the recharge station to node j, respectively, and τ_j the time at which the EV should be at the next node j. The total time required at the recharging station also includes the battery recharging time, given by $(Q_{bat} - y_i) \cdot r_r$, where r_r represents the recharging rate.
The ALNS framework is now capable of evaluating and constructing routes that explicitly account for the operational limitations of electric vehicles (EVs). As a result, the algorithm can effectively generate and analyse routing solutions that consider both travel distance and energy consumption, in contrast to conventional vehicle models that assume an unlimited driving range.

3 Experimental results

This section presents the experimental outcomes obtained from applying the Adaptive Large Neighborhood Search (ALNS) algorithm to the Pickup and Delivery Problem with Time Windows (PDPTW). The objective is to minimize the total travel distance while satisfying all pickup–delivery precedence and time-window constraints. Three iteration budgets (100, 500, and 1,000 iterations) are examined to analyze convergence behavior, operator adaptation, and the influence of parameter configurations. All experiments are conducted under identical conditions with fixed random seeds to ensure reproducibility.

3.1 Small Number of Iterations (100)

The initial experiment is conducted on the lrc104 instance with a cooling rate of $\alpha = 0.95$ and an initial temperature of $T_0 = 1000$. The goal of this test is to observe the early behavior of the Adaptive Large Neighborhood Search (ALNS) algorithm and to evaluate its convergence pattern within a short computational budget of 100 iterations. The algorithm achieves a final total distance of 914.56 after 100 iterations, improving from an initial distance of approximately 1898.39.
Figure 1 shows the evolution of the total route distance over the 100 iterations. The "best distance" curve decreases rapidly during the first 20 iterations, reaching around 950, and stabilizes after iteration 85. In contrast, the "current distance" curve fluctuates strongly, reflecting the acceptance of both improving and worsening solutions. These fluctuations demonstrate the simulated annealing acceptance mechanism, which enables temporary cost increases to promote diversification and escape local optima.

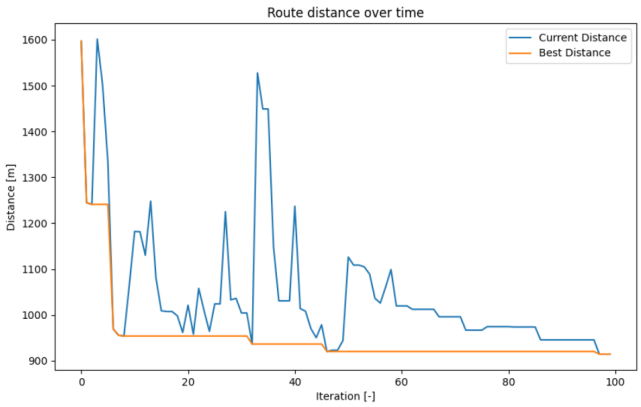

Figure 1: Evolution of the distance over time.

The corresponding temperature schedule is presented in Figure 2. The temperature decays exponentially from 1000 °C to approximately 100 °C, following the geometric cooling rule $T_{k+1} = \alpha T_k$. This controlled reduction allows the algorithm to explore the solution space freely at the start and gradually restrict acceptance to more promising moves as the search progresses.

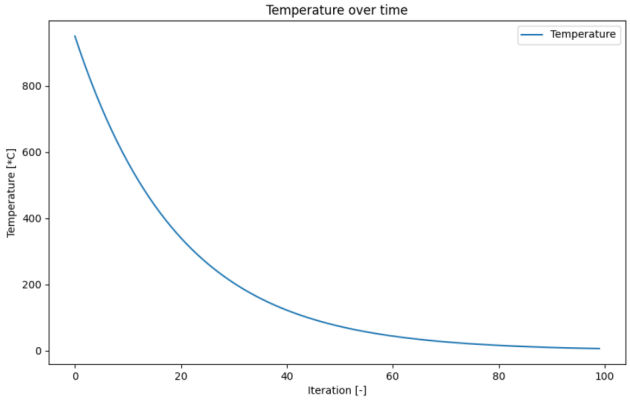

Figure 2: Evolution of the temperature over time.

Figure 4 and Figure 3 present the evolution of operator weights for the repair and destroy phases, respectively. In the repair phase (Figure 3), Repair Op 3 (Regret-2 Insertion) obtains the highest weight throughout most of the run, maintaining values between 0.45 and 0.55. This indicates that regret-based reinsertion is the most effective repair strategy for this instance, as it balances local optimization with global improvement potential. Repair Op 2 (Greedy Insertion) holds a moderate weight (= 0.35–0.40), while Repair Op 1 (Random Insertion) remains least influential (= 0.15–0.25), consistent with its exploratory role in early iterations.

The destroy operator weights (Figure 4) reveal a more dynamic adaptation pattern. Destroy Op 2 (Worst Removal) and Destroy Op 4 (Time-Oriented Removal) emerge as the dominant operators, both reaching weights close to 0.40 at different phases of the run. Destroy Op 3 (Related /Shaw Removal) alternates between moderate and high influence, suggesting its utility when spatially correlated requests are present. Finally, Destroy Op 1 (Random Removal) gradually decreases below 0.20 as the algorithm shifts from exploration to intensification. This adaptive adjustment confirms that the weighting mechanism effectively emphasises the most productive operators while maintaining sufficient diversity.

Figure 3: Evolution of repair operator weights over time.

Figure 4: Evolution of destroy operator weights over time.

Overall, the short-run experiment demonstrates that the ALNS quickly identifies high-quality feasible solutions within a limited number of iterations. The observed convergence pattern, together with the adaptive reweighting of operators, indicates a well-balanced interplay between diversification and intensification.

3.2 Medium Number of Iterations (500)

To examine the effect of a longer runtime, the same instance (lrc104) was solved for 500 iterations using identical parameters ($T_0 = 1000, \alpha = 0.95$) The goal was to determine whether additional iterations could yield further improvement once the algorithm had already converged in the short run.

The algorithm reached a final total distance of 873.21 m, representing an improvement of approximately 4.5 percent compared to the 100-iteration result. However, as shown in Figure 5, almost all improvement

occurs within the first 150 iterations, after which the cost curve flattens completely. This behavior indicates early convergence caused by the rapid cooling schedule, which drives the temperature to near-zero levels and significantly reduces the acceptance probability for worse solutions. Consequently, the search becomes fully deterministic, and exploration effectively stops.

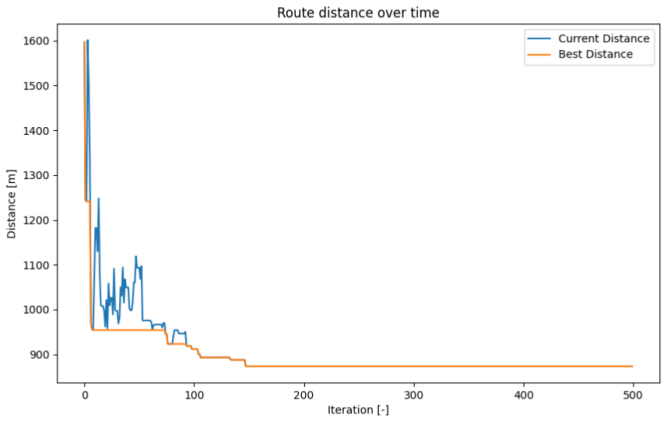

Figure 5: Evolution of the distance over time for an increased number of iterations.

3.3 Large Number of Iterations (1000)

The final experiment was conducted with an extended computational budget of 1,000 iterations on the lrc104 instance, using the same configuration ($T_0 = 1000, \alpha = 0.95$).The objective was to assess whether a longer runtime could provide further improvement beyond the 500-iteration solution.

The algorithm achieved a final total distance of 873.21 m, which is identical to the best solution obtained in the previous experiment. As illustrated in Figure 6, the total distance curve remains stable after iteration 200, confirming that the algorithm had already converged to a near-optimal solution early in the run. The flat trajectory of the "best distance" curve and the limited oscillations of the "current distance" line demonstrate that the temperature reached near-zero levels, preventing the acceptance of worse solutions and halting further diversification.

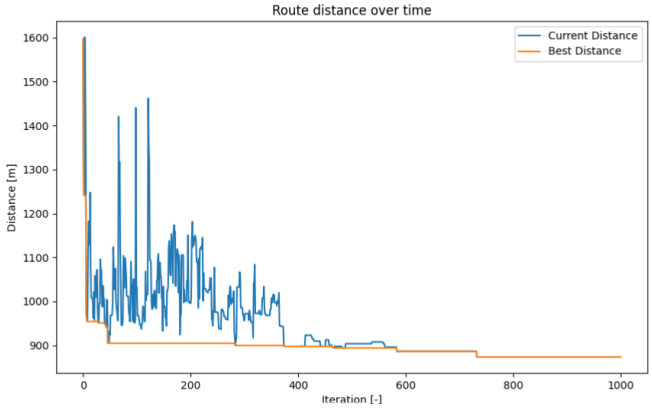

Figure 6: Evolution of the distance over time for an significant increased number of iterations

This stagnation behavior highlights the trade-off between solution quality and exploration. While a lower cooling rate accelerates convergence, it can also lead to premature freezing of the search process. For this instance, the ALNS reached a high-quality solution quickly, but additional iterations did not yield any measurable improvement.

3.4 EV data comparison

For the final subsection, a small comparison is made between the results for "regular" vehicles (RV), which may be assumed to be diesel vehicles, and the newly implemented EV data and feasibility algorithm. For this, the parameters are set to $\alpha = 0.95, T_0 = 1000$, and with only a small number of

iterations (n = 100). The instance "lrc104" will be used again for this.

The first comparison that is made, is with respect to the distance of the route. The analysis for the (RV) situation can be seen in Figure 8, whereas the EV situation can be seen in Figure 7.

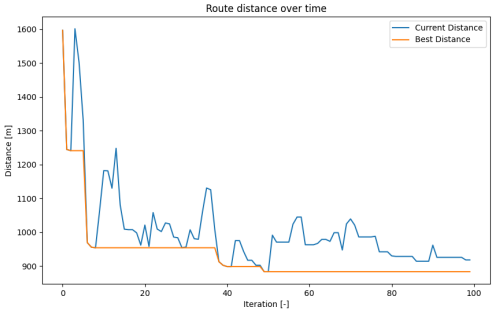

Figure 7: Evolution of the distance over time for RV.

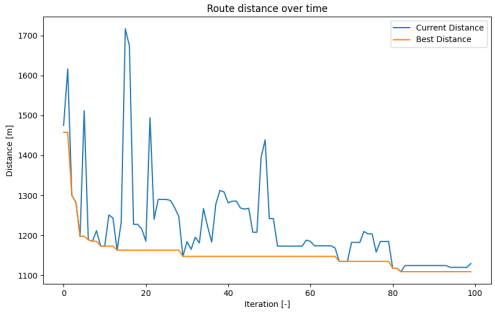

Figure 8: Evolution of the distance over time for EV

The figures indicate that the overall trends in the plots appear relatively similar at first glance. The total distance per iteration fluctuates throughout the search process. However, the average distance generally decreases over time, which can be explained by the progressively reduced acceptance probability for worse solutions. A noticeable difference can be observed in the minimum and maximum distances achieved. In the regular vehicle (RV) scenario, the best solution achieved a total distance of 883.31, whereas the electric vehicle (EV) scenario resulted in a significantly higher best distance of 1109.20.

It was initially expected that the EV routes would be of comparable or even shorter length than those of the RV case, due to the additional range limitations encouraging more compact routing. However, the opposite occurred: the EV routes were considerably longer. The exact cause of this difference remains unclear. One possible explanation considered was the influence of the destroy and repair operators within the ALNS algorithm.

The operator-weight plots provide some insight into this behaviour. For the repair operators, the patterns are broadly similar across both RV and EV cases, with the third operator consistently achieving the highest average weight and the first operator the lowest. In contrast, the destroy operators show less consistent behaviour, with higher variability and apparent randomization in their selection probabilities. Consequently, there appears to be no strong or systematic preference for specific destroy operators, aside from a few isolated instances.

Figure 9: Evolution of repair operator weights over time for RV.

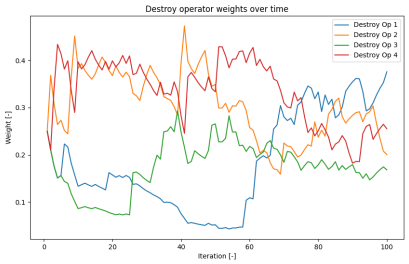

Figure 10: Evolution of destroy operator weights over time for RV.

Figure 11: Evolution of repair operator weights over time for EV.

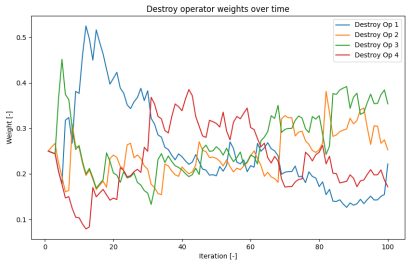

Figure 12: Evolution of destroy operator weights over time for EV.

Unfortunately, the analysis above has not resulted in a clear answer on how the EV routes have become significantly longer, and would therefore require a more in-depth analysis. One suggestion that could not be tested, but would theoretically be feasible, is the option of more vehicles driving shorter distances. While the shortest distance might decrease, due to the number of increased vehicles, and thus arcs traversed, the total route length would increase.

4 Conclusions & Discussion

In the report above, the PDPTW problem has been approached and analysed using the ALNS heuristic. The different steps of this heuristic have been implemented, tested, and potentially extended to take into account additional components, such as electric vehicles.

The algorithm integrated several destroy and repair operators combined with a simulated annealing acceptance criterion, enabling a balance between exploration and exploitation. The implementation successfully reduced the total route distance from 1898.4 to 873.2 for the lrc104 instance, showing that the ALNS can effectively find high-quality feasible solutions within a limited number of iterations.

A key observation was that the temperature decayed rapidly, leading to early convergence and reduced improvement during later iterations. Although the adaptive operator weighting mechanism performed well—favoring the Regret-2 Insertion and Worst Removal operators—the search occasionally stagnated due to limited diversification. Future improvements could include a slower or adaptive cooling schedule, dynamic acceptance thresholds, or the introduction of reheating phases to enhance exploration and prevent premature convergence.

One possible improvement of the current electric vehicle (EV) implementation concerns the feasibility-checking algorithm. Currently, the algorithm only checks feasibility for no charge at all, or tries to insert a charging stop after the last node visited. This approach provides a quick approximation of feasibility but may overlook more optimal or necessary charging opportunities along the route. A more comprehensive strategy would evaluate all relevant points along the route where a charging stop could be inserted and then select the option that yields the most feasible and cost-effective solution. Implementing this would have required more significant changes to the algorithm structure and logic, which was unfortunately not possible due to time constraints.

A second improvement involves relaxing the assumption of full recharging at every charging stop. In the current implementation, vehicles are always recharged to full capacity, which simplifies feasibility checks but can lead to unnecessarily long charging times and less flexible routing decisions. Allowing partial recharging would make the problem formulation more realistic and increase the variety of feasible solutions. It could also enable the algorithm to identify more efficient routes by balancing the trade-off between charging duration and route continuity.