Optimization of Routes of Electric Vehicles taking into account the Status of Charging Stations

Gonçalo Cravino Fernandes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. António Manuel Raminhos Cordeiro Grilo
Prof. Hugo Gabriel Valente Morais

Examination Committee

Chairperson:
Supervisor: Prof. António Manuel Raminhos Cordeiro Grilo
Member of the Committee: Prof. João Paulo Carvalho

October 2023

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I wish to express my deepest gratitude to all those who have played a significant role in the successful completion of my master's thesis.

Foremost, I extend my immense thanks to my esteemed thesis advisors, Prof. António Grilo and Prof. Hugo Morais, for their unwavering support, invaluable guidance, and profound insights that have been instrumental in shaping the outcome of this thesis. Your expertise and dedication have been a constant source of inspiration.

To my incredible family, friends, and my unwaveringly supportive girlfriend, I owe an immeasurable debt of gratitude. Your enduring encouragement and nonstop belief in my abilities sustained me through the challenges encountered on this academic journey.

This thesis stands as a testament to the collective efforts, belief, and support of those who have been part of this endeavor, and for that, I am genuinely appreciative.

Abstract

In the last decade, the rise of electric vehicles has brought benefits such as reduced reliance on fossil fuels. However, the limited-range batteries and charging challenges have created the need for advanced electric vehicle route planning. This thesis explores three different methodologies to solve the electric vehicle route planning problem, minimizing both route duration and cost. The first methodology consists of, a search-based approach utilizing the Dijkstra's algorithm, to obtain the optimal path for our problem formulation. The second methodology, consists of a Meta-heuristic approach utilizing a Genetic algorithm, to obtain a close-to-optimal solution in a faster runtime. The third and final methodology, consists of a reinforcement learning approach utilizing the Deep Q-Learning algorithm, to obtain a close-to-optimal solution in real time. The experimental results obtained showed the capability of the Genetic algorithm to obtain solutions with comparable quality to those achieved by Dijkstra's algorithm, but with significantly improved runtime efficiency. The results obtained from the Deep Q-Learning algorithm, although underwhelming showed the potential of the algorithm utilization.

Keywords

Electric Vehicle Route Planning, Dijkstra, Genetic Algorithm, Deep Q-Learning

Resumo

Na última década, a adoção dos veículos elétricos trouxe benefícios, como a redução da dependência de combustíveis fósseis. No entanto, a autonomia limitada das baterias e os desafios na recarga, criaram a necessidade de um planeamento de rotas mais avançado para veículos elétricos. Esta tese explora três metodologias diferentes para resolver o problema de planeamento de rotas para veículos elétricos, minimizando tanto a duração da rota como os custos. A primeira metodologia consistem, numa abordagem baseada em busca, utilizando o algoritmo de Dijkstra, para obter o caminho ótimo para a nossa formulação do problema. A segunda metodologia consiste numa abordagem meta-heurística utilizando um algoritmo genético para obter uma solução quase ótima num tempo de execução mais rápido. A terceira e última metodologia consiste em uma abordagem de reinforcement learning utilizando o algoritmo Deep Q-Learning, para obter uma solução quase ótima em tempo real. Os resultados experimentais obtidos demonstraram a capacidade do algoritmo genético de obter soluções de qualidade comparável às obtidas pelo algoritmo de Dijkstra, mas com uma eficiência de tempo significativamente melhor. Os resultados obtidos com o algoritmo Deep Q-learning, embora menos impressionantes, demonstraram o potencial para a utilização do algoritmo.

Palavras Chave

Planeamento de Rotas para Veículos Elétricos, Dijkstra, Algoritmo Genético, Deep Q-Learning.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

CS	Charging Station
DE	Differential Evolution
DNDQL	Dueling Network Deep Q-Learning
DQL	Deep Q-Learning
DQN	Deep Q Network
EV	Electric Vehicle
EVCS	Electric Vehicle Charging Station
EVRC	Electric Vehicle Routing and Charging
EVRP	Electric Vehicle Route Planning
GA	Genetic Algorithm
MDP	Markov Decision Process
OCM	Open Charge Map
ORS	OpenRouteService
SPSP	Single Pair Shortest Path
SSSP	Single Source Shortest Path
QL	Q-Learning
RL	Reinforcement Learning
SoC	State of Charge
VRSA	Vila Real de Santo António

1

Introduction

Contents

1.1 Context and Motivation

In the last decade, the utilization of Electric Vehicle (EV)s has increased significantly [4]. With all the advantages that EVs have brought especially in terms of their independence from fossil fuel, their limited-range batteries have created a new constraint in Electric Vehicle Route Planning (EVRP). EV's battery sizes have been continuously increasing [1], as depicted in FIgure 1.1, to the point where they are now capable of making long-distance routes without the constant need for charging [4].

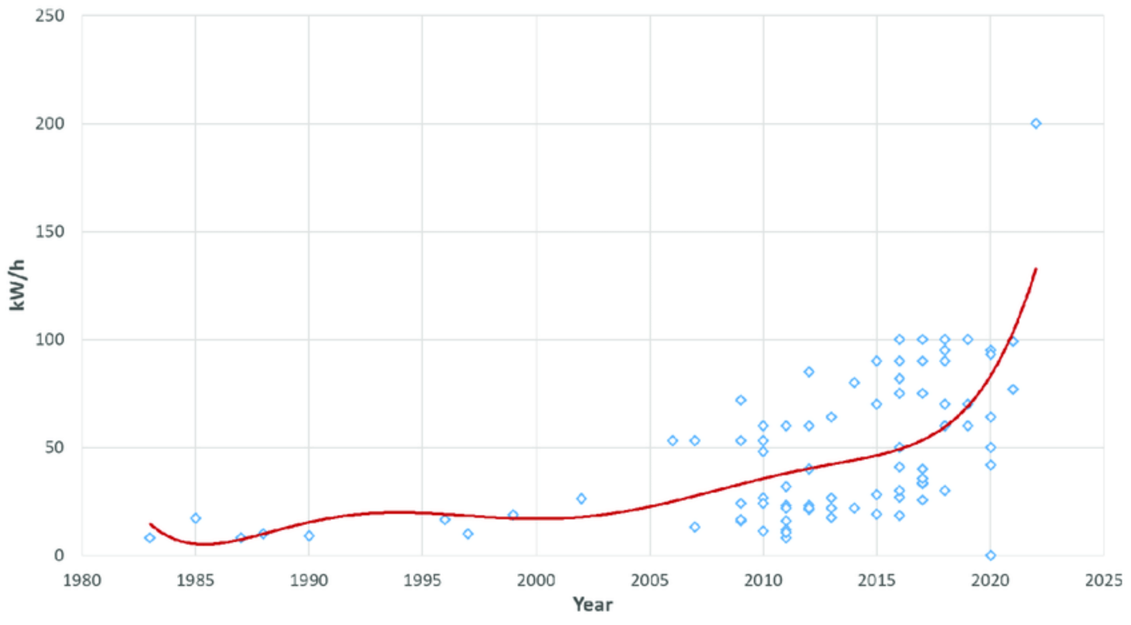


Figure 1.1: Evolution of the battery capacity since the mid 80s until now [1].

However, due to the time associated with the charging of EVs, users tend to be overcautious about their EV range, especially when performing long-distance routes. This compounded with the possible waiting times due to the lack of infrastructure [5] shows the need for solutions for EVRP problems. In addition to waiting times at charging stations due to rising electricity prices, EV users are increasingly more interested in minimizing energy consumption and charging costs [6]. It has been concluded that, in comparison with conventional route planning, EVRP is more challenging mainly due to the time variables associated with EV charging such as the state of charge, the power of the charging station and the occupancy state of the charging station [7], [8]. This routing problem also becomes more complex when taking into consideration the different charging costs of charging stations and the possibility of prioritizing charging costs and energy consumption over route duration.

With this context in mind, the focus of this work is to explore the different algorithms capable of finding the optimal route from a starting location to a goal location in a manner that optimizes both time and cost. Our specific goal is to find the necessary charging stations in which to make a charging operation, in order to achieve our final location. To find the correct charging stations, our algorithm needs to be able to select the best charging stations to achieve our cost and time minimization goal, on the characteristics of the charging stations such as location, charging cost, and charging power. By leveraging these station characteristics, the algorithm should be able to find the route that achieves the best balance between route duration and route cost.

1.2 Problems and Objectives

The main purpose of this master thesis is to develop a methodology that allows choosing an optimal EV route in order to minimize energy consumption, charging cost, and route duration. More specifically, we want to implement an algorithm that from the characteristics of the road network and charging infrastructure is able to find the fastest route in terms of driving time and charging time while minimizing the cost of charging. In order to fulfill this objective, the following research question must be answered:

- How to design an optimization algorithm that is able to find the fastest route in terms of driving time and charging time while minimizing the cost of recharging?
- What are the most adequate optimization algorithms for EVs routing problems?

1.3 Related Projects and Scientific Outputs

The work carried out as part of this thesis was developed under the scope of the following research project:

- **Horizon Europe EV4EU** – Electric Vehicles Management for carbon neutrality in Europe, funded by the European Union under grant agreement no. 101056765. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor the grating authority can be held responsible for them.

1.4 Outline

This document is structured as follows. The present chapter introduces the primary motivation and contributions of this thesis.

Section 2 presents some background on the definition of graph search, the shortest path problem, and the algorithms used to solve it.

Section 3 explores the advantages and disadvantages of the different current approaches to EVRP and an explanation of their methodology.

Section 4 gives a formulation for the routing and charging problem proposed for this thesis and explains the methodology that will be used to solve the optimization problem.

Section 5 presents the results obtained from the experimental setup.

Finally, in Section 6 we present the final conclusions achieved from this work and present insights into possible future work to be explored.

2 Background

Contents

To better understand the formulation of the electric vehicle route planning problem and its solution, in Section 2.1 and 2.2, its analyzed the fundamentals of graph theory and the shortest path problem, and the most used algorithm to solve shortest path problems such as our route planning problem. Section 2.3 details the fundamentals of Meta-heuristic algorithms and most in-depth the fundamentals of genetic algorithms. Section 2.4 and 2.5 presents the concepts necessary to understand and apply reinforcement learning approaches to electric vehicle route planning, and the most applicable algorithms to this problem.

2.1 Graph Theory

Graph theory consists of the study of the relationship between edges and vertices. A graph $G = (V, E)$ consists of a finite set V of vertices and a finite set E of edges. In an $edge(x, y) \in E$ directed from the vertex x to the vertex y, the vertex x is called the tail of the edge, and the vertex y is the head of the edge.

A path consists of a sequence of vertices connected by edges, i.e. in $G = (V, E)$, P is a path if for the sequence of vertices $(u_1, u_2, ..., u_k) \in V$, the edge $e(u_{i-1}, u_i) \in E$ (where $1 < i < k$). Given two vertices $u_s, u_F \in V$ u_F is called reachable if a path $P = (u_s, ..., u_F)$ starting in the vertex u_s and ending at vertex u_F exists in G.

If a weight is associated with an edge $e \in E$ by the weight function ω, then the weight of a given path $P = (u_1, ..., u_k)$ is defined by the sum of the weight associated with the edges $e(u_{i-1}, u_i) \in E$ (where $1 < i < k$) i.e., $\omega(P) = \sum_{i=1}^{k} \omega(e(u_{i-1}, u_i)$.

Given a vertex u_S, a vertex u_F, and the path P starting at u_s and ending at u_F, the weight of P is called the distance between u_S and u_F. In the case where an edge $e \in E$ has $\omega(e) < 0$ then $\omega(P)$ can be also negative. If there is no path P between two vertices then the distance between those two vertices is infinity. In the case where for any edge $e \in E$, $\omega(e) \geq 0$ there is at least one path P between vertex u_S, a vertex u_F with the smallest path weight. This path P is the shortest path between u_S and u_F.

2.2 Shortest Path Problem

The shortest path problem consists of several problems associated with the computation of the shortest path between two given vertices in a given graph.

The problems we are going to focus on mainly in this thesis are:

- The Single Pair Shortest Path (SPSP), which consists of finding the path P between the source vertex u_S and the target vertex u_T with the smallest path weight;
- The Single Source Shortest Path (SSSP), which consists of finding the paths from a source vertex u_S with the smallest path weight to the all other vertices in the graph.

In order to solve these problems, multiple algorithms have been developed, being the most predominantly used the Dijkstra's algorithm, the Bellman-Ford algorithm, and the A* algorithm.

2.2.1 Dijkstra's Algorithm

Dijkstra's algorithm is one of the widely used algorithms to solve SPSP and SSSP in a directed graph with only positive edge weights. The algorithm was first published by Edsger Dijkstra in 1959 [9]. The algorithm receives as input a directed graph, an edge weight function $\omega \rightarrow R_0^+$, and a source vertex. The output of the algorithm is the shortest path tree from the source vertex to all the other vertices in the graph, with this shortest path tree we are able to obtain the shortest path and distance value from the source to any given vertex of the graph.

Algorithm 1 Dijkstra's algorithm

```
    Input: graph $G(V, E)$, weight function $\omega$ , source vertex $s \in V$
    Output: Shortest path from s to all other vertex
  for $u \in V$ do
      dist(u) $= \infty$
      prev(u) $= null$ dist(s) $= 0$
  add u to Queue
      while Queue is not empty do
          remove u from Queue
          for all edges$(u, v) \in E$ do
              if dist(v) $>$ dist(u) $+ \omega(u, v)$ then
                  dist(v) $=$ dist(u) $+ \omega(u, v)$
                  prev(v) $= u$
```

The algorithm works by maintaining a set of distances for each node in the graph. These distances represent the shortest known distance from the source node to that node. The algorithm initially sets all distances to infinity, except for the distance to the source node, which is set to zero. It then repeatedly selects the node with the smallest distance and updates the distances to its neighbors. This process continues until all nodes in the graph have been processed, at which point the algorithm has calculated the shortest distances from the source to all other nodes. Adaptations of this algorithm can be seen used in [8, 10] to solve EVRP.

2.2.2 Bellman-Ford Algorithm

The Bellman-Ford algorithm [11] like Dijkstra's algorithm is widely used to solve SPSP and SSSP in a directed graph with the advantage over the Dijkstra algorithm of also working with negative edge weights but at the cost of not being as fast as Dijkstra's, which leads to the algorithm only being preferred in graphs with negative edge weights.

Algorithm 2 Bellman-Ford algorithm

```
  Input: graph $G(V, E)$, weight function $\omega$, source vertex $s \in V$
  Output: Shortest path from s to all other vertex
for $u \in V$ do
    dist(u) $= \infty$
    prev(u) $= null$ dist(s) $= 0$
    for  V - 1 times do
        for all edges$(u, v) \in E$ do
            dist(v) $= min\{$dist(v)$,$ dist(u) $+ \omega(u, v)\}$
```

In [12] we see the authors use the Bellman-Ford algorithm to solve their EVRP formulation due to the existence of negative weighted edges.

2.2.3 A* Algorithm

The A* algorithm [13] is an extension of Dijkstra's algorithm through the use of a heuristics function the algorithm is able to find the shortest path to a given vertex faster than Dijkstra's algorithm. In [7] we can see the use of the A* algorithm to accelerate the shortest path search for their electric vehicle route.

A* works by maintaining a priority queue of nodes that need to be visited. The priority of a node is determined by a heuristic function, which estimates the cost of the cheapest path from that node to the goal. When the algorithm is run, it initially adds the start node to the priority queue, and then it repeatedly extracts the node with the lowest priority from the queue and processes it.

Processing a node involves expanding it to its neighbors, which are the nodes that can be reached from the current node with a single edge. For each neighbor, the algorithm calculates the cost of the cheapest path from the start to that neighbor, which is the sum of the edge weights along the path. The algorithm then updates the priority of the neighbor based on the cost and the heuristic estimate and adds it to the priority queue. This process continues until the goal node is reached or there are no more nodes in the queue to process.

2.3 Meta-Heuristic Algorithms

Meta-heuristic algorithms are a set of problem-solving algorithms that unlike exact algorithms instead of finding the optimal solution no matter the time and computational cost this algorithm focus on obtaining a close to optimal solution at a better time a computational cost. These algorithms are very useful in applications where obtaining a solution quickly is more valuable than obtaining the optimal solution. Some examples of this algorithms are:

- Genetic algorithms: These algorithms are inspired by the idea of natural selection. The algorithm creates a population of possible solutions, and by applying genetic mechanisms to the members of the population with the highest fitness, the population evolves to have a population of better solutions.
- Ant Colony Optimization: This algorithm is based on the search behavior of ant colonies. The algorithm creates a population of artificial ants to explore solution paths.
- Particle Swarm Optimization: This algorithm is based on the social behavior of animals that participate in large groups such as fish. The idea is for each particle to move through the solution space adjusting its trajectory based on its own experience and the experience of the other particles.

2.3.1 Genetic Algorithms

Genetic algorithms are a problem-solving algorithm that works based on the idea of natural selection and evolution. The basis of the algorithm is to create a population of possible solutions and assign them a given fitness value depending on their performance for the given problem. By assigning a greater likelihood of being used to generate a new generation to the best-fitting solutions, the algorithm evolves to create better solutions with each generation.

The Genetic algorithm is characterized by representing the problem solution as a chromosome where parts of the solution can be separated into genes in order to apply mutation and crossover mechanisms to these chromosomes.

These mutation and crossover mechanisms are the key mechanisms that prevent the algorithm from converging in local minimums and allowing close to optimal solutions to be found. The crossover mechanism is the mechanism that combines the chromosomes from two different solutions in order to create a solution with the best parts of both solutions. The mutation mechanism is the mechanism that allows the introduction of new solution paths to the algorithm by changing part of a solution´s chromosome with new genes.

2.4 Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework for modeling decision-making. A Markov decision process is defined by a set of states and actions. These actions allow for the transitions between states and the outcome of these transitions allows us to evaluate these state-action pairs. In an MDP an agent in, a given state, at a given time has a set of actions that can be performed based on its current state and goal. By taking an action, the state of this agent changes to a new one, a reward is given to the agent based on this action which could be positive or negative. The goal in an MDP is to find a policy, which is a set of rules that the agent can follow to maximize the expected reward over time.

2.5 Reinforcement Learning

Reinforcement learning is a field of machine learning that focuses on training an agent by rewarding (and punishing) the agent depending on the actions taken, in other words, is learning to act by trial and error. In reinforcement learning, every time an action is performed by an agent in a given state, reward is given to the agent and the goal of the agent is to maximize the sum of the reward.

Reinforcement learning, in contrast with supervised learning does not need labeled training to be trained to learn the correct behavior, but instead, the training process rewards the model when it acts as intended and punishes it otherwise. In this aspect, reinforcement learning is more similar to biological learning where learning comes from rewards and punishments from trial and error.

Reinforcement learning algorithms can be divided into two categories, model-based, and model-free algorithms. In model-based algorithms, as the name suggests, the agent models its environment and from this model predicts future rewards without needing to perform those future actions. Model-free algorithms, in contrast, do not need a model of the environment of the agent and can only know the reward of a given action by performing it, this means that model-free algorithms will repeat the same actions multiple times updating their action-taking strategy based on the outcome of their actions, to maximize rewards.

There are multiple reinforcement learning algorithms that have been developed over the years, being the most commonly used:

- Q-Learning: This algorithm is based on the idea of learning an optimal action-value function, which estimates the expected reward for taking a particular action in a given state. Q-learning uses an iterative update rule to improve the estimates of the action-value function over time.
- Deep Q-Learning: This algorithm is a variant of Q-Learning that uses deep neural networks to approximate the action-value function.
- SARSA: This algorithm is similar to Q-learning, but it uses an on-policy learning approach, which means that the agent's behavior is based on its current policy.
- Actor-Critic: This algorithm consists of two neural networks, the actor and the critic, where the actor-network, learns to take actions, while the critic network learns to evaluate the actions taken

by the actor. The critic network then provides feedback to the actor-network, which helps it to learn to take better actions over time.

- Policy Gradient Methods: these algorithms optimize a policy by making small changes to it at each time step in the direction that increases the expected reward, as estimated by the gradient of the expected reward with respect to the policy parameters.

For our approach, we are going to focus on the Q-Learning and Deep Q-learning algorithms that due to it being a model free Reinforcement Learning (RL) algorithm which is necessary for a complex environment like ours, where it is difficult to obtain a reliable model, and due to its trial and error learning approach that helps to deal with the complexity and unpredictability of the interaction of our agent and its environment.

2.5.1 Q-Learning (QL)

Q-Learning [14] is a model-free, policy-free reinforcement learning algorithm. This algorithm aims to find the best action the agent can perform for a given state. The Q-learning algorithm is considered off-policy for being able to learn from taking random actions, making the need for a policy not necessary. The way the algorithm works is by having a quality function for every state-action pair the agent can perform, by exploring the environment, the model updates this quality function with the reward obtained from each action performed by the agent for a given state. Given enough training, the Q-learning algorithm is sure to converge to an approximation of the best action-value function for a given goal.

2.5.2 Deep Q-Learning (DQL)

Deep Q-Learning is a deep reinforcement learning algorithm that picks up on the concepts of Q-learning but uses a deep neural network to approximate the Q-function. In traditional Q-learning, the Q-function is typically represented as a table of state-action values. This can be impractical for large or continuous state spaces because the size of the table grows exponentially with the number of states and actions. Deep Q-learning addresses this problem by using a neural network to approximate the Q-function. The neural network takes as input the current state of the environment and outputs a predicted Q-value for each possible action. The Q-values are then used to choose the next action in the same way as in traditional Q-learning.

Due to its complexity EVRP is impractical to solve with a look-up table of state-action and therefore the use of neural networks to compute the state-action values is necessary. In [15–17] we see the use of neural networks to improve the Q-learning algorithm with positive results

One of the key challenges in using deep Q-learning is that the Q-function is not differentiable, which means that standard backpropagation cannot be used to train the neural network. Instead, the network is trained using a variant of stochastic gradient descent called the Q-learning update rule. This update rule adjusts the network weights to minimize the error between the predicted Q-values and the target Q-values, which are the expected rewards of taking each action in the current state.

2.5.3 Dueling Network Deep Q-Learning (DNDQL)

The Dueling Network DQL [18] is a variation of the DQL algorithm that was created to improve the efficiency and effectiveness of deep Q-learning, particularly in situations where it is challenging to estimate the values of different actions accurately. The main difference between DQL and DNDQL is the Q-value estimation. While DQL uses a neural network to estimate its Q-values, the DNDQL decomposes the Q-value of a state-action pair into two components:

- State Value: which estimates the value of being in a particular given state.
- Action Advantage Value: which estimates the advantage of taking each possible action in the given state, capturing the difference in expected rewards between different actions in the same state.

This decomposition improves the algorithm's learning efficiency and can lead to more stable and faster convergence for reinforcement learning tasks in complex environments.

3

Related work

Contents

Since the focus of this thesis is on the optimal routing of electric vehicles considering the state of charging stations, Section 3.1 explores the use of search-based and heuristic-based solutions to Electric Vehicle Routing and Charging (EVRC). In section 3.3 it is analyzed the use of reinforcement learning in EVRC. Section 3.4 explains the use and importance of realistic parameterization of EV consumption and charging characteristics.

3.1 Conventional Search-based Approach to EVRP

The classical approach to this EV route planning is by using graph search algorithms such as Dijkstra, Bellman-Ford, and heuristic accelerated search algorithms such as A-star [3,7,8,10,12]. These search algorithms are single-label algorithms but they can be adapted to have a label set for each vertex and therefore tackle multivariable problems like EVRC. In [12] the authors use an adaptation of the Bellman-Ford algorithm to optimize EVRC taking into account the energy consumption of the EV depending on its velocity, and in [8] the authors use an adaptation of Dijkstra's algorithm to minimize not only travel time, but also, the cost of energy recharging. Some authors also used heuristic accelerated algorithms [3,7], which prove to improve the runtime in comparison to normal search-based algorithms. To use these algorithms to solve our EVRP problem, the road network is converted into a graph network where the charging stations are nodes that make part of the graph. This algorithm requires, however, some pre-processing, as converting the entire road network into a graph is inefficient and makes the algorithm's runtime significantly worse, therefore, by only converting a small part of the road network, the part where we are most likely to find our solution, we can speed up the algorithm runtime and diminish the complexity of the system [7]. This pre-processing of the road network appears to be a good solution for short routes in small road networks where it is possible to find the sample of the road network where the optimal or close to optimal solutions will be. When scaling the problem to long-distance routes due to the complexity of the graph problem and the runtime of the algorithms being directly tied to the number of nodes in the graph i.e. to the size of the route, these algorithms become unsuited as the runtime is either excessive or the sample of the road network is too small to obtain a meaningful solution.

3.2 Meta-Heuristic Algorithms in EVRP

One of the major shortcomings of the classical SPSP algorithms, like Dijkstra, is their time complexity being exponential, which means that for graphs with a large number of nodes, their runtime increases significantly making these algorithms less suited for these graphs. To solve this problem, meta-heuristic algorithms such as Genetic Algorithm (GA), have been used to solve EVRP. Meta-heuristic algorithms manage to solve these search problems in a more reasonable runtime than algorithms like Dijkstra.

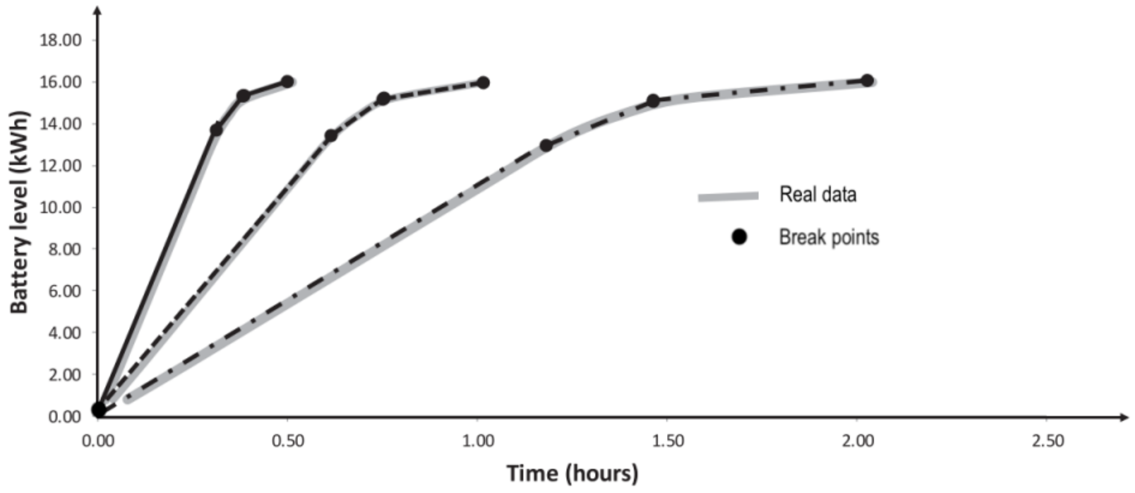
They achieve this by returning a near-optimal solution instead of the optimal solution in a much faster runtime. Due to this, these algorithms can be incredibly useful in use cases where runtime is more important than optimality. In [19] the authors use a genetic algorithm to route electric delivery vehicles to distribute cargo throughout multiple sites. In their implementation of the genetic algorithm, they used the order of sites to visit as its chromosome. In their problem, Shao et al. [19] want to minimize the monetary cost associated with these routes. To achieve this they use the direct monetary characteristics of the route, such as the charging cost, and convert the non-monetary characteristics of the route by associating a monetary cost for the distance traveled and the time charging and loading the vehicle. Shao et al. [19] applied their methodology to a discrete representation of a realistic road network in the Beijing urban area. They concluded that ”The results indicate that the GA yields acceptable performance on computational time, convergence, and solution quality. ” [19]. In [20], the authors used a variation of the Genetic algorithm called the re-insertion Genetic algorithm to optimize the route of electric delivery vehicles to distribute cargo to multiple clients. The algorithm used in [20], differs from the normal Genetic algorithm by first using a nearest neighbor-like mechanism to generate the initial population, which proves to obtain faster converging time in comparison to a random generation process. The algorithm also changes the mutation process of the genetic algorithm, by instead of using a random process to create the mutations in the chromosome, the algorithm uses a function to determine the correlation between stops and removes and reinserts in a different order the stops with the highest correlation between them. This methodology speeds up the convergence of the algorithm due to focusing on changing the segments of the chromosome that might impact its performance the most. C. Li et al. [20] like [19] uses the order of stops the vehicle needs to make to complete its route as the chromosome. Their fitness is calculated as the inverse of the sum of the cost of the energy consumed, the charging cost, and a cost associated with time window penalties(taking more than the defined time to complete the route). To evaluate the performance of their algorithm, C. Li et all. [20], apply their algorithm in the New York apple production area and use the algorithm to visit multiple apple producers in a given time window. From their results, the authors concluded that not only the algorithm proved to achieve good results performing the task, but also outperformed classical implementations of the Genetic algorithm in terms of convergence speed and performance. In [21], the authors use a type of evolutionary algorithm called Differential Evolution (DE) to solve the route and charge schedule of a EV transport shuttle in an airport area. The DE algorithm uses the same genetic operators selection, crossover, and mutation as the GA, but applies them in a different order. The algorithm starts by creating new chromosomes from the previous generation by mutating them and applying the crossover mechanism between these new chromosomes. After creating this new population, only then it selects the chromosomes that have the target fitness for the new generation. Similar to the other implementations in [21] the chromosome consists of the sequence of locations the transport EV needs to travel to complete its route. Barco et all. [21] compare the results obtained from their algorithm to the results obtained from a GA, and their results show the DE was capable of achieving a significantly faster convergence time with a performance very close to the GA.

3.3 Reinforcement Learning in EVRP

In recent years, in order to overcome the problems derived from classical SPSP algorithms and meta-heuristic-based algorithms, reinforcement learning algorithms have been used to solve EVRP. In [15] the authors use DQL to optimize the route of multiple EV's to charging stations in order to minimize waiting times and traffic. This centralized control is applied in a small network where the state of roads, charging stations, and EVs is known which makes the algorithm obtain good results in reducing traffic and waiting times in charging stations and shows promising results at dealing with the uncertainty associated with realistic traffic behaviors. In [16], it is proposed a similar approach to optimize the route of EVs to charging stations but instead of multiple vehicles, the authors focus on a single vehicle to charging station routing, by not controlling the parameters of the entire network, the authors implement a feature extraction mechanism in order to adapt to traffic conditions, charging station waiting times, and energy prices. In this paper, we can also see that this approach obtained good results in minimizing the recharging cost associated with the trip. In [17], the author uses a dueling Deep Q-Learning algorithm to optimize the route of an electric vehicle to its final goal. The use of DNDQL shows to be useful to better evaluate the quality of different actions that lead to similar rewards. The approach on [17] is more suited to EVRC having similar goals to the approaches presented in Section 3.1 but without the scaling downsides associated with search-based approaches. In [15, 16], is proposed the use of RL applied with great success to small local networks which although presenting great results do not deal with long-distance routes with multiple recharges where these algorithms are most useful. In [16], the minimization of recharging cost shows great results, which entails that the minimization of recharging cost applied to an approach similar to the one in [17] can lead to an even better level of route optimization.

3.4 EV Consumption and Charging

The introduction of more realistic models of the physical characteristics of EVs was first introduced by Zundorf [22]. Zundorf, instead of using a linear relationship between charge percentage and charging time, and like some other authors assuming that all the recharging actions were full recharging, used a charging curve composed of multiple linear functions to approximate the realistic charging curve between charge percentage intervals. In an EV charging curve, we can observe that from 0% to 80% the charging curve behaves very linearly, but for the last 20%, this curve stops being linear due to the decrease in input current in order to protect the battery. Due to this, for this last 20%, Zundorf divides into intervals from 80% to 90%, and from 90% to 100%, using a different linearization for each interval. Since the publication of [22], most research in EVRC has taken into consideration some kind of realistic charging curve, in [8] the EVs are constrained to charging to a maximum of 80%, in order to only deal with the linear part of the charging curve. In Figure 3.1, we can see that the authors in [2] used a piecewise linear approximation for different types of charging stations, and in Figure 4.10, we can see the approach used in [3] where the authors divide the charging curve into two charge percentage intervals, from 0 to 80% and from 80% to full charge. They then from data collected from the charging of an EV, make a linear regression to obtain the curve for the first interval. For the second interval, instead of a second linearization, they make a polynomial regression. By using a polynomial regression for the last charge interval, the authors don't need multiple linearizations to describe the charging curve correctly, as the charging curve closely resembles a polynomial function at this charge percentage. In [7], the authors use the normal values of current and voltage in the charging of a lithium-ion battery to obtain their realistic charging model.



Type of CSs	Slow	Moderate	Fast
Charging power (kWh/h)	11	22	44
Piecewise linear approximation	— · —	------	——

Figure 3.1: Piecewise linear approximation for different types of CS charging an EV with a battery of 16 kWh used in [2].

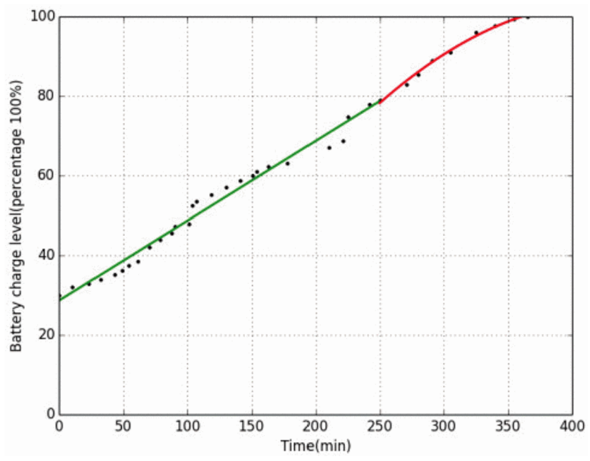


Figure 3.2: Charging curve used in [3].

Some authors have also used the relationship between an EV's velocity and its energy consumption to reduce the total energy consumption, increasing the distance the EV can last without needing to charge, and even reducing Electric Vehicle Charging Station (EVCS) waiting times. In [12], the authors use a realistic model of the forces applied to the vehicle during motion, to forecast the energy consumption of the EV for different velocities. With this forecast applied to their algorithm, the obtained results show that by driving at a slower speed to conserve energy, the total route time ends up being lower due to avoiding extra charging stops. In [7], the authors analyze the energy consumption/speed curve of two real-life electric vehicles, a Renault Zoe and a Tesla Model S, and find a function that best approximates the consumption curve of these vehicles. From this curve, and due to the use of their database to know how long the EV will have to wait at the charging station, the algorithm can choose a route with a speed that minimizes energy consumption and uses the time the EV would be waiting in the charging station.

3.5 Summary of Literature Review

From the analysis of the literature reviewed throughout this section, we can notice that the problem of electric vehicle route planning, from a start location to a target location, with multiple charging operations, is mainly solved with classic search-based algorithms that, although optimal, require large amounts of computing runtime and processing power. This leads them to be unsuited when scaling this problem to larger road networks. From the literature focused on the use of reinforcement learning, we can observe that the results obtained from its use in small networks that optimize multiple criteria, most importantly the cost of the charging operation, are very promising [15, 16]. Although these papers only focus on current location to single charging station routes, it can be observed in [17], the use of reinforcement learning for electric vehicle route planning, from a start location to a target location, with multiple charging operations, but with a reduced set of optimization goals, most notably the optimization of the charging cost from charging stations with different associated charging prices. With all this, we can find a lack of literature in large road networks electric vehicle route planning, from a start location to a target location with multiple charging operations, that not only optimize route duration and energy spent but also the cost of the multiple charging operations.

Table 3.1 presents a comparison of the different objectives and challenges observed in the papers reviewed in this section, and the algorithms used to solve them.

Table 3.1: Summary of literature review.

Ref	objective	methodology	missing
[15]	Centralized EV routing to charging station to minimize traffic and waiting times.	RL DQL	Only solves small local routes to CS. It does not account for CS energy price. Does not take into account mid-travel charging.
[16]	Single EV optimal route to charging station taking into	RL DQL	Only solves small local routes to CS. Energy prices are an approximation. Does not take into account mid-travel charging.
[17]	Single EV optimal route to final destinantion	RL DDQL	Does not take into consideration CS energy price.
[19]	Single delivery EV optimal route through predefined stops	Genetic algorithm	Does not solve large distance routes.
[20]	Single delivery EV optimal route through predefined stops	Re-insertion Genetic algorithm	Focus on the cost of the energy spent during the route. Does not diferentiate charging stations.
[21]	Centralized delivery EV optimal routing through predefined stops	Differential Evolution	only solves small routes around case study area. Only optimizes routes duration
[12]	Single EV optimal route to final destination using realistic energy consumption and realistic charging	Adapted Bellman-Ford	Need of historical data for speed prediction. Difficulty scaling due to graph size complexity. Does not take into account CS energy prices.
[8]	Single EV optimal route to final destination using day ahead energy pricing.	Dynamic programming Adapted Dijkstra´s algorithm	Need of historical data for speed prediction. Difficulty scaling due to graph size complexity. Day-ahead energy pricing works well for private CS but not for public ones.
[10]	Centralized EV routing using mobile charging stations.	Dijkstra´s algorithm	Requires mobile charging stations that follow predetermined routes. Only solves local routes. Does not take into account CS energy prices.
[3]	Single EV optimal route to final destination using realistic charging.	A* algorithm	Difficulty scaling due to graph size complexity. complexity. Does not take into account CS energy prices.
[7]	SIngle EV optimal route to final destination using centralized Database.	A* algorithm	Difficulty scaling due to graph size complexity. Need for a centralized database to obtain predicted waiting times of each CS based on historical data.
	Objective	**methodology**	
This Thesis	Single EV long distance optimal route and charging using real charging station data	Djikstra’s vs Genetic algorithm vs RL DDQL	

4 Methodology

Contents

4.1 Problem Formulation

In the Section 4.1a, is presented the Formulation for our route and charging optimization problem in a mathematical formulation based in the formulation for electric vehicle route and planning formulated in [10, 16, 17], where all the variables are defined in Table 4.1.

Table 4.1: Formulation Variables.

Symbol	Description	unit
V	Set of nodes	
s	Starting node	
g	Goal node	
(i, j)	Path between node i and node j	
td_{ij}	Drive time between node i and node j	hour
tc_j	Charging time of charging operation in node j	hour
cc_j	Charging cost of charging operation in node j	Euro €
x_{ij}	Binary decision variable to identify selected route	0-1
y_j	Binary decision variable to identify selected charging station	0-1
γ	Multi-objective optimization weight	

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j \in E, i \neq j} \Big[(cc_j y_j)\gamma + (td_{ij}x_{ij} + tc_j y_j)(1-\gamma) \Big] && && \text{(4.1a)}\\
\text{subject to} \quad & \sum x_{sj} - \sum x_{js} = 1 && \forall j \in V, && \text{(4.1b)}\\
& \sum x_{gi} - \sum x_{ig} = -1 && \forall i \in V, && \text{(4.1c)}\\
& x_{ij} - y_j \geq 0 && \forall (i,j) \in V, && \text{(4.1d)}\\
& x_{ij} = \{0, 1\} && \forall (i,j) \in V, && \text{(4.1e)}\\
& y_j = \{0, 1\} && \forall j \in V && \text{(4.1f)}
\end{aligned}$$

Constraints 4.1b and 4.1c ensures the restriction imposed by the EVRC, where the EV always starts at a vertex s and always arrives at a vertex g by following a sequential path of vertices $\in V$. Constraint 4.1d ensures the EV is only able to receive energy from charging stations that are in a reachable path. Constraint 4.1e defines the binary decision variable x_{ij} which indicates whether the EV has followed the path from i to j or not. Constraint 4.1f defines the binary decision variable y_j which when $y_j = 1$ indicates that the path from i to j leads to a charging station where the EV has received energy and when $y_{ij} = 0$ indicates the path from i to j does not lead to a charging station and the EV has not received energy.

4.1.1 Markov Decision Process Formulation

The MDP for our route and charging problem is composed of a state space (S) and an action space (A). In every state $s \in S$, the state is composed by the State of Charge (SoC) of the EV and its coordinates. The action space is composed of the actions that link the states in S as depicted in Figure 4.1.

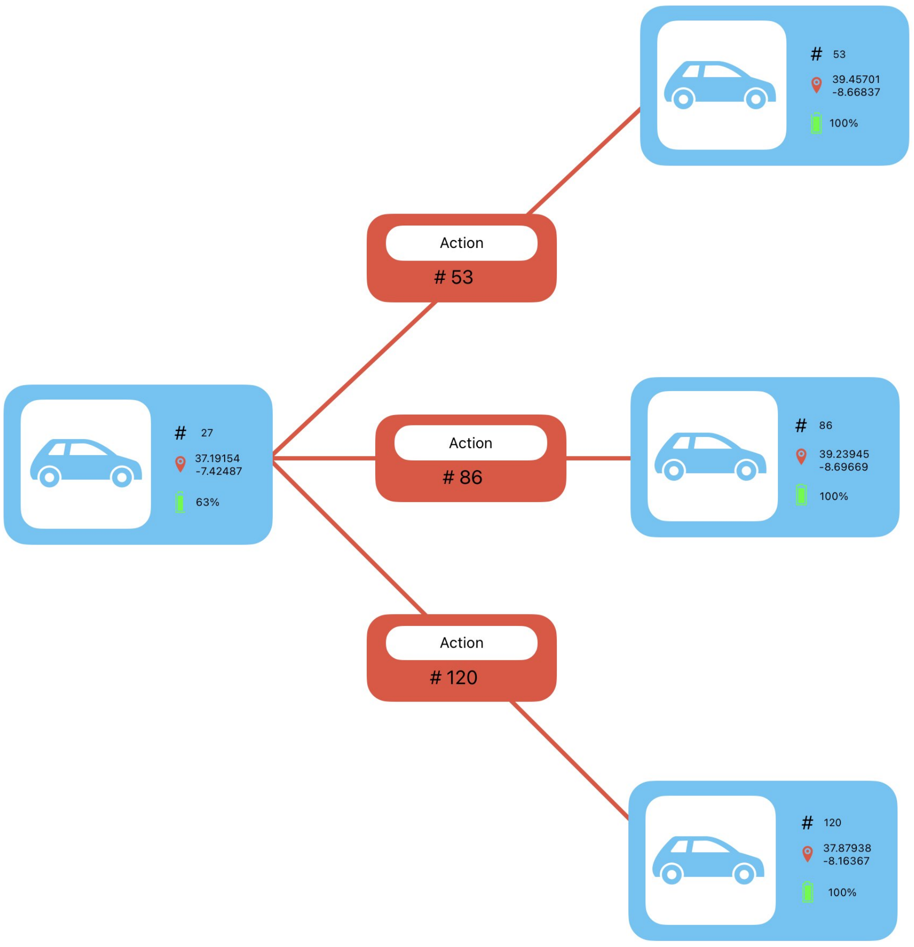


Figure 4.1: State and action representation.

Each action describes a path for the EV to drive, which implies a driving time and energy consumption associated with the length of this path. The action also defines the existence and characteristics of a charging station such as the number of charging ports, the type and power associated to each charging port, and the price associated with the charging action. For each state transition associated with an action, a reward must be given to the agent. This reward must take into account several factors:

- The cability of the EV of achieving the next station given the SoC. In order to prevent the agent from running out of energy during a route between two stations, the reward must penalize actions that attempt to achieve unreachable stations.

- The energy cost and time spent in a recharging action, for the same amount of energy recharged a station that can provide a faster charging stop time (waiting time + charging time) and cheaper charging cost should obtain a higher reward value.

- The number of recharging actions already performed, this is to prevent unnecessary charging stops by penalizing recharging actions.

From the reward given to an action performed by the agent in a given state we obtain the state-action value, and from the cumulative sum of the state-action values rewarded to get to a given state we obtain the state value.

4.2 Environment Formulation

In order to implement our solutions to the EVRP problem, we need to be able to model not only the real-life road network and charging infrastructure, but also, the behavior of a typical EV. This section will focus on the technologies utilized to model these parameters and obtain the data, required for the correct implementation of our algorithms.

4.2.1 Routing API

The basis of EVRP settles in deciding which charging station a given EV should stop to charge. In order to verify if the chosen charging stations are the best, we need to estimate the best route between them, and correctly estimate the time and energy required to perform these routes. For our approach, we relied on an open-source software called OpenRouteService (ORS) [23], which provides multiple services based on crowd-sourced geographical data from OpenStreetMap [24]. For our solution, we used a local client of their software which allows us to execute unlimited requests of routes, and used their route service by making a query with an initial coordinate and final coordinate to the service. This query provided us with an estimation of the time required to perform the route, the road distance in km of the route, and an array of coordinates detailing the said route.

4.2.2 Charging Station Pre-processing

As stated in Section 4.2.1, the basis of EVRP settles in deciding which charging station a given EV should stop to charge. Therefore, the model of this charging infrastructure needs to be as close as possible to reality. In order to achieve this, we relied upon the acquisition of data from an open-source API called Open Charge Map (OCM) [25]. With this API we were able to make a query to obtain the data from all charging stations in their database for a given geographic area. The data provided by this

API ranged from its geographic coordinates to the number and power of the charging station ports. As stated, this API relied on the definition of a geographic area of where we wanted to find charging station data. To define this area, the API gave us multiple methods from which we decided to use the bounding box method, where we would send two geographic coordinates, and the API would consider the area to be the box created by these two coordinates latitude and longitude values. In order to find the best two coordinates for a given trip the driving optimal route is calculated using our Routing API presented in Section 4.2.1 and from this route, we find the extreme north, south, east, and west points of our route and add an extra 20km in order to not remove potentially useful charging stations.

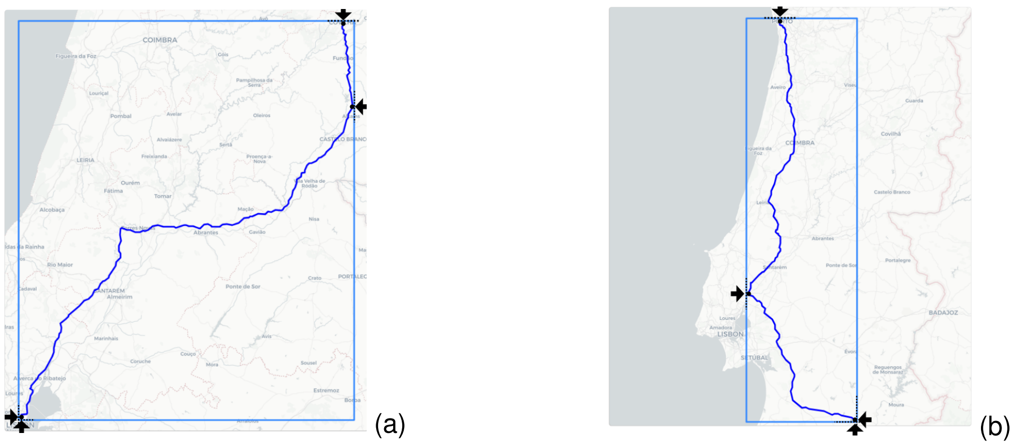


Figure 4.2: Bounding box for a given route (a) - route from Lisboa to Covilhã (b) - route from Porto to Beja.

However, in some cases, this approach still resulted in an excessive number of impractical charging stations, mainly because they were situated at considerable distances from the optimal driving route. In order to fix this issue we filtered the stations obtained from the API by their location by checking if their distance to the optimal driving route didn't surpass a certain threshold. To achieve this for a given threshold of 10Km we would choose a point in the optimal driving route every 10km and find which stations are located less than 10km away from this point.

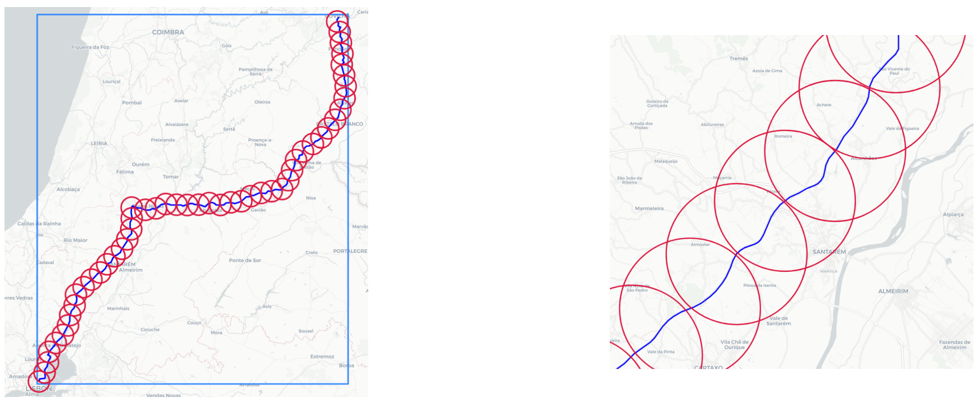


Figure 4.3: Station filtering where the radius of each circle is the chosen threshold.

4.2.3 Electric Vehicle Model Formulation

To achieve reliable results with our solution to the EVRP problem we need to have a good electric vehicle model. To achieve this we need to model the key parameters of the EV for our problem. These parameters are the EV's battery capacity and the energy consumption rate. With these two values, we are then able to compute the EV's driving range. With this model of the EV and with the information provided by the charging station API and the Routing API we are able to compute the energy consumed by each route and the time taken to recharge at each station.

4.3 Electric Vehicle Route Planning using Dijkstra

In order to solve the problem detailed in Section 4.1, we will employ the use of the Dijkstra's algorithm to find the optimal route for our formulation. Dijkstra's algorithm is a search-based algorithm used to find the shortest path between nodes in a given graph. By systematically exploring and evaluating potential routes, it enables us to determine the most efficient solution to our problem.

4.3.1 EVRP Graph Formulation

In order to use our search algorithm the first step is to convert our environment into the form of a graph that our algorithm can then use to find the optimal route. In our environment, we can see that the road infrastructure and the charging stations already behave similarly to a graph where the roads connect the multiple charging stations just like edges connect multiple nodes in a graph, therefore, the most logical approach is to use each charging station as the graph node and the graph edges represent the cost associated to moving from one station to another.

Nodes

Each charging station has different characteristics, these characteristics are what make a charging station a good or a poor choice for recharging. Therefore, these parameters need to be well described in their corresponding node. For this, we described each node with a dictionary containing the most important of these characteristics, this being its geographic coordinates in order to calculate the route characteristics to reach this Charging Station (CS), and the number of ports and their respective characteristics in order to calculate the charging characteristics of the CS.

```
CS_node = {
    "index": defined index
    "Latitude": latitude coordinate,
    "Longitude": longitude coordinate,
    "number of ports": number of charging ports,
    "ports": [
            {"A": port amperage, "V": port voltage, "KW" port power,
               "type": port type of current},
            {"A": port amperage, "V": port voltage, "KW" port power,
               "type": port type of current},
            ...
            ]
    }
```

Listing 4.1: Stop node implementation

Edges

In a weighted graph, the edges of the graph describe the cost associated with moving from one particular node to another. In our environment, this cost refers not only to the time cost associated with driving from one node to another, but also, the time cost associated with the charging operation performed at the new node and its associated monetary cost.

States

In our algorithm, we want to find the path with the lowest cost from the initial state to the goal state. We defined a state as a tuple containing the current node, the EV battery state percentage, and the path done to achieve the current node.

```
State = [  CS_node , battery_state, [path]]
```

Listing 4.2: State implementation

4.3.2 Dijkstra's Implementation Diagram

In figure 4.4 is depicted the flowchart design of our implementation of the Dijkstra algorithm to solve our EVRP problem.

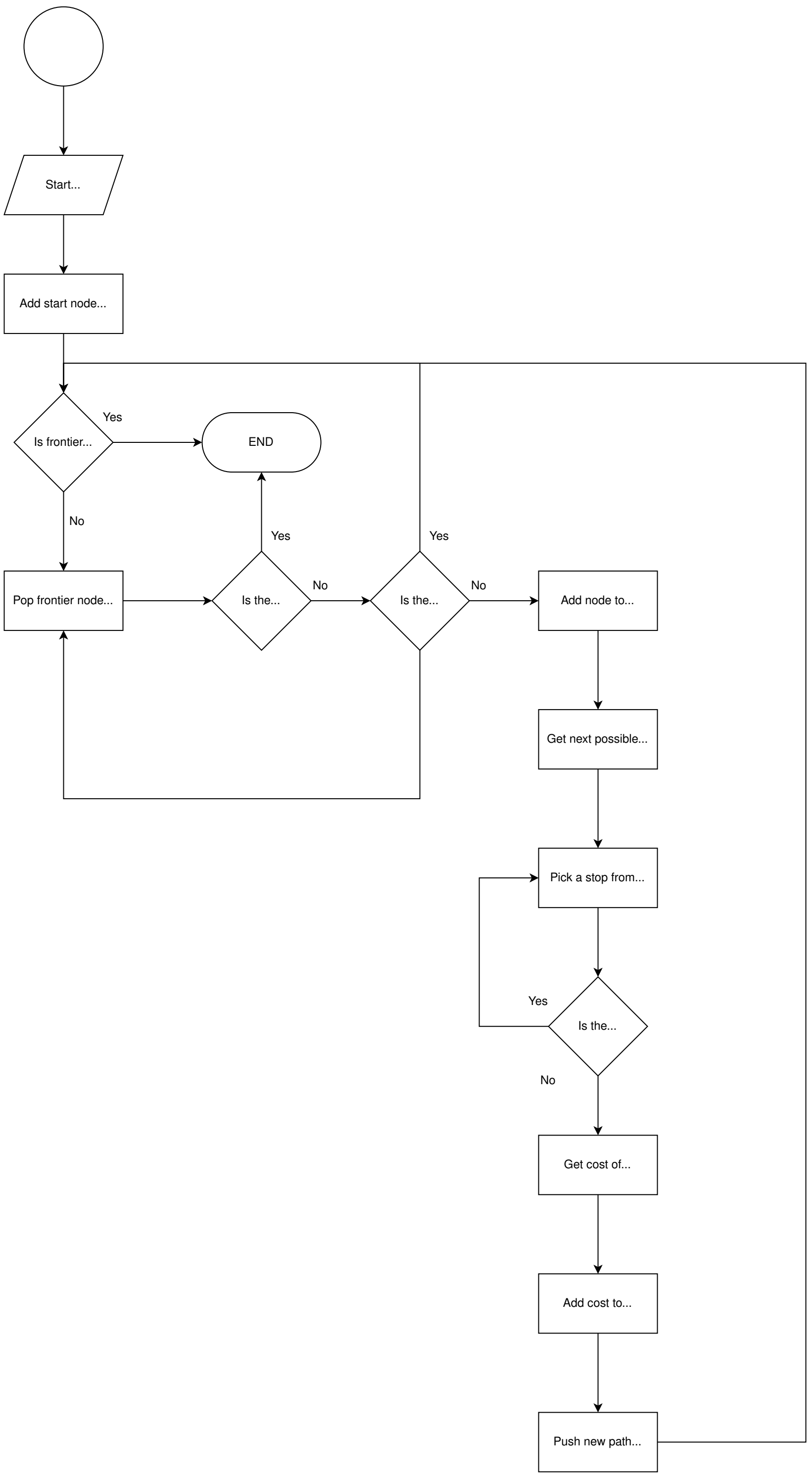


Figure 4.4: Implemented Dijkstra's algorithm diagram.

4.4 Electric Vehicle Route Planning using GA

In order to solve our problem formulation defined in Section 4.1, we are going to employ the implementation of a Genetic Algorithm. Genetic Algorithms operate by utilizing genetic mechanisms that mimic the principles of natural selection, crossover, and mutation to iteratively enhance solution populations across generations. This approach enables us to systematically refine and evolve solutions to the specified problem.

In Figure 4.5 is depicted the flowchart design of our implementation of the GA to solve our EVRP problem

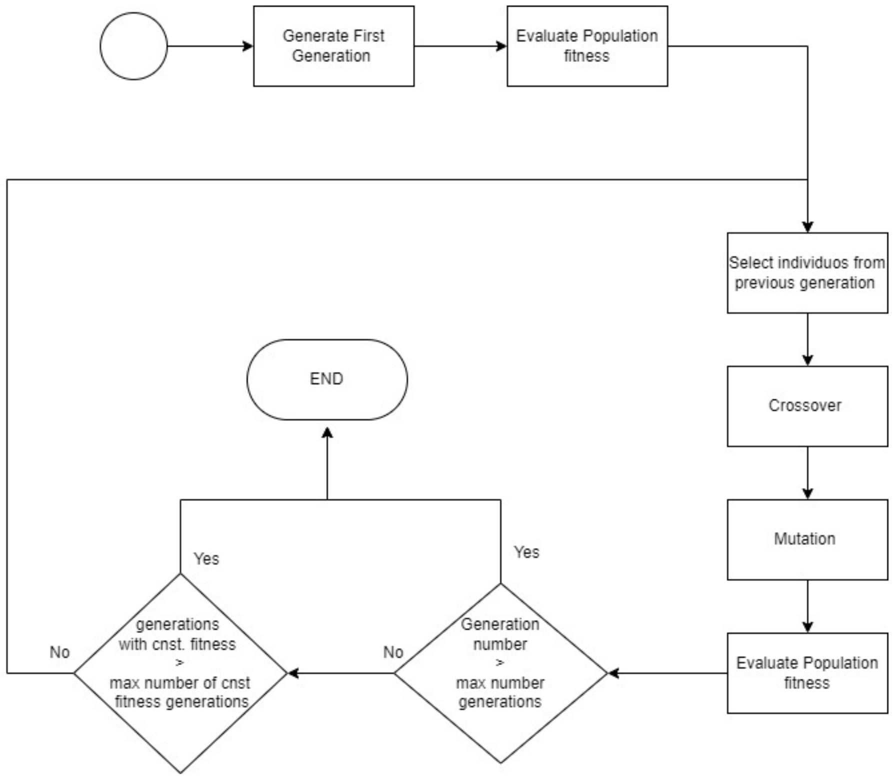

Generate First Generation
Evaluate Population fitness
Select individuos from previous generation
Crossover
Mutation
Evaluate Population fitness
END
Yes
Yes
No
No
generations with cnst. fitness > max number of cnst fitness generations
Generation number > max number generations

Figure 4.5: Implemented GA diagram.

4.4.1 Chromosome

The first step in any GA implementation is to define our chromosomes. This chromosome describes the behavior of each individual and is in this chromosome that the crossover and mutation mechanisms will make changes.

In our implementation, the chromosome consisted of a list where the first element contains the number of stops of the individual. The second element consists of a list of the ordered stop nodes that define the individual's route. The last element consists of a list with the order index of the stop nodes of the individual's route. This implementation can be seen in the Listing 4.3.

```
chromosome = [ Number_of_stops, [stop_nodes], [path]]
```

Listing 4.3: Chromosome implementation

In order for the algorithm to function as intended only valid chromosomes were accepted. For a chromosome to be deemed valid it must satisfy these four rules:

1. The number of stops must be between the maximum and minimum number of stops for a given route;
2. Each stop must be reachable from the previous stops;
3. The first stop must be reachable from the starting position;
4. The target position must be reachable from the last stop.

4.4.2 First Generation Creation

To create viable individuals, the algorithm starts by defining for the given start and target position the list of stops reachable from the start position, the list of stops that can reach the target position, the minimum number of stops, and the maximum number of stops. The minimum number of stops was calculated by getting the road distance between the initial position and the final position and then dividing this distance by the EV driving range. The maximum number of stops is then defined as $max\_N\_stops = 2 \times min\_N\_stops + 1$.

With these lists and values, to create each viable individual the algorithm first chooses the number of stops of the individual from a random integer between the minimum number of stops and the maximum number of stops. After that, the first stop is randomly selected from the list of stops reachable from the start position, this ensures the 3rd rule is always met. The subsequent stops until the last stop are randomly selected from the stops that are not reachable from the start position and do not reach the goal position. Finally, the last stop is randomly selected from the list of stops that reach the goal position also ensuring the following of the 4th rule. After the initial creation of the individual, in order to ensure that the 2nd rule is followed, the individual is evaluated to check if all stops are reachable from the previous stop. If during the evaluation, the algorithm detects that one stop is not reachable from the previous stop a new stop is randomly selected until a reachable stop is found.

4.4.3 Fitness Function

The objective of the GA is to find the solutions most suited to solve our problem. In order to find this solution there needs to be defined a fitness parameter to evaluate how good the solution of a given individual of the population is. This fitness parameter needs to correctly describe the performance of the individual in solving the problem at hand. For our problem, the fitness value was computed from two components:

- The time component (C_{time}) - consisting of the time associated with driving between stations and the time associated with the recharging operations.
- The cost component (C_{cost}) - consisting of the cost associated with the recharging operation.

The time component is defined in Equation 4.2 by adding the total driving time of the chromosomes route with the total charging time. In order to penalize unnecessary stops, an extra term is added to represent the time cost of stopping in a charging station. This cost represents the real-time cost of arriving at the charging station, the time of connecting the car to the charging station, and the time of disconnecting and paying for the charging operation. This charging station cost is constant for every station.

$$C_{time} = \sum t_{driving} + \sum t_{charging} + \sum t_{CS} \tag{4.2}$$

The cost component is only comprised of the sum of the cost of all of the charging operations of the route and is defined in Equation 4.3.

$$C_{cost} = \sum m_{charging} \tag{4.3}$$

In order to combine these components in a single cost, a weight γ is used to find an optimization balance between the two components. This is combined cost is defined in Equation 4.4.

$$C_{combined} = (C_{cost})\gamma + (C_{time})(1-\gamma) \tag{4.4}$$

This combined cost, however, increases as the time component and the cost component of the route increases. This would mean that the fittest individuals would be the ones that took the most time and cost consuming route. Therefore, for the fittest individuals to be the most time and cost saving routes, our fitness score consists of the inverse value of the combined cost. This is defined in Equation 4.5.

$$Fitness = \frac{1}{C_{combined}} \tag{4.5}$$

4.4.4 Selection Mechanism

To create a new generation, a method to select individuals is necessary not only to select good individuals for our new generations, but also, to select good individuals to use in our crossover and mutation mechanisms. In order to create a new generation that tends to be fitter than the one before the first logical idea would be always to select the top performers of each generation. While this solution might improve the fitness from one generation to the next, it will quickly diminish the solution diversity of the population, which can lead to the algorithm becoming stuck at a local minimum. In order to select good individuals, the selection mechanism should tend to select the best-performing individuals but also select not as good performing individuals every now and then. In order to achieve this selection method we utilized two common selection algorithms:

- The tournament selector.
- The biased roulette selector.

The tournament selector consists of randomly choosing two individuals from the population and choosing the one with the highest fitness value. This method ensures that any individual from the population can be selected except the worst-performing individual of the population, which will always have a lower fitness value than any other individual in the population. Although this method tends to select higher fitness value individuals, it is not guaranteed that the method will choose individuals from the top 10% best performers more times than from the 50% best performers.

The biased roulette selector solves this problem by assigning probabilities of being chosen to each individual of the population depending on its fitness value, i.e. the higher the fitness value of an individual, the higher the probability of being selected. For our implementation, we summed the fitness values of the entire population, and then randomly selected a number from 0 to the sum of the fitness values. Then from the highest fitness value individuals to the lowest, we sum the fitness values until this sum is higher than the selected number and choose the last summed individual. This ensures that the best-performing individuals will always have a greater probability of being selected.

In order to get the benefits of both methods, whenever a selection of an individual of the population was needed, one of these methods was chosen randomly with the same probability for either one.

4.4.5 Crossover Mechanism

The crossover mechanism is an important part of the generation creation. In each generation, we will find individuals that might excel at some parts of its solution, by using the crossover mechanism the algorithm is able to combine the best parts of some individuals, to obtain a better-performing individual than the two individuals separately.

The basis for the crossover mechanism is to select two individuals and combine parts of their chromosomes to create a new individual chromosome. In our problem, the sequence of the stops is as important as the stops themselves. Therefore, the combination of the two chromosomes needs to preserve good sequences of stops.

To achieve this, in our implementation, the algorithm begins by selecting two individuals from the old generation using our selection mechanism to be our parent individuals. Then the number of stops of the new ”child” individual is chosen to be the number of stops of one of the parent individuals.

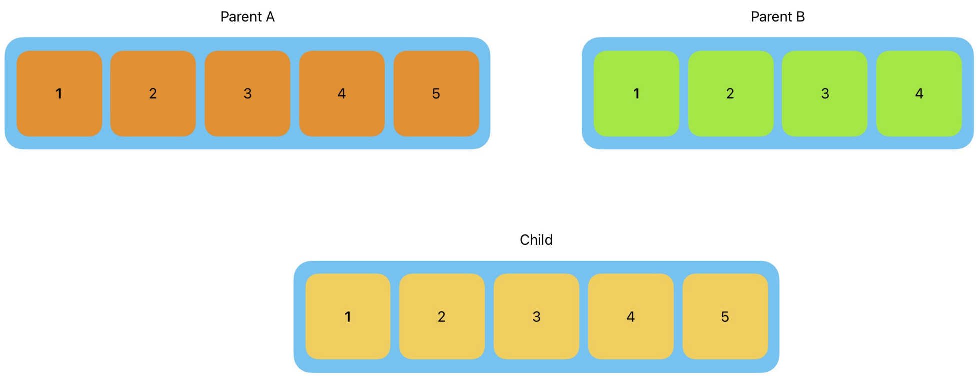


Figure 4.6: Parent chromosomes and child chromosome.

After this, in order to preserve sequences of stops, the algorithm splits the child’s chromosome into a randomly chosen point, between the first stop and the last stop. This split creates two chromosome segments, the first segment from the 1st stop to the splitting point, and the second segment from the splitting point to the final stop.

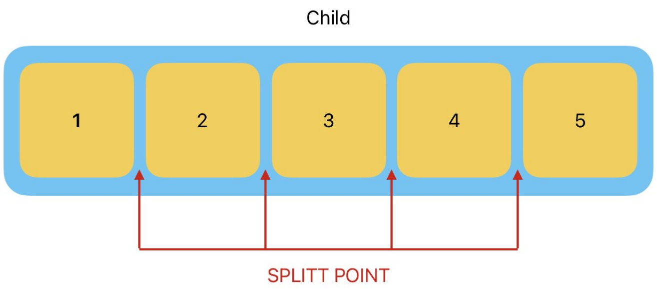


Figure 4.7: Possible split points.

In order to create this new chromosome consisting of two chromosome segments, we create 4 chromosome segments from the parent chromosome. For the first chromosome segment of the child, for a given number of stops N, we use the first N stops of each parent to create two chromosome segments. For the second chromosome segment of the child, for a given number of stops X, we use the final X stops of each parent to create the other two chromosome segments.

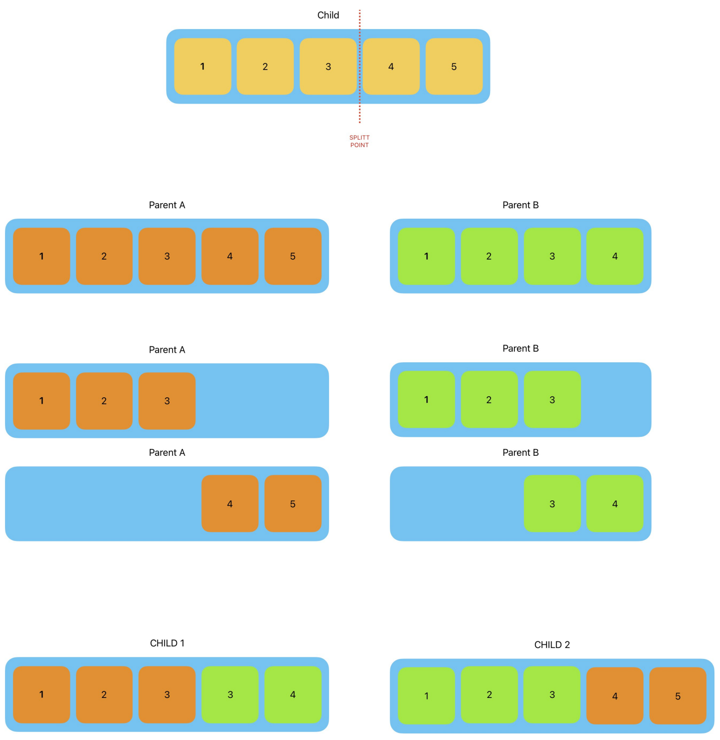


Figure 4.8: Child chromosomes created from parent chromosome segments

To find the best combination of the parent chromosomes, two child chromosomes are created. In one of the child chromosomes, the first segment comes from parent A and the second segment from parent B. On the other child chromosome, the first segment comes from parent B and the second segment from parent A. Then each child is evaluated and the best child is the one chosen for the new generation. If during the creation of the two child chromosomes, we end up with an invalid chromosome for one of them, the algorithm automatically chooses the other child. In the case where both child chromosomes are invalid, the algorithm restarts the crossover mechanism with a new set of parents.

4.4.6 Mutation Mechanism

The mutation mechanism is the mechanism that brings genetic variability to the solution population. During the creation of the new generation, the selection mechanism only allows the selection from already seen individuals of the population, and while the crossover mechanism allows the algorithm to create new individuals, the new individuals are merely combinations of the chromosomes of already seen individuals of the population. This means that without any other mechanism, the algorithm would only be able to find a local optimal solution based on the initial population. By having a mutation mechanism we allow the introduction of new chromosome combinations that haven't yet been seen in the population. This allows us to achieve a solution that is closer to the optimal solution without depending on the initial population chromosomes.

For our implementation, we consider a mutation as changing one of the stops of the chromosome with a new stop. Due to the nature of our problem, swapping one stop for another random stop would in the majority of cases, lead to an invalid chromosome due to having an unreachable stop. To prevent this, instead of choosing a random stop from the entire stop list, we only consider stops that are within a 20 km radius of the stop we want to replace. Depending on the number of stops of the chromosome we want to apply the mutation to, a number of mutations are randomly selected between 1 mutation and $\frac{N\_stops}{2}$. The algorithm then randomly selects which of the stops of the chromosome suffers a mutation.

4.4.7 Fixing mechanism

During the creation of the new generation, as a result of crossover and mutation, these mechanisms only focused on checking the validity of these chromosomes, but some of these chromosomes might contain redundant segments. For example, we can have a chromosome that has 5 stops but stop 5 is reachable from stop 2 making stops 3 and 4 unnecessary. In order to verify this, every new candidate for the new generation is tested for unnecessary stops before being added to the new generation.

To achieve this, an algorithm was developed that checked for every stop i where $i \neq$ final stop if existed any stop j where $j \geq i+1$ that can be reached from i and maintains the chromosome viable.

This fixing mechanism helps to accelerate the algorithm by removing viable but low-fitness solutions from being added to the new generations.

4.5 Electric Vehicle Route Planning using RL

In order to solve the MDP proposed in Section 4.1.1 we use a RL based approach to find the optimal strategy. This approach is based on the QL algorithm [26] that uses the Bellman equation to recursively update the action-value function.

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'}(Q(s',a')) - Q(s,a)) \tag{4.6}$$

The classical QL algorithm action-value function is based on a look-up table which in our case would require a multi-dimension table, in order to make this approach more suitable for our problem, we instead use a deep neural network to approximate the value-action function.

4.5.1 OpenAI Gym Environment

In order to implement the DQL algorithm in our problem, we developed a custom gym environment using the openAI gym RL API.

To create a custom gym environment we had to define:

- The action space;
- The observation space;
- the reset and step function;
- the reward function.

Action Space

In our problem, we define an action as choosing which station to travel to from the current state, therefore, our action space is comprised of the total possible stations the agent can choose from.

Observation Space

The observation space is comprised of all the variables important in our environment. In our implementation, we used a Python dictionary to define our observation. This dictionary was comprised of four keys:

- ’agent’ - containing the agent’s current station index, latitude, longitude, and battery state;
- ’path’ - listing all the already visited stops;
- ’start’ - containing the start geographic coordinates.
- ’goal’ - containing the goal geographic coordinates;
- ’stops’ - consisting of an array containing all station indexes, geographic coordinates, charging power, and charging cost.

Reset and Step Function

For the correct function of the gym environment, we need to develop a reset function that assigns the initial state values for each variable of the system. And a step function that for the given action chosen, computes its reward value updates the observation, and checks for termination of the episode

4.5.2 Reward Funtion

The reward function determines the behavior our agent takes. In order to define this behavior the reward function must return positive rewards when the agent takes steps that reflect the intended behavior and return negative rewards when the agent takes unintended actions.

For our problem, the intended behavior we want our agent to learn is:

- To choose the minimum number of reachable stations necessary to reach the goal
- To minimize the route duration and associated cost.

In order to achieve the first point, our reward function cannot simply give a positive reward when the agent takes a valid action. These rewards would incentivize the agent to choose the maximum number of steps allowed in order to maximize the reward obtained from choosing a reachable station. By giving the agent a positive reward for choosing a reachable station, we might be training the agent to choose reachable stations, but we are not training the agent to find the minimum number of stations to the goal. Therefore instead of giving positive rewards when the agent chooses a valid action, the functions return a less negative reward. What this means is, that when the agent takes a valid action, the reward function returns a negative reward, but the absolute value of this negative reward, is not as penalizing as the negative rewards obtained from invalid actions. This still incentivizes the agent to choose reachable stations, as they will return a less negative reward, but it also incentivizes it to choose the minimum number of stations necessary to reach the goal, as adding unnecessary stops only returns more negative rewards.

The reward received for a given valid action, must then be able to do two things: Inform the agent about the quality of the action by itself, but also inform the agent about the quality of the action in the context of the previous path. In order to achieve this, our valid action reward consists of two components.

An action component, in which the value represents the quality of the action by itself. This component consists of the combination of the time component and cost component of the route, between the current agent state and the agent's state after the chosen action. This combination is very similar to the ones described in Section 4.1 and 4.4.3, and is defined in Equation 4.7.

$$Action\_component = c_{charging} + (t_{driving} + t_{charging})(1 - \gamma) \tag{4.7}$$

The other component is a path component that consists of the sum of past action rewards up to the current agent state.

$$Path\_component = \sum reward \tag{4.8}$$

Considering that our algorithm's objective is to minimize the route duration and cost, like in Section

4.4.3, by utilizing the combination of these components our algorithm would be looking for the most time and cost consuming route. Luckily for this algorithm, due to the fact that we need to use negative rewards, by rewarding the agent with the negative combination of these components, the algorithm will as intended minimize the route's duration and cost.

However, with this reward function, we still have a problem, the agent has no incentive to look for the goal. This is due to the fact that in order for the agent to maximize rewards, in some cases it is as viable to choose multiple actions with very small negative rewards until the agent achieves the maximum number of actions per episode, as it is to find the goal. Therefore in order to incentivize the agent to look for the goal, when the agent performs a valid action that archives the goal a supplementary positive reward is given to the agent.

$$Reward\ Function(action) = \begin{cases} reward = -(Path\_componment + Action\_component) & ,if \begin{array}{c} action = valid \\ goal = not\ reached \end{array} \\ reward = (-(Path\_componment + Action\_component)) + goal\ reward & ,if \begin{array}{c} action = valid \\ goal = reached \end{array} \\ reward = -Invalid\_action\_penalty & ,if\ action = not\ valid \end{cases} \tag{4.9}$$

With all this, our final reward function consists of the presented expression.

4.5.3 Deep Q Network Architecture

One of the most important parts of the DQL algorithm is its Q-value aproximator. For the estimation of our Q-value for our DQL, we needed to design our Deep Q Network (DQN), the neural network that will be our Q-value function approximator.

Neural Network

For the design of this DQN, we used a neural network consisting of 3 fully connected layers. The first layer has 1024 neurons with a ReLU activation function, the second layer has 256 neurons with a ReLU activation function, and in the final layer, the output layer, the number of neurons is equal to the number of possible actions of the agent and a linear activation function. The size of this neural network input is equal to the size of the environment observation, as for each step the agent feeds its environment observation to the neural network.

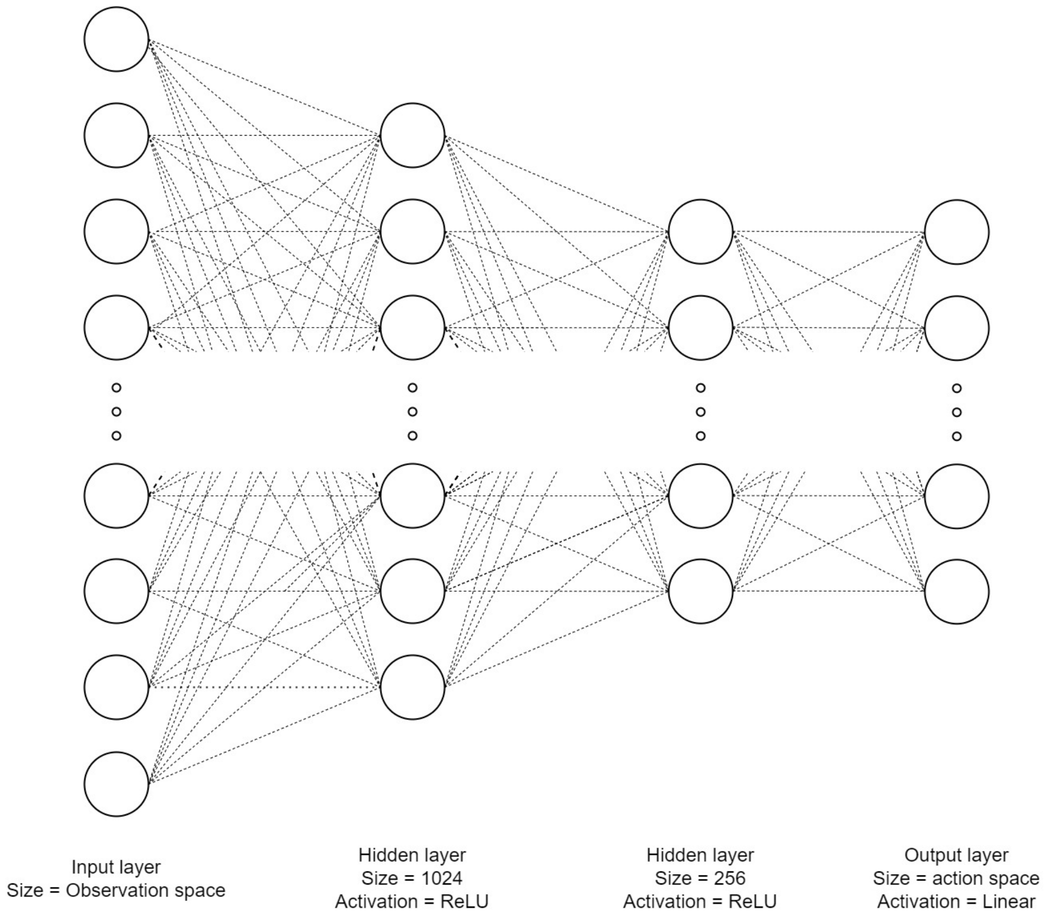


Figure 4.9: Neural network used for DQL.

Dueling Networks

Due to the complexity of our environment, we used a dueling network architecture to improve the convergence speed of our DQL implementation. The decomposition of the Q-value into two networks is useful to better evaluate the quality of different actions that lead to similar rewards [17]. This improves significantly the convergence speed of our algorithm, especially in a complex environment with a high-action space like ours. For our implementation, we used the neural network architecture previously explained. The re-composition of the Q-values for a given state is computed from the value of each network with the following expression:

$$Q(s, a) = V(s, a) + [A(s, a) - AVG(A(s))] \tag{4.10}$$

Where Q is the Q-value for action a when the agent is in the state s. $V(s, a)$ is the value from the state value network and $A(s, a)$ the value of the action advantage network.

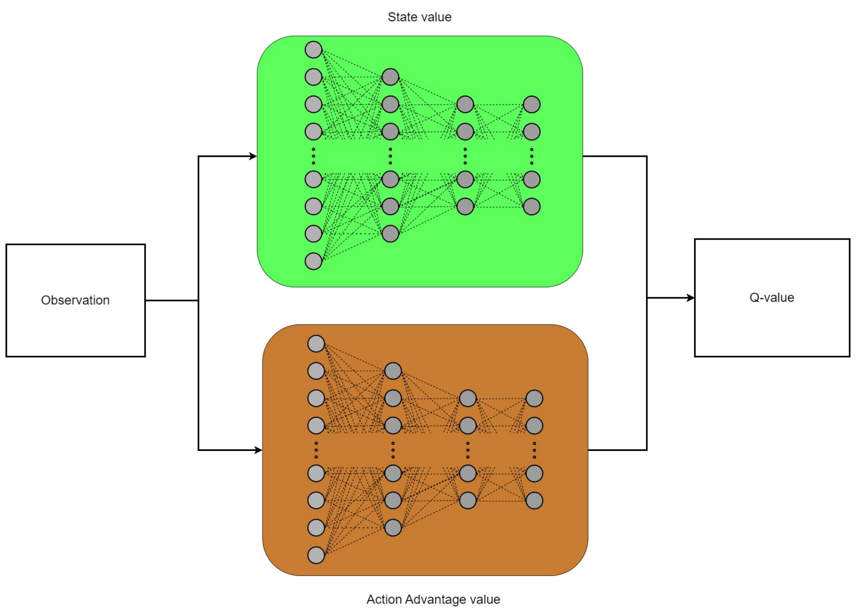


Figure 4.10: Dueling Networks architecture.

4.5.4 Epsilon Greedy Policy

In order for our agent to explore its environment, we need to give them an exploration strategy. In the beginning, our agent doesn't know anything about its environment, and its action-value function approximator is unable to give a good prediction for correct actions. Therefore, we need our agent to explore its environment in order to improve its action-value function. In order to explore the environment, one solution is for the agent to choose actions randomly from the action space. Although this will most certainly lead to bad decisions, it will allow our agent to learn from these bad decisions. This method works well in the beginning when the agent knows nothing about the environment, but after a while, this strategy becomes ineffective. This is because the agent will be able to explore the first steps of the environment, but to explore the environment after the initial steps, the agent needs to be able to first advance in the environment, and for this to happen by random chance would take a lot of time. Therefore, a method is needed that can have the benefits of random exploration, with the benefits of exploiting the known environment. To achieve this we decided to use the Epsilon greedy policy. This policy gives a probability for the decision to either be random or to be the action with the highest value in our action-value function approximator. This allows the agent to progress in the environment with the decisions from the action-value function, and to still be able to explore the environment with random decisions. Just by setting a fixed epsilon, we can improve the exploitation of the environment, but we know that it is more advantageous for the agent to focus on exploring the environment in the beginning, and to focus on exploiting it after already having explored parts of the environment. To achieve this behavior, we used a variable epsilon. The variable epsilon begins with a high probability of choosing a random action in order to explore our environment, and with each step, this probability would decrease, which will increase the probability of choosing an action based on the action-value approximator. By choosing more actions based on the action-value approximator, the agent is able to advance in the environment to explore actions closer to the goal state.

5 Results

Contents

In this chapter, we present and analyze the results obtained in the experiments performed during this work. We start by choosing our optimization parameters and obtaining our baseline results to compare them to the other results obtained from the different algorithms.

5.1 Experimental Setup

In order to validate the behavior of our algorithms, an experimental setup needs to be designed. The experimental setup intends to test the performance of the algorithm in a close to real-life utilization of the algorithms. In order to achieve this, some parameters needed to be defined, such as the parameters for the EV model, the parameters for the formulation, and the routes where the algorithms were going to be tested.

5.1.1 Setup Parameters

EV formulation

As explained in section 4.2.3 to achieve realistic results with our algorithms we need to have a good EV model. To achieve this, for our battery capacity we based our model on a study [27] which found the average EV battery capacity to be 40 kWh. For our energy consumption rate, we based our model on the average consumption rate of EV found in [28], which found an average energy consumption rate of 0.2kW per Km, which gives a 200 km driving range to our EV. With this model of the EV and with the information provided by the charging station API and the Routing API, we are able to compute the energy consumed by each route and the time taken to recharge at each station.

Cost function Gamma

In order to solve the problem formulated in section 4.1, we need to select a gamma value to combine the two costs. In order to obtain a good optimization of both costs, the gamma value must be chosen considering its effect on each cost. From the formulation we can see that for a gamma value of 0, we would obtain an optimization of only the trip cost, and for a gamma value of 1, we would obtain an optimization of only the trip duration, this can be observed in Table 5.2. Therefore, we can expect to obtain an increase in the trip cost and a reduction of trip duration the closer the gamma value comes to 1, and the exact opposite the closer the gamma value comes to 0.

Table 5.1: Dijkstra results for the three selected routes with single cost optimization

Dijkstra	Short Trip (265 possible CS)		Medium Trip (463 possible CS)		Long Trip (548 possible CS)	
	Time	Cost	Time	Cost	Time	Cost
Gamma = 0	2:58:20	3.23	4:58:48	11,29	8:50:15	39.33
Gamma = 1	3:24:57	1,82	6:16:26	5.27	10:30:48	11,82

In figures 5.1 and 5.2 we can see the effect of different gamma values on the trip cost and time duration.

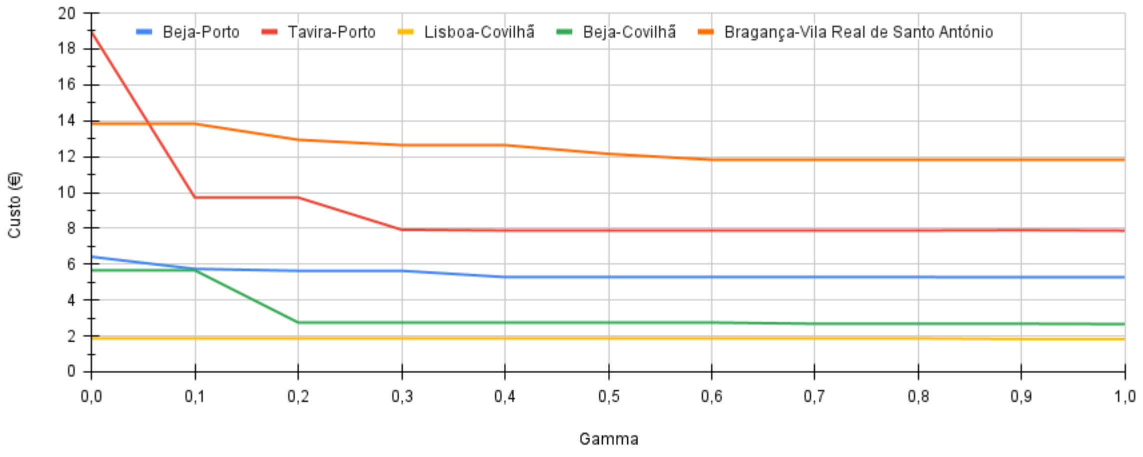


Figure 5.1: Different gamma effect on the trip cost for different routes.

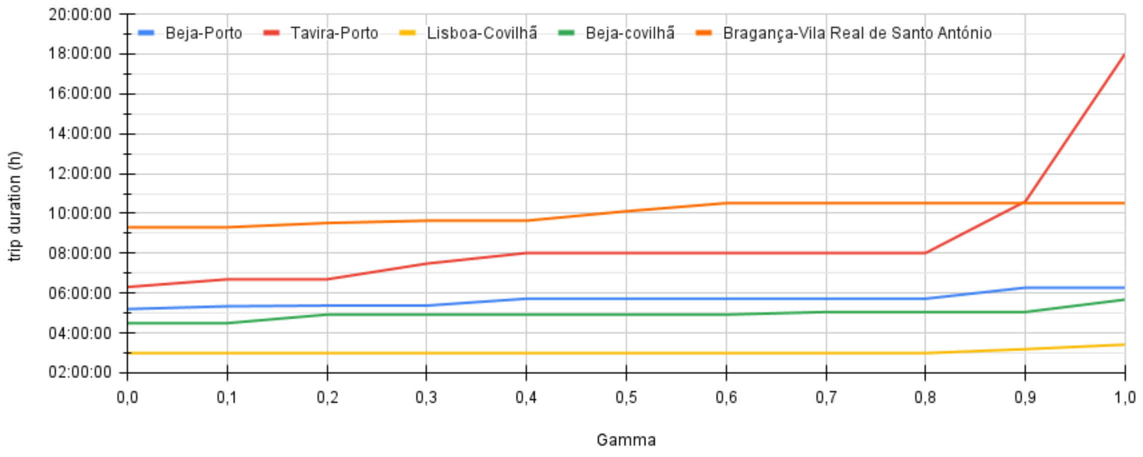
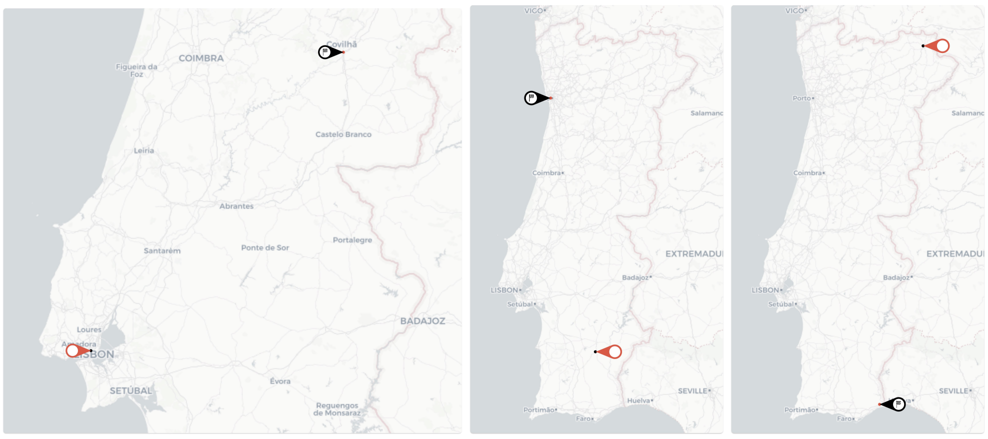


Figure 5.2: Different gamma effect on the trip duration for different routes.

In Figure 5.1 we can observe that for values of gamma superior to 0.5, the trip cost converges with the optimal trip cost and we obtain trip cost values very close to the optimal trip cost. In Figure 5.2, we can observe that for values of gamma close to 1, the trip duration starts to significantly increase and the lower the gamma value, the closer the trip duration comes to the optimal trip duration value. From this observation, we decided to use a gamma value of 0.5 as this allows us to achieve a good balance between time optimization and cost optimization

5.1.2 Routes

In order to compare the different algorithms, our experimental setup consisted of three routes. A short route, from the city of Lisboa to the city of Covilhã, this trip's fastest road route has a distance of 277km, contains 265 possible charging stations and requires at least 1 recharging operation to be completed. A medium route, from the city of Beja to the city of Porto, this trip's fastest road route has a distance of 448km, contains 463 possible charging stations, and requires at least 2 charging operations to be completed. And finally, a long route, from the city of Bragança to the city of Vila Real de Santo António (VRSA), this trip's fastest road route has a distance of 778km, this route has 548 possible charging stations and requires at least 5 recharging operations to be completed. These routes are represented below in Figure 5.3.



(a) **(b)** **(c)**

Figure 5.3: (a) Short trip (b) Medium trip (c) Long trip

5.2 Baseline

In order to evaluate the performance of our algorithms, a baseline needs to be measured in order to have a reference point to compare our algorithms. In this section, we analyze the data obtained from Dijkstra's algorithm to be used as our Baseline.

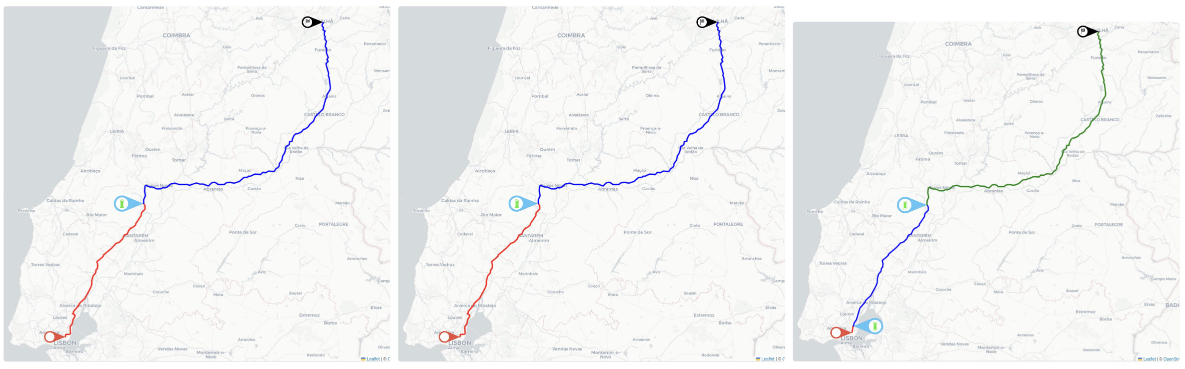
5.2.1 Baseline results

In Table 5.2, we can observe the results obtained with our Dijkstra's algorithm for the three defined routes.

Table 5.2: Baseline results for the 3 selected routes

Dijkstra	Short Trip (265 possible CS)		Medium Trip (463 possible CS)		Long Trip (463 possible CS)	
	Time	Cost	Time	Cost	Time	Cost
Gamma = 0	2:58:20	3.23	4:58:48	11,29	8:50:15	39.33
Gamma = 0.5	2:59:23	1,86	5:43:15	5,28	10:06:27	12,15
Gamma = 1	3:24:57	1,82	6:16:26	5.27	10:30:48	11,82

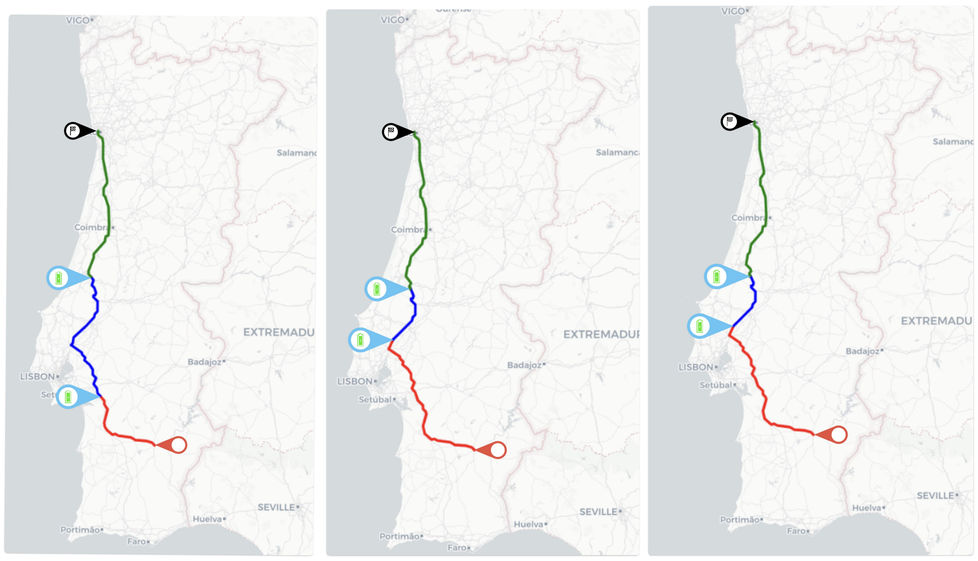
As expected, we can see that the results from a gamma value of 0,5 give us the most balanced results for each of the routes. In Figure 5.4, 5.5 and 5.6, we can see the routes associated to the results in Table 5.2.



(a) **(b)** **(c)**

Figure 5.4: (a) gamma = 0 (b) gamma = 0.5 (c) gamma = 1

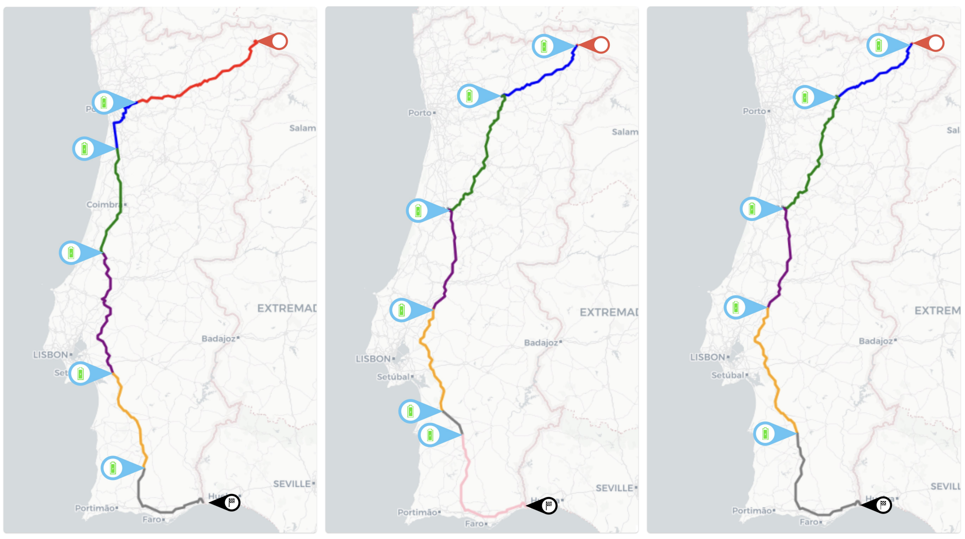
It is possible to see that in Figure 5.4, solutions (a) and (b) make almost the exact same route but route (b) ends up saving 1.37 euros in comparison to route (a). This is due to route (b) selecting a charging station very close to the one selected in route (a), but with a slightly lower charging rate, increasing the total route duration by 1 minute but with a much cheaper charging cost.



(a) **(b)** **(c)**

Figure 5.5: (a) gamma = 0 (b) gamma = 0.5 (c) gamma = 1

In Figure 5.5, we see a similar result where route (b) and (C) are very similar but where route (b) by selecting charging stations with a slightly faster charging rate than route (c) can improve the duration of the trip by 24:21 minutes from the route (c) by only increasing the trips cost by 0,33 euros. These results show the importance of utilizing algorithms for EVRP where choosing charging stations with a few kilometers of distance can make a big difference in trip duration and cost.



(a) **(b)** **(c)**

Figure 5.6: (a) gamma = 0 (b) gamma = 0.5 (c) gamma = 1

In Figure 5.6 we can see that for longer routes the charging operations become very complex which once again demonstrates the importance of these algorithms.

The results in Table 5.2 prove that Dijkstra's algorithm performed as expected, but when looking at the algorithm runtime we can find the expected downside of this algorithm. In Figure 5.7, we can observe the real-life runtime of the algorithm for the different numbers of charging stations of the three routes.

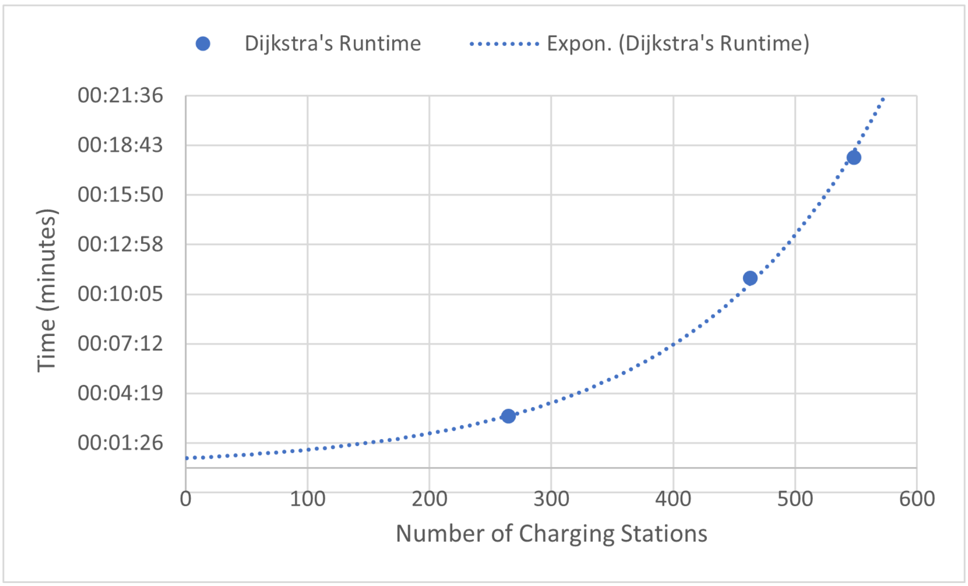


Figure 5.7: Dijkstra's algorithm runtime for different number of charging Stations.

From the analysis of Figure 5.7, we can clearly observe an exponential growth in the algorithm runtime with the increase of Charging stations. This exponential growth is the main downside of Dijkstra's algorithm and the main problem we want to fix with our implementation of the other algorithms.

5.3 Genetic Algorithm

In this section, we are going to analyze the results obtained from our implementation of the genetic algorithm, on the three defined routes. with these results, we will then compare them to the baseline results obtained in the previous section.

5.3.1 Hyperparameters Tunning

In order to have the best performance of our genetic algorithm some hyperparameters needed to be decided. One of these parameters is the stopping criteria. Besides the max number of generations, our genetic algorithm has another stopping criteria, that is if the best solution of a population is the same for more than a defined number of generations our algorithm stops. This stop criteria is to prevent the algorithm from wasting computational power to continue to find a new solution when the algorithm already has converged in a good solution.

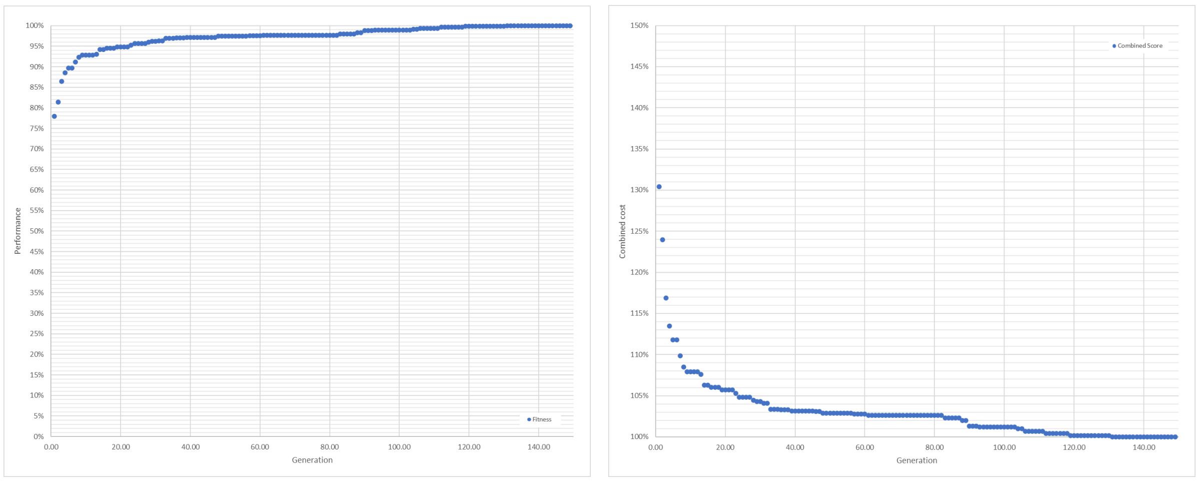


Figure 5.8: Genetic algorithms performance along generations.

As we can see in Figure 5.8, the performance of the GA improves along the generation, but after the 40th generation, the increase in performance per generation starts to diminish. Meanwhile, as we can see in Figure 5.9, the runtime increases linearly along the generations, which means that for the later generations, the small increase in performances requires a big increase in runtime. Therefore, in order to get the best performance per runtime possible, we need to find a good number of generations for our stopping criteria.

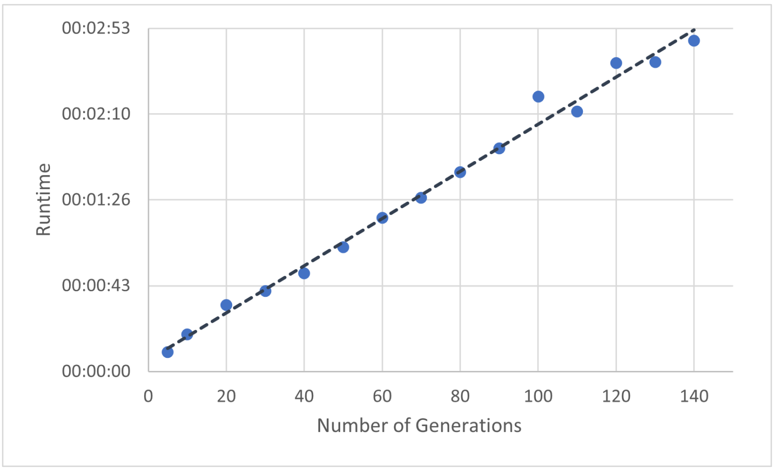


Figure 5.9: Genetic algorithm runtime along generations performing the medium trip.

In Figure 5.10, we can see that by increasing the number of generations of our stopping criteria, the performance of our algorithm increases, but beyond 50 generations the increase in performance is mostly negligible. Meanwhile, due to the mostly linear increase of the runtime, beyond 50 generations we are just increasing the runtime to obtain mostly the same performance. Therefore we decided on 50 generations with the same fitness as our stopping criterion.

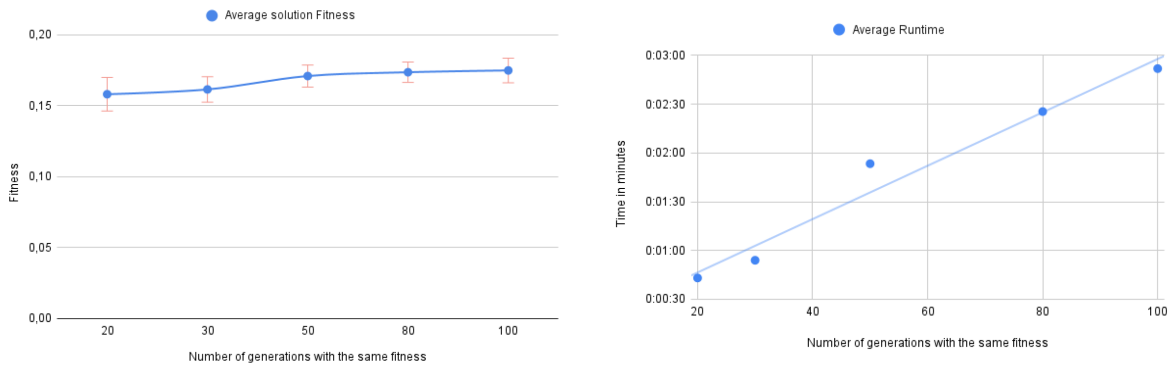


Figure 5.10: Algorithm's performance with different stop criteria values.

Another parameter that needs to be selected is the size of the population, which not only affects the performance and convergence speed of the algorithm but also affects its runtime. By increasing the population size of each generation, we introduce more chromosomes per generation, which increases the genetic diversity of our population. This leads to an increase in the speed of convergence per generation and also helps to prevent the algorithm from converging into a local optimum, which improves the overall performance of the algorithm. However, the added computational power needed in each generation to compute more new solutions increases the overall runtime of the algorithm.

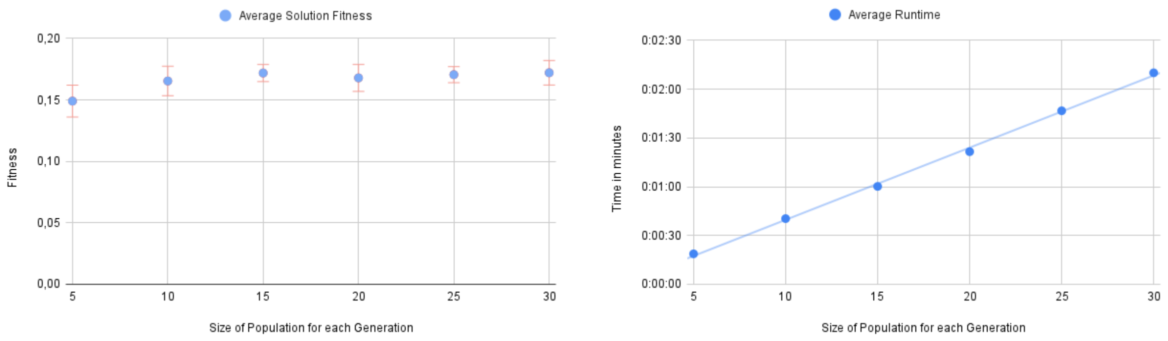


Figure 5.11: Algorithm´s performance with different sizes of population.

In Figure 5.11, we can see that with the increase of the population size, the algorithm's performance increases but stabilizes beyond a population size of 15 chromosomes. Meanwhile, we can also observe that beyond a population of 15, the increased runtime does not increase the algorithm's performance. From this observation, we concluded that a population of 15 chromosomes per generation would give us the best performance-to-runtime ratio.

5.3.2 Genetic Algorithms Results

To compare the performance of the GA to our baseline, we tested its performance using the parameters in Table 5.3.

Table 5.3: GA experimental results parameters.

Parameters	**values**
max number of generations	150
Population size	15
Generations with constant fitness limit	50
Gamma(γ)	0,5
Crossover rate	0,7
Mutation rate	0,2

Table 5.4 presents the results obtained performing the three routes described in section 5.1.2.

Table 5.4: Results obtained from Genetic Algorithm.

	Short Trip				Medium Trip				Long Trip			
	Fitness	Time	Cost	Runtime	Fitness	Time	Cost	Runtime	Fitness	Time	Cost	Runtime
Dijkstra	0,4119	2:59:23	1,86	0:03:28	0,181	5:43:15	5,28	0:11:00	0,089	10:06:27	12,15	0:18:01
Genetic Agorithm (std dev)	0,4100 (0,006)	3:01:01 (0:04:39)	1,86 (0,0007)	0:00:43 (0:00:14)	0,171 (0,008)	5:55:13 (0:20:08)	5,79 (0,423)	0:01:53 (0:00:35)	0,080 (0,003)	10:44:23 (0:31:08)	14,23 (0,89)	0:09:18 (0:02:54

From the results in Table 5.4 and figures 5.12,5.13, we can see that the GA can reliably achieve results very close to the optimal solution in a very reasonable runtime.

In Figure 5.12, we can observe the performance of the GA against the performance obtained from the Dijkstra's algorithm.

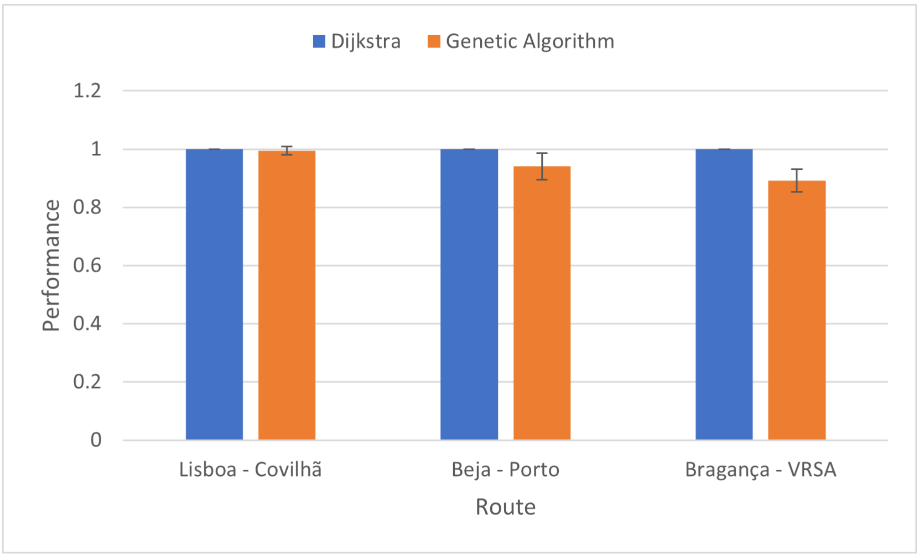


Figure 5.12: Comparison of the GA performance results against the baseline results.

From the data in Figure 5.12, we can conclude that the performance of the GA comes very close to the optimal solution achieving a solution with a performance of at least 90% of the optimal solution.

In Figure 5.13, we can observe the runtime of the GA associated with each route in comparison to the runtime of the Dijkstra's algorithm.

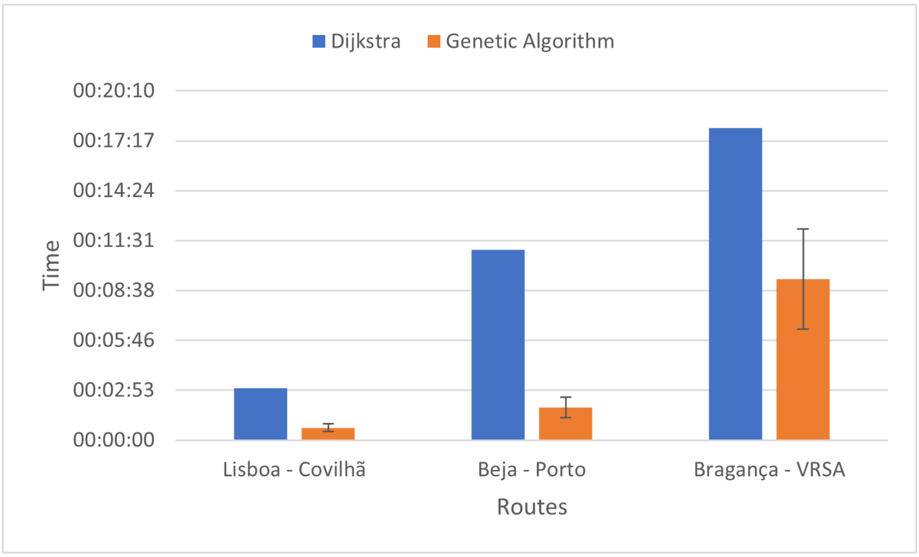


Figure 5.13: Comparison of the GA runtime results against the baseline results.

From the observation of Figure 5.12 and 5.13, we can conclude that the GA is able to achieve a very reliable and close to optimal solution in a fraction of the runtime of the Dijkstra's algorithm. These results show the strength and the potential the GA has to solve this problem, especially in very time-sensible applications.

5.4 Deep Q-Learning

In this section, the results obtained from the DQL algorithm are going to be analyzed. First, we will analyze the results from a simplified environment to validate the algorithm's behavior. Then we will analyze the performance of the algorithm in the experimental setup routes, which will allow us to evaluate its performance with the algorithms explored in the previous sections.

5.4.1 Reward Function

As explained in Section 4.5.2 for the correct behavior of our algorithm, the tuning of the reward function is imperative. From the formulation presented in Section 4.5.2, there are three reward values that need to be tuned. For a valid action from one station to a reachable station, the reward of this action consists of two values. The first value is the combination of the associated costs of the trip, for this cost we need to define the gamma value, which based on the findings of section 5.1.1, we chose to be $gamma = 0.5$. The second value consists of the cumulative rewards the agent received in previous steps. The absolute value of these two values is summed and rewarded to the agent but with a negative value. This is to encourage the agent to take the least amount of steps necessary as explained in section 4.5.2. For a valid action like the one described before, but that also takes the agent to a station that is able to reach the goal. For this action besides the reward from the action, the agent receives an additional goal reward for achieving the goal. This goal reward is positive and with a value equal to the magnitude of the cumulative reward, in order for the agent to end the episode with a cumulative reward value above 0, but with a value where the agent is able to distinguish the best paths from less optimal paths. The last reward value is for an invalid action where the agent chooses an unreachable station, for this action, the agent receives a penalty equal to $reward = -5000$ in order to penalize this behavior.

5.4.2 Simplfied Experimental Setup

Due to the complexity of the DQL algorithm, we started by testing its capabilities in a simplified version of our environment. This simplified environment consisted of a 100 by 100 units 2D plane with a distribution of 20 points, where each point represented a charging station. This setup is represented in Figure 5.14.

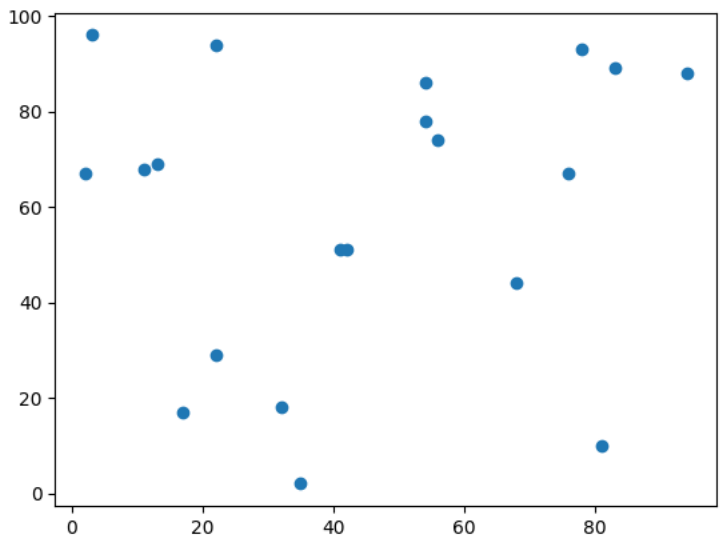


Figure 5.14: Simplified Environment

To represent the distance between charging stations, we utilized the Euclidean distance between the points. This simplified setup, due to the much less computation complexity to calculate its parameters, allowed us to train an agent in a much faster runtime, allowing us to verify implementation details much faster.

In Figure 5.15 we can see the data from the training of 200000 steps.

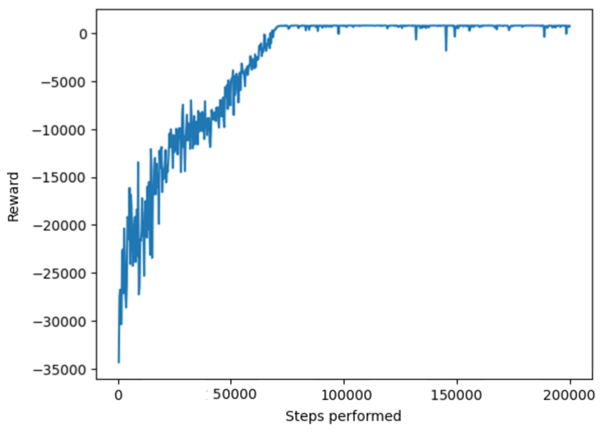


Figure 5.15: Training data from Simplified Environment

From the training data, we can see that the algorithm is successful at converging the agent rewards to around zero, which implies the algorithm during the training process is consistently reaching the goal. The spikes in the final steps of training are due to our epsilon-greedy policy not reaching an epsilon of zero, but instead maintaining an epsilon of 0.01 once it decays to this value.

In figures 5.16 and 5.17, we can see the obtained results with the training of 200000 steps.

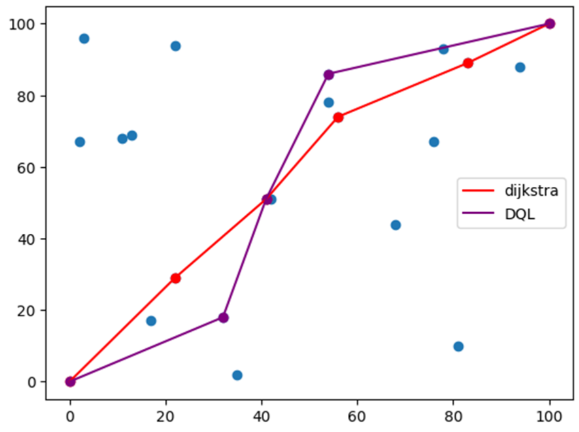


Figure 5.16: Example of a solution obtained in simplified environment

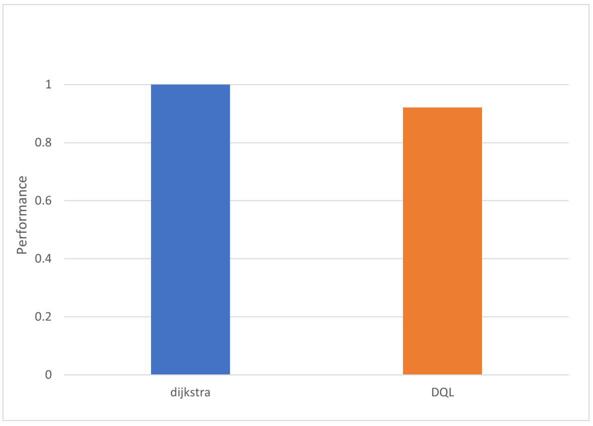


Figure 5.17: Performance obtained from simplified environment

As observed, the algorithm's solution achieves a reasonable performance in finding the route from the start position to the goal position, validating the algorithm's behavior. The performance achieved by the algorithm is not optimal as it achieves only about 90% of the performance of Dijkstra's algorithm, this shows that even for a simplified version of the algorithm, a longer training than the training of only 200000 steps would be required to achieve optimal results.

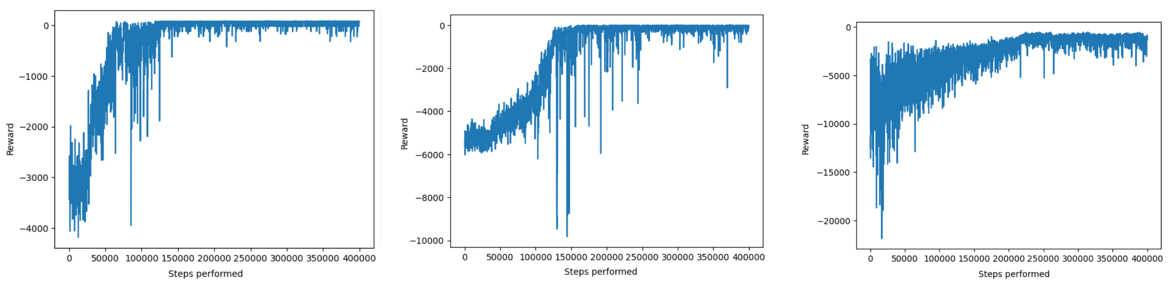
5.4.3 Deep Q-Learning Results

After validating the algorithm in the simplified setup, we tested its behavior in the three selected routes to compare its performance to the Dijkstra's algorithm and the Genetic algorithm's performance.

Table 5.5: DQL experimental results parameters.

Parameters	values
max number of steps per episode	10
starting epsilon(ϵ)	0,9
Min epsilon(ϵ) value	0,01
Epsilon(ϵ) decay rate	5e-5
Sequential memory size	5000
DQN optimizer	Adam
DQN learning rate	1e-3
Training size	400 000

In Figure 5.18, we can see the training data of the algorithm for the three selected routes, this training had a duration of 400000 steps which took around 10 hours of training using the parameters defined in Table 5.5. We can see that for the routes with a higher number of possible charging stations, the agent's rewards take longer to stabilize around zero and have a bigger interval between the highest reward and lowest reward. Due to the action space of the agent being directly correlated with the number of charging stations in the environment, this data demonstrates the difficulty the algorithm has in learning with the increase of its action space.



(a) (b) (c)

Figure 5.18: Training data for (a) Short Trip (b) Medium Trip (c) Long Trip

This increase in difficult learning can be observed in Table 5.6 and in Figure 5.19, where, while the solution of the algorithm in the short trip achieves a reasonable performance in comparison with the other algorithms, for the medium trip the performance of the algorithm is very poor, and for the long trip, the algorithm was unable to converge into a valid route during the 10-hour training.

Table 5.6: Results obtained from the three tested algorithms

	Short Trip				Medium Trip				Long Trip			
	Fitness	Time	Cost	Runtime	Fitness	Time	Cost	Runtime	Fitness	Time	Cost	Runtime
Dijkstra	0,4119	2:59:23	1,86	0:03:28	0,181	5:43:15	5,28	0:11:00	0,089	10:06:27	12,15	0:18:01
Genetic Agorithm (std dev)	0,4100 (0,006)	3:01:01 (0:04:39)	1,86 (0,0007)	0:00:43 (0:00:14)	0,171 (0,008)	5:55:13 (0:20:08)	5,79 (0,423)	0:01:53 (0:00:35)	0,080 (0,003)	10:44:23 (0:31:08)	14,23 (0,89)	0:09:18 (0:02:54
Deep Q-learning	0,3466	3:21:00	2,42	-	0,118	6:56:24	9,9	-	DNC	DNC	DNC	-

In Figure 5.19 we can better see how the performance of the DQL algorithm compares to the other algorithms.

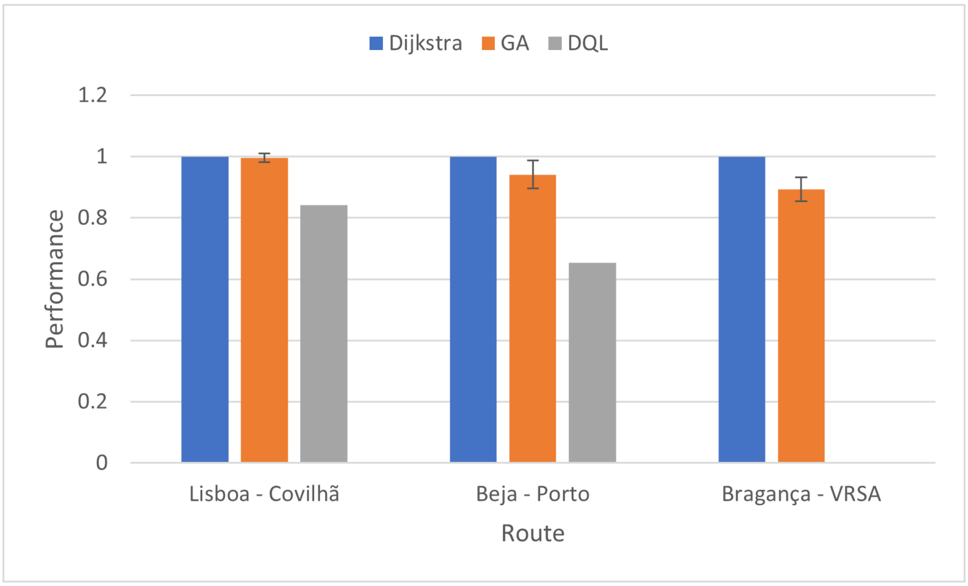


Figure 5.19: Comparison of the three algorithms performance.

By analyzing the figure, we can see that for the short trip the performance of the algorithm is about 15% worse than the other algorithms. For the medium trip, we can see that the algorithm can only achieve a solution that performs 35% worse than the optimal solution obtained with Dijkstra's algorithm. This drop in performance combined with the algorithm not being able to provide a valid solution for the long trip, shows that in order to obtain a quality solution the algorithm would need to be trained for a much longer period of time. However, the capability to provide solutions of reasonable quality for the smaller maps shows the potential of the algorithm. With these results, we can conclude that with proper training, the algorithm can achieve a reliable solution. This combined with the advantage of obtaining a solution instantly and being able to respond to real-time changes in the environment, gives the DQL algorithm very high potential to be used in more dynamic and time-sensible applications.

6

Conclusion and Future work

Contents

6.1 Conclusion

In this work, we have analyzed different approaches to find the optimal route of an EV in order to minimize the route duration and also the recharging cost.

We started by exploring the most classical approach, of using the Dijkstra's algorithm a search-based algorithm widely used to solve routing problems. From our implementation, we observed that while the algorithm is more than capable of achieving good results in solving the problem, the time complexity of the algorithm makes it unsuited to solve medium to large routes in a reasonable runtime.

To improve on the results obtained from the search-based approach we explore the use of two other methodologies.

We first explored the usage of Genetic Algorithms, a type of meta-heuristic approach. Meta-heuristic algorithms are utilized for their ability to find nearly optimal solutions within significantly reduced computation time compared to optimal algorithms like Dijkstra's algorithm. The results obtained from our implementation of the Genetic algorithm showed a great improvement in comparison to the Dijkstra's algorithm results, obtaining near to optimal solutions in a fraction of the runtime. These findings show the potential of these algorithms for time-sensitive applications.

Finally, we explored the application of the Deep Q-Learning algorithm, a prominent reinforcement learning technique. Deep Q-Learning is particularly suited for complex environments, as it leverages deep learning to model intricate, nonlinear relationships. However, the results we found in our implementation, showed a very poor performance in comparison to the results obtained from the other algorithms. These results where mainly due to the very high action space of our environment which required much more extensive training in order for the DQN to be able to better approximate the Q-values for our environment. Nevertheless, our initial testing of the algorithm showed that after adequate training the algorithm is able to achieve reliable solutions in real time. This capability to obtain a solution instantly is incredibly powerful and shows the potential this algorithm can achieve with the appropriate training.

In conclusion, our study demonstrates the trade-offs and advantages of different route optimization approaches. The choice of method depends on the specific requirements of the task, with traditional algorithms like Dijkstra's offering simplicity but limited scalability, while meta-heuristic approaches like Genetic Algorithms provide efficiency gains. Deep Q-Learning, although promising, requires substantial training but offers the potential for real-time solutions in complex scenarios.

6.2 Future Work

In this section, we detail potential enhancements that future research and development efforts can introduce to address limitations in our work and augment its overall functionality.

6.2.1 Dinamic Station Status

One of the big problems of the real-life application of these algorithms is that the station's status changes dynamically. One station that might be available at the initial route estimation, might no longer be available. This can lead to unexpected waiting times, which can lead to unnecessary increases in the route duration. One solution for this problem would be, to be able to know the station's status in real-time, through a centralized system, that receives real-time data from the entire charging station network. With a system like this, we would then be able to update our route in real-time in order to adapt to the new network status. This real-time data, paired with the capabilities of the DQL algorithm of responding to these changes in real-time, make the use of this algorithm very promising, in creating a system capable of dealing with the real-life dynamic behavior of charging stations and minimizing unexpected waiting times.

Bibliography

[1] J. Sanguesa, V. Torres, P. Garrido, F. Martinez, and J. Marquez-Barja, “A review on electric vehicles: Technologies and challenges,” *Smart Cities*, vol. 4, pp. 372–404, 03 2021.

[2] S. R. Kancharla and G. Ramadurai, “Electric vehicle routing problem with non-linear charging and load-dependent discharging,” *Expert Systems with Applications*, vol. 160, p. 113714, Dec. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0957417420305388

[3] Y. Zhang, Q. Ma, X. Zhang, M. Gao, P. Yang, H. He, X. Hu, and B. Aliya, “Integrated Route and Charging Planning for Electric Vehicles Considering Nonlinear Charging Functions,” in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*. Nanjing: IEEE, May 2018, pp. 660–665. [Online]. Available: https://ieeexplore.ieee.org/document/8465298/

[4] *Global EV Outlook 2020: entering the decade of electric drive?* Paris: IEA Publications, 2020, oCLC: 1232229382.

[5] “Is This The Future? Waiting To Charge At Tesla Supercharger Station.” [Online]. Available: https://insideevs.com/news/580338/is-this-future-waiting-charge-tesla-supercharger-station/

[6] K. Connolly, “Soaring energy costs could threaten future of electric cars, experts warn,” *The Guardian*, Sep. 2022. [Online]. Available: https://www.theguardian.com/environment/2022/sep/12/soaring-energy-costs-could-threaten-future-of-electric-cars-experts-warn

[7] S. Schoenberg and F. Dressler, “Reducing Waiting Times at Charging Stations with Adaptive Electric Vehicle Route Planning,” *IEEE Transactions on Intelligent Vehicles*, pp. 1–1, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9672714/

[8] C. Liu, M. Zhou, J. Wu, C. Long, and Y. Wang, “Electric Vehicles En-Route Charging Navigation Systems: Joint Charging and Routing Optimization,” *IEEE Transactions on Control Systems Technology*, vol. 27, no. 2, pp. 906–914, Mar. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8126871/

[9] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959. [Online]. Available: http://link.springer.com/10.1007/BF01386390

[10] D. Kosmanos, L. A. Maglaras, M. Mavrovouniotis, S. Moschoyiannis, A. Argyriou, A. Maglaras, and H. Janicke, “Route Optimization of Electric Vehicles Based on Dynamic Wireless Charging,” *IEEE Access*, vol. 6, pp. 42 551–42 565, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8402042/

[11] R. Bellman, “On a routing problem,” *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958. [Online]. Available: https://www.ams.org/qam/1958-16-01/S0033-569X-1958-0102435-2/

[12] F. Morlock, B. Rolle, M. Bauer, and O. Sawodny, “Time Optimal Routing of Electric Vehicles Under Consideration of Available Charging Infrastructure and a Detailed Consumption Model,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5123–5135, Dec. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8886708/

[13] P. Hart, N. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: http://ieeexplore.ieee.org/document/4082128/

[14] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, May 1992. [Online]. Available: http://link.springer.com/10.1007/BF00992698

[15] K.-B. Lee, M. A. Ahmed, D.-K. Kang, and Y.-C. Kim, “Deep Reinforcement Learning Based Optimal Route and Charging Station Selection,” *Energies*, vol. 13, no. 23, p. 6255, Nov. 2020. [Online]. Available: https://www.mdpi.com/1996-1073/13/23/6255

[16] T. Qian, C. Shao, X. Wang, and M. Shahidehpour, “Deep Reinforcement Learning for EV Charging Navigation by Coordinating Smart Grid and Intelligent Transportation System,” *IEEE Transactions on Smart Grid*, vol. 11, no. 2, pp. 1714–1723, Mar. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8845652/

[17] Y. Zhang, M. Li, Y. Chen, Y.-Y. Chiang, and Y. Hua, “A Constraint-based Routing and Charging Methodology for Battery Electric Vehicles with Deep Reinforcement Learning,” *IEEE Transactions on Smart Grid*, pp. 1–1, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9924526/

[18] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,” 2015. [Online]. Available: https://arxiv.org/abs/1511.06581

[19] S. Shao, W. Guan, B. Ran, Z. He, and J. Bi, "Electric Vehicle Routing Problem with Charging Time and Variable Travel Time," *Mathematical Problems in Engineering*, vol. 2017, pp. 1–13, 2017. [Online]. Available: https://www.hindawi.com/journals/mpe/2017/5098183/

[20] C. Li, Y. Zhu, and K. Y. Lee, "Route Optimization of Electric Vehicles Based on Reinsertion Genetic Algorithm," *IEEE Transactions on Transportation Electrification*, vol. 9, no. 3, pp. 3753–3768, Sep. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10025751/

[21] J. Barco, A. Guerra, L. Muñoz, and N. Quijano, "Optimal Routing and Scheduling of Charge for Electric Vehicles: A Case Study," *Mathematical Problems in Engineering*, vol. 2017, pp. 1–16, 2017. [Online]. Available: https://www.hindawi.com/journals/mpe/2017/8509783/

[22] T. Zündorf, "Electric vehicle routing with realistic recharging models," *Unpublished Master's thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany*, 2014.

[23] "Openrouteservice." [Online]. Available: https://openrouteservice.org/

[24] "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org/

[25] "Open Charge Map - Electric Vehicle Charging Locations Near You." [Online]. Available: https://map.openchargemap.io/

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, second edition ed., ser. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[27] M. Etxandi-Santolaya, L. Canals Casals, T. Montes, and C. Corchero, "Are electric vehicle batteries being underused? a review of current practices and sources of circularity," *Journal of Environmental Management*, vol. 338, p. 117814, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301479723006023

[28] M. Etxandi-Santolaya, L. Canals Casals, and C. Corchero, "Estimation of electric vehicle battery capacity requirements based on synthetic cycles," *Transportation Research Part D: Transport and Environment*, vol. 114, p. 103545, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361920922003716