UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

A PROGNOSTIC DECISION-MAKING APPROACH UNDER UNCERTAINTY FOR AN ELECTRIC VEHICLE FLEET ROUTING PROBLEM

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

DIEGO GUSTAVO TRONCOSO KURTOVIC

PROFESOR GUÍA:
MARCOS ORCHARD CONCHA

MIEMBROS DE LA COMISIÓN:
JORGE SILVA SÁNCHEZ
DIEGO MUÑOZ CARPINTERO

Este trabajo ha sido parcialmente financiado por:
FONDECYT Chile Grant Nr. 1210031,
and the Advanced Center for Electrical and Electronic Engineering,
AC3E, Basal Project FB0008, ANID.

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO
POR: DIEGO GUSTAVO TRONCOSO KURTOVIC
FECHA: 2023
PROF. GUÍA: MARCOS EDUARDO ORCHARD CONCHA

UN ENFOQUE DE TOMA DE DECISIONES DE PRONÓSTICO BAJO INCERTIDUMBRE PARA UN PROBLEMA DE ENRUTAMIENTO DE FLOTA DE VEHÍCULOS ELÉCTRICOS

Los Vehículos Eléctricos (EVs) han ganado popularidad durante los últimos años dada su baja emisión de Gases de Efecto Invernadero, contribuyentes al Cambio Climático. Investigadores han trabajado en las limitaciones de los EVs, tales como capacidad y manejo de la batería e infraestructura urbana de carga. El Problema de Ruteo de Vehículo Eléctrico (EVRP) representa un desafío relevante dado que afecta la cadena de suministros y por su rol en la transición a la electromovilidad. Diferentes métodos han sido usados e incorporan elementos realistas, por ejemplo, modelos de consumo energéticos, condiciones estocásticas y dinámicas, estaciones de carga, y otras.

En esta tesis, diseñamos, implementamos y testeamos un algoritmo basado en Monte Carlo Tree Search para resolver el EVRP en línea. Es una metodología incipiente para resolver el EVRP y tiene potencial dada su estructura de Árbol, funcionamiento en tiempo real, evaluación de escenarios futuros y procedimiento de búsqueda. Nuestra metodología soluciona el problema en línea exitosamente y es capaz de adaptar la ruta dada nueva información. Futuros trabajos apuntan a añadir nuevos elementos al EVRP, estudiar el alcance de esta metodología en áreas donde la información del futuro es valiosa, explorar diferentes estrategias con flotas de EVs, y otros.

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO
POR: DIEGO GUSTAVO TRONCOSO KURTOVIC
FECHA: 2023
PROF. GUÍA: MARCOS EDUARDO ORCHARD CONCHA

A PROGNOSTIC DECISION-MAKING APPROACH UNDER UNCERTAINTY FOR AN ELECTRIC VEHICLE FLEET ROUTING PROBLEM

Electric Vehicles (EVs) have gained popularity over the past few years, given their potential to reduce the emission of Greenhouse Gases, which contribute to Climate Change. Researchers have put substantial effort into handling the limitations EVs have, for instance, battery capacity, battery management and charging infrastructure in urban areas. The Electric Vehicle Routing Problem (EVRP) represents a relevant challenge since it directly affects the supply chain of many businesses and could help transitioning to electromobility. Several methods have been used to solve the EVRP incorporating realistic elements, for example, energy consumption models, dynamic and stochastic conditions, charging stations, and others.

In this thesis, we designed, implemented and tested a Monte Carlo Tree Search-based algorithm to solve the EVRP on-line. This is an incipient methodology to solve the EVRP and has great potential given its Tree structure, real-time functioning, evaluation of future scenarios and search procedure. Our work shows how the proposed methodology successfully solves the problem online and can update the route given new information. Future efforts aim to add new elements for the EVRP, study the scope of this methodology in other areas where uncertain prognostic information is valuable, explore different strategies in the EVs fleet case, and others.

A mi familia.

Agradecimientos

Las palabras y símbolos en este documento no son más que la portada de un gran libro de lo que fue mi caminar estos casi dos años de tesis. Durante este trabajo he vivido un variopinto de emociones. Desde la amarga frustración y el cansancio hasta la genuina felicidad y satisfacción del logro y la superación. Cada tropiezo y desafío ha significado crecimiento y aprendizaje. Al finalizar este enriquecedor proceso, tengo más preguntas que al comenzar y más curiosidad por el conocimiento. Espero poder responder algunas de estas interrogantes de aquí en adelante.

En primer lugar agradezco a mi familia, cuyo amor, apoyo y cariño han sido invaluables e inconmensurables desde que tengo memoria. Mi gratitud no conoce límites. A todos ustedes los admiro profundamente, me enorgullecen y los amo.

A Vesna y Gustavo por sus enseñanzas, sacrificio y esfuerzo. A Iván, Nicolás y Svenka por todas las conversaciones, anéctodas, historias, consejos y risas. A mi Tata y a mi tío Oso por su ejemplo, resiliencia y templanza. Parte de ustedes está en mi.

Al profesor Marcos Orchard por su confianza y apoyo desde que era un plancomunero con muchas ganas y pocos conocimientos. Su guía y consejo han sido una gran ayuda durante todos los años que nos conocemos.

A la profesora Christ Devia, con quien he podido aprender de un nueva área desconocida e intrigante que me cautiva y me genera reflexiones constantemente.

A mi pareja Catalina, por su amor, comprensión, apoyo, compañía y por los momentos juntos, particularmente en este tiempo de estrés.

A mis amigos Alonso, Daniel, Kurt, Lucas y a mi amiga Francisca por las largas conversaciones y buenos momentos a lo largo de muchos años.

A todos ustedes, gracias.
Diego.

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Why Electric Vehicles?

Exhaust gasses from Conventional Vehicles (CV) are the primary source of air pollution, particularly in highly dense populated areas [1]. According to the International Energy Agency, in 2021, 24 % of total CO_2 emissions are caused by the subject of Transportation [2]. This gas constituted 80.1 % of Greenhouse Gas emissions in the United States in 2019 [3], being one of the most important causes of Climate Change [1].

To overcome this problem, a transition to Hybrid Electric Vehicles (HEV) and Electric Vehicles (EV) arises as a clear alternative in several countries [4], notwithstanding the social effects that the use of vehicle cause in cities and urban areas. Many countries have adopted strategies to stimulate the purchase of HEV or EV, or to restrict it for CVs [5, 6, 7, 8, 9]. Among those actions, it is possible to find tax incentives, subsidies, road priority, or access to restricted traffic zones. According to [10], the most relevant factor to promote EV adoption are charging stations' density, fuel price and road priority.

There are mainly three EV categories: Battery Electric Vehicles (BEV), Plug-in Hybrid Electric Vehicles (PHEV) and Fuel Cell Electric Vehicles (FCEV) [11, 12]. BEVs are propelled only by electric motors, and their onboard battery provides their power. Some of their benefits are the absence of tailpipe emissions, high efficiency (particularly when compared with CV) and a simple powertrain design. PHEVs use an electric motor and an internal combustion engine (ICE), and their specific use depends on the configuration (series, parallel, series-parallel and complex). PHEVs are usually understood as a transition technology because they allow short trips to be made in electric mode and longer ones to be made with alternative fuels. Finally, in FCEVs, electricity is generated from hydrogen. They do not produce pollutants since electricity is obtained from chemical reactions, not burning fuel. The most significant drawbacks are their restrictive cost and the duration of the fuel cell that generates the electricity. This work will focus on BEVs, addressing their progress and challenges. For simplicity, from now on, the term EV will refer to BEV.

The sales of EVs have sky-rocketed in the last few years. In the US there are 50 different light-duty EV models, and more than 130 are projected for 2023 ([13]). According to [14], by 2035, EV and ICE will be cost-competitive, even with low oil prices, due to lower costs in batteries' manufacturing (which represents approximately 25 % of the total EV price [15]). The adoption of EVs requires solving problems inherent to their use, for example, the location of charging stations and the different technologies they use, battery management, high cost, short driving range and long charging time [16, 5]. EVs management is also related to the

so-called "range anxiety", which is the phenomenon where drivers fear that the current range will not satisfy their needs, leading them always to charge the EV to have a minimum battery level [17]. This action is also related to concerns about protecting the battery's health, given the recommendations to maintain the battery's charge between certain boundaries and not at the maximum capacity.

1.1.1. Battery: Basic Concepts

There are several battery technologies used in EVs. Among them, it is possible to find Lead-acid, Nickel-Metal-Hydrite (NiMH), Nickel-Cadmium (NiCd) and Lithium-Ion (Li-Ion) [18].

1. Lead-acid: This battery is also used in ICE vehicles for ignition, starting and other electrical functions. Its main disadvantages are its weight and low range for EV application.

2. NiMH: This type of battery is used in Hybrid EVs (HEV) due to its longer life cycle and lighter weight, compared to Lead-acid. They also can hold more energy. Its disadvantages are a higher self-discharge rate and a reduction in its cycle life if it repeatedly experiences rapid discharges.

3. NiCd: This technology provides a longer life cycle than NiMH because it can tolerate deep discharges. Its main drawbacks are the existence of the "memory effect". This is the reduction in capacity if it is recharged without being completely discharged; and a low relative electrical capacity.

4. Li-Ion: Most common technology due to its advantages: high energy density, lighter package, low self-discharge rate and relatively good temperature performance (extreme environment affects its performance).

1.1.1.1. State-of-Charge

The State-of-Charge (SOC) is usually defined as a ratio between the energy available in the battery and the maximum nominal capacity.

$$SOC = \frac{Q_{available}}{Q_{nom}} \tag{1.1}$$

It is a critical measure since it allows us to know how much energy is left in our storage system and decide upon it. As it looks simple to understand and conceive, it is challenging to estimate since there is no method to measure it directly. Several factors influence its value, such as manufacturing imperfections, mechanical damage, temperature, consumption profile and aging [19].

Reaching both SOC limits produces extreme chemical reactions, which increase the polarization impedance, which is highly related to the battery degradation [20]. As a result, the routing task and battery management are fundamental for the proper adoption of EV technologies, as they provide guidelines and recommendations on when to charge the EV battery, the amount of charge, the impact associated with different charging technologies and the capacity of each charging station. The latter is critical in the case of EV fleet management.

1.1.1.2. Future trends

In recent years, researchers have put effort into developing new technologies, aiming at overcoming the drawbacks of Lithium-Ion batteries, such as environmental impact and cost, or improving their capabilities, such as energy density and safety.

Lithium metal batteries (LMB) are expected to be protagonists in the next generation of batteries, replacing Lithium Ion batteries (LIB). This asseveration is based on the high energy capacity of LMBs (higher than 500 $\frac{Wh}{kg}$). The theoretical energy densities of $Li\text{-}O_2$ and $Li\text{-}S$ are 3505 and 2600 $\frac{Wh}{kg}$. The main drawback of LMBs is the Li dendrite growth, which causes low cycle efficiency and causes safety issues. While researchers have tried different approaches to tackle this problem, such as matrix design and electrolyte modification, it is still an unresolved challenge.

Due to limited Lithium resources and lower costs, sodium-ion batteries (NIB) arise as a prospective candidate. The estimated energy density of NIBs is competitive with graphite/ $LiMn_2O_4$. Polyvalent ions, such as Magnesium-Ion batteries (MIB) and Aluminium-Ion batteries (AIB), may create systems with capacities several times higher than LIBs'. Notwithstanding the high capacity, good cycle and safety performances of MIBs, oxidation and polarization of MG^{+2} are their main limitation. Al could reach twice the $Mg's$ capacity as anode, as the atomic weight of Al is low and it has a high energy density [21].

1.2. Health Aware Decision Making (HADM)

In this subsection, we will briefly explain the concept of Health Aware Decision Making. For a more detailed discussion, please refer to [22].

The concepts of System Health Management (SHM) and Decision Making (DM) have historically been developed separately. The authors from the previous work state that the typical functions of SHM are monitoring, fault detection, diagnosis, mitigation and recovery, and those of DM are all the processes aimed at accomplishing operational objectives. When a prognostic stage is incorporated in the SHM module, it is often called Prognostics and Health Management (PHM) [23]. As a consequence, in many implementations, these processes are separated, as illustrated in the following diagram:

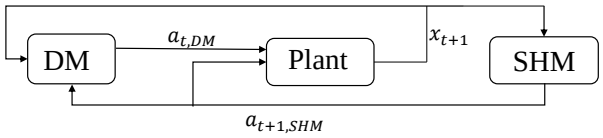


Figure 1.1: Diagram illustrating a standard SHM design. Adapted from [22].

It is important to note that, according to [22], SHM should be unified with DM for greater operational effectiveness and resilience, and they defined this unification. While some previous works have considered this spirit [24, 25, 26], there needed to be a structured and detailed discussion of this new conceptualization. In [27], the authors also encourage the research and trends regarding the relationship between prognostics and the decision-making processes.

To create their unifying approach, they rely on two fundamental principles:

1. State-based system modeling framework.

2. Utility function describing operational preferences for the system ($U(s)$).

In simple terms, the essential concepts for the unification are a system model and a reward function for states and actions in the future. Specifically, the critical property of the utility function is that it allows us to map multiple variables into a single number and translate a potentially complex DM formulation into a utility maximization problem.

In this regard, whereas prognostic algorithms cannot predict given the absence of knowledge about the future, the Utility function calculation process allows us to explore different scenarios by taking sequential actions and evaluating them depending on the state they lead. As a result, the uncertain future information is more appropriately addressed.

This proposal can be illustrated as follows:

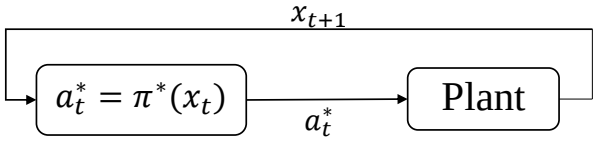


Figure 1.2: Diagram illustrating a HADM implementation. Adapted from [22].

To summarize, HADM proposes that DM and SHM can be carried out within the same process instead of two different systems exchanging information. In this regard, for example, the *fault detection* stage would be incorporated within the DM process as a *faulty state*, and appropriate actions could be available using the exploration of future states.

1.3. Electric Vehicle Routing Problem (EVRP)

Vehicle Routing Problem (VRP) is a problem inspired by the classic Travelling Salesperson Problem (TSP). The TSP consists of one salesperson who must travel to several cities and return to the starting one, restricted to visiting each city only once. The objective is to find the shortest path that satisfies these constraints.

The VRP is a similar problem, but a vehicle system and its functioning are included. In this case, the vehicle must travel to different nodes (cities can be a particular case) subject to its range capacity, fuel consumption, velocity restrictions, and others. These constraints have inspired different works and methods to approach them, varying on how these elements are incorporated and solved. One variant is the Capacitated Vehicle Routing Problem (CVRP), where vehicles have a maximum weight to carry. Another variant is where each node has a time window within which it must be visited. Thus, if the vehicle arrives outside this time window, there are two options depending on the formulation of this constrain: soft time windows, which allow the vehicle to arrive at the node outside its time window at a cost; and hard time windows, where reaching the node is only allowable if the arriving time is within the time window boundaries, this variation is the so-called Vehicle Routing Problem with Time Windows (VRPTW). Naturally, the variant combining these two is the so-called Capacitated Vehicle Routing Problem with Time Windows (CVRPTW). Other variants include Pickup and Delivery, Open or Closed Routes, Multiple depots (starting and ending places) and others.

According to [28], two main factors determine the difficulty and applicability of a specific VRP formulation. On the one hand, we must differentiate between Static and Dynamic

versions. On the other hand, we should differentiate between Deterministic and Stochastic variants; Static and Deterministic are the easiest variants, whereas Dynamic and Stochastic versions are the hardest. Static vs Dynamic refers to conditions that may change due to unknown dynamic factors (traffic, weather, unexpected events and others). Deterministic vs Stochastic conditions consider variables that cannot be exactly determined beforehand, such as energy consumption, travel time, client demands, and others. In a static and deterministic approach, a solution found would not change throughout policy execution. On the other hand, an initial solution could change in a dynamic and stochastic variation. Reasonably, more realistic variants are more applicable; therefore, they have caught more attention [29].

There are different ways to model the VRP, depending on the vehicle type, requirements and the constraints of every scenario. In this regard, some works have approached and solved this problem by trying to keep the formulation generic so that the solution can be applied to more scenarios. Others have solved particular scenarios such as Rescue Missions, Waste Collection, Periodic routes, Medical Tasks, UAV delivery and others [28].

The VRP can be extended to a multi-agent scenario where a fleet must be routed to fulfill the needs, restricted to constraints inherent to the type of vehicles used.

When dealing with a fleet of vehicles, new alternatives and scenarios arise, for example, the communication, coordination and even competition among the vehicles, the type of vehicle: CV, hybrid scheme or EV, and the ramification each category has, as previously mentioned, and others.

In [28], they claim that one of the emerging research directions in VRP in terms of the formulation is to include new challenging extensions, which reflect contemporary challenges, demands and opportunities, for instance: EVRP and last-mile delivery. They point out two other main directions, which are not directly considered in this work: Rich VRP (whose focus is on complex implementation-oriented attributes) and specific businesses or industry settings, for example, autonomous air/ground delivery and bike sharing.

1.3.1. Methods to solve the EVRP

The VRP is one of the most researched problems in the Operational Research (OR) field due to its practical relevance in the Last Mile stage of a supply chain, where the reduction of costs and maximize service quality is of utmost importance [30, 31, 32]. This problem is also studied because of its computational complexity. The VRP has been classified as an NP-Hard problem due to the latter characteristic. This tag means it cannot be solved in polynomial time. As follows, we explain the main methods used to solve the VRP [33].

1.3.1.1. Exact methods

Exact methods compute every possible outcome before they give a solution. As such, they are computationally expensive and escalate poorly with the size of the instance. These methods are classified in three categories: Lagrange Relaxation-based, Column Generation (CG) and Dynamic Programming. The first methods penalize violations of inequality constraints using Lagrange multipliers. CG is used to solve large Linear Programming problems. This method consists in solving a subproblem taking only a subset of the original variables, and adding variables to this new formulation only if they may improve the objective function [34].

1.3.1.2. Approximate methods

On the other hand, the need for faster high-quality solution have led researchers to develop heuristic and metaheuristic algorithm.

1.3.1.2.1. Heuristic methods

Heuristic algorithms can be classified into construction and local search heuristics. The former group can be divided into sequential and parallel methods. In sequential methods, feasible solutions are built by adding nodes sequentially to a first node. Parallel methods work in several solutions simultaneously.

Local search algorithms modify initial solutions improving them. These modifications can be intra-route or inter-route, depending on if the change is within the same route or between different routes. The goal in these procedures is to minimize the number of vehicles required or the traveled distance.

1.3.1.2.2. Metaheuristic methods

Unlike the heuristic methods, metaheuristc algorithms seek to obtain better solutions by exploring a larger subset of the solution space. These methods are more advanced than the previous ones. Population-based and Local Search are presented in the following paragraphs.

Population-based methods is an example of these algorithms. They obtain a new solution by combining existing ones or by making them cooperate. Within this group, Evolutionary Algorithms (EA) arise as an alternative [35]. Its functioning is inspired in the evolution process in living beings. Genetic Algorithms (GA) is the more classical representative of this class of methods. As can be inferred from the name, this proposal is inspired by a biological process. From an initial set of solutions, two samples are selected (*parents*), according to a metric, and are combined to obtain new solutions *children*. A *mutation* procedure takes place at this stage to increase the variability of the population. This procedure is repeated starting from the *children* until a criteria is met. Memetic Algorithms (MA) also belong to the EA class. It is similar to GA, with the difference that an improvement method is applied to every children. A third method from the population-based class is Scatter Search (SS). Unlike EA, it needs a smaller set of good and diversified. Subsequently, local improvements methods are applied to those solutions.

Another category in this large family of population-based metaheuristic methods are the swarm-based algorithms. Within this group Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) arise as the most common techniques. ACO is also inspired in nature. Ants leave a pheromone path where they have been, as such, in computational models, artificial ants also leave a path and the goodness of the final destination. With more iterations, more ants will follow the best paths. PSO initiates with a group of particles, whose individual movement is given by the best solution the particle has found and the best solution any has found.

Apart from population-based methods, another family is the Local Search methods. The main idea behind these algorithms is to improve the current solution by exploring a more promising neighbourhood in the solution space. The most common technique in this family is Tabu Search (TS). It consists on moving from an initial solution to its best neighbour even if the objective function deteriorates. In this way, the likelihood of avoiding a local optimum is increased. Succesive neighbours are examined and stored (and forbidden or declared *tabu* for a number of iterations). This is repeated t times and then the best *non-tabu* move is chosen [36].

Another option is the Guided Local Search (GLS), where to avoid local optimum, the solution evaluation is modified to penalize more heavily undesired features. In the case of VRP, those features are the edges [37]. Simulated Annealing is also part of the Local Search (LS) methods. It is inspired in how atoms behave with temperature changes and based on statistical mechanics. With higher temperatures, more movements occur, which translates in local changes to an initial solutions, even if it worsens it. Changes take place while temperature is decreasing until no new changes are allowed [38, 39].

The Large Neighbourhood Search (LNS) explores a wide search space. The initial solution in this algorithm is destroyed and repaired according to *destroy* and *reinsert* operators. A variant of this method is the Adaptive LNS, where the probability of choosing certain operators depends on how good the generated solution was [40]. Variable Neighbourhood Search (VNS) proposes a different scheme. Starting from any initial solution, a *shaking* step is performed by randomly selecting a solution from the neighbourhood. Then, an improvement algorithm is applied and the whole process is repeated until no incumbent solutions are found. After that, it moves to a new neighbourhood and the process is repeated until a criteria is met [41, 42].

In addition, Greedy Randomized Adaptive Search Procedure (GRASP) starts with an initial solution x, obtained through a greedy randomized process and find a local optimum with local search starting from x. This process is carried out iteratively until a stopping criteria is met and best solution found is chosen [43]. Iterated Local Search (ILS) is an alternative which starts from an initial solution and uses a LS procedure to improve it, obtaining s^*. Afterwards this solution is *perturbed* and improved similarly to the initial one and gets $s^{*'}$. Finally, an acceptance method chooses between the s^* and $s^{*'}$, to obtain the next starting point. This is repeated until a criteria is met [44].

1.3.1.3. New methods

The authors from [28] discuss how, despite the success and efficacy of the previous methods, several new ideas and approaches from other optimization domains can contribute to the VRP and be an alternative to existing techniques, both addressing new, specific variants or offering innovative schemes for known VRPs. In their opinion, these Computational Intelligence (CI) methods have potential for future development and more frequent usage in the VRP. These emerging methods include game-theoretic and bi-level optimization methods, hyperheuristic approaches, cognitively motivated techniques and Monte Carlo simulations.

Based on this, the Upper Confidence Bound for Trees (UCT) arise as an attractive solution in the mentioned work. UCT is an extension of the Monte Carlo Tree Search (MCTS) algorithm (both will be discussed further in the following Chapter). UCT, unlike MCTS, which make use of uniformly distributed simulations, optimally balances the exploration and exploitation of different solutions, which is particularly useful in problems with large action spaces [45]. One of the main advantages of MCTS is its knowledge-free characteristic, which means that only the rules of a problem or the capacity for distinguishing one result from another are needed, although the use of heuristics could improve the performance. Furthermore, its tree structure and search policies make its decision-making process easier to understand.

Only one work used UCT for a Capacited VRP with Traffic Jams [46] (Traffic Jams increase the cost of traversing the specified edge) according to [28], up to that date, and very few others have used UCT in other variants [47]. In regard to the incorporation of a Reinforcement Learning (RL) framework to solve the problem, they mention two works: one is a static version for VRP and the other for a version of the TSP [48, 49]. In addition, in

[50], several more recent efforts using RL are reviewed, however, none of them is applied to the EVRP. Moreover, only one of those implemented UCT for a warehouse routing case.

Our work will focus on the EVRP, which means that the fleet is made up exclusively of EVs [51]. This case brings up challenges mentioned in the previous section and constraints such as range limit, charging stations availability, non-linear charging functions and charging time, battery management, energy consumption models and other EV's features.

In a survey by Erdelic et al. [52], the authors state that several real-life elements have not been sufficiently studied in the VRP, for instance: dynamic traffic conditions, uncertainties in demand, travel time, time windows, service time, charging time and others.

Recently, in [53], the authors discuss Monte Carlo Tree Search implementations to the VRP. Most of them do not address the EVRP ([54, 55]). In [56], they present a survey about the Rich VRP mentioned before and point out the inclusion of uncertainty for more realistic implementations as a future trend. Additionally, in [57], the authors encourage the study of robust implementations, capable of dealing with all possible scenarios, and the need for progresses in the speed of the simulations.

1.4. Hypotheses

Based on the previous discussion, this thesis will test the following hypotheses:

1. Solving an adequately formulated Electric Vehicle Routing Problem fulfills all operational requirements efficiently.
2. A real-time algorithm based on Monte Carlo simulations can solve the Electric Vehicle Fleet Routing Problem.
3. It is possible to update the solution online, given new information, with the proposed algorithm.

1.5. Objectives

1.5.1. Main objective

The main objective of this thesis is to design, implement and test a real-time algorithm based on Monte Carlo simulations to solve a properly formulated Electric Vehicle Fleet Routing Problem. It will be able to update the solution given new measurements and information online.

1.5.2. Specific objectives

Specific objectives are listed as follows:

- To formulate an Electric Vehicle Fleet Routing Problem that fulfills all operational requirements with realistic features.
- To design, implement and test a real-time algorithm, based on Monte Carlo simulations, that solves the Electric Vehicle Fleet Routing Problem, incorporating Health Aware Decision Making guidelines.
- To study the performance of the proposed solution in an online case-study.

1.6. Thesis outline

The thesis is organized as follows: Chapter 2 presents the theoretical background on where we based our work; Chapter 3 explains the methodology applied to solve this problem; Chapter 4 presents a case study inspired by the operation of EV Fleet used for delivery porpoises; Chapter 5 presents results obtained; Chapter 6 offers a discussion of the results and what may be the advantages, restrictions, progresses, extensions and different challenges ahead and Chapter 7 concludes our thesis giving final remarks.

Chapter 2

Theoretical Background

We formulated the EVRP as a Markov Decision Process within a Reinforcement Learning framework and solved it by implementing a Monte Carlo Tree Search-based algorithm. This Chapter briefly explains these concepts.

2.1. Markov Decision Process (MDP)

Decision Theory provides a formal framework for Decision Making under Uncertainty (DMU), combining probability theory with utility theory [58]. One way to work with problems whose utility function is defined by a sequence of actions is by using Markov Decision Processes (MDP).

An MDP models fully observable sequential decision problems with four main elements [58]:

1. States: Represents the system's state at a specific time.
2. Actions: The set of possible actions given a state. If the state is terminal, there are no available actions.
3. Transition Model: Defines the probability of transitioning from state s to s' after executing action a. It is defined as $Pr(s,a,s')$.
4. Reward function: It gives a reward after executing action a in s to reach state s'.

The Markovian property tells us that the next state and reward depend only on the current state. An extension has been defined where full observability is not assumed. In this case, the formulation is called Partially Observable Markov Decision Process (POMDP) and the term *state* is often replaced by *observation*. MDPs have been used in very different areas, such as clinical tasks [59] and cyberattacks [60].

Both formulations aim at finding an optimal policy that maps a state to action through action-state pairs, receiving a reward depending on the state it reaches. Chosen actions influence not only immediate rewards but also subsequent ones. As a consequence, MDPs need to leverage between immediate and delayed rewards.

At each step, the agent (learner in the environment) receives an environment's representation (which includes itself) and chooses and acts upon it. After executing an action a, a new state and a reward are presented.

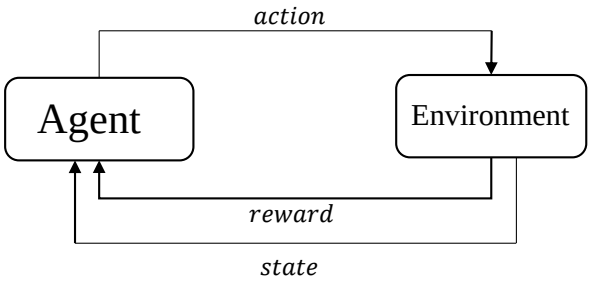


Figure 2.1: Basic diagram of the Reinforcement Learning concept.

Every time step can be something other than a real-time measure. For example, it may be positions on a map. Likewise, actions range from the charge applied to a battery or to deciding to take one road or another.

To end this Section, a quote from [61] summarizes very well the MDP:

> The MDP framework is a considerable abstraction of the problem of goal-directed learning from interaction. It proposes that whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment: one signal to represent the choices made by the agent (the actions), one signal to represent the basis on which the choices are made (the states), and one signal to define the agent's goal (the rewards). This framework may not be sufficient to represent all decision-learning problems usefully, but it has proved to be widely useful and applicable.

2.1.1. Non-Markovian Decision Process

There are two scenarios where the Markovian property is broken. The first is when observations are limited and partial. In that scenario, the current state must be estimated based on past information or a latent space. In this case, rewards and dynamics are not markovian. The second scenario is when only rewards are not markovian. As explained before, the former is a POMDP, while the latter is known as Non-Markovian Reward Decision Process (NMRDP). When both scenarios are included in a formulation, it is named Non-Markovian Decision Process (NMDP) [62].

Markovian rewards are suited for applications when the history is not relevant to the *Reward* function. Games are an excellent example of this. In chess, in simple modeling, the final state has three possible outcomes: win, draw or loss. It is irrelevant if it took many or few moves to reach that final state or the number of pieces a player lost. However, this is not true for most applications, where the history and past information are relevant to the current state.

An example is the case of an assistance robot, which has to ensure that a person takes medication after every meal. In this example, the condition of *person ate meal* influences the decision-making at the current state and the possible actions *give or not the medication*. Another example is routing problems, where the reward depends not only on reaching the depot but on having visited all clients and fulfilling other requirements. This is because the only information of returning to the depot is not enough to account for other relevant characteristics in the problem, such as visiting all clients, avoid crossing the battery's threshold or reaching all clients within their time-window. Other options where Non-Markovian rewards

are present are problems where a particular state must be met precisely once or every k steps. In conclusion, a Non-Markovian reward needs more than the current state to deliver a value.

An advantage of Markovian rewards formulations is the use of dynamic programming techniques to solve the problem. These techniques cannot be directly applied when the non-markovian property is not met. Different researchers have tried transforming NMRDPs into MDPs to use MDPs' solvers in the problem. In [63], the authors discuss that in an initial NMRDP $G_s = (S, A, R)$, the objective is to obtain an MDP $G_{ES} = (ES, A, R_{ES})$, where R_{ES} is the expanded Reward, which is now markovian and, thus, any off-the-shell planner can be applied. They found strong correspondence between MDP and NMRDP. An MDP $G_{ES} = (ES, A, R_{ES})$ is an expansion of an NMRDP $G_s = (S, A, R)$ if there are functions $\tau : ES \to S$ and $\sigma : S \to ES$ three conditions are met:

1. For all $s \in S$, $\tau(\sigma(s)) = s$.
2. For all s, $s' \in S$ and $es \in ES$ if $Pr(s,a,s') = p \geq 0$ and $\tau(es) = s$, there exists only one es', $\tau(es') = s'$, such that $Pr(es,a,es') = p$.
3. For any feasible trajectories $\langle s_0, \ldots, s_n \rangle$ in G_s and $\langle es_0, \ldots, es_n \rangle$ in G_{es}, where $\tau(es_i) = s_i$ and $\sigma(s_0) = es_0$, we have $R(\langle s_0, \ldots, s_n \rangle) = R_{ES}(es_n)$.

If an MDP G_{ES} can be produced from a NMRDP G_s, then optimal policies for G_s can be found by solving G_{ES}.

The same authors, in [64], used boolean temporal variables to save relevant information from the current trajectory. In [65], the authors use working memory slots to save past observations and take them into account in the decision-making process. Camacho et Al. proposed an approach to solve NMRDP using Linear Temporal Logic, a language for expressing temporal properties over a sequence of states [66], and a Deterministic Finite-State Automaton (DFA) [67].

2.2. Reinforcement Learning

There are three main areas in Machine Learning (ML): Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL). In SL, the model learns to map inputs to outputs from a dataset labeled by experts. After this map is created, the model is tested by predicting labels for unseen data, and a merit metric is used to measure its performance. The limitations of SL are the necessity of an expert to label data, which may be costly, and errors in this labeling (caused, for example, due to noisy sensors). Besides, the trained model could not perform better than the expert.

On the other hand, UL intends to find patterns or structures in unlabeled data, for example, to identify operating profiles from a system's data. UL has been mostly used for dimensionality reduction, feature extraction and clustering [68]. The third area, RL, learns to map states to actions to maximize a numeral signal called *Reward*. The model is not told what actions to execute; instead, it discovers which actions yield the most reward by trial-and-error. In the most challenging cases, actions affect immediate rewards and subsequent ones. These two characteristics distinguish RL from other areas of ML [61].

At first sight, RL may be included within the UL framework, as its functioning is not based on labels, however, besides the two mentioned characteristics, RL tries to maximize a *reward* and not find structures as the primary goal.

To obtain the highest reward, the agent must exploit those actions with high outcomes in the past, however, these actions may be hidden behind unexplored scenarios and to find them, the agent must choose "suboptimal" actions. On the other hand, exploring scenarios without exploiting good actions would make the agent lose the highest reward. In other words, the idea is that neither full exploitation nor exploration are the key to accomplishing the objective. This dilemma is the *Exploitation Exploration Dilemma*, which will be further discussed later.

As said before, a critical part of the RL paradigm is the reward the agents receive as it guides the behavior and decision selection. It is also one of the most challenging elements of an RL implementation since the designer would undoubtedly want to avoid bias in the search for the best action and instead let the agent act to find it. A simpler scenario is where the reward is sparse. For example, in a chess game, if the sequence of actions leads to a win, the reward is 1; if it leads to a loss, the penalty is -1; and 0 if there is no winner. The advantage of is kind of reward function is that it avoids a significant intervention by the designer in the DM process, decreasing the chance of inducing a bias. However, it is uninformative for the agent and could take significant computational resources to achieve the expected result. Theoretically, general-purpose RL algorithms should deal with sparse rewards; in practice, it may take prohibitively large computational resources. In [69], different approaches to improve sparse rewards designs are discussed. Some of them are based on intrinsic reward, a concept inspired by psychological studies, where the agent explores just for curiosity.

Another approach in *reward* function design is to incorporate elements in the function to provide clues for the agent as to which directions may be beneficial. These types of reward functions are called "dense reward functions". In the same chess problem, a reward would be given if the agent captures any of the opponent's pieces. The latter option creates challenges in balancing minor penalties with a specific global objective. For instance, how does the designer balance the reward in a case where if the agent loses a piece, it could allow it to win the game in the next few steps? Alternatively, in [70] where a *closeness* metric was added to hint a bicycle to move towards a target. As a result, the agent learned to drive in circles around the starting point. In brief, especially in more complex scenarios, the *reward* function plays a key role.

2.2.1. Temporal Difference Learning

Temporal Difference (TD) Learning is the core idea of Reinforcement Learning algorithms [71]. Its main idea is to update a state's value $V(s_t)$, which is defined as the total expected reward of being in state s at time t, according to the immediate reward r_{t+1} and the value of the following state at the next time $V(s_{t+1})$. As such, TD is a mechanism to implement the ideas of RL in a problem, allowing to *learn*. As can be inferred, states with higher values are preferred and will guide the agent's exploration and decision-making.

TD Learning combines Monte-Carlo (MC) and Dynamic Programming (DP) ideas. As the former, it can learn from raw experience without a known model; as the latter, it can gain knowledge from other states' estimates. Both methods use the experience to estimate the value of the states, however, the main difference between MC methods and TD is the time each of them waits until they update each state's value. In MC, a simulation must wait until it reaches a terminal state to obtain an outcome and update the values. A typical MC rule to update a specific value at time t is:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[G_t - V(s_t) \right], \tag{2.1}$$

where G_t is the outcome following time t and α is a constant step-size parameter.

On the other hand, TD methods only need to wait for one step $(t+1)$ to obtain a reward and update the state's value. The simplest update rule is as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]^{1}, \tag{2.2}$$

where R_{t+1} is the reward obtained at the transition to s_{t+1}. This TD method is called TD(0), which is a specific case of TD(λ) and *n-steps* methods. When n= 1, it is called *one-step* method.

TD is called a *bootstrapping* method since it updates the estimate based on an existing value, as seen in Eq.2.2.

It has been proven that both MC methods and T(0) converge to the optimal value under certain circumstances. A straightforward question in this regard: Which one is faster? This question remains open, however, in practice, TD methods usually converge faster.

To generalize for n steps, it is valuable to consider that in MC methods, the updates are in the direction of the complete return:

$$G_t \doteq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{T-t-1} R_T, \tag{2.3}$$

where T is the last time step of the sequence and G_t is defined as the *target*.

Both methods update their estimate state's value $V(s_t)$ based on experience. Whereas in MC methods, the *target* is the return, in *one-step* updates, the *target* is the reward at time t plus a discounted estimated value of the next state $(0 \leq \gamma \leq 1)$. In other words, in the latter case, the *target* is an estimate of the return.

$$G_{t:t+1} \doteq R_{t+1} + \gamma V(S_{t+1}) \tag{2.4}$$

The extension to *n-steps* is shown below:

$$G_{t:t+1} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}), \tag{2.5}$$

such that $n \geq 1$ and $0 \leq t \leq T - n$. If $t + n \geq T$, all the missing terms are set to zero and $G_{t:t+n} \doteq G_t$.

It is important to note that values R_{t+n} and V_{t+n-1} are available only at time $t+n$. The expression for the *n-steps* value learning algorithm is:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \quad 0 \leq t \leq T \tag{2.6}$$

To summarize, the role of Temporal Difference Learning in Reinforcement Learning is to be a method to estimate the state-value function $V(s_t)$ based on adjusting the *TD error*, which is the difference between current estimate for $V(s_t)$, the discounted value estimate of $V(s_{t+1})$ and the reward R_{t+1}. In this manner, the system can make correct decisions from the current state, based on expected future rewards.

2.3. Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search is a best-first search algorithm based on Monte-Carlo simulations to find optimal decisions in a given scenario, for example, planning problems. This class of

[1] A detailed derivation of this expression can be found at the Appendix A.1.

algorithms explore by building the Tree following the most promising nodes according to a certain metric.

MCTS has gained popularity since its success in achieving a new milestone when the AlphaGo team defeated the World Champion of the game Go using Deep Learning and MCTS [72, 73], and due to its applications in several areas [74, 75, 76, 77, 78]. It is an anytime algorithm, which means that it can always offer a solution, although more computational time usually leads to better solutions.

The MCTS has four main stages:

1. Selection: Select an action not previously selected or chooses the best children from the current node.

2. Expansion: Expand the tree if the action does not exist in a tree node.

3. Simulation: Choose actions according to a policy from the selected node until it reaches a terminal state.

4. Backpropagation: A reward is obtained and backpropagated to all previous nodes.

The construction of the Tree is illustrated in Fig.2.2 with a toy example. The Tree is initialized with one node, which corresponds to the first state. Let us assume that there are three available actions at this state: B, C and D. As there are available actions, in the first simulation the *Expansion* stage takes place. One of these actions is chosen and expands the Tree creating a new node that represents the state of the system after executing the chosen action in the previous node. Then, in the *Simulation* stage, one action is chosen (according to a policy) from the available set in Node B and is executed, reaching a new state (without creating a new node), where new available actions are presented. This procedure is carried out until a terminal stage is reached (this stage and the final state are represented by the gray line and the diamond-shaped figure). Finally, in the *Backpropagation* stage, depending on the terminal state, a reward R is obtained from Node B and propagated upwards (represented by the blue arrow). In the second simulation, the first node now has two available actions (C and D), as B was already chosen. An option is selected and the *Expansion* and *Simulation* stages are carried out as previously explained. In the third simulation we only have one available action and the process is carried out similarly. In the fourth simulation, however, the Node A has no available actions as all of them have been executed to create new nodes. When there are no more available actions in a node, the *Selection* stage takes place, instead of the *Expansion* one. The best child is selected (represented by the red arrow) and the process is repeated from this node. In this case, Node B is selected and the *Expansion* stage again takes place, if there are available actions. If the node represents a terminal state, the *Simulation* stage consists only in evaluating and state and backpropagating the reward R.

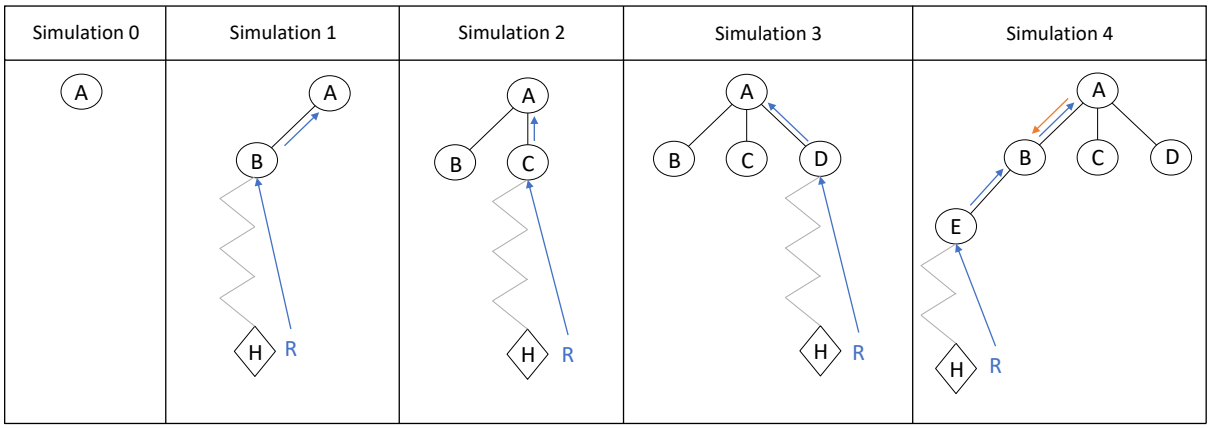


Figure 2.2: Construction of Tree.

Please, do not confuse the *Simulation* stage with the number of simulation as it is presented in Fig.2.2. The former refers to the stage where several actions are selected and executed until a terminal state is reached and evaluated, while the later indicates the iterations of the Tree.

These four stages repeat until one criterion is met, usually the number of simulations or maximum computational budget. Finally, the node that maximizes or accomplishes a metric is selected to choose the action for the actual state-transition execution in the system. In this step, the systems transits from the initial node to the best child node, and the search continues until it reaches a terminal node (a node that represents a terminal state) in the Tree.

There is no universal standard on how to choose the action to be executed after all the simulations. Different approaches have been used, for example, selecting the node with the highest average reward, the most visited node, using more complex expressions and others [79, 80]. In [81], the authors tested and discussed how different criterions affected the obtained solutions. They concluded that when each node had few visits, a max reward rule was inappropriate since lucky nodes benefit more. On the contrary, this option is appropriate when more visits have taken place. This idea is developed in [47]. The *max-visit* rule usually takes significant computational time to converge, particularly if the difference between the best and second best node is small [82]. In the mentioned study ([47]), the authors elaborate how in sparse reward applications, primarily games, the outcomes of the *Simulation* stage are usually uninformative: +1 if it is a win, 0 for a draw and -1 in case of a loss. In other applications, these kinds of rewards are unsuitable since many metrics are taken into account, and it is necessary to differentiate between wins and losses and better wins. With this in mind, they discuss how nodes leading to good rewards on average can be chosen instead of nodes with best rewards hidden by bad rewards (worse average reward), specifically in a routing problem. To solve this, they proposed the *two-phase* method, where to choose the action to execute, the P best nodes (in terms of average reward) are selected, and among them, the one with the best individual reward is chosen.

This on-line procedure is carried out until the system reaches a terminal state. A diagram illustrating this process is shown as follows:

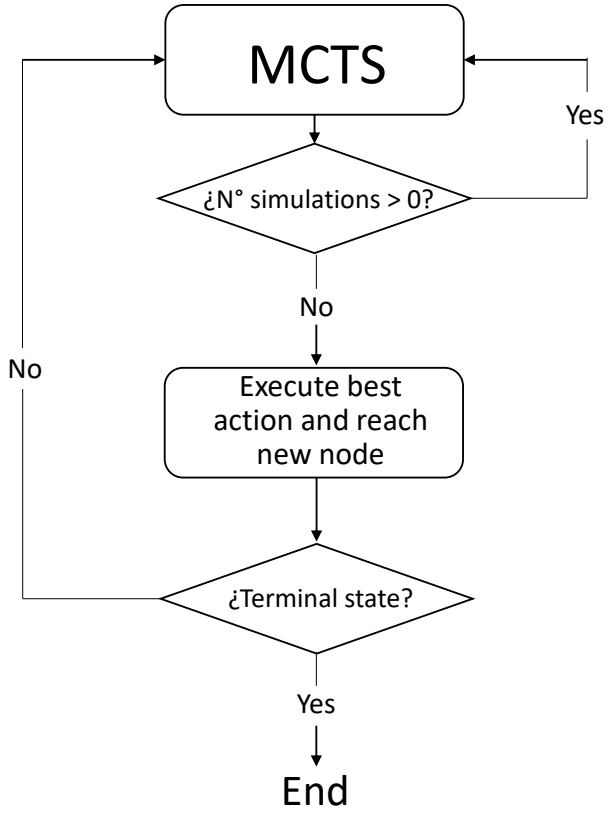


Figure 2.3: Basic diagram of the MCTS functioning. The number of simulations is reset every after every execution of the best action.

There are two options in the literature on how to continue with the Tree after the actual execution of the best action (Fig.2.3): Reset the Tree and only keep the current node and its parents, or keeping the Tree below the node. In deterministic scenarios, keeping the structure below would be beneficial since no new information would be obtained if the Tree is erased and built again.

2.3.1. How does MCTS search?

MCTS is a Bandit-based method [83]. These methods are a family of sequential decision problems where one must choose between K arms to pull to maximize a reward (for example, a slot machine with K arms). As we initially do not know the reward distribution of each arm, the Exploration-Exploitation Dilemma begins to take place.

The main idea behind this dilemma is that the first rewards obtained in a node may not represent well how good it is in the long term, and the highest reward could be hidden under low initial rewards in another node [84].

In this regard, the *Regret* is defined as the loss due to choosing suboptimal choices instead of the following the globally optimal policy. To ensure that the optimal choice is not missed, all actions must have non-zero probabilities of being chosen. Consequently, it is important to place an upper confidence bound on the rewards observed so far.

Based on this, in [85], the authors proposed policies that associate quantities called *upper confidence index* to each machine, as an estimate of the expected reward of each option, and choose the arm with the highest index value. These policies have changed throughout the time and currently are not always a function of the reward.

Moreover, the authors in [86] show policies with uniform logarithmic regret, this is, the amount of times that the optimal action is chosen growths exponentially with more simulations, at least asymptocally. In particular, the authors proposed a policy that has finite-time regret logarithmically bounded for arbitrary sets of reward distributions, with bounded supports. This policy, named Upper Confidence Bound (UCB), has been used for several years to choose the action. One of the simplest UCB variations is the UCB1. It states that the chosen action should be the one that maximizes the following expression:

$$A_t = \arg\max_a \left[Q_t(a) + \sqrt{\frac{2ln(t)}{N_t(a)}} \right] \tag{2.7}$$

For practical purposes, the authors tuned more finely the bound of Eq.2.7. They used:

$$V_a(N_t(a)) \stackrel{\text{def}}{=} \left(\sum_{\tau=1}^{N_t(a)} Q_t(a)_\tau^2 \right) \frac{1}{N_t(a)} - Q_t(a)^2 + \sqrt{\frac{2ln(t)}{N_t(a)}} \tag{2.8}$$

Which is an upper confidence bound for the node's variance. This expression means that the variance is, at most, the variance of the sample plus the second term. Then, they replaced the second term of Eq.2.7 with:

$$\sqrt{\frac{ln(t)}{N_t(a)} min(\frac{1}{4}, V_a(N_t(a)))}, \tag{2.9}$$

where $\frac{1}{4}$ is an upper bound on the variance of a Bernoulli distribution. Finally, the *UCB1-Tuned* can be expressed as:

$$A_t = \arg\max_a \left[Q_t(a) + \sqrt{\frac{ln(t)}{N_t(a)} min(\frac{1}{4}, V_a(N_t(a)))} \right] \tag{2.10}$$

This expression outperformed the others significantly, however, they could not prove a regret bound.

Years later, an Upper Confidence Bound for Trees (UCT) was developed [87], whose expression includes an *exploration factor*, so the designer can leverage which parts of the expression influence the most in the action selection:

$$A_t = \arg\max_a \left[Q_t(a) + 2C_{exp} \sqrt{\frac{2ln(t)}{N_t(a)}} \right] \tag{2.11}$$

In Eq.2.11, the expression comprises two parts: The first one is the average reward the action a has delivered historically, and the second part is a fraction between the time that has elapsed and the number of times that action has been chosen. As the reader can notice, if a given action has been selected on few occasions, the second term increases its influence. This part promotes the selection of those actions which do not have the best average reward but have been less explored, so they may lead to new scenarios, which, in turn, may produce better rewards. As mentioned, the C_{exp} factor is in charge of weighting these two elements [88].

In [89], in a Go application, the authors added two new parameters. The first is a p factor to leverage the second term in Eq.2.10, which by default is 1. This factor leads to the following

formula:

$$A_t = \arg\max_a \left[Q_t(a) + p\sqrt{\frac{ln(t)}{N_t(a)} min(\frac{1}{4}, V_a(N_t(a)))} \right] \tag{2.12}$$

This factor balances exploitation and exploration similarly to C_{exp}.

The second parameter is the first-play urgency (FPU). It aims to solve the problem of choosing actions that have yet to be explored, particularly when many actions and nodes far from the root are rarely visited. The authors assigned an urgency value to nodes to visit them at least once before exploiting them and updated them according to the UCB1 expression. Small FPU values proved to increase the performance of their implementation.

It is worth mentioning that most applications of MCTS have been in games with at least two players; thus, most of its advances and progress aim in that direction. However, some efforts have been developing a single-player MCTS [90]. In terms of the UCT expression, in [91], they add a third element in order to "inflate" the standard deviation of unvisited actions and explore them sooner:

$$A_t = \arg\max_a \left[Q_t(a) + c \cdot \sqrt{\frac{ln(t)}{N_t(a)}} + \sqrt{\sigma^2(a) + \frac{D}{N_t(a)}} \right], \tag{2.13}$$

where D is a constant and σ^2 is the action's reward variance.

With a similar approach, the work in [92] proposed a *Best Arm Identification* for the *Selection* stage for a two-player game, where the confidence on each node's values is smaller than a threshold.

A different and simpler option to solve the aforementioned dilemma is the ϵ-*greedy* rule, where the node with highest average reward is player with probability $1 - \epsilon$ and a random node is chosen with probability ϵ. As ϵ is constant, this rule has a linear regret growth over time, which differs significantly from previous policies. To fix this, a $\frac{1}{n}$ rate was introduced to diminish this exploration rate as more plays are made. This change allowed the authors from [86] to prove a logarithmic bound on the regret.

2.3.1.1. TD Learning in MCTS

As explained, before (Subsection 2.2.1), TD Learning uses *bootstrapping* to estimate the current state-value to ultimately allow a better decision-making process. In this regard, the structure of MCTS allows the implementation of a TD Learning strategy, since each node represents a state of the system and node's children can be understood as the state in time *t+1* for a specific node in time *t*.

Inspired by results where TD methods usually outperform MC methods, in [93] the authors proposed a *general Temporal Difference values in UCT (TD-UCT)* framework and made modifications in the Selection and Backpropagation stages to incorporate TD Learning. The first change is in the UCT formula where the first term in Eq.2.11 is replaced, and the new expression to choose a node is defined by:

$$A_t = \arg\max_a \left[\omega_{TD} \cdot V_{TD}(s,a) + (1 - \omega_{TD}) \cdot Q_{MC}(s,a) + c \cdot \sqrt{\frac{ln(t)}{N_t(a)}} \right] \tag{2.14}$$

Then, they proposed three variants that change in complexity, parameters and functionality. Two depend only on the reward obtained and the distance from the root to the node, and the last bootstraps from previous estimates.

The first one is the *TD-UCT Single Backup*, where the authors simplified the TD mechanics and removed the estimation of $V_t(s_{t+1})$, assuming that all nodes have converged to the outcome obtained in the Simulation stage. As such, the expression for the estimation of $V(s_t)$ is:

$$V(s_t) \leftarrow V(s_t) + \alpha(\lambda\gamma)^{d_L} \cdot \delta_1, \tag{2.15}$$

where d_L is the distance to the leaf node, γ and λ lower the importance of future rewards and δ_1 is the *temporal difference error*, used to adjust each node's value. In Eq.2.2, it corresponds to the expression inside the parenthesis. In this case, the *TD error* is:

$$\delta_1 = \gamma^P \cdot R_i - V(leaf), \tag{2.16}$$

where P is the number of steps in the Simulation stage and *V(leaf)* is the current value of the leaf node of the tree.

A second variant is called *TD-UCT Weighted Rewards*, where they remove the bias *V(leaf)* simplify the previous alternative reducing the number of parameters and set ω_{TD}, α and λ to 1. As consequence, in Eq.2.14, $Q_M C(s,a)$ disappears from the expression.

Finally, the third proposal uses bootstrapping in its dynamic. After the Simulation stage, is node's value is updated as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha(\lambda\gamma)^{d_L} \cdot \gamma^P \cdot \delta_t, \tag{2.17}$$

where δ_t is defined as:

$$\delta_t = R_i + \gamma \cdot V(s_{t+1}) - V(s_t) \tag{2.18}$$

In this last equation, it is possible to see the bootstrapping backup, which decreases its step rate exponentially.

We combined the latter TD-UCT variant with Eq.2.13, obtaining the following expression:

$$A_t = \arg\max_a \left[\omega_{TD} \cdot V_{TD}(s,a) + (1-\omega_{TD}) \cdot Q_{MC}(s,a) + c \cdot \sqrt{\frac{ln(t)}{N_t(a)}} + \sqrt{\sigma^2(a) + \frac{D}{N_t(a)}} \right] \tag{2.19}$$

The authors in the previous article did not offer a rigorous argument on why those modifications performed better or worse. Moreover, most of their article focus on testing their variants, and they pointed out in the conclusions that a better understanding of the ideas and dynamics presented there could motivate new MCTS enhancements. In this regard, while this expression is new and has not been tested on other scenarios, it may help add new tools to MCTS modifications.

2.3.2. Other modifications

Recently we have discussed some variations for the first stage (*Selection*). Others include adding fixed values to unexplored nodes to visit them before exploiting the known nodes.

Some variations take advantage of the specific knowledge to include heuristics, and others consider the maximum value of each node instead of the average for the UCB formula. Many of these expressions do not hold a rigorous formulation and are used based on trial and error. The determination parameters' values, such as C_{exp} and D, follow the same methodology in several works.

In the *Simulation* stage, a straightforward and fast implementation is randomly choosing the actions. The advantage of this choice is that it does not require previous knowledge about the application, however, the sequence of actions may not be realistic. In this regard, different improvements have been implemented, such as the introduction of evaluation functions, Recurrent Neural Networks to guide the trajectories [94], pattern recognition to choose paired actions or domain-specific policies.

Finally, some valuable strategies have been tested in the *Backpropagation* stage. One of them is the proposal explained before, which applies the concept of TD in this stage. In [95], the authors analyzed similar approaches with the idea of incorporating TD in this process. Strategies based on RL algorithms such as SARSA and Q-Learning [61] are reviewed. Others, inspired by the idea that rewards found later in the search are more accurate than those found at the beginning of the process, have tried different variations [96]. In [97], they divided the number of times N a certain node has been visited, linearly or exponentially, and weighted each segment in the same way. For example, suppose a node has been visited 300 times, divided into three segments, and weighted each segment linearly. In that case, the resultant reward array is the concatenation of the first 100 values, the second 100 values repeated twice and the last 100 repeated three times.

Chapter 3

MCTS-based Method for solving the EVRP

In this Chapter, we explain how our proposal fulfills our objectives by applying the methodologies in Chapter 2 and how we incorporated the different elements mentioned in Chapter 1 in the implementation.

We first modeled the map where we solved the EVRP as a directed graph D and characterized each node depending on its functionality: depot, charging station and depot. Then we introduce the energy consumption model of our EV and its parameters. At this point, we explained how we added uncertainty in the planning by incorporating a stochastic and dynamic *traffic* variable, which directly affects the velocity in a given edge and, hence, the energy consumption and travel time. Following, we described the State-Space Model of our EV and how it changes on every node. Details about the specific implementation of the MCTS are described and explained, where the *Reward* function is introduced.

3.1. Dynamic and Stochastic EVRP

Consider the directed graph $D = \{B \cup N \cup CS, A\}$, where $B = \{0, 1, \ldots, b\}$ are the b nodes representing depots, $N = \{b+1, \ldots, b+n\}$ are the n clients to be visited, $CS = \{b+n+1, \ldots, b+n+l\}$ are the l charging stations, the set $V = \{B \cup N \cup CS\}$ represents all the nodes in the graph and A all the arcs between non-identical nodes in our graph. This graph shows one depot, ten clients and four charging stations.

Each node, independently of its category, will have a position in two dimensions (x,y), and to calculate the distance between two nodes, we compute the euclidean distance. This distance has an impact on both energy consumption as well as travel time.

Several parameters and variables are introduced in this Section. To facilitate their understanding and order, those related to the power consumption model are described in the following table:

Table 3.1: Parameters used in our case study.

Description	Parameter	Value	Unit
Mass (EV+driver+load)	m	1961	[kg]
Rolling resistance coefficients	C_r	1.75	-
	c_1	4.575	-
	c_2	1.75	[s/m]
Air mass density	ρ_{air}	1.2256	[kg/m^3]
Frontal area of EV	A_f	2.3316	[m^2]
Gravitational acceleration	g	9.8066	[m/s^2]
Aerodynamic EV drag coefficient	C_d	0.28	-
Road inclination	θ	0	[°]
Maximum battery capacity	Q	24	[kWh]
SOC upper bound	SOC^+	0.95	-
SOC lower bound	SOC^-	0.20	-

The energy consumption model used in this implementation was obtained from [98] (recommended for a detailed discussion), and its parameters were obtained from [99]:

$$\begin{aligned} P(t) = \Big[& ma(t) + mg \cdot cos(\theta) \cdot \frac{C_r}{1000}(c_1 v(t) + c_2) \\ & + \frac{1}{2}\rho_{air} A_f C_d v(t)^2 + mg \cdot sin(\theta) \Big] \cdot v(t) \end{aligned} \tag{3.1}$$

Table 3.2 summarizes expressions related to kinematics and state transitions.

Table 3.2: Parameters for kinematics and state-transition expressions.

Description	Parameter	Unit
Travel time from node i to node j	$tt_{i,j}$	[min]
Distance from node i to node j	$d_{i,j}$	[m]
Velocity to travel from node i to node j	$v_{i,j}$	[m/s]
Reference velocity	$v_{default}$	[m/s]
$Traffic_{distribution}$	τ_x	-
$Traffic_{distribution}$ mean	μ_t	-
$Traffic_{distribution}$ standard deviation	σ_t	-
Sample from the $traffic_{distribution}$	τ	-
Energy consumption when traveling from node i to node j	$e_{i,j}$	[SOC]
Charge at Charging Station j	c_j	[SOC]
Charging time at Charging Station j	ct_j	[min]
Service time at Client j	st_j	[min]
Waiting time before serving Client j	wt_j	[min]
Demand at Client j	d_j	[kg]

The estimation of travel time is much simpler as it follows the following expression:

$$tt_{i,j} = \frac{d_{i,j}}{v_{i,j}(t)} \tag{3.2}$$

Uncertainty can be involved in several stages and be given by different aspects. For example, there may be uncertainty in travel time due to unknown and dynamic traffic conditions or energy consumption because of incorrect vehicle parameters, noisy sensors or other reasons. In this case, we modeled the uncertainty as an adimensional traffic variable, affecting the value of the velocity as follows:

$$v_{i,j}(t) = \frac{v_{default}}{\tau(t)}, \tag{3.3}$$

where $\tau(t)$ is a sample from τ_x, which is of the form $X_i \overset{iid}{\sim} N(\mu_{\mathbf{t}}(t), \sigma_{\mathbf{t}})$.

We modeled *traffic(t)* as a gaussian random variable. If *traffic*$\geq$ 1, the velocity will be slower than the set value. This event takes place during rush hours.

Taking into account this model, in Eq. 3.2, the $v(t)_{i,j}$ is no longer a constant value, as it will change every time it is instantiated. As a result, travel time and energy consumption will also vary.

With this in mind, the new expressions for the energy consumption model and the travel time calculation when the EV travels from node i to j are:

$$P(t) = \Big[ma(t) + mg \cdot cos(\theta) \cdot \frac{C_r}{1000}(c_1 \boldsymbol{v(t)} + c_2) + \frac{1}{2}\rho_{air} A_f C_d \boldsymbol{v(t)}^2 + mg \cdot sin(\theta)\Big] \cdot \boldsymbol{v(t)}, \tag{3.4}$$

and:

$$tt_{i,j}(t) = \frac{d_{i,j}}{\boldsymbol{v}_{i,j}(\mathbf{t})} \tag{3.5}$$

Our EV's State-Space Model consists of four states: Current node, Stage-of-Charge (SOC), time and weight. The transition formulation for the latter three, when traveling from node i to j and from time k to $k+1$ is as follows:

$$\begin{aligned} x_1[k] =& x_1[k-1] - e_{i,j} + c_j \\ x_2[k] =& x_2[k-1] + tt_{i,j} + st_j + ct_j + wt_j \\ x_3[k] =& x_3[k-1] - d_j \end{aligned} \tag{3.6}$$

3.2. MCTS implementation

3.2.1. Uncertainty

Every node in the Tree contains a state of our system, which is represented by Eq.3.6. As explained before, the stochastic traffic variable induces stochasticity in the travel time and energy consumption, which cannot be represented as such in a particular node in our work. To implement the MCTS, in both *Expand* and *Simulation* stages, we took the average value of the τ_x. Thus, the average $tt_{i,j}$ and $e_{i,j}$ were used to expand the Tree and calculate the reward. This decision was made in the spirit of planning with the expected value of the available information. It is appropriate to mention that these average values in the *Simulation* stage were used to calculate the reward. Details on the specific implementation of these stages are given in the following Subsection.

3.2.2. Stages

3.2.2.1. Selection

As was discussed in the previous Chapter, there are no universal criteria for the *Selection* stage, and many expressions depend on specific applications or previous knowledge. In this research, we tested four different expressions and compared the results in terms of rewards obtained: UCT (Eq. 2.11), Single-Agent UCT (Eq. 2.13), TD-UCT (Eq. 2.14) and Single-Agent TD-UCT (Eq. 2.19).

3.2.2.2. Expansion

In our work, the *possible actions* to expand the Tree were all the nodes in the graph, except those already visited. It is worth mentioning that even actions leading to unfeasible states were allowed and explored, for example, returning to the depot without visiting all the clients or reaching nodes which caused a depletion of the battery.

3.2.2.3. Simulation

In the *Simulation* stage, the depot node is available only if all clients have been visited. The available actions are all those nodes (including charging stations) whose time windows can be met at the next step (with a tolerance) with a uniform random chance. In other words, all clients who would be visited late, with a certain probability, are removed. If no more clients satisfy the first condition, all remaining clients are available to visit. Thus, if the EV does not visit a client within its time window, it will be visited after visiting clients within their time windows. This design prioritizes the constraint of reaching clients inside their time windows. We also included a variation in this stage. Whereas the default strategy to choose actions is uniformly random, we changed it to a "Nearest Neighbour" strategy. We implemented this policy to reduce the distance traveled, which is also an important variable in this problem.

3.2.2.4. Backpropagation

In the last stage, we tested two options to backpropagate the reward obtained at the previous stage:

1. Updating the average reward of each node as:

$$Q_t(r) \leftarrow Q_t(r) + \frac{reward - Q_t(r)}{N_t(r)}, \tag{3.7}$$

   where $Q_t(r)$ is the average of all rewards seen by this node, N_t is the number of times this node has been updated and *reward* is the reward obtained in the specific simulation.

2. Adding to the previous rule, the expresion from Eq.2.17.

3.2.3. Reward

The reward is assigned by evaluating the sequence of actions from the current state at each step. This is clearly a *non-Markovian* formulation since it considers the whole history and not just the current state. In a *Markovian* setup, which is the norm in MCTS, the evaluation function in the *Simulation* stage would only consider the final state to return a value. In this situation, the Tree could bypass unfeasible nodes or client nodes where its time window has not been respected.

An illustration of this explanation is found in Fig.3.1.

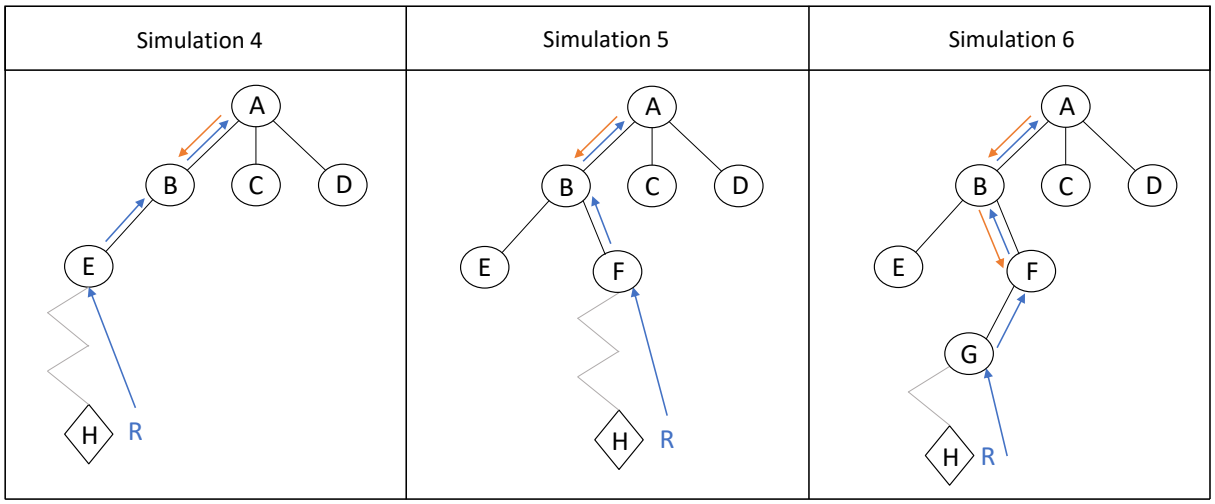


Figure 3.1: Diagram to illustrate which information is represented in reward R.

In the usual implementation of MCTS, the *Simulation* stage considers only from the *selected* node onwards. For example, in Fig.3.1, continuing the same idea from Fig.2.2, in Simulation 6, it begins from the current node at this step, which is node A since it is the first step. Then, it *selects* nodes B and F, which expands the Tree to create G. In this node, the *Simulation* stage would take place choosing actions until reaching a final node H, whose state is evaluated obtaining a reward R. In this toy example, the reward R does not take into account the information from states above node G.

To illustrate why this approach is not suitable for our work, imagine each node in this Tree as a client, if client B was visited late, there is no way of incorporating this information in the evaluation function, since the *Simulation* stage starts from node G. On the contrary, in our Non-Markovian Reward formulation, all the sequence of states from node A is taken into account for the evaluation. As such, relevant information from upper nodes is taken into consideration and is reflected in reward R.

Consequently, the agent could obtain apparent higher rewards given that certain clients have not been visited properly or the EV ran out of battery in a past node. To bring this valuable information, an option would be to extend our state-space formulation to incorporate a binary variable to acknowledge if the EV's battery has crossed the threshold along the path. While this could work for the battery restriction, it is not suitable for the time-windows requirement since. Therefore, the whole sequence starting from the initial state in each step is considered (in the Tree illustrated in Fig.3.1, the initial state would be in node A, in the first step). This allowed us to consider time-windows violations as well. This formulation meets the requirements indicated in Chapter 2, hence, an MDP formulation and solution can be applied.

As mentioned before, the *Reward* function is critical and very challenging to design [100]. It encourages the agent (in this case, the EV) to pursue or avoid specific actions with the expectation of not introducing biases. There are several metrics we want to optimize, and a great advantage of Utility Theory is that it allows us to include all of them in the *Reward* function and gives us a tool to change how we weigh each one of these metrics easily.

The terms used in our *Reward* function and its form can be seen as follows:

Table 3.3: Parameters for *Reward* function.

Description	Parameter
Time outside client t's time window when the EV arrives	time out
Discount factor	γ
Length of the sequence of nodes	n
Time the EV returns to the facility	time
Distance traveled	distance
Price of energy	e_{cost}
Amount of SOC charged	$e_{charged}$
Probability of running out of battery	risk
Auxiliar constants	k
Weights	c

$$\begin{aligned} Reward = \; & c_1 \cdot (distance - k_1) - c_2 \cdot (time - k_2) \\ & - c_3 \cdot \sum_{t=1}^{n} \sqrt{(time\ out_t)} * \gamma^t \\ & - c_4 \cdot (e_{cost} * e_{charged} - k_3) \\ & - c_5 \cdot risk, \end{aligned} \tag{3.8}$$

The risk is obtained by propagating the uncertainty of the energy consumption for each arc ahead and computing the probability that the minimum SOC threshold is met along the path. k_1, k_2 and k_3 are constants employed to make it easier to compare different sequences with similar metrics.

As it is possible to notice, our function encourages the EV to meet the client's time window and avoid running out of battery while accomplishing the task of visiting all clients. However, it also penalizes the distance the EV travels and how much time the EV spends along the route. Finally, it also penalizes the risk (the probability) of crossing the lower SOC bound. It is essential to state that all sequences of actions that lead to the violation of a hard constraint, for example, crossing the SOC threshold, receive a constant high penalty.

It can be argued that this *Reward* function induces biases as it encourages avoiding long distances trips, for example. It is not possible to formulate this problem using sparse rewards since more than one solution can be found that satisfies all the requirements. As such, it is necessary to include other elements to discriminate between good and best solutions. For instance, if two solutions met all clients within their time windows and, for the sake of argument, let us suppose the EV took the same amount of time, paid the same price and took the same risk, the only metric to choose one solution or the other would be the distance, which is a continuous variable. Otherwise, the weight for each metric is up to the designer, and different solutions may arise changing those values.

3.2.4. Best Action

After the criteria to stop the MCTS has been met in a specific node, the *best action* must be chosen for the action's online execution. In other words, after exploring the Tree, an action is selected for the EV to actually transit from one to another. We tested two criteria for this

purpose: *max-visit* and *two-phase*. The former is widely used and is more robust, while the latter was tested in a routing problem, given the nature of the rewards (Chapter 2). After the execution of this action, a new node is reached and the search continues.

To execute this chosen action, we created a *Step* function. This function takes place in the second rectangular block in Fig.2.3. Unlike the *Simulation* stage, where the average value from the $traffic_{distribution}$ is used to calculate the reward, here a random sample from τ_x is used to compute the velocity, travel time and energy consumption in the transition from one state to the other. Consequently, executing the chosen action may lead to a different state from what was expected. These new outcomes replace the previous states' values in the node and, depending on how different the execution of the action was compared to the original values, the planning is reset, erasing all nodes onwards and recreating the Tree.

After the *Step* function has been executed, the EV moves from the initial node to the best child and all the search continues from this node. For example, in Fig.3.1, if after six simulations the decision-maker chooses node B as the best child, this would be the initial node from that moment on. For instance, now the *Simulation* stage would consider all nodes starting from node B to evaluate the sequence and obtain the reward R. As can be imagined, nodes C and D will no longer be explored and can be erased to save computational resources.

Our implementation also includes the option to *wait*. If the EV arrived before a client's time window opened, with a tolerance, it had to wait until the time window began. Otherwise, it delivered the package. This action is also included in the *Expansion* and *Simulation* stages.

To summarize, the algorithm can be shown as follows:

Algorithm 1 *Monte Carlo Tree Search*

Input : Initial State, Number of simulations N_{sim}, Maximum computation time t_{max}, exploration factor C_{exp}, standard deviation inflation factor D and TD factors ω_{TD}, α, λ and γ.

Output: Best action.

create r_0 from state s_0
while r_0 is not terminal state **do**
    $A_0 \leftarrow BestAction(r_0)$
    $s_0 \leftarrow Step(s_0, A_0)$
    create r_0 from state s_0
function BestAction(s_0)
    while $N_s \geq 0$ AND *computational time* $\leq t_{max}$ **do**
        $r_l \leftarrow TreePolicy(r_0)$
        $reward \leftarrow Simulation(r_l)$
        $Backpropagation(r_l, reward)$
    return $BestChild(r_0, c, D)$
function TreePolicy(r)
    while r is not terminal **do**
        if r not fully expanded **then**
            return $Expand(r)$
        else
            $r \leftarrow BestChild(r, c, D)$
    return r
function Expand(r)
    choose $a \in$ *untried actions*
    add a new child r' to r
    return r'
function BestChild(r, c, D, ω_{TD})
    $A_t =$ **UCT-variant(s)**
    return A_t
function Simulation(s)
    while s is not terminal **do**
        choose $a \in$ *untried actions* to get the nearest node.
        s = Move(s,a)
    $Reward \leftarrow reward\ for\ chosen\ sequence$
    return $Reward$
function Backpropagation$(r, reward, \alpha, \gamma, \lambda, d, l)$
    while r has a parent **do**
        $N_t(r) \leftarrow N_t(r) + 1$
        $Q_t(r) \leftarrow Q_t(r) + \frac{reward - Q_t(r)}{N_t(r)}$
        $\delta = reward + \gamma \cdot V_{t+1}(r) - V_t(r)$
        $V_t(r) \leftarrow V(r) + \alpha((\gamma \cdot \lambda)^d) \cdot (\gamma^l) \cdot \delta$
        $r \leftarrow parent\ of\ r$

Chapter 4

Case Study: Electric Vehicle Fleet for delivery purposes

In this Chapter, we explain the specific case study in which we tested the methodology explained in the previous Chapter. In the first part, specific details about the environment and the map are presented, such as the traffic and the characteristics of each type of node. Then, we present and discuss which results are valuable for our purposes and how they were obtained.

4.1. Map

As mentioned earlier in this research, we aim to solve the EVRP with a stochastic and dynamic map. Stochasticity was introduced by including a *traffic* variable, whose effects on travel time, energy consumption and velocity are explained in the previous Chapter. To account for dynamism, in this work, the $\mu_{traffic}(t)$ depends on the hour of the day, and its value is pictured in Fig. 4.1 with $\sigma_{traffic}$ value equal to 0.1. As a result, particularly in rush hours, it can be very sensitive to which hours the EV departs from the depot and travels to each node. In this case, the EV departs from the depot at 8 am.

The *traffic* values were calculated to obtain the same velocity dynamic from [35]. These values are obtained from real data from a busy traffic area in Santiago, Chile.

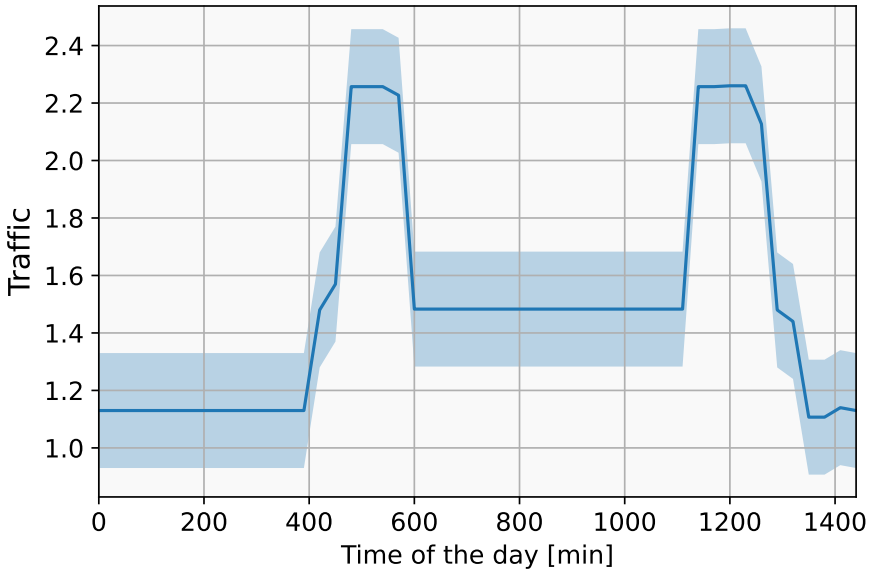


Figure 4.1: $Traffic_{distribution}$ as a function of time. The shaded zone represents two standard deviations.

In the case of the velocity, the $velocity_{default}$ was set to 40 $\left[\frac{km}{h}\right]$. This value was divided by the *traffic* value accordingly.

Besides the location, nodes have different characteristics depending on their type. Depots and Path nodes only have a location. Client nodes have these:

1. Demand: Weight of the package the client must receive.
2. Time Window: Range of time when the client must be visited.
3. Service time: Amount of time each EV will remain in the node to deliver the package.

Charging Stations have:

1. Capacity: It restricts how many EVs can charge simultaneously.
2. Technology: It can be *slow*, *normal* or *fast*. Each has a specific price and affects the time the EV remains in the node.

In this regard, some works implement a full charge every time an EV goes to a charging station [101], which is suboptimal if, for example, there is only one client left to be visited and is near to both the EV current position and the final depot. In this work, we calculated the amount of charge as a function of the number of remaining clients and the energy consumption among those.

For example, let us suppose there are three clients yet to be visited (A,B and C), and let us assume a deterministic energy consumption for every pair: *ec(A,B)* = 1 [kWh], *ec(A,C)* = 2 [kWh] and *ec(B,C)* = 3 [kWh]. The average energy consumption among clients is 2 [kWh]. Finally, the charge the EV will get is:

$$charge = (n_{clients}) \cdot e_{clients} \tag{4.1}$$

In this example, the EV would get a charge of 8 [kWh]. In our implementation, the sampling to calculate this energy consumption is uniformly random. Now, this expression could lead to *charges* of 0.15 or 0.1 [SOC], which is unrealistic if we consider the time it takes to arrive at a charging station and the whole procedure in real life. Consequently, to solve this issue, we set 0.3 [SOC] as a minimum amount to charge every time an EV goes to a Charging Station. If the Charging Station's technology is *slow*, the EV charges up to 0.95 [SOC].

We used the same non-linear charging time curve as [35].

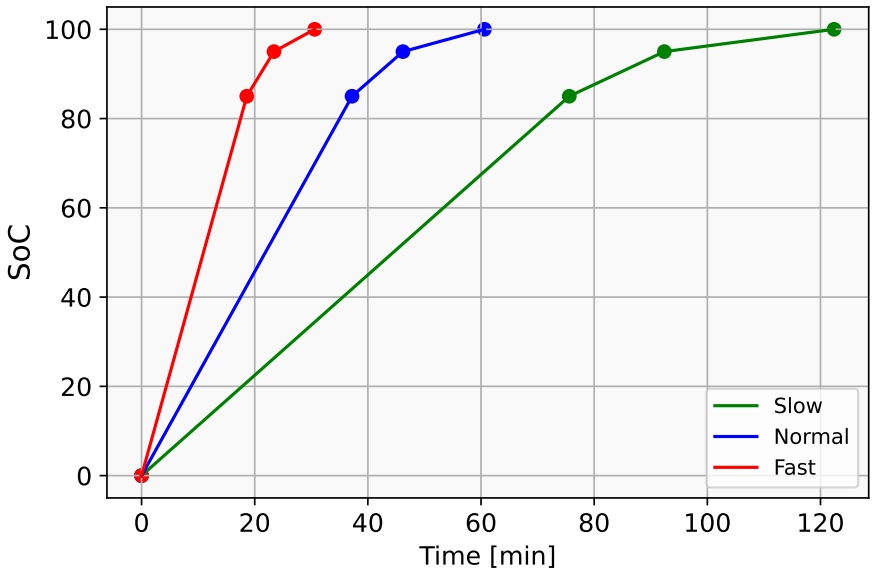


Figure 4.2: Charging functions depending on Charging Station's technology.

The price for charging is $ 0.169 per [kWh] [102] multiplied by a factor depending on the type of technology: 1 for *slow*, 2 for *normal* and 3 for *fast*. Slow charging stations always charge the battery up to 95 %.

Also, in the battery, as previously said, it is recommended to avoid extreme charge levels. As our work solves the problem for "one run" (thus, the battery's State-of-Health is not affected significantly), it is pertinent to discuss concerns about our decision policy in the long run. Considering this information and previous works, we have set an upper and lower boundary for the maximum and minimum charge allowed in each EV, being 95 % and 20 %, respectively.

In Fig. 4.3, we show the map where we tested our methodology:

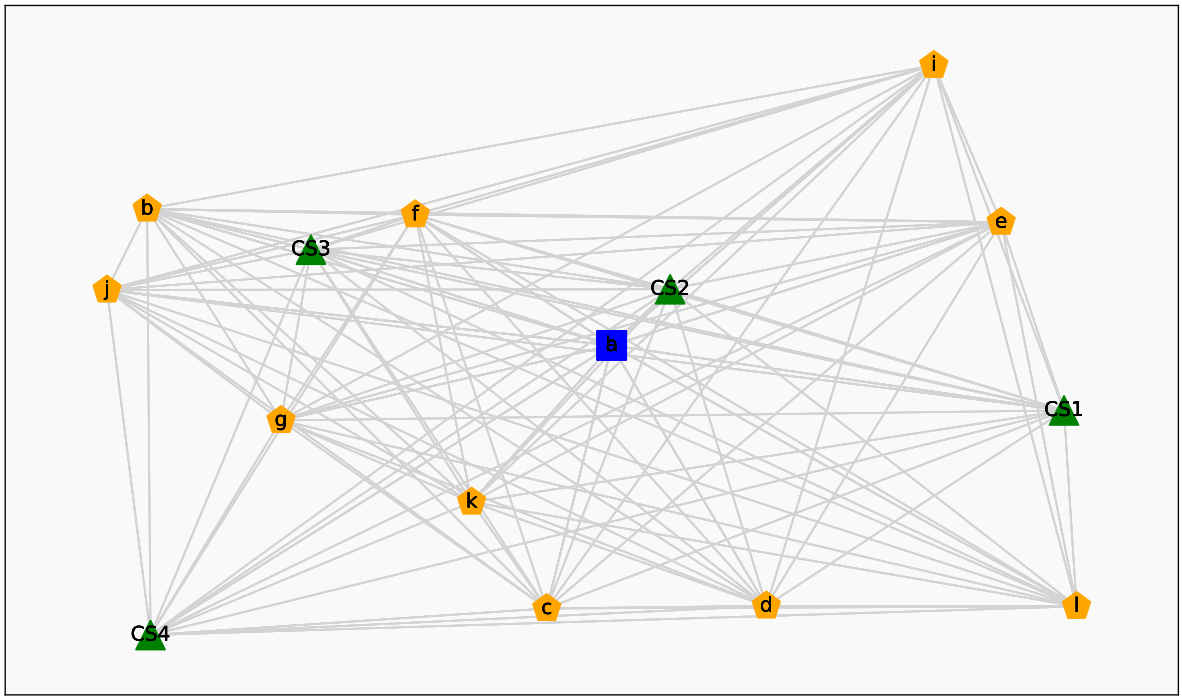


Figure 4.3: Graph. The blue node represents the depot, the yellow ones the clients and the green ones the charging stations.

The parameters of each node are in Table 4.1.

Table 4.1: Parameters of each node type. The depot is represented as two nodes in the same location for implementation purposes.

Node	Pos X [km]	Pos Y [km]	Type	Demand [kg]	Time Window [min]	Service Time [min]	Technology
a	0.0	0.0	Depot	-	-	-	-
b	-13.67	6.0	Client	10	[636.17 - 793.61]	12.64	-
c	-1.91	-11.57	Client	60	[707.78 - 875.44]	11.2	-
d	4.55	-11.46	Client	80	[621.2 - 804.27]	12.06	-
e	11.46	5.42	Client	30	[690.58 - 891.09]	8.57	-
f	-5.78	5.75	Client	50	[543.33 - 727.99]	11.0	-
g	-9.73	-3.3	Client	40	[861.48 - 1039.28]	9.35	-
h	0.0	0.0	Depot	-	-	-	-
i	9.48	12.32	Client	30	[812.19 - 991.02]	12.62	-
j	-14.85	2.44	Client	30	[626.22 - 785.13]	11.3	-
k	-4.12	-6.89	Client	50	[959.79 - 1143.95]	11.32	-
l	13.68	-11.48	Client	60	[742.99 - 899.4]	10.59	
CS1	13.31	-2.86	CS	-	-	-	Slow
CS2	1.72	2.47	CS	-	-	-	Normal
CS3	-8.85	4.21	CS	-	-	-	Fast
CS4	-13.57	-12.74	CS	-	-	-	Normal

4.2. Best route and special scenarios

To choose the best route, we ran simulations doing a parameter sweep with a fixed reward function. These parameters influence the UCT formula (Eq.2.13) and, therefore, change the

obtained sequence. We defined the best route as achieving the maximum reward obtained among all the rewards obtained in the simulation stage, this sweep.

As discussed in Chapter 2, different UCT variants may improve the search and results depending on the application, and there are no universal methods. With this in mind, we ran this parameter sweep for the four different UCT expressions and with the two criteria to select the action to execute, i.e., eight different configurations were tested to find which was more suited to our work in terms of convergence and rewards.

MCTS is a real-time algorithm, and since this work does not aim to research the efficiency of our implementation, computational time is of the essence. Thus, after choosing the best route, we studied which parameters allowed us to reach this solution minimizing the computational effort.

As every iteration has stochasticity, different results may arise even with the same parameters. To study the consistency of the results, we ran our implementation with fixed parameters several times and chose those which consistently achieved the best sequence.

Then, we tested our algorithm to know if it could adapt the planned route to unexpected and different scenarios: obsolete planning, arc blocked, different initial guess and pick-up.

Finally, we tested our algorithm in a multi-agent case, a scenario where there is a fleet of EVs (instead of one) to fulfill EVRP's requirements. Particularly, we implemented our method with two EVs, however, it can be scaled to n.

As the computational cost increases exponentially with more EVs (since the search space rises exponentially), we tested the different scenarios and carried out a detailed analysis of our approach in the single-agent case (one EV). Notwithstanding this, the functionality can be extended to the multi-agent case, as will be discussed later.

4.3. Contingency management and different scenarios

After choosing the best route and selecting those parameters which minimize the computational cost, we wanted to test how this method deals with unexpected scenarios and new information changes the route, given its relevance in real applications where contingency management is of utmost importance.

If the difference between the planned and actual states is significant, the whole schedule is obsolete since its roots are no longer comparable. In this case, the initial plan is removed and the search is *reset*.

At this point, it is important to make clear the difference between *update* and *reset* the route. As explained in the previous Chapter, after the best action's execution, the new current node in the Tree (the one that represents the actual state) updates the EV states' values according to the outcome of the transition (which depends on the sample obtained from the $traffic_{distribution}$). It is worth to recall that all nodes are created using the expected value of the $traffic_{distribution}$. After the actual execution of the action, the obtained sample could be different from the one used to create the node, and it is with this new sample that the EV's states are changed, which changes the nodes too as consequence (as the node includes the EV's states).

It is also relevant to remember that the *Simulation* stage considers from the current node onwards (which has been updated according to the action's execution outcome).

With this in mind, *update* refers to new simulations that consider now the new updated state. On the other hand, *reset* refers to an option where the Tree below the specified node is erased and built again. The difference between choosing an option the other relies on the

discrepancy between the expected states' values and the actual ones, after the best action's execution.

Besides, in this Section the effect of an initial guess of the solution is presented, as this implementation can help to study the robustness of our work. In the following, these scenarios will be explained.

4.3.1. Obsolete Planning

The first scenario occurs when the execution of an action to travel to the next node results in a very different outcome from the initial plan in terms of travel time or energy consumption.

We provoked a 20 % drop in the EV's battery SOC to induce this scenario. As a result, the difference between the planned SOC at the destination node is significantly (arbitrarily) different from the current SOC, and the update is triggered.

4.3.2. Arc blocked

The second scenario was a variant of the Canadian Traveler Problem [103]. It takes place when the EV has decided on the next node to visit, but it is impossible to reach it along the planned arc and must choose an alternative route, simulating a blocked road. To model this, we created a subgraph between the initial and the destination node.

To create this subgraph, we placed eight new Path nodes. The purpose was to force the EV to visit these nodes before reaching the next client. The coordinates of these nodes were calculated using the central point between the two nodes and sampling a value from a uniform distribution with limits [*central point*$_x \pm$*radio*] and [*central point*$_y \pm$*radio*], where *radio* was the euclidian distance from the central point to one the planned node to visit. A trivial solution could be to visit one node and then the next client. To prevent this, we restricted the edges in this subgraph, so that each node is connected to at most two other nodes.

This subgraph offers different routes between these two nodes and represents a higher energy consumption and time traveling than the original blocked path. As a result, the EV arrives at the destination node significantly later and with less energy than expected. Similar to the previous case, this also triggers an update.

4.3.3. Pick-up

A third scenario to test our approach aims to break the best solution found early by adding a new hard restriction. Given the best route and the order in which each client is visited, the EV must visit a specific node *2* before node *1*, whereas in the best solution found, the order would be *1-2*. We set this new hard restriction and studied how the algorithm adapted. With this new restriction, any sequence which broke the constraint was awarded the maximum penalty. Therefore, the number of unfeasible sequences increases significantly, and a trajectory with a wrong order is penalized as much as running out of battery.

4.3.4. Initial guess

We studied how an initial guess changes the final solution regarding computational time and the reward obtained. We provided three initial guesses for a deeper analysis: *best route*, *unfeasible route* and *near-to-best route*. These options were chosen to study convergence and how it can correct the search when it starts with an unfeasible initial guess and a near-to-best

one.

To initialize these guesses, an entire branch in the Tree was created, where each node was initialized with the following values:

1. Number of visits = 1
2. Average reward = -0.2
3. Value = 0
4. Standard deviation = 0.5

We ran 30 iterations per initial guess and studied how the final outcome and the computational time changed with every guess. Finally, we implemented a Wilcoxon Signed-Rank Test to find significant differences in computational time and a Proportion Test to conclude differences in the outcome.

4.4. Multi-agent: Fleet case

When there is more than one EV, different issues appear not only on the routing side but also in the algorithm's architecture. For the sake of an example, let us suppose that in the single-agent scenario (one EV) if we are at the depot, we have four nodes available to choose from: *A*, *B*, *C*, and *D*. When we have two or more EVs, the possible states are not only those four options but a combination of those, and in the case of two EVs, the number of possible states rises to 12. When both EVs are in the depot, there are duplicated actions, which were removed to reduce the computational effort of the algorithm. For example, if both have the same past (in this case, both are in the depot), an action tuple (*Node 1*,*Node 2*) is equivalent to (*Node 2*,*Node 1*).

New constraints arise in this case: EVs cannot visit the same client at any moment, the charging station capacity must not be exceeded and all EVs must visit at least one client.

There are some strategies to implement the multi-agent case in MCTS. One of them states that each agent should find a route without communication with the other agent [104]. While this strategy may fit zero-sum games, it would not be helpful in our case since both agents (EVs) should cooperate to avoid interfering with one another. The former approach would suit a competitive scenario where, for example, delivery trucks compete with each other to visit more clients or the most profitable. Another method was shown in [105], where they implemented a cooperative approach. In our case, we will prefer the latter.

A procedure presented in [106] was implemented, where the multi-agent actions are transformed into a single-agent structure, allowing a more straightforward implementation and treatment. Each agent executes independent actions, satisfying the new constraints, and all agents fully know the information.

In our work, now each node will include the states of all the EVs. As consequence, after the computational budget has been met, the best action will not be a single one but a tuple of actions, each one to be executed by each EV, respectively. As such, each node will now have 2 or more EVs and their respective states. A simple diagram to illustrate this approach with 2 EVs can be seen as follows:

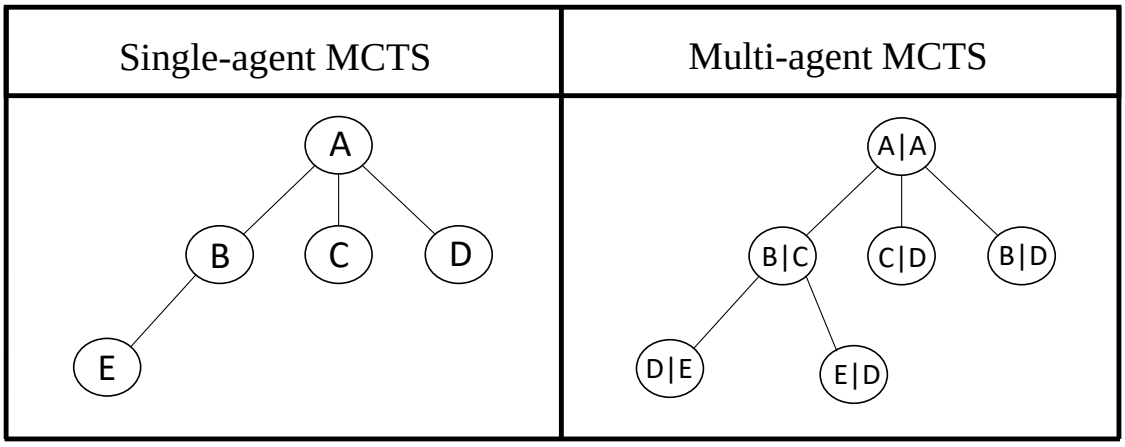


Figure 4.4: The left side represents a single-agent implementation. On the right side, the multi-agent case, where now the actions are a tuple instead of singular elements.

With this formulation, the decision-making process is similar to the single-agent case, ultimately, the node which maximizes a certain metric will be chosen and its associated tuple of actions will be executed with the *Step* function for each EV.

In this formulation, the *Expand* stage include all the tuples of combinations of actions except those tuples where both EVs visit the same client. In the *Simulation* one, the idea of visiting those clients whose time window can be met, as described earlier, changes, as now one EV can visit a client that would make some other clients be visited late if the other EV visits these last clients. In this regard, the number of possible actions increases and, thus, the computational effort. In other words, in the multi-agent case, the available nodes are those combinations of actions whose execution does not leave any client with its time window to be met (with a tolerance) with uniformly random chance. To choose the action-tuple, the first EV will select an action similarly to the single-agent case, and this action will not be available to the remaining EVs if it reaches a client. Then the second EV chooses an action an so on until all EVs select the action to return to the depot.

The *Reward* function changes as we now have more EVs. The *time*, *distance* and $e_{charged}$ are now the sum of these metrics for each EV. The penalties due to *time* out_t are calculated separately for each EV and then added, and the risk is the maximum risk among all the EVs, computed as explained before. These changes are expressed in Eq. 4.2. The *Backpropagation* stage remains just like before as its structure does not change with the number of EVs.

$$\begin{aligned} Reward = \; & -c_1 \cdot ((\sum_{EV=1}^{n_{EV}} distance(EV)) - k_1) \\ & - c_2 \cdot ((\sum_{EV=1}^{n_{EV}} time(EV)) - k_2) \\ & - c_3 \cdot \sum_{EV=1}^{n_{EV}} \sum_{t=1}^{n} \sqrt{(time\ out_t(EV))} * \gamma^t \\ & - c_4 \cdot (e_{cost} * ((\sum_{EV=1}^{n_{EV}} e_{charged}(EV)) - k_3)) \\ & - c_5 \cdot \max_{EV \in EVs} risk(EV) \end{aligned} \tag{4.2}$$

Due to computational resources, and as the features of this work have been satisfactorily implemented in the single-agent (whose extension to multi-agent is discussed in the following Chapters), the multi-agent scenario will not be tested as detailed as the single-agent.

Chapter 5

Results

In this Chapter we present the results obtained from applying our methodology to the described EVRP. They are product of an implementation in *Python 3.7.11* running on a *Dell Latitude 5490* with Windows 10 Pro and Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, 2112 Mhz, 4 Core(s), 8 Logical Processor(s) with 8GB RAM.

5.1. Best solution and hyperparameters' selection

In this Section are shown the results from testing different Selection's strategies and a combination hyperparameters, to obtain a variety of solutions (and their associated rewards). Among these, the best solution was the one that elicited the highest reward. Finally, the hyperparameters that allowed the algorithm to consistently find the solution minimizing the computational time, were selected as the best setup for this case.

The reward function's parameters can be seen in Table 5.1.

Table 5.1: Parameters for *Reward* function in single-agent case.

Description	Parameter	Value
Initial Reward for accomplishing the objective	High Reward	0
Penalty for violating hard constraints	High Penalty	-1
Distance penalty	c_1	0.000002
Time penalty	c_2	0.0001
Out of window penalty	c_3	0.005
Energy penalty (cost and charge)	c_4	0.015
Risk penalty	c_5	0.15
Auxiliar constant 1	k_1	100000
Auxiliar constant 2	k_2	1000
Auxiliar constant 3	k_3	2.5
Discount factor	γ	0.95

The constants k are minimum boundaries obtained experimentally and help differentiate sequences with metrics.

We first made a sweep of hyperparameters' values with four different UCT variants and two *best-child* criterions using the values in Table 5.2. The values for ω_{TD}, λ, γ and α were

0.5, 0.5, 0.5, and 0.01, respectively.

Table 5.2: Hyperparameters used in the sweep run for the single-agent case with different UCT variants the *best-child* options.

Hyperparameter	Values
N_{sim}	50, 100, 200, 300, 400, 500, 1000, 2000
C_{exp}	0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.65, 0.75, 0.85, 1.0
D	0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.75, 1.0, 2.5

The result as a function of the number of simulations can be seen as follows:

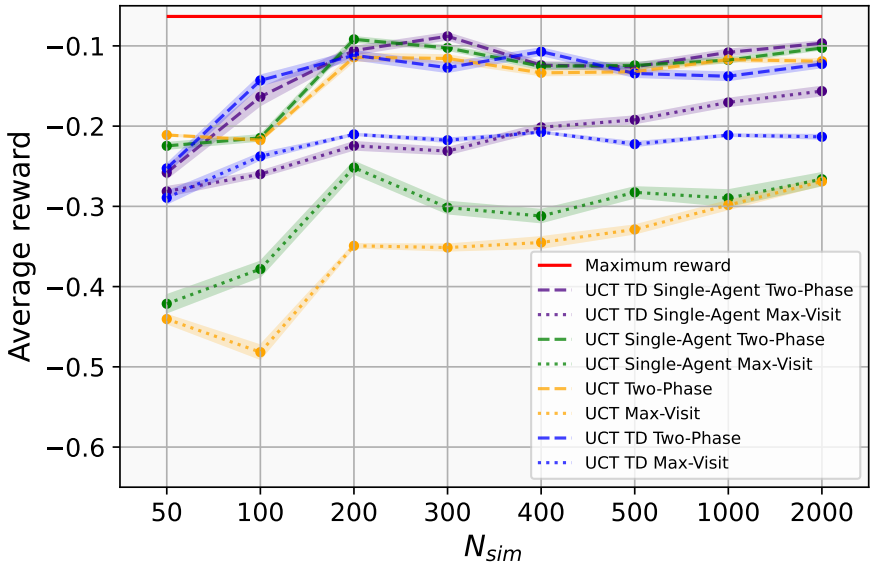


Figure 5.1: Average reward obtained as a function of Number of Simulations with different UCT variants the *best-child* criteria. The shaded area represents the standard error.

It can be appreciated in Fig.5.1 how the average reward obtained increases with more simulations. This effect is particularly notorious with the UCT with both *best-child* options. All UCT variations with the *Two-Phase* standard performed better than those with *Max-Visit*.

The proposed *UCT TD Single-Agent Two-Phase* approach performed better than any other with 2000 simulations, slightly better than *UCT Single-Agent Two-Phase*. Interestingly, it can be noted how with few simulations, higher rewards, on average, were obtained (compared to rewards obtained with 400 simulations). Then it drops, and from 500 simulations onwards, it consistently improves. Also, *UCT TD* outperformed *UCT Single-Agent* in all cases. Therefore, from now on, we used the best setting based on Fig.5.1 to obtain the following results: *UCT TD Single-Agent* with the *Two-Phase* criterion.

To have a first impression of how the implementation's performance changed depending on values C_{exp} and D, we present the following figure:

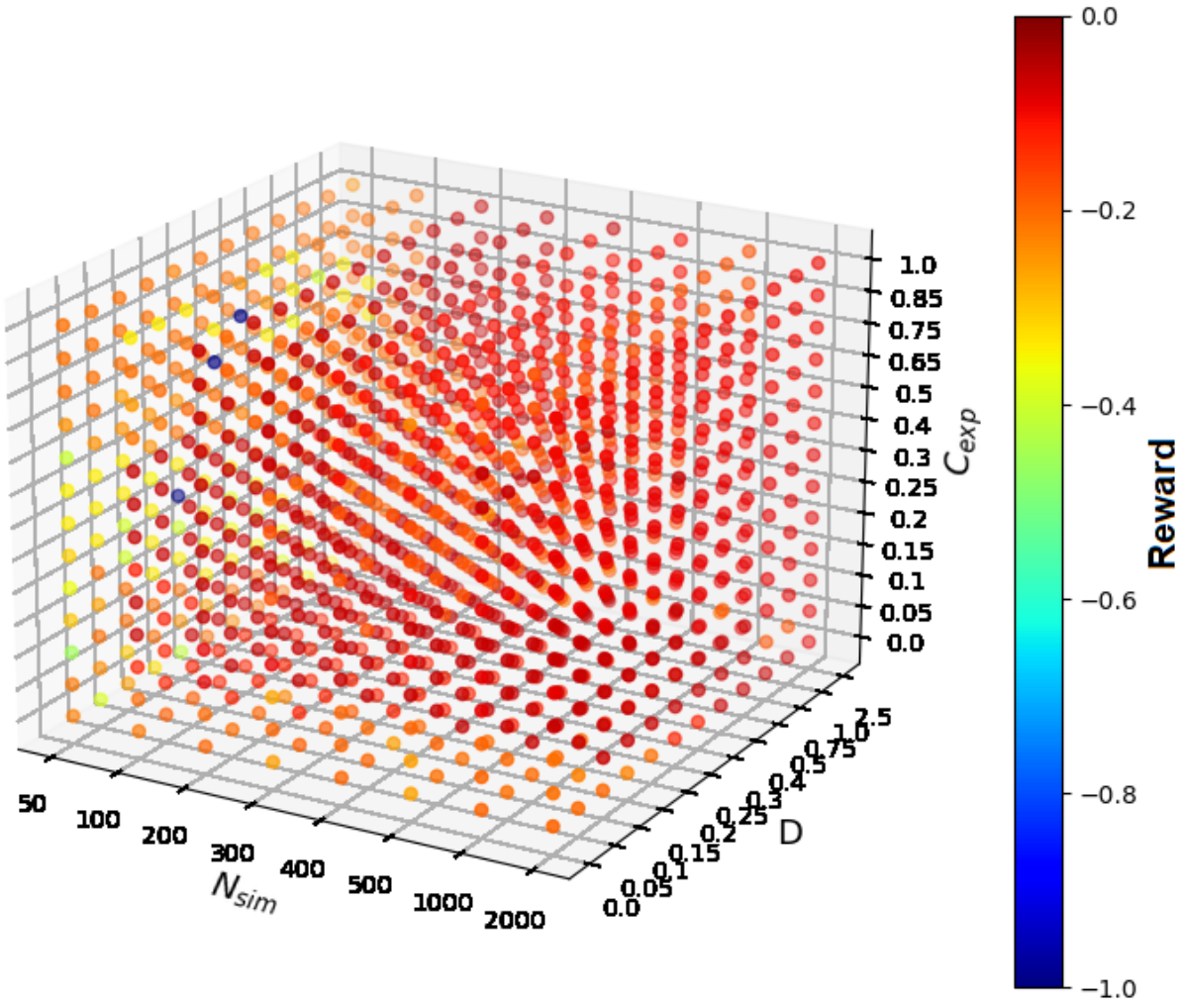
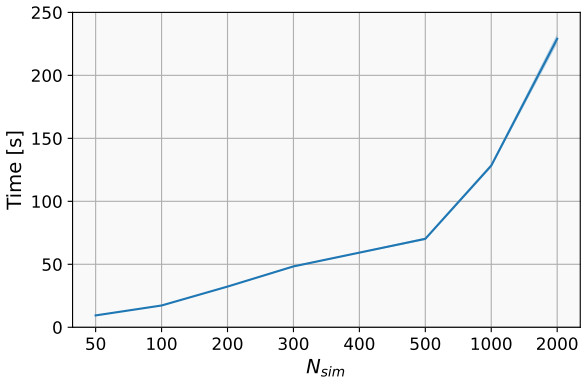


Figure 5.2: Reward obtained as a function of Number of Simulations and C_{exp} and D factors.

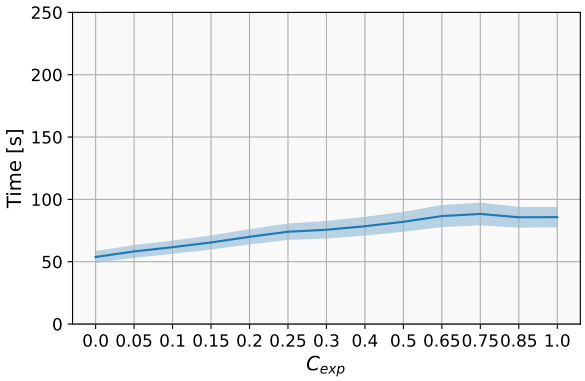
From Fig.5.2, it is possible to observe a tendency in the results. Higher rewards were obtained as the number of simulations increased in one axis, decreasing as the value of C_{exp} and D increased.

The highest reward was obtained with different combinations of hyperparameters. With a similar idea of obtaining the best UCT expression to maximize the reward, now we looked for the best set of hyperparameters that minimized the computational effort.

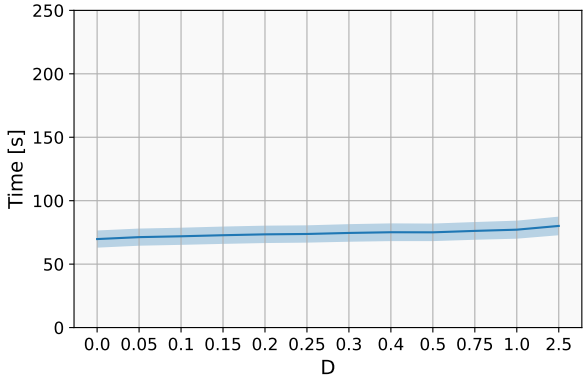
The following figures present how the different hyperparameters affected the computational time.



(a) Time to compute as a function of the Number of Simulations.



(b) Time to compute as a function of C_{exp}.



(c) Time to compute as a function of D

Figure 5.3: Time to compute as a function of hyperparameters, with standard error.

Then, based on Fig. 5.3, it is clear that N_{sim} has the most significant influence in computational time.

Although the highest reward (**-0.063308**) was obtained even with 100 simulations, it could have been luck. Consequently, we restricted N_{sim} to 500, as is the number of simulations where the search consistently started to improve with more simulations. The first set of hyperparameters for this consistency test were the lowest values which found the optimal sequence: 0.025 for C_{exp} and (0.75;1) for D. As 0.75 is near the top of our values with the D hyperparameter, and it was shown how higher values increased the computational time (Fig.5.3), we continued for the next set of values that elicitated the highest reward, with lower hyperparameters' values. These were: 0.05 for C_{exp} and (0.05;0.1;0.15;0.2;0.25;0.3;0.4;0.5) for D.

Notwithstanding the above, we tested the consistency on both setups and included more values in between to consider the sensibility of the search hyperparameters. We ran each configuration ten times and highlighted those hyperparameters-pairs that allowed us to find the best solution all the times, which can be seen in the following Table. In the name of completeness, the results for the first set of hyperparameters can be found in the Appendix (B.1).

Table 5.3: Consistency results for the second set of hyperparameters.

Hyperparameters		$\mathbf{C}_{exp}$		
		0.025	**0.05**	**0.075**
D	**0.075**	-0.215161	-0.222943	-0.075961
	0.1	-0.208501	-0.089738	-0.063308
	0.125	-0.212821	-0.069656	-0.063308
	0.15	-0.162526	-0.063308	-0.063308
	0.175	-0.126787	-0.063308	-0.063308
	0.2	-0.145417	-0.076004	-0.063308
	0.225	-0.126787	-0.063308	-0.063308
	0.25	-0.126787	-0.063308	-0.063308
	0.275	-0.126787	-0.063308	-0.063308
	0.3	-0.126787	-0.063308	-0.063308
	0.325	-0.126787	-0.063308	-0.063308
	0.375	-0.088699	-0.063308	-0.063308
	0.4	-0.063308	-0.063308	-0.063308
	0.425	-0.063308	-0.063308	-0.063308
	0.475	-0.063308	-0.063308	-0.063308
	0.5	-0.063308	-0.063308	-0.063308
	0.525	-0.063308	-0.063308	-0.063308
	0.725	-0.063308	-0.18484	-0.200348
	0.75	-0.063308	-0.139063	-0.155168
	0.775	-0.063308	-0.167501	-0.20397

From these tables and the figures showing the computational effort, we used 500 N_{sim}, 0.05 for C_{exp} and 0.15 for D as the best combination setup.

Finally, the nominal sequence of actions is:
a,f,CS3,b,j,c,d,l,CS1,e,i,g,k,h, whose routing, SOC, time and weight evolution can be represented in Figures 5.4 and 5.5:

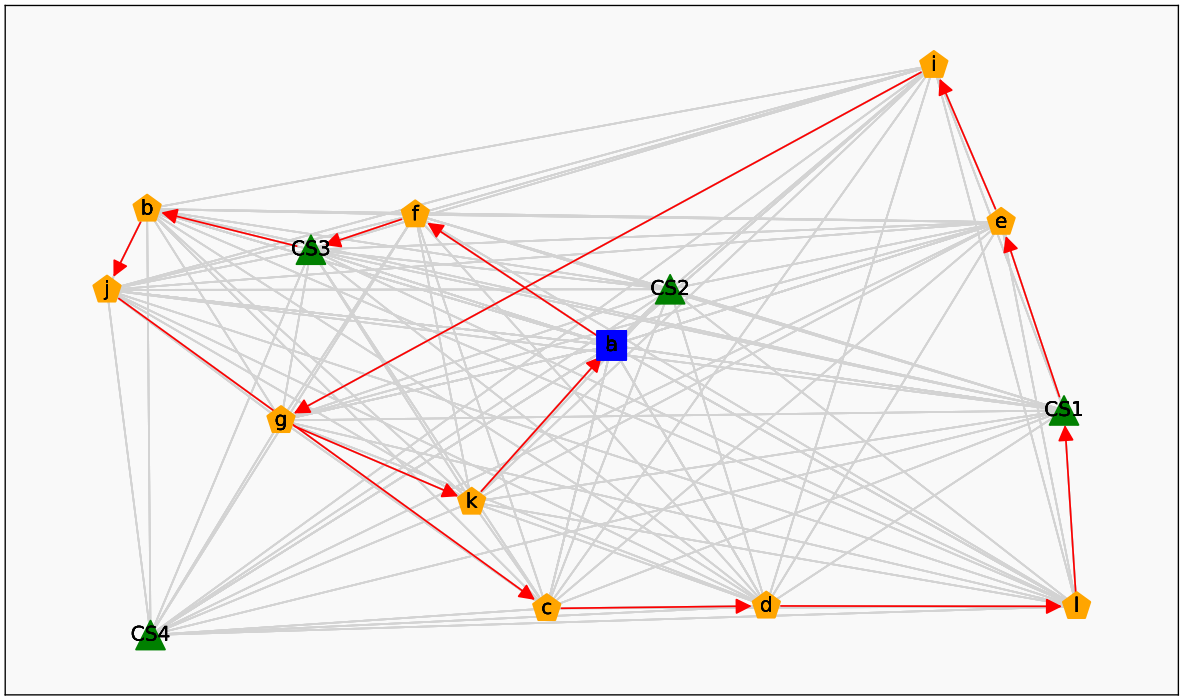
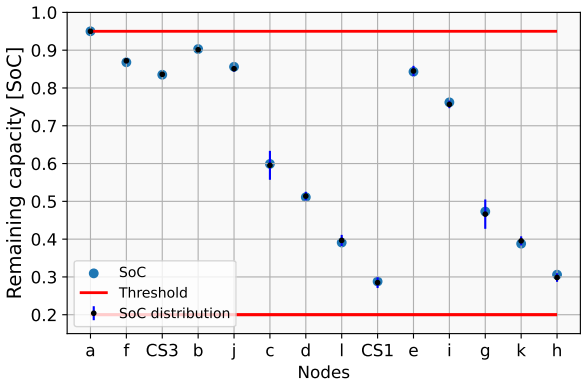
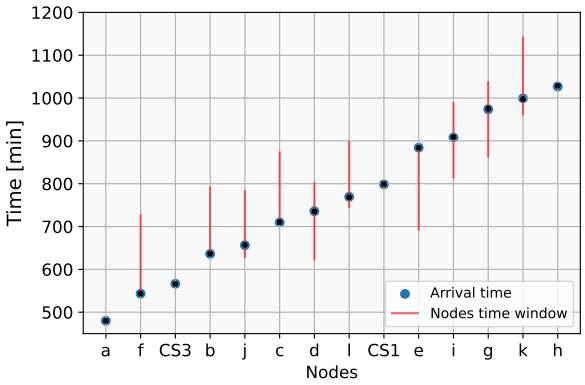


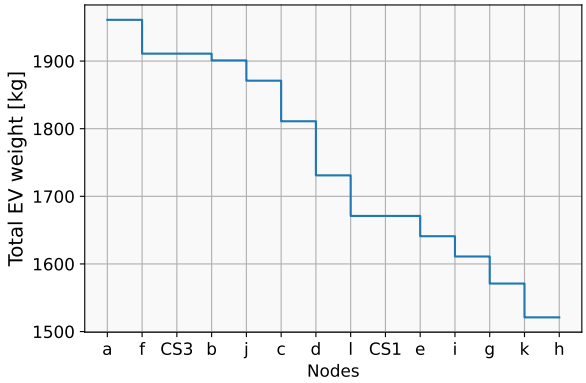
Figure 5.4: Routing solution for one agent.



(a) SOC at the arrival to each node. The SOC distribution refers to the distribution of SOC with two standard deviations, given the SOC before departing from the last node. The light blue dots represent the SOC when the EV arrives at the node.



(b) Time at the arrival to each node.



(c) Weight at the exit of each node.

Figure 5.5: EV's states evolution.

This solution took 48.37 [s], which includes the online execution of the best action encountered in every state.

5.2. Contingency management and different scenarios

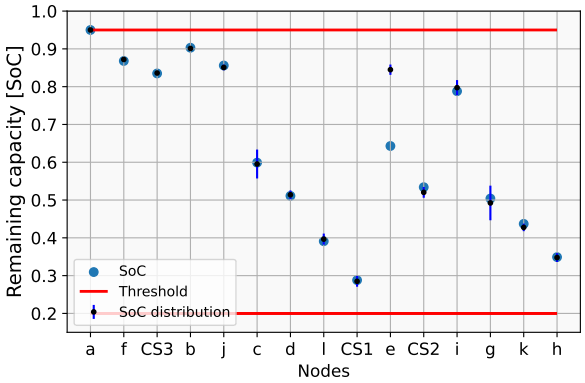
This section shows the results when an unexpected event in the route forces the agent to change the initial plan. We arbitrarily set the threshold to trigger the *reset* in the Tree if the state after the best action's execution was 5 % different or more from the expected one (before the execution). In other words, if the difference between the EV's SOC or time was 5 % or more different to what was expected, the Tree was reset.

It is also shown in this Section the results when a client must be visited before another in order to fulfill the requirements (Pick-up variation), and the effect of initializing the Tree with different initial guesses.

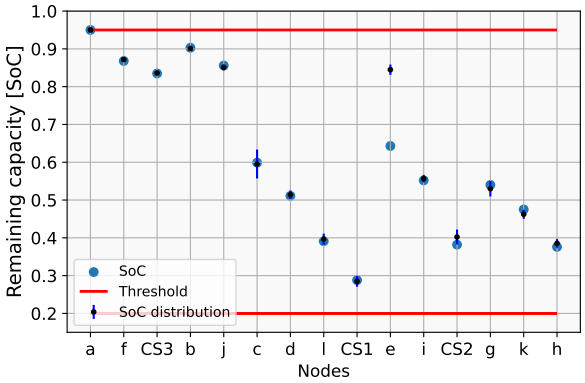
5.2.1. Obsolete Planning

As explained in the previous Chapter, in this case there is a sudden drop in the battery's remaining energy, which may force the EV to change the route. A first case is when the EV

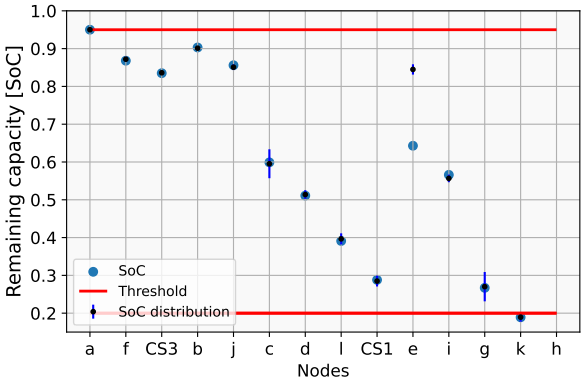
does not update the initial planning, which leads to a violation of the battery's threshold condition and the operational requirements are not met, nor exists the possibility to follow a suboptimal route. On the contrary, if the agent changes the initial planning (by resetting the Tree), acknowledging the new information, the EV includes a new charging station in the schedule. The SOC evolution for each case is represented in the following figures:



(a) SOC at the arrival to each node. When the plan is obsolete and the agent did not reset the route, the EV goes to charge right after node e.



(b) SOC at the arrival to each node. When the plan is obsolete and the agent resets the route, the EV goes to another Charging Station after node i and accomplishes the mission.



(c) SOC at the arrival to each node. When the original plan was obsolete and the agent followed it, the EV violated the hard constraint on the battery's lower threshold.

Figure 5.6: EV's SoC evolution in an Obsolete Planning scenario.

The reward for each of these cases was -0.138553, -0.092388 and -1 (the EV crosses the battery's lower threshold, violating a hard constraint), respectively.

The route found in the second case can be seen as follows:

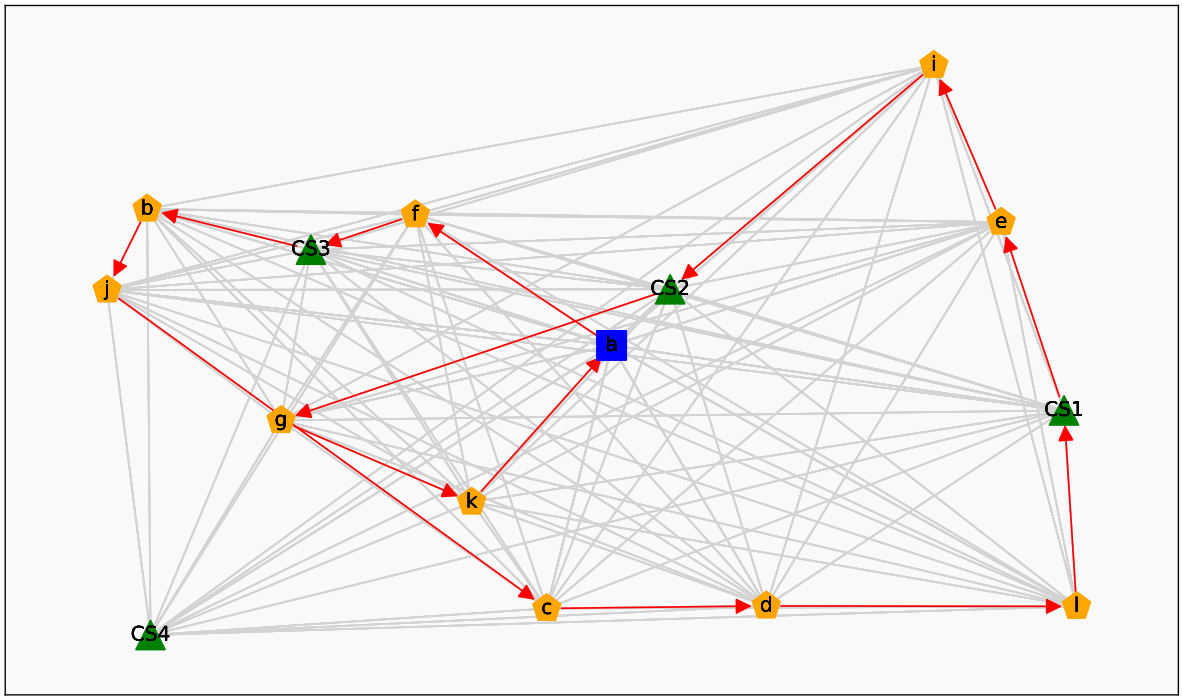


Figure 5.7: Routing solution in an Obsolete Planning scenario.

5.2.2. Arc blocked

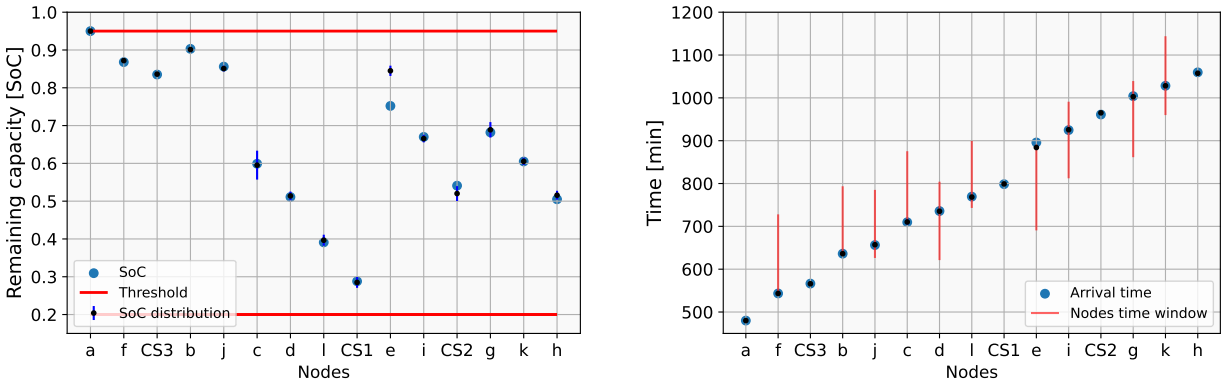
An alternative route requires more time and energy than the original route demands. In terms of SOC, the final result is similar to the previous case, where the difference between the planned SOC and the actual SOC is significantly different and may trigger a *reset* in the Tree. However, the *Arc blocked* scenario also affects the time's state. In conclusion, a *reset* in the Tree may also be triggered by a significant difference between the planned time and EV's current time after the action's execution.

In this subgraph, a routing problem is also carried out. Although much simpler, it also required *Reward* function's values. Those are detailed as follows:

Table 5.4: Parameters for *Reward* function in Arc Blocked case to solve the subgraph routing.

Description	Parameter	Value
Initial Reward for accomplishing the objective	High Reward	0
Penalty for violating hard constraints	High Penalty	-1
Distance penalty	c_1	0.0001
Time penalty	c_2	0.01
Out of window penalty	c_3	0.0045
Energy penalty (cost and charge)	c_4	0.001
Risk penalty	c_5	0.15
Auxiliar constant 1	k_1	24000
Auxiliar constant 2	k_2	670
Auxiliar constant 3	k_3	0
Discount factor	γ	0.95

The following figures show how this alternative route increases the time and energy the EV spent traveling from node *CS1* to node *e*. The final time and energy remaining are significantly lower and higher, respectively, from the initial plan.



(a) SOC at the arrival to each node. The expected remaining SOC is approximately 9 % higher than the actual one.

(b) Time at the arrival to each node. The expected arrival time to node *e* is approximately 10 minutes less than the actual one.

Figure 5.8: EV's SOC and time evolution in an Arc blocked scenario.

In this scenario, all clients remaining are still visited within their time windows.

Similarly to the previous event, the EV went to charge one more time after charging in *CS1*, instead of following nodes *i*, *g* and *k*, as the initial best solution guided. This was caused by the change in the energy consumption due to the subgraph between nodes *e* and *i*. Among the possible solutions, three alternative routes could have been followed: charging in *CS2* after node *e*, charging in *CS2* after node *i* or following the original best solution. Unlike the previous scenario, where the best plan is unfeasible, in this case it can be followed, however, its associated reward is -0.131768, whereas charging in node *CS2* after node *i*, as shown in Fig.5.8.b, leads to a reward of -0.09406. Finally, the reward for charging in *CS2* after node *e* is -0.140966.

The routing for this case is shown below:

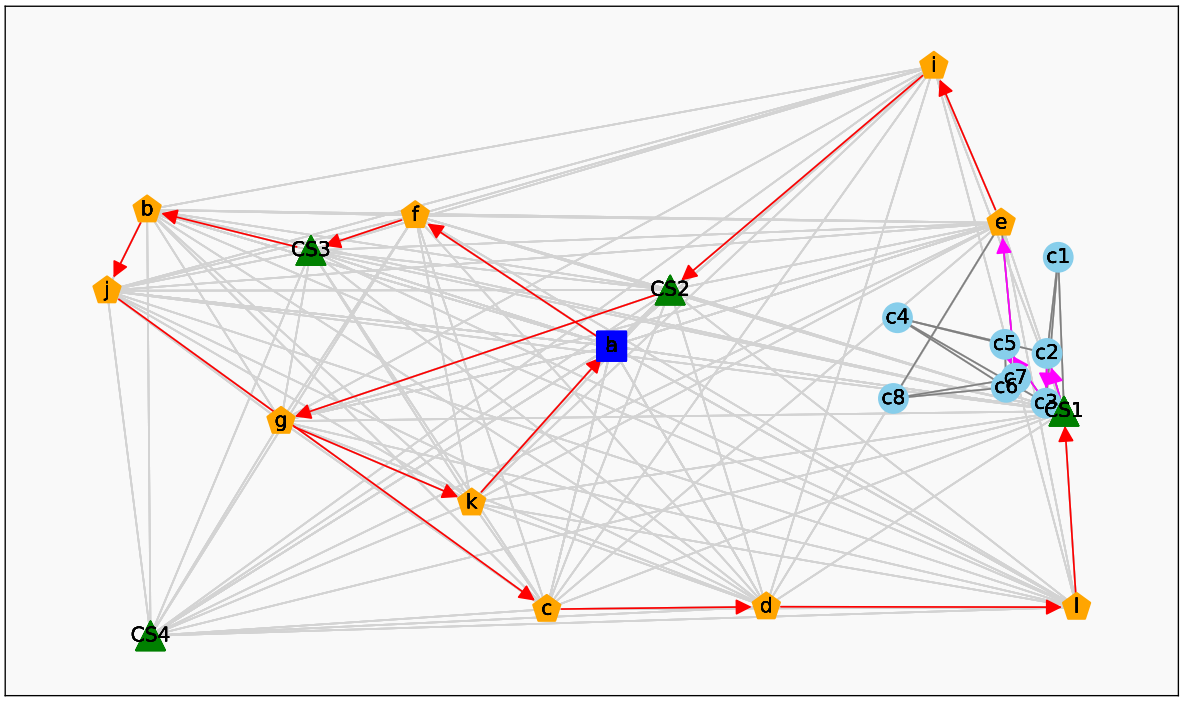


Figure 5.9: Routing solution in the subgraph case.

The subgraph and the routing can be seen in Fig. 5.10.

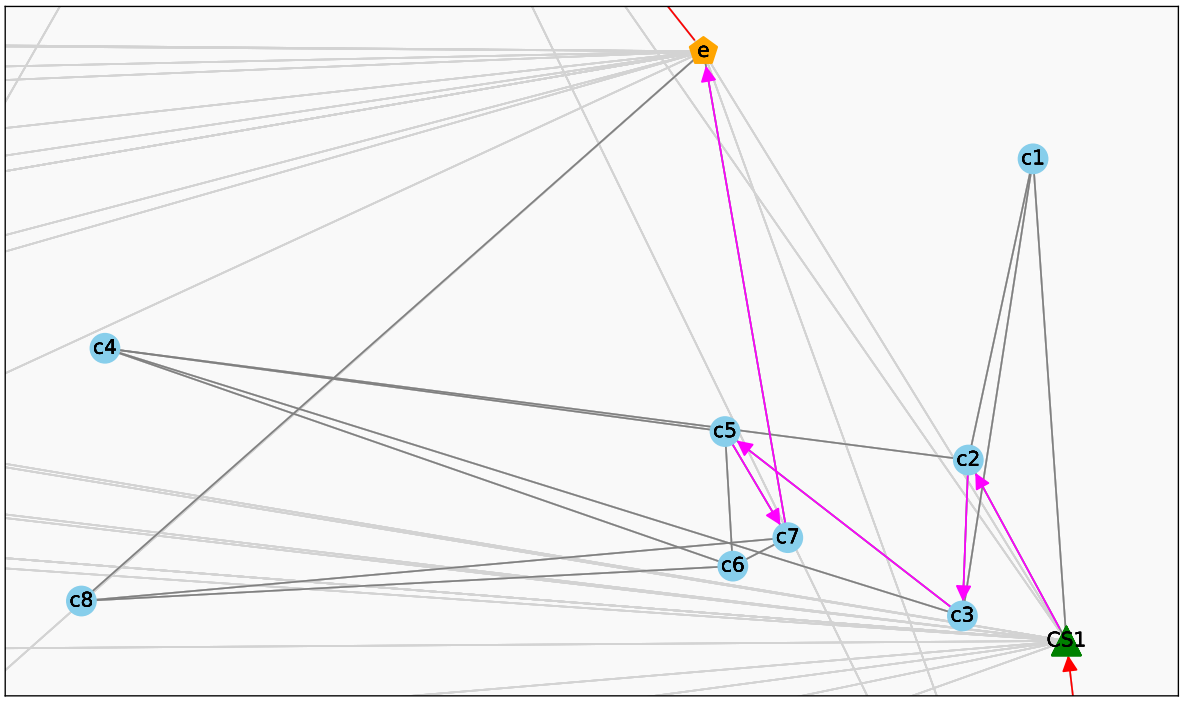


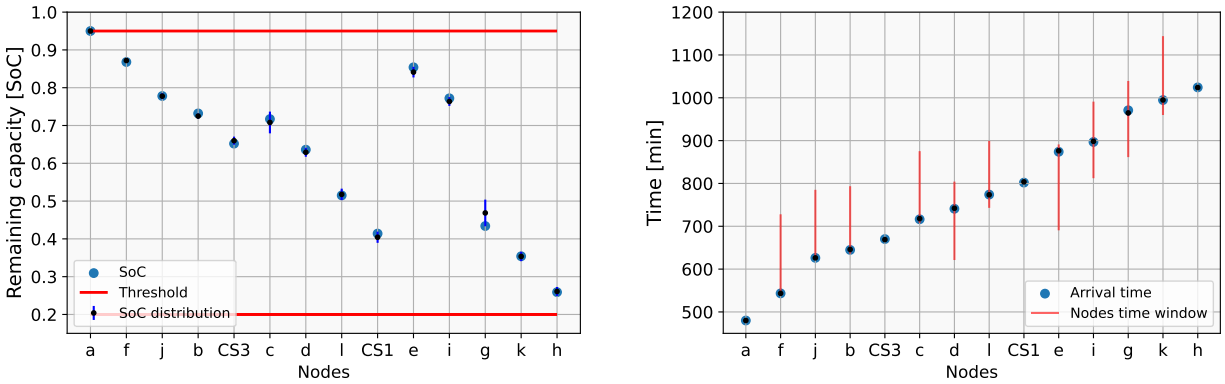
Figure 5.10: Subgraph routing. Gray edges show the arcs in the subgraph and the chosen route is magenta.

5.2.3. Pick-up

This scenario demands the EV to visit a client in a different order than what the best strategy dictates. This changes the structure of the problem since this new restriction influences the planning. The search is significantly more complex because of the new hard constraint. As such, previous exploration values obtained from the consistency run cannot be directly applied here. With this in mind, we ran 10000 simulations with the highest values used before to explore: 1.0 and 2.5 for C_{exp} and D, respectively.

In this scenario, the obtained reward is strictly below the maximum reward obtained before since this new restriction directly affects the original schedule and the previous optimal solution is no longer feasible. In particular, it increases the distance traveled and time spent. This results are presented in Fig.5.11:

In this case, we set the node j to be visited before node b. The results can be seen in the following figures:



(a) SOC at the arrival to each node.

(b) Time at the arrival to each node.

Figure 5.11: EV's SOC and time evolution in a Pick-up scenario.

It can be seen in these two figures how the constraints are satisfied: visiting clients within the time windows and battery charge above the minimum threshold.

As the pick-up implies picking up a load, the weight increases at the node instead of diminishing. This situation is illustrated in the following figure:

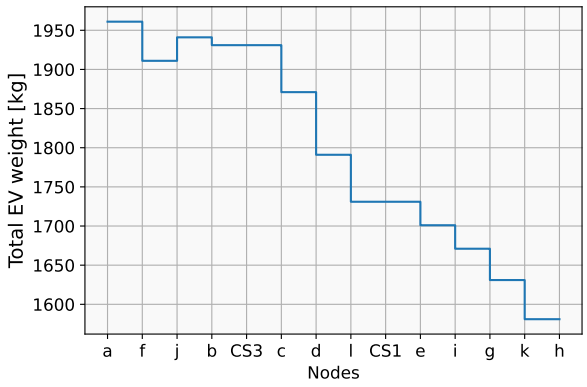


Figure 5.12: Weight after visiting the respective nodes.

The routing with pick-up restriction is shown below:

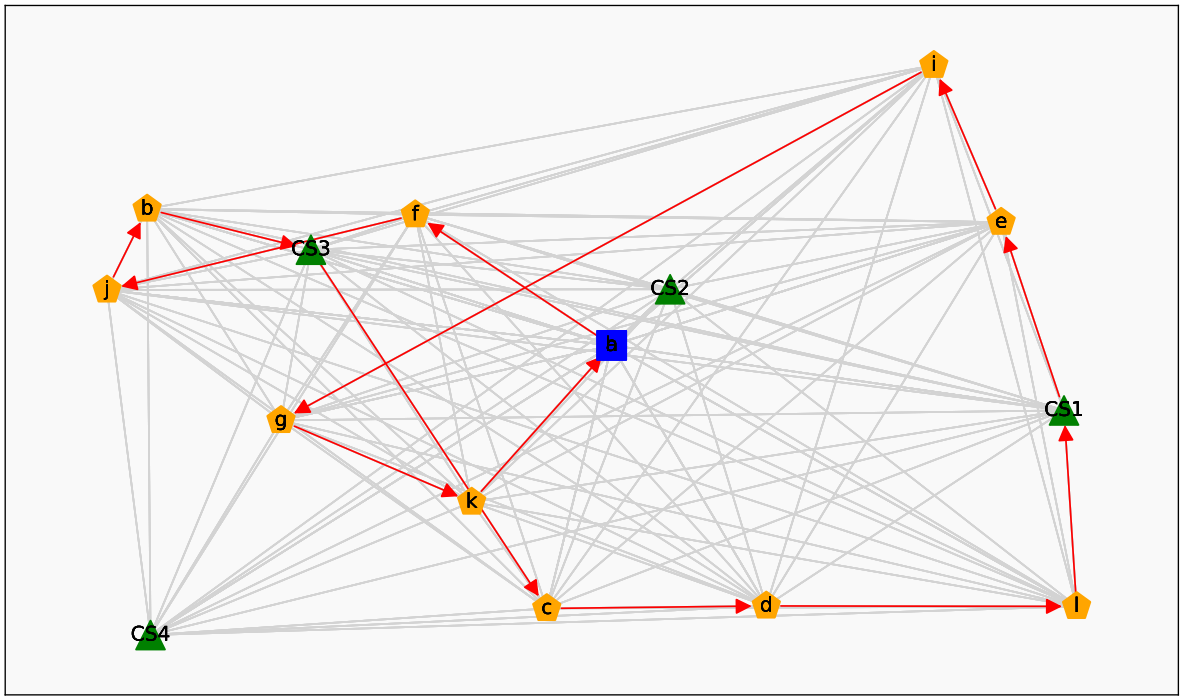


Figure 5.13: Routing solution with pick-up restriction.

5.2.4. Initial guess

We ran 30 executions with the hyperparameters mentioned above (0.05 and 0.15 for C_{exp} and D, respectively) with four different initial solutions: *no solution*, *best solution*, *unfeasible solution* and *near to best solution*. The unfeasible led to crossing the battery threshold, thus, it is tagged as unfeasible (and the reward is the highest penalty). The *near to best solution* swapped the last two clients (g and k).

With 500 simulations, out of these 30 executions, 30 converged to the best solution in the *no initial solution* case. The same result was obtained when the *best solution* was given as input. The median time to compute each execution was 49.55 [s] and 36.52 [s] for each case. A Wilcoxon Signed-Rank Test showed a significant difference in this condition ($p \leq 0.000174$), i.e., the best solution as input significantly reduces the computational time necessary. Comparing the computational time in *near to best solution* with both *no initial solution* and *best solution*, results show a significative time reduction for both cases ($p \leq 0.000174$ and $p \leq 0.0023$, respectively).

When an unfeasible solution was the input, 0 out of 30 executions converged to the best solution, however, it converged to a feasible solution in 30 out of 30 executions, visiting a near Charging Station to avoid running out of battery.

Another result is the reduction in the number of simulations required with the same exploration hyperparameters. With 400 simulations, when no initial guess was given, 5 out of 30 executions converged to the best sequence. On the contrary, when the best solution is the input, 30 out of 30 converged to the best answer. A proportion test showed that this difference is significant ($p \leq 0.001$).

Finally, when the *near to best solution* was the input, 30 out of 30 executions converged to the best solution. When comparing proportions with the 500 simulations case in *near to*

best solution and *no solution*, the proportion test showed ($p \geq 0.999$), i.e., no significant difference was found.

5.3. Multi-agent: Fleet case

This section shows the results of applying our methodology to a case where two EVs are available to fulfill the mission.

Due to the computational cost of this formulation, the results are less detailed than the single agent, where other considerations and capabilities of our algorithm were tested.

In this case, the reward function's parameters and their values are in table 5.5.

Table 5.5: Parameters for *Reward* function in multi-agent case.

Description	Parameter	Value
Initial Reward for accomplishing the objective	High Reward	0
Penalty for violating hard constraints	High Penalty	-1
Distance penalty	c_1	0.000001
Time penalty	c_2	0.0001
Out of window penalty	c_3	0.004
Energy penalty (cost and charge)	c_4	0.0012
Risk penalty	c_5	0.15
Auxiliar constant 1	k_1	105000
Auxiliar constant 2	k_2	1650
Auxiliar constant 3	k_3	0
Discount factor	γ	0.95

We also ran a sweep in hyperparameters for this case. However, we fixed the number of simulations to 10000. The hyperparameters' values are summarized in Table 5.6:

Table 5.6: Hyperparameters used in the sweep run for the multi-agent case.

Hyperparameter	Values
C_{exp}	0.0, 0.25, 0.5, 0.75, 1.0
D	0.0, 0.5, 1.0, 1.5, 2.0, 2.5

The rewards obtained for each configuration are presented in the following table:

Table 5.7: Rewards obtained in hyperparameters sweep with 10000 simulations. Multi-agent case.

Hyperparameters		C_{exp}				
		0	**0.25**	**0.5**	**0.75**	**1.0**
D	**0**	-0.1524693	-0.0554803	-0.0732026	-0.0732026	-0.0827991
	0.5	-0.1463156	-0.0554803	-0.0732026	-0.0732026	-0.0732026
	1.0	-0.0554803	-0.0554803	-0.0732026	-0.0732026	-0.0732026
	1.5	-0.0912375	-0.0554803	-0.0732026	-0.0732026	-0.0732026
	2.0	-0.0784002	-0.0732026	-0.0732026	-0.0732026	-0.0732026
	2.5	-0.0554803	-0.0732026	-0.0732026	-0.0732026	-0.0732026

The following figures and results were obtained using values 0.25 and 0.5 for C_{exp} and D, respectively.

Similar to the single-agent case, a graphical solution for the multi-agent case is shown in the following figure:

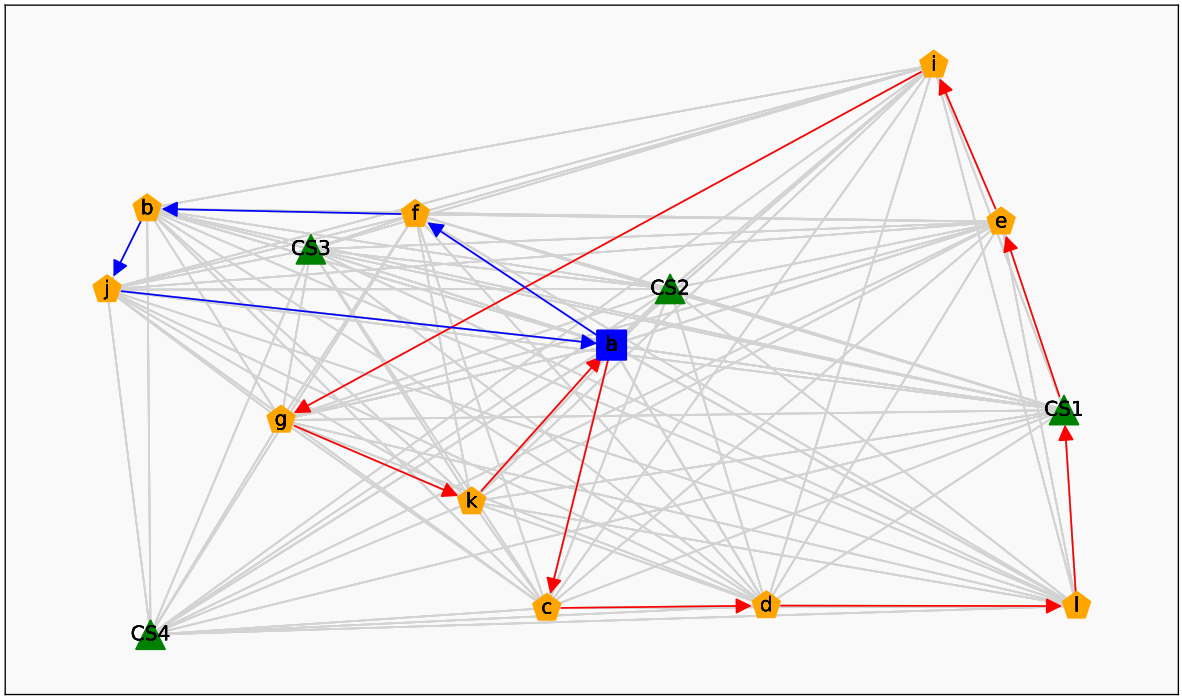
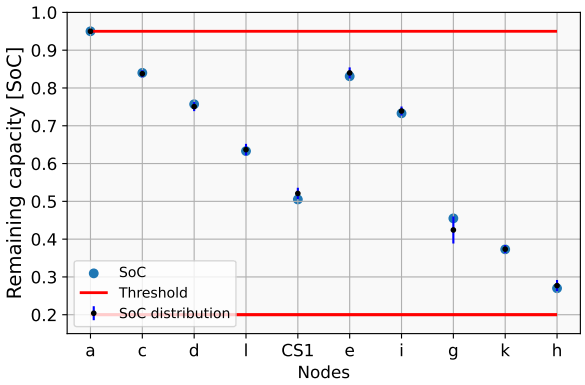
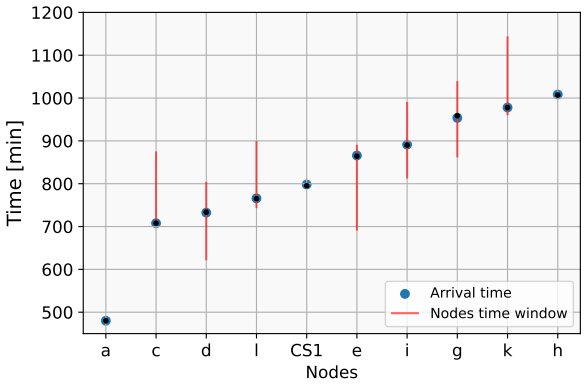


Figure 5.14: Routing solution for 2 EVs.

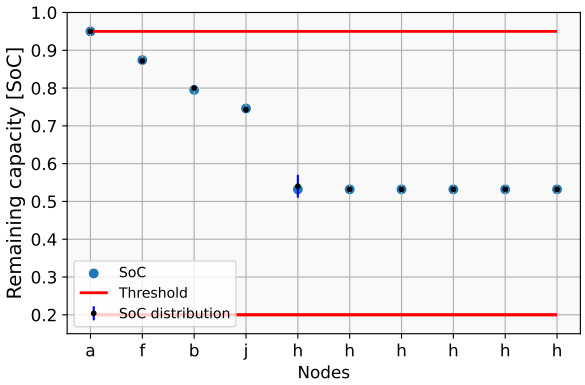
The evolution of the battery's energy, time and weight for both EVs can be seen as follows:



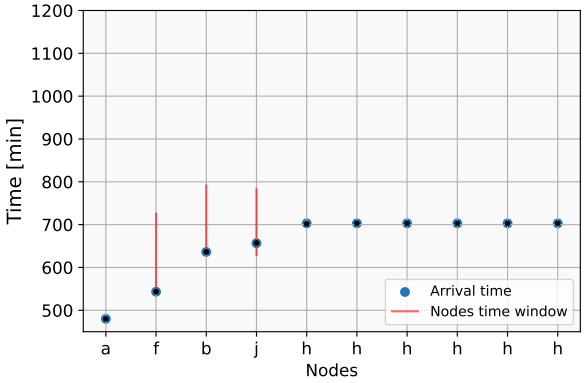
(a) SOC at the arrival to each node for EV1.



(b) Time at the arrival to each node for EV1.



(c) SOC at the arrival to each node for EV2



(d) Time at the arrival to each node for EV2.

Figure 5.15: EV's SOC and time evolution in an Arc blocked scenario.

In both cases, it is possible to see that this solution fulfills the operational constraints.

In Figs.5.15.a and 5.15.b, the last two points are equal because they represent the EV that has already arrived at the depot. Thus, the energy consumption, travel time and uncertainty are 0.

As every simulation returns a reward, it is expected to see that the reward should converge as long as the algorithm progresses. Now, it is relevant to study at which stage the algorithm has already converged to a reward and thus, it is no longer necessary to run more simulations.

The reward evolution for every step can be seen in Fig. 5.16.

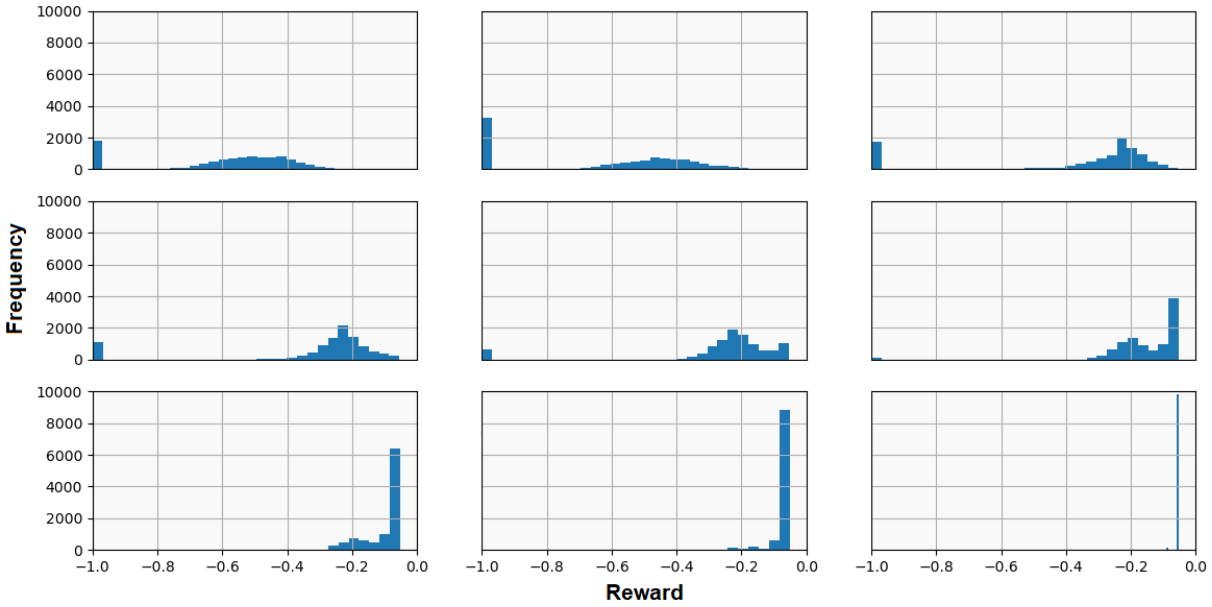


Figure 5.16: Rewards obtained in every step. Progress in the algorithm goes from top to bottom and from left to right

As shown in Fig.5.16, the algorithm converges to the solution approximately after the third step. Consequently, running all of the simulations from that stage onward is unnecessary. In order to have a reference of the savings in computational time, with the specified hyperparameters, one execution took 16.91 [min] to run. If we only run the first three steps and then choose the best child, it took 12.05 [min]. The first steps account for most of the algorithm's computation time to find the solution.

Another option is *early stopping*. If there are 3000 simulations left to run and the best action so far has an advantage of more than 3000 simulations, it could be reasonable to think there is not much gain in running more simulations since the chosen action for this step will not change. One execution took 13.7 [min] to complete with this method, being faster than the usual method but slower than running only the first three steps and then choosing the best child. This idea is not rigorous since more computations may change the chosen action two or more steps ahead, but it did not in this case.

Chapter 6

Discussion

As can be seen in Chapter 5, our implementation solves on-line the Electric Vehicle Routing Problem described in Chapter 1.3. In this work, we have successfully incorporated valuable elements to increase the relevance and applicability of our approach, such as soft time windows, partial charge at charging stations, non-linear charging function, battery management, treatment of uncertainty, dynamic and stochastic map, contingency management, different scenarios and the multi-agent case. Most of these features have been pointed out as important elements to incorporate into the EVRP problem for a more realistic study. In addition, an incipient study in the effect of different UCT expressions and hyperparameters is carried out to show how they affect the performance of MCTS. In this Chapter, several aspects of this thesis are analyzed and discussed, concluding with items that were not directly addressed in this work but represent interesting ideas to discuss and take into account in future efforts.

6.1. Best solution

The best route was defined as the route that maximizes the reward function. In this case, the algorithm consistently converged, with a variety of setups, to a sequence which is the route with the highest reward found. Although the solution was found with as few as 100 simulations per step, it was probably luck for most setups, as discussed in [81], since, on average, with more simulations, the reward decreased for the first hundredths. Notwithstanding this, some setups could have found and converged to the optimal solution with just 100 simulations. For example, values 0.05 and 0.4 for C_{exp} and D, respectively, found the best solution consistently with 100, 200, 300, 400, 500, 1000 and 2000 simulations. With different UCT variants, it was possible to find similar results. We could have chosen those values to run our tests, and savings in computational effort would have been huge, however, we decided to guide our tests according to Fig.5.1, to be based on a more consistent option. With 100 simulations, it took 13.72 [s] to solved the problem on-line.

The inclusion of MCTS' variants and testing more modifications arise as an interesting field for research. For instance, new expression in the *Selection* stage could be tested to improve the search, and a more in-depth analysis could be carried out to study which hyperparameters' values can elicit better rewards with less computational resources. Moreover, changing the values of the parameters in the *Backpropagation* stage could also affect enormously the performance of the algorithm. The idea of creating a map to study the sensitivity of our method as these values change is an open field to put efforts.

In other works, for instance, in [107] the authors implemented three algorithms combined

with UCT and tested their success in the Go game. They included Rapid Action Value Estimation (RAVE) to estimate each action value quickly. The RAVE is based on *all-moves-as-first* (AMAF) heuristic [108]. It updates an action-value every time it is chosen in any simulation, not only in the respective node. It is beneficial when the action-space is large, which causes slow learning. This algorithm significantly improved their winning rate. A study on this technique with MCTS is presented in [109]. Another modification is proposed in [110], where they used maximum entropy to obtain faster convergence.

6.2. Uncertainty in MCTS

One valuable issue is how the *Reward* information is calculated and backpropagated upwards. In this research, the sequence obtained in the *Simulation* stage was executed using the average value of the $traffic_{distribution}$, thus, average energy consumption and time values were also obtained. With this methodology, it was possible to propagate the uncertainty in the EV's energy consumption along the sequence to calculate and penalize the risk of crossing the battery's lower threshold with those actions. By doing so, we are acknowledging the uncertainty in the state-transition. The reward value represents the expected value of this sequence, which is the best guess as we modeled the energy consumption and travel time in arcs as gaussian distributions.

It is up to the designer the probability used to simulate the sequence and obtain the reward. While we used p=0.5 as the best guess given our assumptions, it could be changed to allow a riskier search or a more conservative implementation, for example, p=0.05 or p=0.95, respectively.

This work incorporates the uncertainty in three ways:

1. It incorporates the risk of running out of battery in the reward function, penalizing this probability. If this penalty is high, the agent will avoid sequences with a significant chance of reaching the lower bound of SOC. This risk had an impact on the *Arc Blocked* scenario. The initial plan had a 44 % chance of running out of battery after the detour. Therefore, the agent chose to charge after node i to avoid this risk. The chosen policy has a higher reward because of this penalization. The selected sequence of actions led to a longer trip, it took more time and spent more money, however, the risk is approximately zero. At this point, it is relevant to state that the agent would have preferred the first plan in a deterministic scenario, i.e., where there is no notion of risk (it is equivalent to removing the penalty on the risk). The risk penalty is also a designer's choice; if the penalty had been lower than our choice, the agent would have probably followed the initial plan.

2. Secondly, there is uncertainty in the execution stage when the agent executes the planned action. In this stage, a random traffic sample is instantiated, and its associated energy consumption, travel time and velocity change the EV's states. As explained before, the nodes are created with an average traffic value, which could be significantly different from the actual execution. This difference may trigger an update in the planning, affecting future nodes. This scenario was shown to be within our algorithm's capabilities.

3. In the *Simulation* stage, the available actions are chosen with random probability, thus, it allows finding actions that otherwise would not appear, within an arbitrary criterion, in this case, tolerance to arrive late.

A more straightforward way to incorporate uncertainty in the planning would be to expand and simulate with a random sample of the $traffic_{distribution}$, not the average. This option would undoubtedly add more uncertainty to the modeling and make this implementation significantly faster on the computational side. The cost of this option would be the bias in the *Expand* stage because it would initialize a new node in the Tree with a random sample, which could be very different from the average; thus creating a false idea of what is coming, and finally disturbing the decision-making process. On the other hand, our option provides better planning information for the decision maker since it is the best guess of an unknown future outcome modeled as a gaussian distribution. Our method also more rigorously incorporates the risk in the reward function as it propagates the uncertainty in energy consumption along the sequence, which allowed us to calculate the risk with a formal distribution and not an empirical one. The latter could be as good as the former, as the number of simulations increases for the same sequence, however, our approach requires far fewer resources to estimate the risk.

6.3. Charging Stations implementation

The decision to fix the charging amounts leads to sub-optimal decisions, not in terms of this implementation, however, generally speaking, it is possible to see in Fig. 5.5.a that the EV could have charged less energy and still accomplished the mission without increasing the risk significantly, and reducing the time and money spent on charging, at the same time. An a-posteriori modification could reduce the amount of charge if the route is known. Albeit this approach could work and would reduce the amount of charge and time spent in the route, it would also reduce the amount of money spent and the time the EV spends in the charging station. This change could ultimately affect the whole search if the planning is sensible in that stage, particularly when a full charge is modified, and another sequence could be the new best route. This question remains open, and a continuous charge decision-making approach would be valuable to research, as in [111], where the authors discretize a continuous space and explore each segment, discretizing more as the exploration goes on.

A different solution to include continuous charging quantities would be to solve the problem without the hard restriction on SOC and introduce the Charging Stations a-posteriori along the path to avoid violating the threshold as a second and different optimization problem. An advantage of this solution is that it could allow us to charge what the EV would require, with a certain probability; thus, it would not spend more time and money than needed. However, as said before, introducing these methods as a second stage can ultimately change the best solution.

6.4. Computational efficiency

In terms of computational cost, it is worth discussing it in relative terms since it depends on the efficiency of the implemented code and the hardware it runs on. Whereas this work was written in *Python*, it could run several orders of magnitude faster by only translating it to the *C* language or improving the coding efficiency. Furthermore, the bottleneck in our implementation was not associated with the exploration of scenarios but with code efficiency elements. Therefore, it is relevant to talk about relative performances when the parameters and cases change, not absolute ones. Moreover, it is very likely that our results represent an upper bound on the computational resources needed to solve the problem in both single-agent

and multi-agent cases.

In Fig.5.3, it is possible to see how the computational time increases with the value N_{sim}, C_{exp} and D, and how the Number of Simulations is the factor that affects the most. This behavior makes sense since if we encourage the algorithm to search more before it converges, it will run more simulations and create more branches, using more memory too. Since no new branches are created when the algorithm converges, the computational cost is low.

As was explained, MCTS is an anytime algorithm, which makes it very useful, particularly in real-time scenarios. In this case (Fig. 4.3), it took about 48.37 [s] to obtain the solution, running 500 simulations per step. Only the first step took about 12.87 [s]. In the multi-agent case, it took several minutes. This result is expected as the number of branches to explore increased significantly with 2 EVs. However, it also tells us that a more powerful platform and better code efficiency could be used in real-time situations where thousands of simulations can be run per second. As the fleet EV case tests were not as detailed as the single-agent, other setups can probably converge to the best solution with fewer simulations.

Memory budget and time management have also been studied, such as in [112], where they studied various time-management strategies in different games. In [113], a bounded MCTS is implemented to play *Spades*, *Hearts* and *LOTR:C* in mobiles and computers. The authors in [114] reduced the branching factor by removing superfluous actions, limiting the search depth, introducing specific values to nodes and detecting dead ends. Finally, in [115], they pruned *bad* nodes to more precisely focus on *good* moves; similarly, in [116], the authors pruned nodes with statistically low values. Another example is [117], where they pruned random moves at the beginning hoping that enough good actions would remain available. As this issue was not in the scope of this work, no major effort was dedicated.

In a related discussion, we put effort into testing our approach without removing too many actions to know how we managed the entire action space. In other words, as mentioned before, many actions can be removed to reduce the branching factor. While we could have removed nodes causing delays in the delivery or returning to the Depot without visiting all clients, they were kept to study how our method dealt with them.

6.5. Contingency management and different scenarios

Contingency management is also of utmost importance since real-life situations are frequently very different from what was planned due to various situations such as, but not restricted to: battery failure, traffic jams, blocked roads, new missions, fleet emergencies and others. In Chapter 5, we showed how our algorithm could handle these situations in scenarios where keeping with the original plan would have led to unfeasible states (running out of battery) or risky situations. These tests were carried out only for the single-agent case, however, their extension to the multi-agent is straightforward since each one of these events affects the vehicle individually. In conclusion, there was no need to test it in the scenarios with two or more EVs.

In addition, while the application of our approach to a pick-up scenario is not studied in depth, it would be interesting to direct efforts in this variation due to its relevance in several applications and problem complexity, since it adds a new hard constraint.

At this point, it is valuable to point out how, as the *Reward* function evaluates all actions from the initial state at each step, the most current information (after the *Step* function execution) is always taken into account in this implementation. This can be seen in Fig.5.6.a, where the EV goes to charge acknowledging the current SOC value, however, given that the

Tree is already built and the nodes' values have probably converged prior to this state, it is not able to correct the route as shown in Fig.5.6.b, where that sequence obtains a higher reward than the latter, after the Tree has been reset.

It is also necessary to notice how an initial guess influences the search in this kind of algorithm, not only which sequence of actions but also how this initial guess is initialized in the algorithm. We initialized the algorithm with the initial guess as if it were a Tree branch that has been visited once and has a high reward and uncertainty. This procedure gives the initial guess an advantage over the other actions at the beginning of the search because it will be visited immediately due to its reward and standard deviation. An exciting discussion arises based on the following question: How does the designer introduce an initial guess? What is the optimal value to reduce computational effort? We need to balance its priority (thinking that the initial guess is a good solution) but not biasing too much the search towards this initial guess (in case it is a wrong sequence of actions). If the reward is too high, it will converge quickly, although it could lose time correcting it if the case is the opposite (if it does at all); if the reward is not enough, it will not signify any guidance to the search. In this implementation, it is clear how the initial solution reduced the computational effort significantly and was able to correct an unfeasible initial guess and converged to a feasible solution. Possibly more simulations were required to converge to the best solution. When a *near to best solution* was introduced, it did not require more simulations to correct it to a point where there was no significant difference with the base case. The distance each sequence represented was the main difference between the two solutions (*near to best solution* and *best solution*). One solution to reduce the computational resources to correct this initial guess would be to penalize more the distance or decrease the C_{exp} parameters, so it sticks to the best node so far. Future efforts could dive into this topic.

In regard to using an initial guess as input to guide the search at the beginning, another way to introduce previous information is related to action-pairs. We can notice how some actions are usually together in different obtained solutions. In this regard, clients e and i are usually visited sequentially (in the single-agent case). This information could be relevant to awarding some sequences with this combination of actions. This idea has been used in other areas, such as the Go game, where AlphaGo designers introduce some sequence of actions known to produce good moves in their algorithm [118, 119].

6.6. Multi-agent: Fleet case

In our case, we implemented a cooperative approach, penalizing the total distance traveled by the EVs, however, another approach could be to minimize the maximum distance an EV travels or the amount of charge it takes in the charging station. A very different case is when EVs compete, for instance, if each vehicle is from a different company and both want to visit clients.

Due to the computational effort, few hyperparameters' values were used in this case, however, they were enough to converge to the best solution with 2 EVs and a more sensitive analysis on these values can yield better set of parameters to achieve the best solution faster and\or more consistently.

As the computational effort is of the essence, it is highly desirable to explore strategies to reduce improve the time performance. In this regard, we applied an *early stopping* criteria to avoid "unnecessary" simulations. This method is not rigorously studied, and while the idea is to stop searching when the best action to be executed will not change, we know little about

how more simulations can affect the search in subsequent and deeper nodes. In consequence, while the reduction in computational is significant (~25 %), more effort should be put to study the losses associated to this kind of strategy.

Another new opportunities to explore and investigate are found in [120], where the authors point out four grand challenges for the multi-agent case: the combinatorial complexity, the multi-dimensional learning objectives, the non-stationarity issue and scalability when there are numerous agents [121]. Different strategies and approaches could be tested and studied. For instance, how the EVs communicate and fulfill the requirements. Another case is in [122], where Thakoor et al. proposed a multi-agent scenario where UAVs must surveil an area with limited communications capabilities.

An interesting research can focus on how different levels of cooperation affect the mission. A formulation for this approach can be found in [123], where they added a λ factor to balance how much each vehicle cooperates in an autonomous driving scenario, where 0 means a greedy behavior and 1 a cooperative strategy.

$$J_{i,coop} = J_i + \sum_{j \neq i} \lambda \cdot J_j, \tag{6.1}$$

where J_i is the reward individually computed for vehicle i and j represents the other vehicles.

Contingency Management in this case also opens new opportunities, such as: How would the fleet react if an EV suddenly needs to return to base? Do the other EVs visit the remaining clients or pick up the load? Alternatively, the case where an EV has planned to visit a charging station, but another EV needs to recharge unexpectedly and, as a result, the Charging Station is full. Does it wait? Does it have enough energy to go to another Charging Station?

With our design, in this case-study, it is not relevant which EV visits each client. However, this is not always the case. Suppose each EV is loaded with specific packages. In that case, it will matter which clients it will visit, and it will also affect the contingency case mentioned above, where the remaining EVs must visit the affected EV's clients. How does the troubled EV transfer its load to the other EVs? Do they meet at a central point? Are there warehouses on the map? Contingency management involving two or more EVs is an open and exciting field, and future efforts could aim in that direction.

6.7. New opportunities

In addition to new directions, fields and opportunities mentioned before in each section, there are others questions and ideas which were not directly incorporated in our work, however they arise as attractive areas and are worth dedicating a few words to.

6.7.1. New formulations

As mentioned earlier, there are other scenarios with more specific tasks, such as routing industrial trucks, waste management, regular routes and more. These cases present new constraints and objectives, and different approaches can also be tried, given the new priorities. All these questions and practical problems are worth future research efforts with this methodology. Another direction in the area of specific formulations and problems is to incorporate the knowledge of specific roads in our map, for instance, highways, low-speed roads, avenues, car-free zones, and others.

6.7.2. Reduce manual tuning

The *Reward* function considers different relevant variables, such as arriving to the client on time (with a tolerance), avoiding running out of battery and minimizing the risk of running out of charge, the distance the EV travels, the amount of money it spends and the time it takes. As this work was not focused on implementing the best algorithm or MCTS architecture, the reward function design is not unique and is open to discussion. From our perspective, it is also valuable to point out that we did not award the remaining SOC, given that it could encourage the EV to go to a charging station to increase its SOC and obtain a higher reward. To avoid this, we penalized running out of battery and kept indifferent to the remaining SOC (indirectly, since we penalized the risk of running out of battery). In this regard, it is important to point out that many values were empirically obtained. A possible line of work is to design a reward function using Inverse RL (IRL) [124] with expert knowledge or historical data.

In this same topic, it is important to remember the use of auxiliary variables to improve the efficiency of the search. The use of these tools is usually manually tuned and therefore require the user to spend time finding their values. A very attractive field is to automate this process or switching to *auxiliar variables-free* implementations, to eliminate the need of an expert to handle these parameters and facilitar the extension of this methodology to other instances and domains. This idea is also applicable to the values of the *Reward* function's parameters.

6.7.3. Use of voltage-based battery models

In battery-powered systems, a cutoff voltage is set as a safety measure to protect the asset from overdischarges, which could cause severe damage [125]. Researchers have studied how, when a significant amount of power is required by the system, the voltage drops enough to trigger the cutoff procedure and disconnects the battery, even though there is still latent energy [126, 127]. This phenomenon has been modeled and represents the battery's behavior more accurately, but it comes at a price: computational cost and a specific application.

1. Computational cost: In early versions of this work, we included a battery model from [126], and it increased the computational effort hugely, mainly caused by the prognostic stage where a particle filter-based algorithm was used. It caused this implementation to be unable to run thousands of simulations fast enough on our hardware.

2. Specific application: The model used in this research for energy consumption can be found in several works. Thus, it allows the reader to make an easier comparison with other results in the literature. In this same direction, our work does not aim to represent a specific problem but to show how the EVRP can be solved with several relevant characteristics in a real-time algorithm. Notwithstanding this, in applications or case studies where a specific battery model is used, it would be of great interest to incorporate it in this algorithm design.

6.7.4. Offline stage

Finally, there could also be an offline stage where extensive computational resources could be available and used to solve the problem with several features and variations. This stage's output could serve as an initial guess in an online stage, particularly in contingencies

situations, or several initial solutions from the offline stage could be provided to reduce the search-space. This possibility could significantly reduce the computational cost of this implementation.

Chapter 7

Conclusion

This thesis presents a Prognostic Decision-Making approach to solving the Electric Vehicle Fleet Routing Problem. The problem formulation includes several features to increase applicability in real-life scenarios, such as time-windows for clients, stochasticity and dynamism in travel time and energy consumption, non-linear charging function and partial recharges.

A real-time algorithm based on Monte Carlo simulations was developed and successfully solved the Fleet EVRP. Different settings and parameters were tested to study their performance and convergence to the optimal solution. The computational cost was also addressed in relative terms since our implementation did not aim to be efficient, and better coding and other practices could reduce the computational effort several orders of magnitude. Due to computational resources, the fleet case was less deeply analyzed than the case with one EV, however, results can be straightforwardly extended.

It is possible to notice how Monte Carlo Tree Search can effortlessly include the Health Aware Decision Making guidelines in its design, and how they are also incorporated into the solution formulation. For example, through the exploration and simulation capabilities of future trajectories in the MCTS, and the heavy penalties on undesired states (such as violating the SOC threshold) in the formulation. We hope this pushes the use of these tools in this family of problems.

Our proposed solution was tested online and was able to manage contingency scenarios that directly affected the initial planning. Due to the algorithm's design, new information was immediately incorporated in the *Reward* function after every step execution, allowing a better decision-making procedure.

7.1. Future Work

Many elements can still be included in some aspects of this work. In the architecture dimension of this work, such as variants of the MCTS algorithm, parallelization [128, 129, 130], continuous MCTS [131, 132] and others. In uncertainty treatment, future efforts could focus on characterizing events or backpropagating a more informative metric. Finally, on the EVRP side, setting maximum hours allowed per tour; specific loads for each client; multiple depots; uncertainty in clients' demands or charging stations; different vehicles, power consumption and road models; backhauls cases; a detailed multi-agent analysis; and specific routing scenarios such as waste management, rescue missions, surveillance tasks and other applications, would be of great interest to study.

In terms of methods, the use of Neural-based models is also very scarce in approaches

for this problem [28]. As an extension of this work, we could implement a Deep Reinforcement Learning algorithm to handle larger maps, include more EVs and extract additional information from each simulation.

Extensions of this work could involve different routing problems whose particular constraints should modify those employed here. An example is the incipient problem of Urban Air Mobility (UAM) [133]. The proper use of future information goes beyond routing planning. Many problems with a Prognostic Decision-Making procedure can be in the scope of future efforts from this work, such as those mentioned earlier in biomedical fields, industrial problems, cybersecurity, logistics and others.

Bibliography

[1] J. Wang, Q. Wu, J. Liu, H. Yang, M. Yin, S. Chen, P. Guo, J. Ren, X. Luo, W. Linghu, and Q. Huang, "Vehicle emission and atmospheric pollution in China: Problems, progress, and prospects," *PeerJ*, vol. 7, pp. 1–22, 2019.

[2] International Energy Agency, "Global Energy Review 2021," *Global Energy Review 2020*, pp. 1–36, 2021.

[3] U. Epa and C. Change Division, "Inventory of U.S. Greenhouse Gas Emissions and Sinks: 1990-2019 – Main Text - Corrected Per Corrigenda, Updated 05/2021," 1990.

[4] Z. Li, A. Khajepour, and J. Song, "A comprehensive review of the key technologies for pure electric vehicles," *Energy*, vol. 182, pp. 824–839, 2019.

[5] C. Li, M. Negnevitsky, X. Wang, W. L. Yue, and X. Zou, "Multi-criteria analysis of policies for implementing clean energy vehicles in China," *Energy Policy*, vol. 129, no. February 2019, pp. 826–840, 2019.

[6] P. Jochem, E. Szimba, and M. Reuter-Oppermann, "How many fast-charging stations do we need along European highways?," *Transportation Research Part D: Transport and Environment*, vol. 73, no. May, pp. 120–129, 2019.

[7] Z. Guo, J. Deride, and Y. Fan, "Infrastructure planning for fast charging stations in a competitive market," *Transportation Research Part C: Emerging Technologies*, vol. 68, pp. 215–227, 2016.

[8] M. Günther, C. Kacperski, and J. F. Krems, "Can electric vehicle drivers be persuaded to eco-drive? A field study of feedback, gamification and financial rewards in Germany," *Energy Research and Social Science*, vol. 63, no. May 2019, p. 101407, 2020.

[9] S. Á. Funke, F. Sprei, T. Gnann, and P. Plötz, "How much charging infrastructure do electric vehicles need. A review of the evidence and international comparison," *Transportation Research Part D: Transport and Environment*, vol. 77, no. October, pp. 224–242, 2019.

[10] N. Wang, L. Tang, and H. Pan, "A global comparison and assessment of incentive policy on electric vehicle promotion," *Sustainable Cities and Society*, vol. 44, no. November 2018, pp. 597–603, 2019.

[11] S. Pelletier, O. Jabali, and G. Laporte, "Goods distribution with electric vehicles: Review and research perspectives," *Transportation Science*, vol. 50, pp. 3–22, Feb. 2016.

[12] C. C. Chan, "The state of the art of electric, hybrid, and fuel cell vehicles," *Proceedings of the IEEE*, vol. 95, no. 4, pp. 704–718, 2007.

[13] J. Moore and N. Bullard, "Bnef executive factbook—power, transport, buildings and industry, commodities, food and agriculture, capital," *London: BloombergNEF*, 2021.

[14] N. O. Kapustin and D. A. Grushevenko, "Long-term electric vehicles outlook and their potential impact on electric grid," *Energy Policy*, vol. 137, p. 111103, 2020.

[15] B. Nykvist and M. Nilsson, "Rapidly falling costs of battery packs for electric vehicles," *Nature Climate Change 2014 5:4*, vol. 5, pp. 329–332, 3 2015.

[16] K. Liu, K. Li, Q. Peng, and C. Zhang, "A brief review on key technologies in the battery management system of electric vehicles," *Frontiers of Mechanical Engineering*, vol. 14, no. 1, pp. 47–64, 2019.

[17] D. Pevec, J. Babic, A. Carvalho, Y. Ghiassi-Farrokhfal, W. Ketter, and V. Podobnik, "A survey-based assessment of how existing and potential electric vehicle owners perceive range anxiety," *Journal of Cleaner Production*, vol. 276, p. 122779, 2020.

[18] R. A. Hanifah, S. F. Toha, and S. Ahmad, "Electric vehicle battery modelling and performance comparison in relation to range anxiety," *Procedia Computer Science*, vol. 76, pp. 250–256, 2015.

[19] D. N. T. How, M. A. Hannan, M. S. Hossain Lipu, and P. J. Ker, "State of charge estimation for lithium-ion batteries using model-based and data-driven methods: A review," *IEEE Access*, vol. 7, pp. 136116–136136, 2019.

[20] J. Jiang, W. Shi, J. Zheng, P. Zuo, J. Xiao, X. Chen, W. Xu, and J.-G. Zhang, " Optimized Operating Range for Large-Format LiFePO 4 /Graphite Batteries ," *Journal of The Electrochemical Society*, vol. 161, no. 3, pp. A336–A341, 2014.

[21] E. Fan, L. Li, Z. Wang, J. Lin, Y. Huang, Y. Yao, R. Chen, and F. Wu, "Sustainable recycling technology for li-ion batteries and beyond: Challenges and future prospects," *Chemical Reviews*, vol. 120, no. 14, pp. 7020–7063, 2020. PMID: 31990183.

[22] E. Balaban, S. B. Johnson, and M. J. Kochenderfer, "Unifying system health management and automated decision making," *Journal of Artificial Intelligence Research*, vol. 65, pp. 487–518, 2019.

[23] B. L. Ferrell, "Air vehicle prognostics & health management," *IEEE Aerospace Conference Proceedings*, vol. 6, pp. 145–146, 2000.

[24] E. Balaban and J. J. Alonso, "An approach to prognostic decision making in the aerospace domain," *Proceedings of the Annual Conference of the Prognostics and Health Management Society 2012, PHM 2012*, pp. 396–415, 2012.

[25] E. Balaban, S. Narasimhan, M. J. Daigle, I. Roychoudhury, A. Sweet, C. Bond, J. R. Celaya, and G. Gorospe, "Development of a mobile robot test platform and methods for validation of prognostics-enabled decision making algorithms," *International Journal of Prognostics and Health Management*, vol. 4, no. 1, pp. 1–19, 2013.

[26] E. Balaban, S. Narasimhan, M. Daigle, J. Celaya, I. Roychoudhury, B. Saha, S. Saha, and K. Goebel, "A mobile robot testbed for prognostics-enabled autonomous decision making," in *Annual Conference of the PHM Society*, vol. 3, 2011.

[27] O. Bougacha, C. Varnier, and N. Zerhouni, "A Review of Post-Prognostics Decision-Making in Prognostics and Health Management," *International Journal of Prognostics and Health Management*, vol. 11, no. 15, p. 31, 2020.

[28] J. Mańdziuk, "New Shades of the Vehicle Routing Problem: Emerging Problem Formulations and Computational Intelligence Solution Methods," *IEEE Transactions on*

Emerging Topics in Computational Intelligence, vol. 3, no. 3, pp. 230–244, 2019.

[29] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers and Industrial Engineering*, vol. 99, pp. 300–313, 2016.

[30] S. Gholipour, A. Ashoftehfard, and H. Mina, "Green supply chain network design considering inventory-location-routing problem: a fuzzy solution approach," *International Journal of Logistics Systems and Management*, vol. 35, no. 4, pp. 436–452, 2020.

[31] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy," *Computer Science Review*, vol. 40, may 2021.

[32] J. J. Yoon, "The traveling salesman problem with multiple drones: an optimization model for last-mile delivery," Master's thesis, Massachusetts Institute of Technology, 2018.

[33] G. D. Konstantakopoulos, *Optimization Methods Applied in a Class of Vehicle Routing and Scheduling Problems*. PhD thesis, National Technical University of Athens, Athens, Greece, 2022.

[34] D. Feillet, "A tutorial on column generation and branch-and-price for vehicle routing problems," *4OR*, vol. 8, pp. 407–424, 2010.

[35] J. P. Futalef, D. Munoz-Carpintero, H. Rozas, and M. Orchard, "An evolutionary algorithm for the electric vehicle routing problem with battery degradation and capacitated charging stations," *Proceedings of the Annual Conference of the Prognostics and Health Management Society, PHM*, vol. 12, no. 1, 2020.

[36] G. Barbarosoglu and D. Ozgur, "A tabu search algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 26, no. 3, pp. 255–270, 1999.

[37] F. Arnold and K. Sörensen, "Knowledge-guided local search for the vehicle routing problem," *Computers & Operations Research*, vol. 105, pp. 32–46, 2019.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[39] Á. Felipe, M. T. Ortuño, G. Righini, and G. Tirado, "A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges," *Transportation Research Part E: Logistics and Transportation Review*, vol. 71, pp. 111–128, 2014.

[40] A. Gunawan, A. T. Widjaja, P. Vansteenwegen, and V. F. Yu, "Adaptive large neighborhood search for vehicle routing problem with cross-docking," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2020.

[41] V. C. Hemmelmayr, K. F. Doerner, and R. F. Hartl, "A variable neighborhood search heuristic for periodic routing problems," *European Journal of Operational Research*, vol. 195, no. 3, pp. 791–802, 2009.

[42] M. Bruglieri, F. Pezzella, O. Pisacane, and S. Suraci, "A variable neighborhood search branching for the electric vehicle routing problem with time windows," *Electronic Notes in Discrete Mathematics*, vol. 47, pp. 221–228, 2 2015.

[43] M. G. Resende and C. C. Ribeiro, *Greedy Randomized Adaptive Search Procedures:*

Advances, Hybridizations, and Applications, pp. 283–319. Boston, MA: Springer US, 2010.

[44] P. H. V. Penna, A. Subramanian, and L. S. Ochi, "An iterated local search heuristic for the heterogeneous fleet vehicle routing problem," *Journal of Heuristics*, vol. 19, pp. 201–232, 2013.

[45] C. Barletta, W. Garn, C. Turner, and S. Fallah, "Hybrid fleet capacitated vehicle routing problem with flexible monte–carlo tree search," *International Journal of Systems Science: Operations & Logistics*, vol. 0, no. 0, pp. 1–17, 2022.

[46] J. Mańdziuk and M. Świechowski, "UCT in Capacitated Vehicle Routing Problem with traffic jams," *Information Sciences*, vol. 406-407, no. April 2017, pp. 42–56, 2017.

[47] J. Mańdziuk and C. Nejman, "Uct-based approach to capacitated vehicle routing problem," *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, vol. 9120, pp. 679–690, 2015.

[48] M. Nazari, A. Oroojlooy, M. Takáč, and L. V. Snyder, "RL for Solving the Vehicle Routing Problem," *Nips*, no. NeurIPS, pp. 9861–9871, 2018.

[49] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, pp. 1–15, 2017.

[50] S. M. Raza, M. Sajid, and J. Singh, "Vehicle routing problem using reinforcement learning: Recent advancements," in *Advanced Machine Intelligence and Signal Processing* (D. Gupta, K. Sambyo, M. Prasad, and S. Agarwal, eds.), (Singapore), pp. 269–280, Springer Nature Singapore, 2022.

[51] C. Lin, K. L. Choy, G. T. Ho, S. H. Chung, and H. Y. Lam, "Survey of Green Vehicle Routing Problem: Past and future trends," *Expert Systems with Applications*, vol. 41, no. 4 PART 1, pp. 1118–1138, 2014.

[52] T. Erdelic, T. Carić, and E. Lalla-Ruiz, "A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches," *Journal of Advanced Transportation*, vol. 2019, 2019.

[53] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte carlo tree search: a review of recent modifications and applications," *Artificial Intelligence Review*, 2022.

[54] J. Oxenstierna, *Warehouse Vehicle Routing using Deep Reinforcement Learning*. PhD thesis, Department of Information Technology, Uppsala University, 2019.

[55] J. Mańdziuk and M. Świechowski, "Simulation-based approach to vehicle routing problem with traffic jams," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2016.

[56] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan, "Rich vehicle routing problem: Survey," *ACM Computing Surveys*, vol. 47, 12 2014.

[57] R. Bai, X. Chen, Z.-L. Chen, T. Cui, S. Gong, W. He, X. Jiang, H. Jin, J. Jin, G. Kendall, J. Li, Z. Lu, J. Ren, P. Weng, N. Xue, and H. Zhang, "Analytics and machine learning in vehicle routing research," *International Journal of Production Research*, vol. 61, no. 1, pp. 4–30, 2023.

[58] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall,

3 ed., 2010.

[59] C. C. Bennett and K. Hauser, "Artificial intelligence framework for simulating clinical decision-making: A Markov decision process approach," *Artificial Intelligence in Medicine*, vol. 57, no. 1, pp. 9–19, 2013.

[60] C. Lee, S. M. Han, Y. H. Chae, and P. H. Seong, "Development of a cyberattack response planning method for nuclear power plants by using the Markov decision process model," *Annals of Nuclear Energy*, vol. 166, p. 108725, 2022.

[61] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[62] T. Morimura, K. Ota, K. Abe, and P. Zhang, "Policy gradient algorithms with monte-carlo tree search for non-markov decision processes," 2022.

[63] F. Bacchus, C. Boutilier, and A. J. Grove, "Rewarding behaviors," in *Proceedings of the 13th AAAI Conference on Artificial Intelligence (AAAI-1996)*, pp. 1160–1167, 1996.

[64] F. Bacchus, C. Boutilier, and A. J. Grove, "Structured solution methods for non-markovian decision processes," in *AAAI/IAAI*, 1997.

[65] A. Ez-Zizi, S. Farrell, and D. Leslie, "Bayesian reinforcement learning in markovian and non-markovian tasks," in *2015 IEEE Symposium Series on Computational Intelligence*, pp. 579–586, 2015.

[66] E. A. EMERSON, "Chapter 16 - temporal and modal logic," in *Formal Models and Semantics* (J. VAN LEEUWEN, ed.), Handbook of Theoretical Computer Science, pp. 995–1072, Amsterdam: Elsevier, 1990.

[67] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith, "Decision-making with non-markovian rewards: From LTL to automata-based reward shaping," in *Proceedings of the Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM-17)*, pp. 279–283, 2017. See also University of Toronto Technical Report CSRG-632.

[68] R. Nian, J. Liu, and B. Huang, "A review on reinforcement learning: Introduction and applications in industrial process control," *Computers & Chemical Engineering*, vol. 139, p. 106886, 8 2020.

[69] M. Hensel, "Exploration methods in sparse reward environments," *Studies in Computational Intelligence*, vol. 883, pp. 35–45, 2021.

[70] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, (San Francisco, CA, USA), p. 463–471, Morgan Kaufmann Publishers Inc., 1998.

[71] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning 1988 3:1*, vol. 3, pp. 9–44, 8 1988.

[72] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[73] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, pp. 1140–1144, 12 2018.

[74] O. Trunda and R. Barták, "Using monte carlo tree search to solve planning problems in transportation domains," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8266 LNAI, pp. 435–449, 2013.

[75] J. V. Eyck, J. Ramon, F. Guiza, G. MeyFroidt, M. Bruynooghe, and G. V. d. Berghe, "Guided monte carlo tree search for planning in learned environments," in *Proceedings of the 5th Asian Conference on Machine Learning* (C. S. Ong and T. B. Ho, eds.), vol. 29 of *Proceedings of Machine Learning Research*, (Australian National University, Canberra, Australia), pp. 33–47, PMLR, 13–15 Nov 2013.

[76] X. Liu and A. Fotouhi, "Formula-e race strategy development using artificial neural networks and monte carlo tree search," *Neural Computing and Applications*, vol. 32, pp. 15191–15207, 9 2020.

[77] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), p. 3338–3346, MIT Press, 2014.

[78] J.-M. Horcas, J. A. Galindo, R. Heradio, D. Fernandez-Amoros, and D. Benavides, "Monte carlo tree search for feature model analyses: A general framework for decision-making," in *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A*, SPLC '21, (New York, NY, USA), p. 190–201, Association for Computing Machinery, 2021.

[79] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai," in *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'08, p. 216–217, AAAI Press, 2008.

[80] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. V. den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for monte-carlo tree search," *New Mathematics and Natural Computation*, vol. 04, pp. 343–357, 11 2008.

[81] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4630 LNCS, pp. 72–83, 2007.

[82] T. Vodopivec, S. Samothrakis, and B. Ŝter, "On monte carlo tree search and reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 60, pp. 881–936, 2017.

[83] J. C. Gittins and D. M. Jones, "A dynamic allocation index for the discounted multi-armed bandit problem," *Biometrika*, vol. 66, no. 3, pp. 561–565, 1979.

[84] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pp. 322–331, 1995.

[85] T. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances*

in Applied Mathematics, vol. 6, no. 1, pp. 4–22, 1985.

[86] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, p. 235–256, May 2002.

[87] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006* (J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, eds.), (Berlin, Heidelberg), pp. 282–293, Springer Berlin Heidelberg, 2006.

[88] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo Search," *White paper*, no. 1, p. 22, 2006.

[89] S. Gelly and Y. Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," in *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*, (Canada), Dec. 2006.

[90] M. P. Schadd, M. H. Winands, M. J. Tak, and J. W. Uiterwijk, "Single-player Monte-Carlo tree search for SameGame," *Knowledge-Based Systems*, vol. 34, pp. 3–11, 2012.

[91] M. P. D. Schadd, *Selective Search in Games of Different Complexity*. PhD thesis, Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands, 2011.

[92] E. Kaufmann and W. M. Koolen, "Monte-carlo tree search by best arm identification," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 4904–4913, Curran Associates Inc., 2017.

[93] T. Vodopivec and B. Šter, "Enhancing upper confidence bounds for trees with temporal difference values," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, 2014.

[94] Y. Shen, J. Chen, P.-S. Huang, Y. Guo, and J. Gao, "M-walk: Learning to walk over graphs using monte carlo tree search," in *NeurIPS*, 2018.

[95] P. Khandelwal, E. Liebman, S. Niekum, and P. Stone, "On the analysis of complex backup strategies in monte carlo tree search," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1319–1328, PMLR, 20–22 Jun 2016.

[96] T. Vodopivec and B. Branko˘brankošter, "Forgetting early estimates in monte carlo control methods," *ELEKTROTEHNIŠKIELEKTROTEHNI˘ELEKTROTEHNIŠKI VESTNIK*, vol. 82, pp. 85–92, 2015.

[97] F. Xie and Z. Liu, "Backpropagation modification in monte-carlo game tree search," in *2009 Third International Symposium on Intelligent Information Technology Application*, vol. 2, pp. 125–128, 2009.

[98] C. Fiori, K. Ahn, and H. A. Rakha, "Power-based electric vehicle energy consumption model: Model development and validation," *Applied Energy*, vol. 168, pp. 257–268, 2016.

[99] W. Edwardes and H. Rakha, "Virginia tech comprehensive power-based fuel consumption model," *Transportation Research Record*, vol. 2428, no. 312, pp. 1–9, 2014.

[100] J. Eschmann, "Reward function design in reinforcement learning," *Studies in Computational Intelligence*, vol. 883, pp. 25–33, 2021.

[101] M. Schneider, A. Stenger, and D. Goeke, "The electric vehicle-routing problem with

time windows and recharging stations," *Transportation Science*, vol. 48, pp. 500–520, 11 2014.

[102] B. Alves, "Electricity prices worldwide 2021." https://www.statista.com/statistics/263492/electricity-prices-in-selected-countries/ last accessed 10 Nov. 2022.

[103] Z. Bnaya, A. Felner, D. Fried, O. Maksin, and S. E. Shimony, "Repeated-task canadian traveler problem," *AI Communications*, vol. 28, pp. 453–477, 1 2015.

[104] N. Zerbel and L. Yliniemi, "Multiagent monte carlo tree search," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, (Richland, SC), p. 2309–2311, International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[105] E. Galván-López, R. Li, C. Patsakis, S. Clarke, and V. Cahill, "Heuristic-based multiagent monte carlo tree search," in *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, pp. 177–182, 2014.

[106] M. Crosby, A. Jonsson, and M. Rovatsos, "A single-agent approach to multiagent planning," in *Proceedings of the Twenty-First European Conference on Artificial Intelligence*, ECAI'14, (NLD), p. 237–242, IOS Press, 2014.

[107] S. Gelly and D. Silver, "Combining Online and Offline Knowledge in UCT," in *International Conference of Machine Learning*, (Corvallis, United States), June 2007.

[108] B. B. Max-Planck, "Monte carlo go." *Unpublished*, 1993.

[109] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[110] C. Xiao, R. Huang, J. Mei, D. Schuurmans, and M. Müller, "Maximum entropy monte-carlo planning," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[111] W. Mao, K. Zhang, Q. Xie, and T. Basar, "POLY-HOOT: monte-carlo planning in continuous space mdps with non-asymptotic analysis," *CoRR*, vol. abs/2006.04672, 2020.

[112] H. Baier and M. H. M. Winands, "Time management for monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 301–314, 2016.

[113] E. J. Powley, P. I. Cowling, and D. Whitehouse, "Memory bounded monte carlo tree search," in *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'17, AAAI Press, 2017.

[114] T. Keller and P. Eyerich, "Prost: Probabilistic planning based on uct," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 22, pp. 119–127, May 2012.

[115] J. Duguépéroux, A. Mazyad, F. Teytaud, and J. Dehos, "Pruning playouts in monte-carlo tree search for the game of havannah," in *Computers and Games* (A. Plaat, W. Kosters, and J. van den Herik, eds.), (Cham), pp. 47–57, Springer International Publishing, 2016.

[116] B. Bouzy, "Move-pruning techniques for monte-carlo go," in *Advances in Computer*

Games (H. J. van den Herik, S.-C. Hsu, T.-s. Hsu, and H. H. L. M. J. Donkers, eds.), (Berlin, Heidelberg), pp. 104–119, Springer Berlin Heidelberg, 2006.

[117] Y. Zhuang, S. Li, T. V. Peters, and C. Zhang, "Improving monte-carlo tree search for dots-and-boxes with a novel board representation and artificial neural networks," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 314–321, 2015.

[118] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," Research Report RR-6062, INRIA, 2006.

[119] J. Keller, *Improving MCTS and Neural Network Communication in Computer Go*. PhD thesis, Worcester Polytechnic Institute, 2016.

[120] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," *ArXiv*, vol. abs/2011.00583, 2020.

[121] L. Canese, G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences 2021, Vol. 11, Page 4948*, vol. 11, p. 4948, 5 2021.

[122] O. Thakoor, J. Garg, and R. Nagi, "Multiagent uav routing: A game theory analysis with tight price of anarchy bounds," *IEEE Transactions on Automation Science and Engineering*, vol. 17, pp. 100–116, 1 2020.

[123] D. Lenz, T. Kessler, and A. Knoll, "Tactical cooperative planning for autonomous highway driving using monte-carlo tree search," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, pp. 447–453, 2016.

[124] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 8 2021.

[125] H. Maleki and J. N. Howard, "Effects of overdischarge on performance and thermal stability of a li-ion cell," *Journal of Power Sources*, vol. 160, pp. 1395–1402, 10 2006.

[126] C. Díaz, V. Quintero, A. Pérez, F. Jaramillo, C. Burgos-Mellado, H. Rozas, M. E. Orchard, D. Sáez, and R. Cárdenas, "Particle-filtering-based prognostics for the state of maximum power available in lithium-ion batteries at electromobility applications," *IEEE Transactions on Vehicular Technology*, vol. 69, pp. 7187–7200, 7 2020.

[127] H. Rozas, D. Troncoso-Kurtovic, C. P. Ley, and M. E. Orchard, "Lithium-ion battery state-of-latent-energy (sole): A fresh new look to the problem of energy autonomy prognostics in storage systems," *Journal of Energy Storage*, vol. 40, p. 102735, 8 2021.

[128] T. Cazenave and N. Jouandeau, "On the Parallelization of UCT," in *Computer Games Workshop*, (Amsterdam, Netherlands), June 2007.

[129] T. Cazenave and N. Jouandeau, "A parallel monte-carlo tree search algorithm," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5131 LNCS, pp. 72–80, 2008.

[130] G. M. Chaslot, M. H. Winands, and H. J. V. D. Herik, "Parallel monte-carlo tree search," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5131 LNCS, pp. 60–71, 2008.

[131] M. H. Lim, C. J. Tomlin, and Z. N. Sunberg, "Sparse tree search optimality guarantees

in pomdps with continuous observation spaces," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 4, pp. 4135–4142, 7 2020.

[132] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0c: Alpha zero in continuous action space," *ArXiv*, vol. abs/1805.09613, 2018.

[133] W. N. Chan, B. E. Barmore, J. L. Kibler, P. U. Lee, C. J. O'Connor, K. Palopo, D. P. Thipphavong, and S. J. Zelinski, "Overview of nasa's air traffic management-exploration (atm-x) project," in *AIAA Aviation Forum 2018*, no. ARC-E-DAA-TN57276, 2018.

Annex

Annex A. Theoretical Background

A.1. Temporal Difference Learning

Eq.2.2 is derived as follows:

The TD update is based on the *TD error*, which is the difference between the ultimate correct reward $V(s_t^*)$ and the current prediction $V(s_t)$.

$$\delta_t = V(s_t^*) - V(s_t), \tag{A.1}$$

We also know that the reward at time t is the sum of discounted reward from *t+1* onwards.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{A.2}$$

As such, and taking into account that $V(s_t^*) = G_t$, Eq.A.1 can be expressed and developed as follows:

$$\begin{aligned}
\delta_t &= V(s_t^*) - V(s_t) \\
&= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} - V(s_t) \\
&= R_{t+1} + \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} - V(s_t) \\
&= R_{t+1} + \gamma \left(\sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k+1} \right) - V(s_t) \\
&= R_{t+1} + \gamma \left(\sum_{k=0}^{\infty} \gamma^{k} R_{(t+1)+k+1} \right) - V(s_t) \\
&= R_{t+1} + \gamma V(s_{t+1}) - V(s_t)
\end{aligned} \tag{A.3}$$

Finally, the update rule can be written as follows:

$$\begin{aligned}
V(s_t) &\leftarrow V(s_t) + \alpha \delta_t \\
&\leftarrow V(s_t) + \alpha \left[R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right],
\end{aligned} \tag{A.4}$$

which is Eq.2.2.

Annex B. Results

B.1. Best solution and hyperparameters' selection

The average reward obtained after running ten iterations with each setup can be seen in the following table:

Table B.1: Consistency results for the first set of parameters.

Parameters		C_{exp}	
		0	**0.025**
D	**0.725**	-0.063308	-0.063308
	0.75	-0.063308	-0.063308
	0.775	-0.063308	-0.063308
	0.975	-0.063308	-0.17057
	1	-0.063308	-0.158068
	1.025	-0.104403	-0.202498