
Politecnico di Torino

**Politecnico di Torino**

Master of Science in ICT for Smart Societies
Academic Year 2021/2022
December 2022

# Reinforcement Learning for the scheduling of EV charging systems in a Smart Grid context

Supervisors:

Prof. Michela Meo
Prof. Daniela Renga

Candidate:

Felipe Spoturno Bareño

**Abstract**

In the last decades global warming has aroused increasing attention as new temperature records are being registered worldwide and new violent and unpredictable meteorological events occur. Italy has just registered the second hottest summer in the last 200 years and different countries have registered the hottest temperatures ever. Climatologists agree that these temperatures are five times more probable due to climate change and they are caused by the increasing amount of $CO_2$ and other greenhouse effect gases released on the atmosphere due to human activity. Organizations at national and international level are constantly studying and monitoring the production of greenhouse effect gases and for both Europe and the USA it emerges that in the last years the domestic transportation sector is responsible for about 22-27% of emissions, and that the energy production sector is responsible for about 25% of emissions.

If combined, different enabling technologies that has been developed in the last decades can give an enormous contribution to the reduction of emissions produced by domestic transportation. These technologies include: the improvement of the quality and capacity of electric batteries and their use in electric vehicles, the increased efficiency and cost reduction of photo-voltaic panels for local energy production, the improvement of ICT and electrical infrastructure for the operation of Smart Grids, and the development of new computational algorithms able to control complex systems under uncertainty in a reasonable time.

This work aims at studying different algorithms for scheduling the recharge of batteries in a Battery Swapping Station (BSS) which are stations in which electric vehicles drop their empty battery, and pick a fully charged one to use. These algorithms take into account that the station may be provided a small photo-voltaic plant for local energy production, and that it may be embedded in a Smart Grid in which the cost of electricity varies during the day. The aim of the control algorithm is to reduce as much as possible both the electricity consumption from the grid, and the number of cars that cannot be served due to the lack of charged batteries in the station considering that there is a natural trade-off between these two performance indicators.

The proposed strategy, based on Reinforcement Learning techniques, introduces improvements on all performance indicators with respect to the previously proposed Heuristic approach. In particular, at approximately the same level of electricity costs, this algorithm is able to reduce car losses by about 20%, to reduce the cost per service approximately up to 5%, and provides the possibility to easily change the operating point through a single hyper-parameter that weights the trade-off between costs and losses.

# Contents

# Part I
# Introduction and context

The actual context and motivations for this work are presented in this chapter. It will be shown how the domestic transportation sector is highly responsible of the actual climate crisis, and how the European Commission is designing policies for a transition towards a carbon-neutral economy.

From the group of technologies that will for sure take a big role in the transition towards a green economy, this work focuses on: Smart Grids, Electric Vehicles (EV) and Reinforcement Learning (RL). These technologies will be presented in Sec. 1.2.

Battery Swapping Stations (BSS) and their use in EV recharging systems will be presented in Sec. 2, introducing the benefits it brings in terms of costs to final users, to providers and to the environment.

Lastly, in Sec. 3 after giving a overall introduction to the problem, the specific objectives of this work will be introduced.

## 1 Environmental, economical, political and technological context

Global surface temperatures are increasing rapidly since the industrial revolution and the evidence strongly suggests that this is caused due to the greenhous gas emissions (GHG) produced by human activity [1]. Industrialized countries are the biggest producers of GHG emissions, Europe was in the third place after USA and China in 2015.

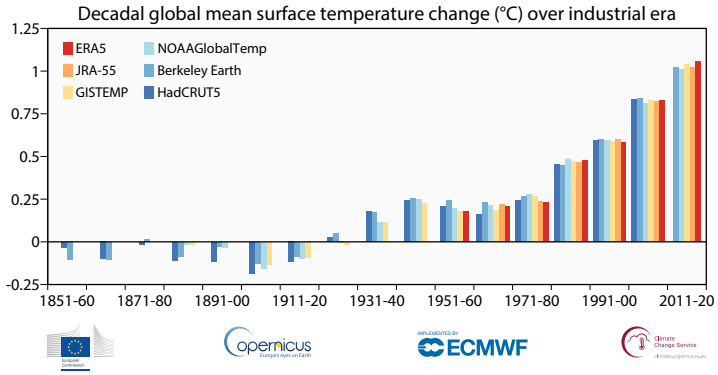


Figure 1: Decadal averages of global air temperature at a height of two meters estimated change since the pre-industrial period according to different datasets: ERA5 (ECMWF Copernicus Climate Change Service, C3S); GISTEMPv4 (NASA); HadCRUT5 (Met Office Hadley Centre); NOAAGlobalTempv5 (NOAA), JRA-55 (JMA); and Berkeley Earth. Credit: Copernicus Climate Change Service/ECMWF. [2]

Organizations at national and international level are monitoring and analyzing GHG emissions, and suggesting policies to implement in order to slow global warming, in particular to avoid reaching an increase of $2°C$ which might be extremely dangerous for the climate and for the environment, according to experts. Apart from industry, in Europe the most GHG emissions come from the energy supply and domestic transport sectors reaching around 50% of total emissions and even if efforts are being done by Europe, carbon neutrality is still far to be reached [3, 4].

Electrification of transport combined with renewable energy production provide a hope in the reduction of emissions both from energy supply and domestic transport, and a set of measures have been taken by the European Commission to boost the development of the industrial technologies that will lead this change.

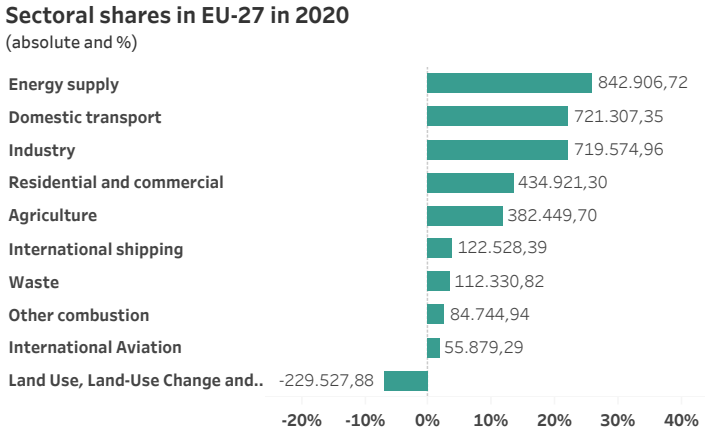


Figure 2: Data on greenhouse gas emissions and removals in ktCO2 eq., sent by countries to UNFCCC and the EU Greenhouse Gas Monitoring Mechanism (EU Member States). [5]

## 1.1 European green deal

To overcome the challenges of the climate change and the degradation of the environment, the European Commission presented in December 2019, what is called the "European Green Deal". Since then, different panels have been opened to discuss on how to face different aspects of the climate crisis, in particular, on March 2020 it was presented the proposal for an European Climate Law that would write into a law the objectives of the European Green Deal with the ambition of granting a "climate-neutral economy" by 2050 [6]. More precisely the objectives of the law are:

- Set the long-term direction of travel for meeting the 2050 climate neutrality objective through all policies, in a socially fair and cost-efficient manner
- Set a more ambitious EU 2030 target, to set Europe on a responsible path to becoming climate-neutral by 2050

- Create a system for monitoring progress and take further action if needed
- Provide predictability for investors and other economic actors
- Ensure that the transition to climate neutrality is irreversible

With transport producing around 25% of EU's GHG [7], the European Green Deal sets the objective for nearly all EU cars, vans, buses and heavy-duty vehicle to be zero-emission by 2050. On this direction, the EU Commission has already approved a ban on the production and selling of new fuel cars from 2035 [8]. Different working groups inside the EU Commission together with external stakeholders are proposing academic and industrial researches and partnerships through the Horizon Europe Research and Innovation funding programme [9] with the objective of boosting the development of knowledge and technologies needed to face the green transition.

## 1.2 Enabling technologies

In order to meet the goals of the European Commission on the reduction of GHG emissions and in particular to have a more green transportation, different technologies that have been developed in the last decades may be combined to produce novel integrated and efficient systems. In particular when talking about Information and Communication Technologies (ICT) the term "enabling technologies" often emerges, to refer to the fact that these technologies have the potential to integrate disciplines that traditionally where segregated and not part of the information domain allowing to design novel, distributed and complex systems able to solve problems in a way that would not be possible before. This work focuses on the combination of the following technologies, from different domains, that integrated through ICT may strongly contribute to the reduction of GHG emissions. These technologies are Electric Vehicles (EV), Smart grids, and Reinforcement Learning (RL).

### 1.2.1 Electric Vehicles

The idea of using electric-powered vehicles is not new, and the principles of work for our modern AC industrial motors have been proposed through different patents by Nikola Tesla back to 1887. Instead, what is enabling the developing and explosion of the electric-mobility market, is the developing of new technologies and means of production for batteries, now capable to provide the autonomy requested by urban transportation at an affordable price. Between 2010 and 2015, electric Li-ion batteries price have more than halved and their production is growing with an exponential trend [10].

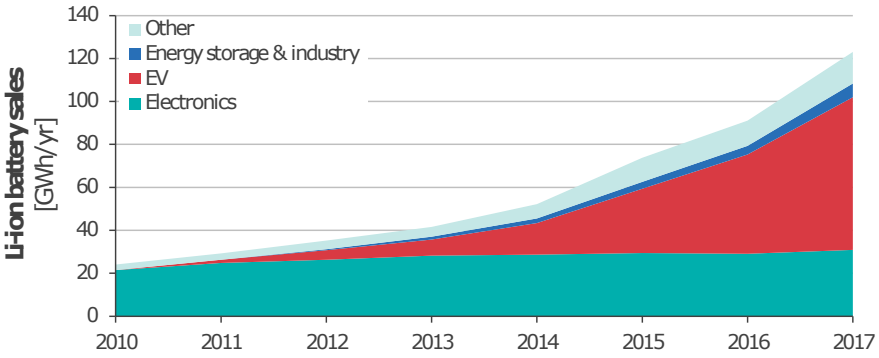


Figure 3: Global historical annual growth Li-ion batteries in main market segments.

However, electric transportation by itself cannot solve the problem of GHG emissions if the electrical energy used by EVs do not come from renewables. Given that renewable energies depend on natural phenomena that are not under human control, and given that electrical production and consumption must match each other perfectly (due to electrical balance equations), there is the need of introducing mechanisms able to manage these constraints. The next Section will focus on how ICT can be used on Smart Grids to produce and store clean energy usable by EVs.

### 1.2.2 Smart Grids

Electric energy distribution emerged as a field of engineering back to 1880s when energy started being produced in power stations. Since then, energy has been produced in mass by plants far away from consumers, which are located mostly in cities or industrial areas. These plants (apart from hydroelectric production) have often been fossil-fuel powered. In the last century different technologies maturated enough to allow to produce clean energy from wind and solar, and in particular at micro-scale. Since electrical energy must be consumed or stored at the exact moment at which it is being produced, there must be a coordination between producers and consumers. A naive solution to the problem, often used in isolated production systems is to turn on the power plant when needed, or modulating the production rate to match the energy demand. Moreover, considering the fact that renewable energies (at mass and micro scale production levels) are produced by natural phenomena that are not under human control, it may be convenient to have a mechanism able to exploit energy surpluses when available. Here ICT comes into place, transforming traditional electrical grids into what is called a Smart Grid [11].

Smart Grids introduce several mechanisms that allow to have in the same electrical infrastructure mass production energy plants, micro-scale producers, and consumers to match their needs collaborating thanks to ICT by exchanging information. Demand Side Management (DSM) mechanisms aim at modifying consumers' demand profile to match the supply, and in particular Demand Response (DR) mechanisms aim at rescheduling the consumers'

profile in time. DR is a particular strategy of DSM, and this work focuses on using a price-based approach in which the price of the electrical energy varies during the day to meet the renewables production profile and electrical infrastructure constraints. For example, given a Smart Grid provided with a great amount of solar energy production in which a price-based DR is implemented, this Smart Grid would set lower electricity prices at the middle of the day when solar production is at its maximum and would set higher electricity prices during the night [11, 12].

To schedule the recharging of EVs batteries in a station embedded in a Smart Grid, implies to consider the price of electricity, the expected power production from local renewable sources (if available), and the rate of arrival for cars, variables that vary during the day. The computational complexity of optimizing such a system can only be faced through Reinforcement Learning, a mathematical optimization tool that will be introduced in the next Section.

### 1.2.3 Reinforcement Learning

This work focuses mainly on the design and implementation of optimization techniques for the scheduling of EVs charging systems. When these systems are embedded in a Smart Grid, and local renewable energy production is available, the number of variables to take into account make practically impossible to compute an exact solution to the problem.

Reinforcement Learning (RL) is a recent branch of computer science gathering ideas that come from different fields of engineering such as control theory, operational research, and artificial intelligence [14]. The technical aspects about RL will be introduced in detail on Sec. 6, its objective is to solve optimization problems in which decisions are made in stages, while the system evolves being influenced by the decision that was made at the beginning of the stage, plus a set of external stochastical variables. As others disciplines under the umbrella of Machine Learning, RL has strongly evolved in the last years thanks to the increased computational capacity and reduction of costs of electronic components and processors, allowing to solve complex problems in a reasonable time. Also the development of ICT technologies and the possibility to coordinate distributed and communicating computational resources is at the core of the new wave of development of RL strategies, since one of its main contributions is the development of control strategies for distributed (or multi-agent) systems, in which for example, remote measurements are obtained through low-power sensors, decisions are computed in a centralized powerful computer, and then actions are sent back through the network to low-power actuators. The most remarkable result of RL in the last years is the development of AlphaGo Zero, a self-trained computer program capable of playing and winning chess matches against top-level (human) professional players [15].

# 2 Battery Swapping Station (BSS)

Battery Swapping Stations are systems in which EVs can drop an empty battery to pick a fully charged one. These systems avoid the recharging waiting time that traditional EVs

recharging stations have, at the price of implementing a more complex infrastructure. BSS are mostly useful for fleet of public transport in a city (taxis, buses) in which EVs must always be ready to provide the service. As regulations and standards in electric mobility evolve, more companies are developing technologies for making battery swapping an actual alternative to traditional charging stations, mostly for the micro-mobility segment (electric bikes and motorcycles) [16].

Figure 4: Promotional image from Gogoro for a BSS [16].

## 2.1 Architecture of the BSS

This work studies a BSS embedded in a Smart Grid context, and particularly focuses on the design of control algorithms for scheduling the recharging of batteries. The main components present in the systems are the following (see Fig. 5):

- **Electrical energy sources**: in order to recharge batteries energy is obtained from a local photo-voltaic (PV) plant, and from the electrical Smart Grid in which electrical price varies during the day. Eventually, PV energy surpluses may be sold to other consumers through the Smart Grid.
- **Charging sockets**: batteries in the system are charged by sockets that can be turned on and off by a centralized Agent. They can also measure the state of charge of the batteries connected to it.
- **Agent**: it is the brain of the system and the main object of study of this work. It takes as input the information for the forecasts for the energy cost, solar production and batteries demand, and also it gets the actual state of the system (the state of charge of the batteries), to produce a set of commands that decide which sockets must be turned on and which turned off.

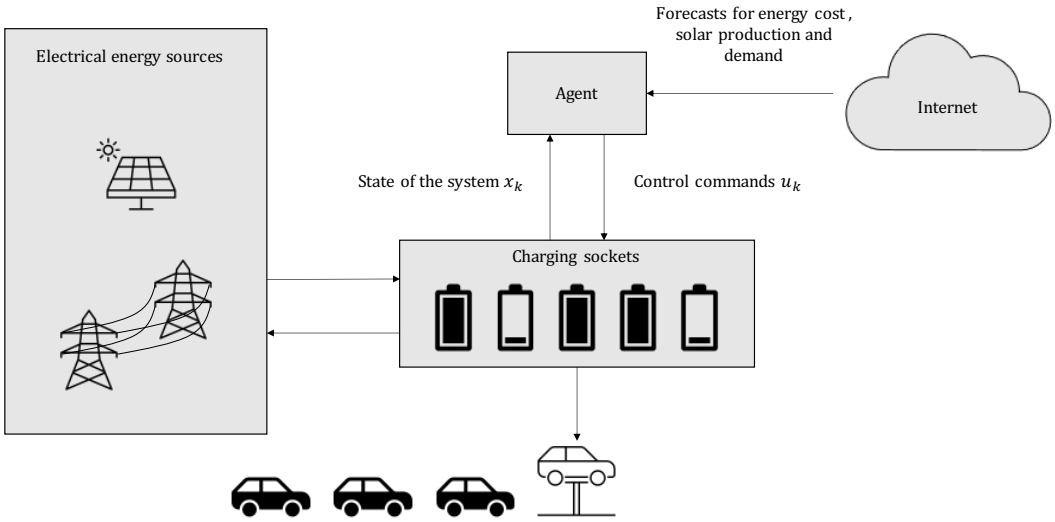


Figure 5: Battery Swapping Station under study.

## 2.2 Battery-as-a-service (BaaS) model

Battery Swapping introduces also some benefits to final consumers in terms of costs, with the boosting of a new business model called Battery-as-a-service (BaaS). In the traditional EV market, each EV is provided with a fixed battery, limiting users with the time it takes to refill the battery as already mentioned, but also increasing enormously the initial and maintenance cost of the Electric Vehicle [17].

In the BaaS model, the final user its not the owner of the EV's battery, since the owner becomes a service provider company. By using a BaaS provider, the user gets access to BSS stations in which it can exchange an empty battery with a fully charged one when needed, without having any ownership on them. BaaS reduces then both the Capital Expenditure (CapEx) of the final consumer since the battery is not acquired by him, and the Operating Expenditure (OpEx) since the maintenance cost of the batteries becomes a responsibility of the service provider. On the other hand, a BaaS model can also benefit service providers giving them the opportunity to give a second-life to aged batteries improving their economical incomes and further reducing the environmental impacts of the overall transportation system [19, 20, 21].

# 3 Objectives of this work

As previously mentioned the aim of this work is to design a scheduling algorithm for batteries within a Battery Swapping Station, to reduce both the car losses due to the lack of charged batteries and to reduce the electrical costs incurred by buying energy from the electrical

grid. In particular, this work aims at:

- **Formalize the recharge scheduling of batteries as a mathematical optimization problem:** previous researchers have introduced some Heuristic algorithms that will be introduced in Sec. 7.1. This work aims at formalizing the recharge scheduling of batteries as an optimization problem able to represent the trade-off between electrical costs and car losses.
- **Design and implement an adaptive algorithm:** after formalizing the recharge scheduling of batteries as an optimization problem, this work aims at exploring different optimization techniques, considering that the resulting algorithm must adapt to the changing external conditions of the system (arrival rates of cars, electrical costs, and photo-voltaic energy production).
- **Improvement of performance:** the proposed algorithm should introduce improvements in terms of performance (costs and losses) with respect to the previously proposed Heuristic algorithm.
- **Controlling the working point of the system through hyper-parameters:** the proposed algorithm should be able to provide a mechanism to easily tune its working point and to choose the trade-off between costs and losses.
- **Introduce a benchmarking mechanism for future developments:** this work aims also at providing a method for benchmarking algorithms in terms of electrical cost and car losses, enabling future researchers to objectively compare the performance of different scheduling algorithms.

# Part II
# Mathematical background

As stated on the previous Part the aim of this work is to formalize the recharge scheduling of batteries in the BSS as an optimization problem. In this Part Dynamic Programming will be introduced as a mathematical tool for minimizing the accumulated cost-function of a Markovian Decision Process (MDP). DP find exact solution for a set of MDP, but it may become computationally infeasible for problems in which there are too many variables in the system. This problem is also known as the curse of dimensionality. To overcome it DP will be extended presenting what is called Approximated Dynamic Programming, a series tools for reducing the computational complexity of DP at the price that solutions become sub-optimal. Afterwards, Reinforcement Learning (RL) be introduces as another tool that aims at further reducing the computational complexity for large problems. These are the basis for understanding the mathematical models and algorithms presented in Part III for optimizing the scheduling of the BSS.

## 4 The mathematical framework of Stochastic Dynamic Programming

### 4.1 The basic problem

Stochastic Dynamic Programming (DP) is a mathematical tool that aims at controlling stochastical dynamic systems in which decisions are made in stages (or time steps). The objective is to minimize a certain cost that is accumulated at each time step, and that depends both on the applied control and on the evolution of the system. The mathematical model under study is also known as a Markovian Decision Process (MDP).

The basic model for DP is composed by two main features [21]:

1. An underlying discrete-time dynamic system

$$x_{k+1} = f_k\left(x_k, u_k, \omega_k\right) \tag{1}$$

   where $k = 0, 1, \ldots, N-1$ indexes discrete time, $x_k$ is the state of the system, $u_k$ is the control or decision variable to be selected at time $k$, $\omega_k$ is a random variable on the evolution of the system and its cost, and finally $N$ is the time-horizon of the system.

2. A cost function that is additive over time

$$\sum_{k=0}^{N-1} g_k\left(x_k, u_k, \omega_k\right) + g_N\left(x_N\right) \tag{2}$$

where $g_k(x_k, u_k, \omega_k)$ is the accumulated cost at each time step (also known as step cost-function), while $g_N(x_N)$ is a terminal cost incurred at the end of the process, thus it depends only on the final state. Note that due to the presence of $\omega_k$ the total cost is generally a random variable.

The evolution of the discrete system and the accumulation of cost dynamics can be seen in a summarized way on Fig. 6.

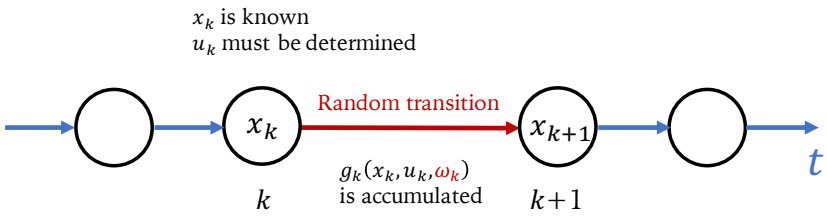


Figure 6: Mathematical framework for DP.

The objective of DP is to arrive to a policy $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ where $\mu_k$ maps states $x_k$ into controls $u_k = \mu_k(x_k)$ in such a way that the accumulated cost is minimized. Under this policy, the evolution of the system becomes:

$$x_{k+1} = f_k(x_k, \mu_k(x_k), \omega_k) \tag{3}$$

the associated expected value for the accumulated cost of the system, under the policy $\pi$, starting from a generic initial state $x_0$ is defined as:

$$J_\pi(x_0) := \mathbb{E}\left[\sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), \omega_k) + g_N(x_N)\right] \tag{4}$$

An optimal policy $\pi^*$ is then a policy that minimizes $J$ over the set of all the possible policies $\Pi$, that is:

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0) \tag{5}$$

## 4.2 Principle of optimality and the DP Algorithm

Dynamic Programming relies on a very simple idea, the principle of optimality [21]. The name has been assigned by Bellman who contributed a great deal to the popularization of DP.

**Principle of optimality:** Let $\pi^* = \left\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\right\}$ be an optimal policy for the basic problem, and assume that when using $\pi^*$, a given state $x_i$ occurs at time $i$ with positive probability. Consider the subproblem whereby we are at $x_i$ at time $i$ and wish to minimize the "cost-to-go" from time $i$ to time $N$:

$$\mathbb{E}\left[\sum_{k=i}^{N-1} g_k\left(x_k, \mu_k\left(x_k\right), \omega_k\right) + g_N\left(x_N\right)\right]$$

Then the truncated policy $\left\{\mu_i^*, \mu_{i+1}^*, \ldots, \mu_{N-1}^*\right\}$ is optimal for this subproblem.

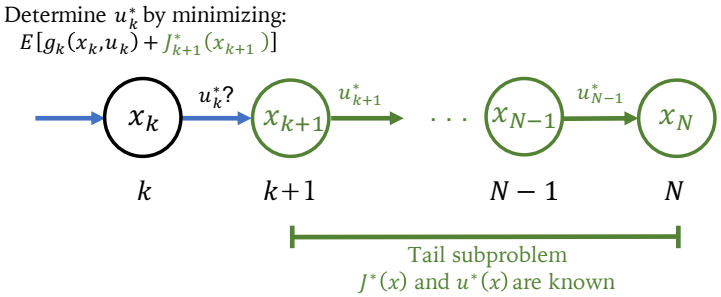


Figure 7: Principle of optimality. At time $k$ if the tail sub-problem is solved $J_{k+1}^*(x)$ and $u_{k+1}^*(x)$ are perfectly known so $J_k^*(x)$ and $u_k^*(x_k)$ can be computed. The solution to the problem can be constructed backwards through the DP Algorithm.

The principle of optimality suggest that an optimal policy can then be constructed by solving the "tail subproblem" involving the last stage, then extending stage by stage the tail subproblem until constructing an optimal policy for all the problem. This strategy of building the solution backwards in time is known as the Dynamic Programing Algorithm.

**Dynamic Programming Algorithm:** For every initial state $x_0$, the optimal cost $J^*(x_0)$ of the basic problem is equal to $J_0(x_0)$, given by the last step of the following algorithm, which proceeds backwards in time from time $N-1$ to time 0:

$$J_N\left(x_N\right) = g_N\left(x_N\right)$$

$$J_k\left(x_k\right) = \min_{u_k}\left\{\mathbb{E}_{\omega_k}\left[g_k\left(x_k, u_k, \omega_k\right) + J_{k+1}\left(f_k\left(x_k, u_k, \omega_k\right)\right)\right]\right\}$$

Furthermore, if $u_k^* = \mu_k^*\left(x_k\right)$ minimizes the right side of the last equation for each $x_k$ and $k$, the policy $\pi^* = \left\{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\right\}$ is optimal.

In the DP Algorithm terminology $g_k\left(x_k, u_k, \omega_k\right)$ is called "step cost-function" at time $k$, while the term $J_{k+1}\left(f_k\left(x_k, u_k, \omega_k\right)\right)$ is the "cost-to-go" function [21].

## 4.3 Infinite Horizon DP

The problem of optimizing Eq. (2) is also known as "Finite horizon DP", because the number of stages is finite. Instead, it may be convenient in some cases, to state the problem in a different way. Infinite horizon DP problems [21] differ from those studied up to now in two respects:

1. The number of stages is infinite
2. The system is stationary (the system equation, the cost per stage, and the random disturbance statistics do not change from one stage to the next)

From the theoretical point of view these problems are conveniently classified as:

- **Stochastic shortest path problems:** In which there is a terminal cost-free state $x_T$ where the system remains blocked, thus, the problem has an horizon in practice, but it is given in the space-state and not in the time-space.
- **Discounted problems:** Problems in which the cost-per-stage exponentially decays with a factor $\alpha$ instead of being accumulated until the end of the problem. It will be explained in detail in Sec. 4.3.1.
- **Average cost per stage problems:** The function to minimize is the mean accumulated cost-per-stage until the end of the problem.

On this work the focus will be on studying discounted problems, which will be presented on the next section.

### 4.3.1 Discounted problems

Discounted problems are a particular subset of infinite horizon Dynamic Programming problems with a particular structure in which the function to minimize is written as:

$$J_\pi(x_0) := \lim_{N\to\infty} \mathbb{E}\left[\sum_{k=0}^{N-1} \alpha^k \cdot g_k\left(x_k, \mu_k(x_k), \omega_k\right)\right] \tag{6}$$

here the final cost $g_N(\cdot)$ is not present since now the system will never reach a final stage, and the cost-per-stage decays exponentially with a factor of $\alpha$. In order to ensure the convergence of this equation [21] a sufficient condition is to have a bounded cost-per-stage function $|g_k(x,u,\omega)| < M$, so that $J_\pi(x_0)$ is bounded by the decreasing geometric progression $\left\{\alpha^k M\right\}$.

### 4.3.2 Value Iteration

Since now the system is stationary, and the concept of time vanishes, the Dynamic Programming algorithm as stated in Sec. 4.2 must be modified to take a new form that is called "Value iteration" (VI) algorithm:

$$J_{k+1}(x_i) = \min_u \left\{ g(x_i, u) + \alpha \cdot \sum_{j=1}^{n} p_{ij} \cdot J_k(f(x_j, u)) \right\}, \text{ for each } i \tag{7}$$

where $k$ is now the iteration for computing the cost-to-go function, $i$ and $j$ are indexes for the states, and $p_{ij}$ are the transition probabilities [21]. It can be proved that VI converges to the optimal cost-to-go function, and in the convergence point $J^*(x)$, in fact, the system verifies what is known as the Bellman's equation:

$$J^*(x_i) = \min_u \left\{ g(x_i, u) + \alpha \cdot \sum_{j=1}^{n} p_{ij} \cdot J^*(x_j) \right\} \tag{8}$$

VI is then an algorithm used to solve a functional equation for $J^*(x)$.

# 5 Approximate Dynamic Programming

## 5.1 The curse of dimensionality

Solving a Dynamic Programming problem analytically is feasible and useful only for some particular cases in which the system is very simple or with a limited amount of stages, and the solution can be easily interpreted. Often DP is only used to produce a numerical solution to a problem, but still in most scenarios, the computational complexity of solving it through exact DP becomes unfeasible. The reason lies in what Bellman called the "curse of dimensionality" which refers to a rapid increase of the required computation and memory storage as the size of the problem increases.

Relaxing the constrains, simplifying the base problem, reducing the state-space, are examples of strategies that can be used to reduce the complexity of solving the exact DP, giving place to what is called Approximate DP, leading to sub-optimal solutions to the problem [21].

## 5.2 Some Approximate DP Strategies

Approximate DP strategies are grouped in two main categories:

- **Approximation in value space:** where the cost-to-go function is approximated to a simpler function $\widetilde{J}$ which reduces the complexity of the problem. And then the optimal control can be computed as:

$$u_k^* = \underset{u_k}{\operatorname{argmin}} \left\{ \widetilde{J}_k(x_k) \right\} \tag{9}$$

- **Approximation in policy space:** the optimal control is chosen from a suitable class of policies based on some criterion; the selection process often uses data, optimization, and parametrical models such as neural networks. The optimal control can e computed as:

$$u_k^* = f\left(x_k; \theta\right) \tag{10}$$

Often both methods can be combined, specially when policy space approximations are used. In this work the focus will be on using approximations in value space, in particular, the strategy explained in Sec. 5.2.1.

### 5.2.1 Problem approximation (Decoupling)

Problem approximation allows to reduce the complexity of the exact DP algorithm by introducing simplifications to the original problem. On this work problem approximation will be introduced through the decoupling of variables in the studied system, to reduce both the state and the control spaces. Note that if a system is composed by $M$ entities, each of which having $\#x$ possible states and capable of applying $\#u$ controls, then the state-space and the control-space on which the optimization must be performed, have cardinalities of $\#x^M$ and $\#u^M$ respectively. Instead, by decoupling the system and treating each entity as a stand-alone unit, the state-space and the control-space for each one of the $M$ optimizations, have cardinalities of $\#x$ and $\#u$ which is enormously smaller. Using decoupling as an approximation to the system is then a convenient approach when dealing with multi-agent systems.

Let the system state, the control variable, and the random variable for the system evolution be written as vectors $\mathbf{x_k}$, $\mathbf{u_k}$ and $\mathbf{w_k}$. In that case, using the Dynamic Programming Algorithm implies solving:

$$J_k^*\left(\mathbf{x_k}\right) = \min_{\mathbf{u_k}} \left\{ \mathop{\mathbb{E}}_{\mathbf{w_k}} \left[ g_k\left(\mathbf{x_k}, \mathbf{u_k}, \mathbf{w_k}\right) + J_{k+1}^*\left(\mathbf{x_{k+1}}\right) \right] \right\} \tag{11}$$

Decoupling of the system allows to split the problem into its $M$ components. In that case it will be written a sub-problem for each component $i$, as following:

$$\mathbf{J_k^{*(i)}}\left(\mathbf{x_k}^{(i)}\right) = \min_{\mathbf{u_k}^{(i)}} \left\{ \mathop{\mathbb{E}}_{\mathbf{w_k}} \left[ g_k^{(i)}\left(\mathbf{x_k}^{(i)}, \mathbf{u_k}^{(i)}, \mathbf{w_k}\right) + \mathbf{J_{k+1}^{*(i)}}\left(\mathbf{x_{k+1}}^{(i)}\right) \right] \right\} \tag{12}$$

as shown in the equation this implies to be able to write a single step cost-function $g_k^{(i)}\left(\cdot\right)$ and a single cost-to-go function $\mathbf{J_{k+1}^{*(i)}}\left(\cdot\right)$ for each component of the system.

# 6 Model-based Reinforcement Learning

## 6.1 From Dynamic Programming to Reinforcement Learning

Dynamic Programming and Approximate DP assume the knowledge of a control table used to get the optimal control to apply to the system. Then, an off-line training generates a

control table, and the on-line algorithm will just apply the control.

**Algorithm 1** DP and ADP Algorithms basic structure.

1. Before starting, compute the mapping control table for each $k$ and $x$ of the system (or an approximation of it) using the Dynamic Programming Algorithm or a variation of it like (e.g. value iteration).
2. Once the system starts, at each instant of time $k$:
   (a) Get the optimal control to apply $u_k^*$ from the pre-computed look-up table given $k$ and $x_k$.
   (b) Apply the control $u_k^*$

Model-based Reinforcement Learning (RL) introduces strategies in which the model of the system is known, as in DP, but the computation of the solution is not made entirely off-line, because it might be infeasible due to the dimensions of the problem [22]. RL relies on the model of the system to interact with it to recommend an action, so RL strategies are basically on-line algorithms. In Sec. 6.2 and Sec. 6.3 two algorithms will be introduced that allow to perform an exploration of the control space to suggest a sub-optimal control strategy for the system.

**Algorithm 2** Model-based RL Algorithms basic structure

1. Before starting, run off-line algorithms to pre-compute/learn/train some models that will be used on-line
2. Once the system starts, at each instant of time $k$:
   (a) Interact with the off-line trained models to get a sub-optimal control $\widetilde{u_k}$
   (b) Apply the sub-optimal control $\widetilde{u_k}$

Model-based Reinforcement Learning algorithms will try to minimize on an on-line basis the following problem:

$$u_k^* = \underset{u_k}{\operatorname{argmin}} \left\{ Q_k \left( x_k, u_k, \omega_k \right) \right\} = \underset{u_k}{\operatorname{argmin}} \left\{ \underset{\omega_k}{\mathbb{E}} \left[ J_k \left( x_k, u_k, \omega_k \right) \right] \right\} \tag{13}$$

here $Q_k(\cdot)$ are functions called $Q$-values, that are nothing but the expected value of the cost-to-go function used in classical DP. Each one of the operations shown in the equation: $\operatorname{argmin}\{\cdot\}$, $\underset{\omega_k}{\mathbb{E}}[\cdot]$, $J_k(\cdot)$, may be computed through different approximation methods, giving place to a wide variety of algorithms that can be used in different combinations for the final optimization.

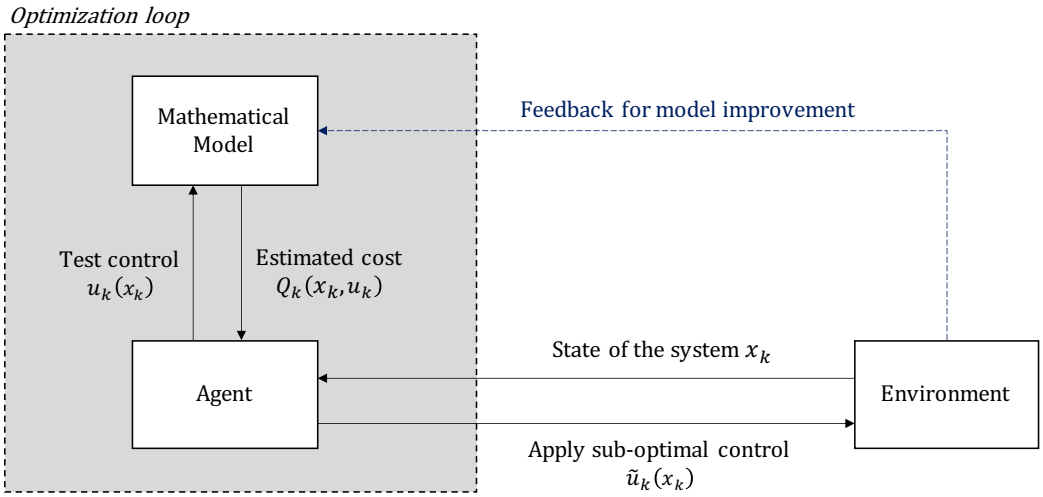


Figure 8: Model-based Reinforcement Learning operating scheme.

## 6.2 One-step lookahead and Multi-step lookahead strategies

One-step and Multi-step lookahead are RL algorithms that explore the control space by interacting with a mathematical model for the evolution of the system and evaluating the associated cost-to-go function [14]. More precisely, given a current state $x_k$ of the system at time $k$, the algorithm tries different controls $u_k$ in search of the one that minimizes the cost-to-go.

In one-step lookahead strategies, the algorithm tries different controls evaluating what will be the immediate evolution of the system (one step) and its corresponding cost. In this case we can write Eq. (13) as:

$$u_k^* = \underset{u_k}{\operatorname{argmin}} \left\{ \underset{\omega_k}{\mathbb{E}} \left[ g_k \left( x_k, u_k, \omega_k \right) \right] + \widetilde{J}_{k+1} \left( x_{k+1} \right) \right\} \tag{14}$$

where it can be seen that the immediate impact of the control is evaluated through the expected value of the step cost-function, and the cost associated to the future state $x_{k+1}$ of the system is approximated through an "approximate cost-to-go function". Another way to visualize this is through the scheme on Fig. 9.

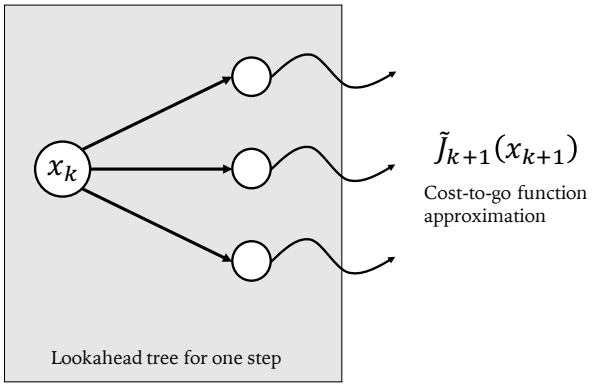


Figure 9: Deterministic one-step lookahead algorithm. Starting from $x_k$, different controls are applied leading to new states $x_{k+1}$ from which the cost-to-go is computed through approximation.

If the system is stochastic, then for each possible control variable, different possible states $x_{k+1}$ will be reached according to the value that $\omega_k$ will assume. This introduces some computational complexity to the exploration of the possible control to applies and is represented in Fig. 10.

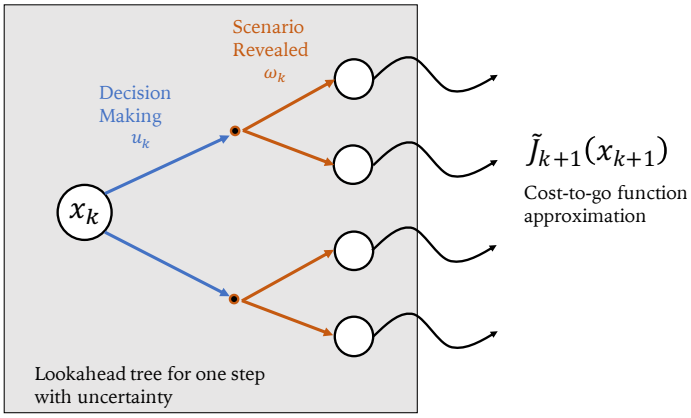


Figure 10: Stochastic one-step lookahead algorithm. Starting from $x_k$, different controls are applied and for each, different values of $\omega_k$ can be assumed, leading to new states $x_{k+1}$ from which the cost-to-go is computed through approximation.

The strategy of one-step lookahead gives the best solution for the problem considering only one step in the future. The algorithm can be generalized considering $l$-steps in the future, so Eq. (14) is modified in the following way:

$$u_k^* = \underset{u_k}{\operatorname{argmin}} \left\{ \sum_{i=k}^{k+l-1} \underset{\omega_i}{\mathbb{E}} \left[ g_i \left( x_i, u_i, \omega_i \right) \right] + \widetilde{J}_{k+l} \left( x_{k+l} \right) \right\} \tag{15}$$

note that even if the system is run for $l$ steps, the algorithm must be used at each time step to compute only the control $u_k^*$ at time $k$. This can be better visualized in Fig. 11.

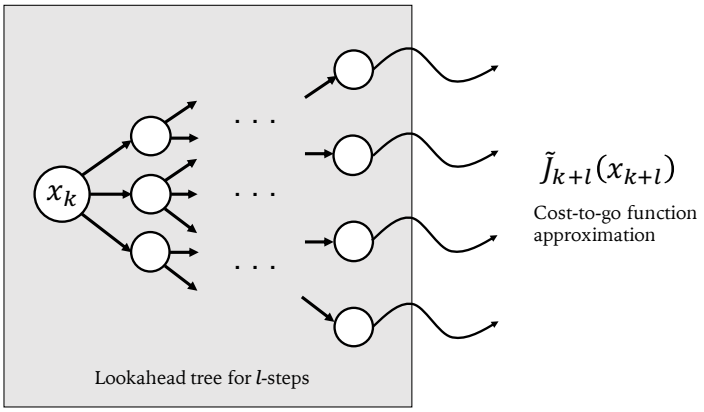


Figure 11: Deterministic $l$-step lookahead algorithm. Starting from $x_k$, different controls are applied leading to new states $x_{k+1}$. The algorithm is run for each new state until arriving to time $k+l$ from which the cost-to-go is computed through approximation.

### 6.2.1 Approximate cost-to-go functions

The computation of the cost-to-go function $\widetilde{J}_k \left( x_k \right)$ in lookahead strategies can be done through different methods [14]. Some of these methods are for example:

- Training a parametrical model (e.g. neural networks)
- Rollout-cost obtained by applying an heuristic (so much less computational complex algorithm) on the evolution of the system
- For discounted problems, the step cost-function may become negligible after $l$ forward steps so the cost-to-go function may be approximately set to zero
- Approximation of the cost-to-go obtained by solving a simpler problem

It will be shown later (see Sec. 10.3) that the method used in this work will be the last one. In particular, given a system that is composed by $M$ elements, its state becomes a vector $\mathbf{x_k}$, and if it can be approximately decomposed into $M$ sub-problems that can be solved through exact DP, then a way for computing $\widetilde{J}_k \left( \mathbf{x_k} \right)$ is:

$$\widetilde{J}_k \left( \mathbf{x_k} \right) = \sum_i \mathbf{J}_{\mathbf{k+1}}^{*(i)} \left( \mathbf{x_{k+1}}^{(i)} \right) \tag{16}$$

where $\mathbf{J}_{\mathbf{k+1}}^{*(i)}$ stands for the decoupled cost-to-go value for the $i$ component of the system obtained by applying the DP Algorithm off-line.

## 6.3 Multi-agent policy iteration

For systems composed by $M$ agents, capable to apply an independent control to the system [14, 23], the control variable becomes a vector with $M$ components:

$$\mathbf{u_k} = (u_1, u_2, \ldots, u_M) \tag{17}$$

If one intends to use, for example, one-step lookahead strategies to optimally control the system, one would need to try $\#u^M$ possible controls, where $\#u$ stands for the number of possible controls to apply. This may become infeasible when $M$ is large. Multi-agent policy iteration is an algorithm that aims at exploring a subset of the control space in linear time, by exploring $\mathbf{u_k}$ one component at a time. In that case Eq. (13) can be written in the following way:

$$\mathbf{u_k}^* = \text{seq. } \underset{\mathbf{u_k}}{\text{argmin}} \left\{ \underset{\omega_k}{\mathbb{E}} \left[ J_k \left( x_k, \mathbf{u_k}, \omega_k \right) \right] \right\} \tag{18}$$

where "seq. argmin" means that the minimization is performed by varying and selecting one component of $\mathbf{u_k}$ at a time. This means that instead of performing an all-at-once minimization, the procedure is to perform a one-agent-at-a-time optimization, in this case the space to explore becomes of dimension $M \cdot \#u$ which is linear. The algorithm proceeds in the following way:

**Algorithm 3** Multi-agent policy iteration algorithm.

1. At each instant of time $k$:
   (a) Initialize the optimal control as $\mathbf{u_k}^* := 0$
   (b) For a number of iterations $N_{IT}$ or until convergence:
      i. For each component $i$ of $\mathbf{u_k}$:
         A. Find a new $\mathbf{u_k}^{*(i)}$ by minimizing the cost in Eq. (18), where the rest of the components are kept fixed to $\mathbf{u_k}^*$.
         B. Update the component $i$ of the optimal control $\mathbf{u_k}^*$
   (c) Apply sub-optimal control $\mathbf{u_k}^*$.

# Part III
# BSS modeling and optimization

In this Part the basic variables, parameters and performance indicators of the system will be presented. On Sec. 7.2 the distribution for the arrival process of cars will be introduced, which is fundamental for the modeling of the system.

This work is entirely based on the usage of a BSS simulator that will be presented in Sec. 7 with the work that the research team has done so far, and the Heuristic algorithm previously proposed.

The approach used to setting up a model for the system, was to start from a very simplified version of it. In Sec. 9 there is a simple model for a station composed by one single socket, then in Sec. 10 there is a complete model for a generic amount of sockets. It will be shown that trying to solve the problem may become computationally infeasible when the number of sockets is high, so a series of modifications to the DP algorithm, based on Reinforcement Learning will be presented on Sec. 10.2 and Sec. 10.3.

| **Variable** | **Description** | **Set of values** |
|---|---|---|
| $M$ | Number of sockets in the station | $\mathbb{N}$ |
| $C$ | Capacity of a battery | $\mathbb{R}^+$ |
| $E_0$ | Maximum lack of energy accepted by a car when requesting a battery | $\mathbb{R}^+$ |
| $\Delta t$ | Discrete time step | $\mathbb{R}^+$ |
| $k$ | Time index | $\mathbb{N}$ |
| $N$ | Number of time steps within a day | $\mathbb{N}$ |
| $\lambda_k$ | Arrival rate of cars at $k$ | $\mathbb{R}^+$ |
| $p_k^{\lambda(n)}$ | Probability of arriving $\geq n$ cars at $k$ | $[0,1] \in \mathbb{R}$ |
| $\mathbb{E}\left(\omega_k^{PV}\right)$ | Expected energy from solar plant at $k$ | $\mathbb{R}^+$ |
| $\alpha$ | Discount factor hyper-parameter | $[0,1] \in \mathbb{R}$ |
| $\beta$ | Trade-off hyper-parameter | $[0,1] \in \mathbb{R}$ |
| $\mathbf{x_k}$ | Lack of energy in batteries at $k$ | $x^M = \{0, \cdots, C\}^M$ |
| $\mathbf{u_k}$ | Energy given to batteries at $k$ | $u^M = \{0, u_{max}\}^M$ (OFF/ON) |

Table 1: System parameters and variables.

The scope of the algorithm is to minimize the accumulated cost of the system during 24 hours (a whole day). The construction of the cost function will be presented on Sec. 8. As input, at the beginning of the day, the algorithm is provided with the predictions for the arrival rate, the solar energy production, and the cost of electricity from the grid, so it can compute a dynamic scheduling according on how events occur during the day.

# 7 The BSS Simulator

In order to study the behavior and performance of different control algorithms for the Agent on the system, an event-based simulator implemented on Python is used. This work continues the work done by previous students R. Monti [24] who build the first version of the simulator, and G. Centonze [25] who further improved the simulator and dimensioned a system for a real application. G. Centonze also designed some initial Heuristic algorithms for the control of the system that will be explained in Sec. 7.1. This works aims at formalizing the problem as an optimization problem, so algorithms will not be Heuristic but will find optimal and sub-optimal solutions for a formally written problem.

## 7.1 Heuristic control algorithm

The Heuristics algorithms already implemented on the simulator are based on postponing strategies. Each time that a battery is fully charged, or each time that a new battery arrives to the system the Agent will control if it is "convenient" to postpone the recharge of any battery. Checking for the convenience of postponing a battery recharge is done only if the solar panel is not producing energy, so the usage of the local produced energy is maximized. Pseudocodes for the heuristics algorithms are shown in Alg. 4 and Alg. 5.

**Algorithm 4** Postponing strategy control algorithm.

Given as input the actual time $t$, and the postponing time $T$:

1. Compute $PV\,(t)$ which is the actual PV production
2. If $PV\,(t) = 0$ and the number of postponed batteries is not greater than $F_{max}$:
   For each battery on the system:
   (a) If it is convenient to postpone the charge given $t$ and $T$ set a turning on timer at $t + T$

**Algorithm 5** PV production-based convenience algorithm.

Taking as input the actual time $t$ and a future time $t'$:

1. Compute $PV\left(t'\right)$ which is the expected PV production at time $t'$
2. If the level of charge for the battery is greater than $C/2$ :
   (a) Return $PV\left(t'\right) > 0$
3. Otherwise:
   (a) Compute $PV\left(t + 1\text{hour}\right)$ and $PV\left(t' + 1\text{hour}\right)$
   (b) Return $PV\left(t + 1\text{hour}\right) + PV\left(t\right) < PV\left(t' + 1\text{hour}\right) + PV\left(t' + 1\text{hour}\right)$

## 7.2 Cars arrival process

As in many queuing system, here it is assumed that the process of arrival for the cars respects the memory-less property. Assuming the memory-less property implies that the inter arrival times $\Delta T_\lambda$ can only follow an exponential distribution. This is partially true in the case of an EV charging station because the arrival rate for cars varies according to the moment of the day. The approximation will be to use and exponential distribution for the inter arrival times that slowly changes during the day.

$$\Delta T_\lambda\left(z, k\right) \sim \lambda_k \cdot e^{-\lambda_k \cdot z} \tag{19}$$

Since the computational optimization of the problem is done on a discrete-time basis, there is another important random variable for the model of the system that is strictly linked to the inter arrival times which is the number of cars arriving in a time window. For exponential inter arrival times, the stochastic counting process of arrivals can be proved to be a Poisson variable. Given a time window $\Delta t$ (the discrete time step of the system), the number of cars arriving to a station is computed as:

$$P\left\{\text{number of arrivals}_k = n\right\} = \frac{\left(\lambda_k \cdot \Delta t\right)^n \cdot e^{-\lambda_k \cdot \Delta t}}{n!} \tag{20}$$

the probability of arriving for more than, or equal, to $n$ cars at time step $k$ will be used during the modeling of the complete system, and it will be written as $p_k^{\lambda(n)}$. It can be computed as:

$$p_k^{\lambda(n)} = P\left\{\text{number of arrivals}_k \geq n\right\} = 1 - \sum_{i=0}^{n-1} \frac{\left(\lambda_k \cdot \Delta t\right)^i \cdot e^{-\lambda_k \cdot \Delta t}}{(i)!} \tag{21}$$

# 8 Key Performance Indicators (KPI) and step cost-function

Several metrics could be taken into consideration to evaluate the performance of the scheduling algorithm used in the system. The two top-level main indicators are the following:

1. **Mean electrical cost**: which includes the electrical cost incurred from buying electrical energy from the grid for charging the batteries.
2. **Mean car losses**: which includes the number of cars that are lost by the system due to the lack of batteries with a sufficient level of energy.

Other important indicators could be taken into consideration for the performance of the system, that may be more related to physical constraints for the station or for preserving batteries health. Some of them may be for example:

- Mean time of a battery in the system
- Mean number of ON-OFF switchings during a battery re-charging cycle
- Peak electrical power consumed by the station

In a scenario in which we consider only the mean electrical cost, and the mean car losses the step cost-cost function used for the Markov Decision Processes will be constructed with an hyper-parameter $\beta \in [0, 1]$ as following:

$$g_k = (1 - \beta) \cdot \text{Mean cost} + \beta \cdot \text{Mean car losses} \tag{22}$$

## 8.1 Benchmarking control algorithms, cost-losses diagram

As written in Eq. (22), there is a natural trade-off in trying to minimize both the mean cost and the mean car losses. In Sec. 10.1 it will be shown that considering only the Mean cost leads to a policy in which batteries are not charged (Always OFF algorithm), while by considering only the Mean car losses leads to a policy in which batteries are charged as soon as they arrive to the station with no interruptions (Always ON algorithm). These two policies set then a limit of performances since no algorithm can be better in terms of costs than the first one, and no algorithm can be better in terms of losses than the latter.

In Fig. 12 there is a diagram used to locate the performance of an algorithm in terms of its mean electrical cost and mean car losses, each algorithm in this diagram will be located on a single point. The ideal working point is to have the mean cost of an Always OFF, and the mean car losses of an Always ON, this is practically impossible to reach, but sets a basis to compare the performance of different control algorithms. Given a generic algorithm for which the cost function is written as in Eq. (22), as $\beta \to 1$ then its performance must approach the Always ON performance, and on the other hand as $\beta \to 0$ its performance must approach the Always OFF performance. While varying $\beta$ from 0 to 1, the trajectory

described by the performance on the cost-losses diagram will be a curve moving from Always OFF to Always ON. The best algorithms will have a curve that will be the closest possible to the ideal working point.

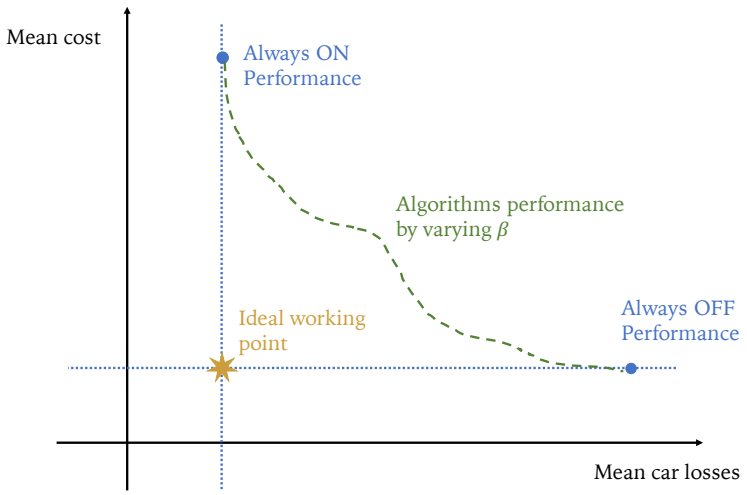


Figure 12: Cost-losses diagram for the benchmarking of control algorithms. No algorithm can perform better than Always OFF in terms of costs, and no algorithm can perform better than Always ON in terms of losses. This sets an ideal working point that is useful for comparing the performance of different algorithms while varying $\beta$ in their step cost-function.

# 9 Single-socket station modeling

The following model represents a single-socket station, without a waiting queue. A car would arrive and pick the battery under charge only if is charged to a certain threshold $E_0$, otherwise, it leaves the station without changing the battery.

The system is represented through a discrete-time and discrete state-space model. The time will be encoded through an integer number $k$, so that the time passing between $k$ and $k+1$ will be constant and equal to $\Delta t$ which will be the time granularity of the system. The state is encoded as the lack of energy of the battery in the socket at each time $k$, so $x_k = 0$ would be a fully charged battery, while $x_k = C$ represents an empty battery. The control variable is the charging energy $u_k$ that is being given to the battery at each slot of time (it could be both a binary variable on/off or a variable admitting different levels of charging powers).

The stochasticity of the system is given by two main random variables:

1. $\omega_k^\lambda$ which is a Bernoulli variable indicating the arrival of a car, with a time-variable probability distribution that can be simply represented as:

$$\omega_k^{\lambda} = \begin{cases} 1 \text{ (arrival)} & \text{with prob. } p_k^{\lambda} \\ 0 \text{ (no arrival)} & \text{with prob. } 1 - p_k^{\lambda} \end{cases} \tag{23}$$

2. $\omega_k^{PV}$ which represents the amount of energy that comes from the solar plant. This variable takes values from the same space of $x_k$ and $u_k$, and we assume to know at least, its expected value at each time step.

With this, the system evolves in time according to the following state transitions:

$$x_{k+1} = \begin{cases} C & \text{if } \omega_k^{\lambda} = 1 \text{ and } [x_k - u_k]^+ \leq E_0 \\ [x_k - u_k]^+ & \text{otherwise} \end{cases} \tag{24}$$

basically the state of the system is changed by charging the battery, and a completely empty battery is left on the system after a car arrival if it changes its battery. It can also be written in a summarized way as:

$$x_{k+1}\left(\omega_k^{\lambda}, u_k\right) = \omega_k^{\lambda} \cdot \mathbb{I}\left\{[x_k - u_k]^+ < E_0\right\} \cdot C + \left(1 - \omega_k^{\lambda} \cdot \mathbb{I}\left\{[x_k - u_k]^+ \leq E_0\right\}\right) \cdot [x_k - u_k]^+ \tag{25}$$

At each time step, the system accumulates a cost, that depends on the control and events of the system. The cost-function can be written as:

$$g_k\left(x_k, u_k, \omega_k^{\lambda}, \omega_k^{PV}\right) = (1 - \beta) \cdot a_k \cdot \left[u_k - \omega_k^{PV}\right]^+ + \beta \cdot \omega_k^{\lambda} \cdot \mathbb{I}\left\{[x_k - u_k]^+ > E_0\right\} \tag{26}$$

where $\mathbb{I}\{\cdot\}$ stands for the indicator function, $[\cdot]^+$stands for a function that allows having only positive values or 0, $a_k$ is the deterministic cost of the energy taken from the grid at time $k$, $\beta \in [0, 1]$ is an hyper-parameter used to weight the trade-off between the cost of the energy from the grid and the cost of loosing a car. The Dynamic Programming Algorithm can then be applied in order to minimize the operation cost of the system, on a time window that starts at $k = 0$ and ends at $k = N$ that could potentially be the first and last hour/minute/second of a day.

## 9.1 Finite-Horizon DP Algorithm

The Dynamic Programming Algorithm aims at minimizing the accumulated cost of the system. Since the cost-function is cumulative, the algorithm runs backwards in time trying to minimize at each step the cost-to-go function called $J_k(x_k)$, starting at $k = N$ and arriving to $k = 0$. In that sense it is important to define a final cost that will give the initial condition for the recurrent algorithm. In this case we can simply set $J_N(x_N) = 0$ (so every final state is equally penalized at the end of the day). The algorithm computes the optimal cost called $J_k^*(x_k)$ as:

$$J_k^*(x_k) = \min_{u_k} \left\{ \mathop{\mathbb{E}}_{\omega_k^\lambda, \omega_k^{PV}} \left[ g_k\left(x_k, u_k, \omega_k^\lambda, \omega_k^{PV}\right) + J_{k+1}^*(x_{k+1}) \right] \right\} \tag{27}$$

We can apply the algorithm to our system:

$$J_k^*(x_k) = \min_{u_k} \left\{ \mathop{\mathbb{E}}_{\omega_k^\lambda, \omega_k^{PV}} \left[ (1-\beta) \cdot a_k \cdot \left[u_k - \omega_k^{PV}\right]^+ + \omega_k^\lambda \cdot \beta \cdot \mathbb{I}\left\{[x_k - u_k]^+ > E_0\right\} + J_{k+1}^*(x_{k+1}) \right] \right\} \tag{28}$$

Since $\omega_k^\lambda$ and $\omega_k^{PV}$ are independent random variables, we can split the expected value arriving to:

$$J_k^*(x_k) = \min_{u_k} \left\{ \mathop{\mathbb{E}}_{\omega_k^{PV}} \left[ (1-\beta) \cdot a_k \cdot \left[u_k - \omega_k^{PV}\right]^+ \right] + \mathop{\mathbb{E}}_{\omega_k^\lambda} \left[ \omega_k^\lambda \cdot \beta \cdot \mathbb{I}\left\{[x_k - u_k]^+ > E_0\right\} + J_{k+1}^*(x_{k+1}) \right] \right\} \tag{29}$$

$$J_k^*(x_k) = \min_{u_k} \left\{ (1-\beta) \cdot a_k \cdot \left[u_k - \mathbb{E}\left(\omega_k^{PV}\right)\right]^+ + \mathbb{E}\left[\omega_k^\lambda\right] \cdot \beta \cdot \mathbb{I}\left\{[x_k - u_k]^+ > E_0\right\} + \mathop{\mathbb{E}}_{\omega_k^\lambda} \left[ J_{k+1}^*(x_{k+1}) \right] \right\} \tag{30}$$

and since $\omega_k^\lambda$ is a Bernoulli variable, then $\mathbb{E}\left[\omega_k^\lambda\right] = p_k^\lambda$ at each time step:

$$J_k^*(x_k) = \min_{u_k} \left\{ (1-\beta) \cdot a_k \cdot \left[u_k - \mathbb{E}\left(\omega_k^{PV}\right)\right]^+ + p_k^\lambda \cdot \beta \cdot \mathbb{I}\left\{[x_k - u_k]^+ > E_0\right\} + \mathop{\mathbb{E}}_{\omega_k^\lambda} \left[ J_{k+1}^*(x_{k+1}) \right] \right\} \tag{31}$$

Finally the last term can be written from Eq. 25 as:

$$\mathop{\mathbb{E}}_{\omega_k^\lambda} \left[ J_{k+1}^*(x_{k+1}) \right] = p_k^\lambda \cdot J_{k+1}^*\left(x_{k+1}(1, u_k)\right) + (1 - p_k^\lambda) \cdot J_{k+1}^*\left(x_{k+1}(0, u_k)\right) \tag{32}$$

From this, the algorithm can be computationally run, to optimize the scheduling of the charging of the battery knowing different parameters of the system that vary with time: $p_k^\lambda$, $\mathbb{E}\left(\omega_k^{PV}\right)$, and $a_k$. By the end of the "off-line" run of the algorithm, it would produce a lookup table that stores $(u_k^*, J_k^*)$ for each discrete state $x_k$, that can be then inspected "on-line" by the controller to find the optimal control to apply given the current state of the system.

| | $k=0$ | $k=1$ | $\cdots$ | $k=N$ |
|---|---|---|---|---|
| $x_0 = 0$ | $\left(u_0^*(x_0), J_0^*(x_0)\right)$ | $\left(u_1^*(x_0), J_1^*(x_0)\right)$ | $\cdots$ | 0 |
| $x_1$ | $\left(u_0^*(x_1), J_0^*(x_1)\right)$ | $\left(u_1^*(x_1), J_1^*(x_1)\right)$ | $\cdots$ | 0 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $C$ | $\left(u_0^*(C), J_0^*(C)\right)$ | $\left(u_1^*(C), J_1^*(C)\right)$ | $\cdots$ | 0 |

Table 2: Example of a control table after running the DP Algorithm.

## 9.2 Infinite-Horizon DP through value iteration

It will be shown on Sec. 11.4 that using Finite-Horizon DP on a daily basis may cause some misbehaviors as the end of the day approaches. In order to overcome this problem, Infinite-Horizon DP can be used.

Even if the system is not stationary in an hourly basis, it can be said that it is approximately stationary in a daily basis, since the solar radiation, car arrival rates, and electricity price may be similar from one day to the next one. Solving Infinite-Horizon DP through Value Iteration, as explained in Sec. 4.3.2, implies iterating to solve the Bellman equation, and selecting a discount factor $\alpha$ for the cost function, so Eq. (27) is simply modified in the following way:

$$J_k^*(x_k) = \min_{u_k} \left\{ \mathop{\mathbb{E}}_{\omega_k^\lambda, \omega_k^{PV}} \left[ g_k\left(x_k, u_k, \omega_k^\lambda, \omega_k^{PV}\right) + \alpha \cdot J_{k+1}^*(x_{k+1}) \right] \right\} \tag{33}$$

In order to properly implement Value Iteration on our system, the DP Algorithm is performed on a daily basis and once the beginning of the day is reached ($k = 0$) the accumulated cost $J_0^*(x)$ is set as the initial condition for a new DP Algorithm iteration setting $J_N^*(x) := J_0^*(x)$. This can also be visualized on Fig. 13. The iteration is performed for limited number of times, allowing the system to "forget" the very initial condition that was set on Sec. (9.1) that established $J_N^*(x) := 0$.

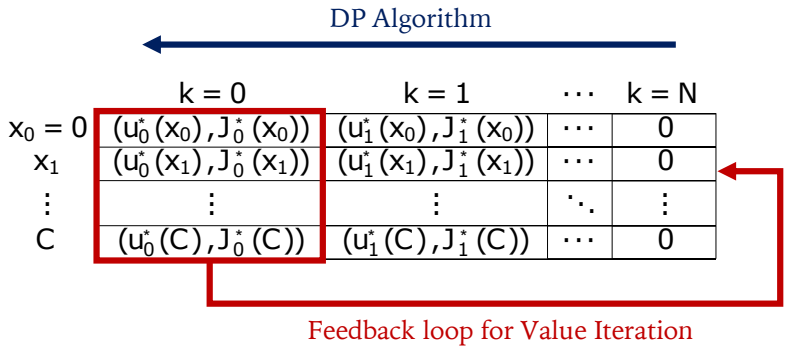


Figure 13: Implementation of Value Iteration for a daily-basis stationary system.

# 10 $M$-socket station modeling

An $M$-socket station can be thought exactly as a single-socket station in which the state and the control variables of the system are column vectors $\mathbf{x_k}$ and $\mathbf{u_k}$. Something that is convenient in that case is to order the state of energy demand of the batteries ($\mathbf{x_k}^{(i)}$, with $0 \leq i < M-1$) in an increasing way. So the state of the system would actually be $\mathbf{x'_k} = f(\mathbf{x_k})$ where $f$ is a permutation of the states. In this way $\mathbf{x'_k}^{(i)} \leq \mathbf{x'_k}^{(j)}$ for each $i \leq j$. Note that

in this way, the first element of the vector $\mathbf{x'_k}^{(0)}$ will be the most charged battery, and it is the battery that is going to be given to the customer in case of an arrival.

As an extension to the case of a single socket system, now the random variable $\omega_k^\lambda$ becomes a binary vector $\mathbf{w_k^\lambda}$ indicating the amount of cars (from 0 to $M$) arriving in the slot of time, so:

$$\mathbf{w_k^{\lambda(i)}} = \begin{cases} 1 \text{ (arrival of } \geq i+1 \text{ cars)} & \text{with prob. } p_k^{\lambda(i)} \\ 0 \text{ (arrival of } < i+1 \text{ cars)} & \text{with prob. } 1 - p_k^{\lambda(i)} \end{cases} \tag{34}$$

The evolution of the system and the cost of each step becomes:

$$\begin{aligned} \mathbf{x'_{k+1}} = C \cdot \mathbb{I}\left\{ f\left( [\mathbf{x'_k} - \mathbf{u'_k}]^+ \right) < E_0 \right\} \circ \mathbf{w_k^\lambda} \Big) \\ + \left[ \mathbf{1} - \mathbb{I}\left\{ f\left( [\mathbf{x'_k} - \mathbf{u'_k}]^+ \right) < E_0 \right\} \circ \mathbf{w_k^\lambda} \right]^T \cdot f\left( [\mathbf{x'_k} - \mathbf{u'_k}]^+ \right) \end{aligned} \tag{35}$$

$$g_k\left(\mathbf{x'_k}, \mathbf{u'_k}, \mathbf{w_k^\lambda}, \omega_k^{PV}\right) = (1-\beta) \cdot a_k \cdot \left[ \sum_i \mathbf{u'_k}^{(i)} - \omega_k^{PV} \right]^+ + \beta \cdot \mathbb{I}\left\{ f\left( [\mathbf{x'_k} - \mathbf{u'_k}]^+ \right) > E_0 \right\}^T \cdot \mathbf{w_k^\lambda} \tag{36}$$

where $\circ$ stands for the element-wise product (or Hadamard product), and $\mathbb{I}\{\cdot\}$ is applied to a vector verifying the condition for each component. Now the DP algorithm would minimize for each state, at each time step:

$$J_k^*\left(\mathbf{x'_k}\right) = \min_{\mathbf{u'_k}} \left\{ \mathop{\mathbb{E}}_{\mathbf{w_k^\lambda}, \omega_k^{PV}} \left[ g_k\left(\mathbf{x'_k}, \mathbf{u'_k}, \mathbf{w_k^\lambda}, \omega_k^{PV}\right) \right] + \mathop{\mathbb{E}}_{\mathbf{w_k^\lambda}} \left[ J_{k+1}^*\left( f\left( \mathbf{x'_{k+1}} \right) \right) \right] \right\} \tag{37}$$

This problem is computationally expensive since the number of solutions to explore for each time is exponential on $M$, involving at least $\#\mathbf{x'} \cdot \#u^M$ operations. The space for $\mathbf{x'}$ is composed by all the sorted combinations for $x$, which is:

$$\#\mathbf{x'} = \begin{pmatrix} \#x + M - 1 \\ M \end{pmatrix} \tag{38}$$

For example for $\#x = 25$, $\#u = 2$ (for an on/off controller) and $\#M = 20$, this would imply at least $\sim 1.8 \cdot 10^{18}$ operations per time step for the construction of a Dynamic Programming lookup table. Also, there is the problem of the memory needed to hold all the actions to take for each one of the states.

Instead of solving the complete table produced by the DP Algorithm which is clearly infeasible, to reduce the computational complexity of the optimization, a series of strategies will be applied and will be introduced in the following sections.

## 10.1 Exact policy solutions (Always OFF and Always ON algorithms)

The problem can be exactly solved for the cases in which $\beta = 0$ or $\beta = 1$, and it can directly be seen by writing down the explicit cost sum. In the case $\beta = 0$ on the cost function only

the electrical cost is considered and accumulated, so the discounted cost becomes:

$$J\left(x_k\right)_{\beta=0} = \lim_{N\to\infty} \sum_{k=0}^{N-1} \alpha^k \cdot a_k \cdot \left[\sum_i \mathbf{u'_k}^{(i)} - \mathbb{E}\left[\omega_k^{PV}\right]\right]^+ \tag{39}$$

the sequence inside the sum is always positive, and can be minimized if $\mathbf{u'_k}^* = 0$, leading to a system in which batteries would never be charged. This also allows to say that no other algorithm's performance in terms of electricity consumption can be better than $\mathbf{u'_k}^* = 0$, that will be called Always OFF Algorithm.

On the other hand but analogously, by considering $\beta = 1$ the sum becomes:

$$J\left(x_k\right)_{\beta=1} = \lim_{N\to\infty} \sum_{k=0}^{N-1} \alpha^k \cdot \mathbb{I}\left\{f\left(\left[\mathbf{x'_k} - \mathbf{u'_k}\right]^+\right) > E_0\right\}^T \cdot \mathbf{p_k^\lambda} \tag{40}$$

the sequence inside the sum can be minimized if the following condition is satisfied $f\left(\left[\mathbf{x'_k} - \mathbf{u'_k}\right]^+\right) \leq E_0$, in order to satisfy it on as many slots of time as possible there is nothing to do but to apply the maximum possible rate of charge so $\mathbf{u'_k}^* = \mathbf{u_{max}}$, that on an ON/OFF control it means that all the batteries would always be charging. In this case, it must be considered that the system evolves according to Eq. (35), but it can be seen that applying $\mathbf{u'_k}^* = \mathbf{u_{max}}$ can only reduce the cost for the following state. Again, no other algorithm will perform better in terms of car losses than using $\mathbf{u'_k}^* = \mathbf{u_{max}}$ which will be called Always ON Algorithm.

## 10.2 Approximate DP (decoupled problem approximation)

The main idea of problem approximation is, as its name suggests, to simplify the original problem to a simpler one. One of the strategies is to decouple variables that may be coupled in the original problem to reduce the dimensionality of the whole. For example, our $M$-socket station, could be thought as a set of $M$ single-socket station.

The algorithm in that case would produce, for each socket, a DP lookup table following the classical DP algorithm. In that case the cost is also decoupled into a vector $\mathbf{J_k}\left(\mathbf{x'_k}\right)$, and the DP algorithm for each component becomes:

$$\mathbf{J_k^{*(i)}}\left(\mathbf{x'_k}^{(i)}\right) = \min_{\mathbf{u'_k}^{(i)}} \left\{ \mathop{\mathbb{E}}_{\mathbf{w_k^{\lambda(i)}}, \omega_k^{PV}} \left[ g_k^{(i)}\left(\mathbf{x'_k}^{(i)}, \mathbf{u'_k}^{(i)}, \mathbf{w_k^{\lambda(i)}}, \frac{\omega_k^{PV}}{M}\right) + \mathbf{J_{k+1}^{*(i)}}\left(f\left(\left[\mathbf{x'_k} - \mathbf{u'_k}\right]^+\right)^{(i)}\right)\right]\right\} \tag{41}$$

Note that in this case, the power coming from the PV is simply distributed uniformly between each one of the socket, eliminating the possibility of further optimization through an intelligent distribution of it. In this case, the workflow of the controller would be the following:

**Algorithm 6** Decoupled problem approximation.

1. Compute $M$ lookup tables running the DP algorithm for each one of the sockets at the beginning of the day.
2. At each instant of time $k$:
   (a) Re-order the states computing $\mathbf{x'_k} = f(\mathbf{x_k})$
   (b) For each socket $i$:
      i. Look in its DP lookup table the optimal control $\mathbf{u'}_{\mathbf{k}}^{(i)}$ to apply knowing $\mathbf{x'_k}^{(i)}$
   (c) Compute the controls to apply to each socket by performing $\mathbf{u_k} = f^{-1}(\mathbf{u'_k})$

## 10.3 Reinforcement Learning (Multi-agent policy iteration + One step lookahead + decoupled problem approximation)

One step lookahead strategies are a halfway between simple problem approximation and solving the complete problem. Since, solving the complete problem is infeasible due to its dimensionality, these strategies perform an on-line optimization over all the possible controls to apply, given the current state of the system, and then for the cost-to-go function use an approximation (potentially through problem approximation). At each time step the algorithm would compute:

$$\mathbf{u'_k}^{*}(\mathbf{x'_k}) = \underset{\mathbf{u'_k}}{\operatorname{argmin}} \left\{ \underset{\mathbf{w_k^\lambda}, \omega_k^{PV}}{\mathbb{E}} \left[ g_k\left(\mathbf{x'_k}, \mathbf{u'_k}, \mathbf{w_k^\lambda}, \omega_k^{PV}\right)\right] + \widetilde{J}_k\left(\mathbf{x'_{k+1}}\right) \right\} \tag{42}$$

the cost-to-go approximation in Eq. (42) can be computed in several ways as already introduced in Sec. 6.2.1. Here it is proposed to use the cost-to-go tables obtained by running the Dynamic Programming at the beginning of the day for the decoupled system, and to sum the associated cost for each socket as shown in Eq. (43). In this way an approximated overall cost-to-go of the system is obtained, after applying a control $\mathbf{u'_k}$.

$$\widetilde{J}_k\left(\mathbf{x'_{k+1}}\right) = \sum_i \underset{\mathbf{w_k^{\lambda(i)}}}{\mathbb{E}} \left[ \mathbf{J}_{\mathbf{k+1}}^{*(i)} \left( f\left( \left[\mathbf{x'_k} - \mathbf{u'_k}\right]^{+} \right)^{(i)} \right) \right] \tag{43}$$

At each time step, all the possible actions are evaluated starting from the actual state of the system, which are $\#u^M$ possible controls. These kind of algorithms allow to have "collaboration" between the agents, because for at least one time step the complete problem is considered. Summarizing, the algorithm would work in the following way:

**Algorithm 7** One step-lookahead + decoupled problem approximation.

1. Compute $M$ lookup tables running the DP algorithm for each one of the sockets at the beginning of the day.
2. For each instant of time $k$:
   (a) Re-order the states computing $\mathbf{x'_k} = f(\mathbf{x_k})$
   (b) Find $\mathbf{u'_k}$ minimizing the cost in Eq. (42) by brute-force (this step may take some time to finish)
   (c) Compute the controls to apply to each socket by performing $\mathbf{u_k} = f^{-1}(\mathbf{u'_k})$

In the previous scenario of having $\#u = 2$ and $\#M = 20$ it would imply at least $\sim 10^6$ operations for each time step, which are several orders less than solving the complete problem through Dynamic Programming, but it is too much time consuming yet. In order to further reduce the search space for the optimal control, in problems like this in which the control variable is a vector that can be thought as $M$-agents operating each on its own socket, the control can be evaluated one at a time. In that case the algorithm would start with a base policy (for example applying $\mathbf{u'_k} = 0$) and then, minimizing the cost one-agent-at-a-time instead of all-agents-at-once:

$$\mathbf{u'_k}^*(\mathbf{x'_k}) = \text{seq. } \underset{\mathbf{u'_k}}{\text{argmin}} \left\{ \underset{\mathbf{w_k^\lambda}, \omega_k^{PV}}{\mathbb{E}} \left[ g_k\left(\mathbf{x'_k}, \mathbf{u'_k}, \mathbf{w_k^\lambda}, \omega_k^{PV}\right)\right] + \sum_i \underset{\mathbf{w_k^{\lambda(i)}}}{\mathbb{E}} \left[ \mathbf{J_{k+1}^{*(i)}} \left( f\left( \left[\mathbf{x'_k} - \mathbf{u'_k}\right]^+ \right)^{(i)} \right) \right] \right\} \tag{44}$$

where "seq." represents the fact that the minimization is performed one component of $\mathbf{u'_k}$ at a time (sequentially). Under this philosophy, the complexity is reduced to $\#u \cdot \#M = 40$ operations, for each iteration of the algorithm, which becomes linear on $\#u$.

**Algorithm 8** One step-lookahead + Multi-agent policy iteration + decoupled problem approximation.

1. Compute $M$ lookup tables running the DP algorithm for each one of the sockets at the beginning of the day.
2. At each instant of time $k$:
   (a) Re-order the states computing $\mathbf{x'_k} = f(\mathbf{x_k})$
   (b) Initialize the optimal control as $\mathbf{u'_k}^* := 0$
   (c) For a number of iterations $N_{IT}$ or until convergence:
      i. For each socket $i$:
         A. Find a new $\mathbf{u'_k}^{*(i)}$ by minimizing the cost in Eq. (44), where the rest of the components are kept fixed to $\mathbf{u'_k}^*$.
         B. Update the component $i$ of the optimal control $\mathbf{u'_k}^*$
   (d) Compute the controls to apply to each socket by performing $\mathbf{u_k}^* = f^{-1}(\mathbf{u'_k}^*)$

## 10.4 Computational complexity of algorithms

An important aspect on every optimization problem is the computational complexity of the optimizing algorithm, which gives a clear idea on how the parameters of the system affect the optimization time. In the following subsection the computational complexity (worst case) will be computed for each one of the algorithms described so far. In particular, the on-line computation complexity of each algorithm will be computed, which is the part of the algorithm that is ran in at the beginning of each slot of time.

### 10.4.1 Heuristics

Since as exposed on Alg. 4 the heuristic algorithm would try to optimize the scheduling by exploring the possible values of time for the postponing from 0 to $T_{max}$, then the computational complexity of the algorithm is the following.

$$O(M \cdot T_{max}) \tag{45}$$

### 10.4.2 Approximate Dynamic Programming

By applying Approximate Dynamic Programming (see Alg. 6), there are two main stages, the first one in which the batteries states are sorted, and the second one in which for each socket the control policy is retrieved from the control table.

$$O(M \cdot \log(M) + M) = O(M \cdot \log(M)) \tag{46}$$

### 10.4.3 Reinforcement Learning

After ordering the state of batteries, the RL algorithm would explore the control space for each socket (see Alg. 8). For each battery, for each possible control to apply, the future states are reordered and the cost-to-go is computed from the DP tables. The computational complexity becomes the following:

$$O\left(M \cdot \log\left(M\right) + N_{IT} \cdot M \cdot \#u \cdot \left(M \cdot \log\left(M\right) + M\right)\right) = O\left(M^2 \cdot N_{IT} \cdot \#u \cdot \log\left(M\right)\right) \quad (47)$$

# Part IV
# Results

In this section the results of implementing both the ADP algorithm and the RL algorithm will be shown. Firstly, the DP output tables will be shown since both the ADP and RL algorithm depend on them, and then the impacts of both $\alpha$ and $\beta$ on the performance on the algorithm will be presented. After presenting the behavior of the control tables coming from DP, the performances for the Heuristic, the ADP and the RL algorithms will be presented in Sec. 12.1, with also in Sec. 13, their performance in terms of other indicators that were identified as secondary.

Each algorithm has access to the state of the system $x_k$ which is the discrete variable for the lack of energy on each battery, and it was set to has 100 possible states, so it directly represents the percentile lack of energy with respect to fully charged battery of capacity $C$. The discrete time basis was selected to be $\Delta t = 5\text{min}$. All through this work the system has a number of sockets that is $M = 20$ as in the work presented by previous researchers. Batteries are charged until a maximum value of $0.9 \cdot C$, so every socket is turned off once the state of the battery reaches $x_k = E_0 = 0.1 \cdot C$.

All the statistical results computed in this Section were obtained by simulating the BSS through the first two weeks of the year. Only the final result presented in Sec. 14 was computed through a complete year since simulating the RL performance is time consuming.

## 11 Dynamic Programming control tables

The output of the DP Algorithm are two tables: the control table and the cost-to-go table. The first one indicates the optimal control to apply $u_k^*(x_k)$ at a given time and at a given state $(k, x_k)$, while the latter, for the same state variable indicates the cost-to-go function value $J_k^*(x_k)$. The algorithm runs backwards in time taking as input the information for the arrival rate for cars, the expected production of solar energy, and the cost for electrical energy from the grid in an hourly basis. Fig. 14 shows for example the input curves for a specific day over the year.

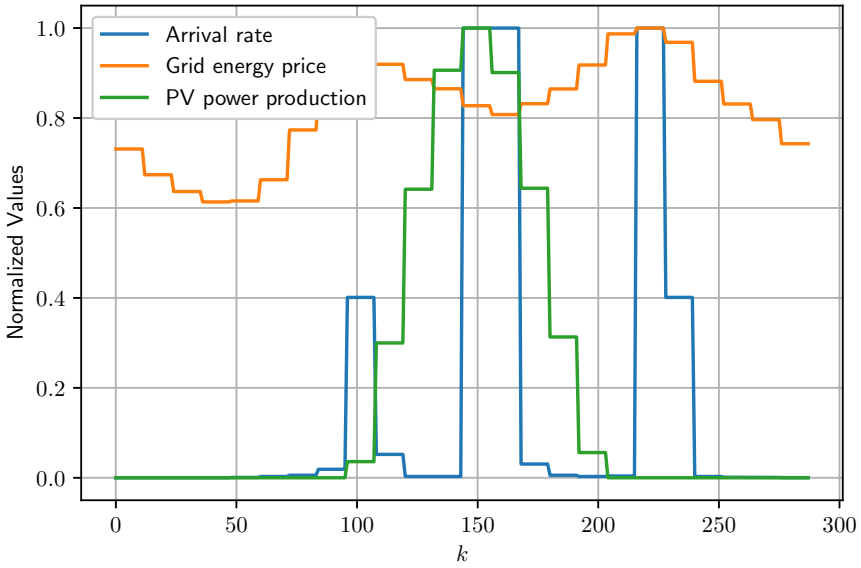


Figure 14: Normalized input curves for the fifth most charged battery.

## 11.1 Control and cost tables

The following tables were produced for the fifth most charged battery, with the hyperparameters $\alpha = 0.9$ and $\beta = 0.9$. It can be seen how the control table presents a very complex behavior, with different holes for the scheduling of the battery for certain values of $(k, x_k)$. These output tables were generated taking as input the curves shown at Fig. 14. From input curves it can be seen that cars may potentially arrive during three main moments of the day, this generates an increment in the cost-to-go function values, specially for low levels of charge. On the control table, it can be seen that a battery at low level starts its charging cycle at around $k = 50$ preparing it for the arrival rate peak at $k = 100$.

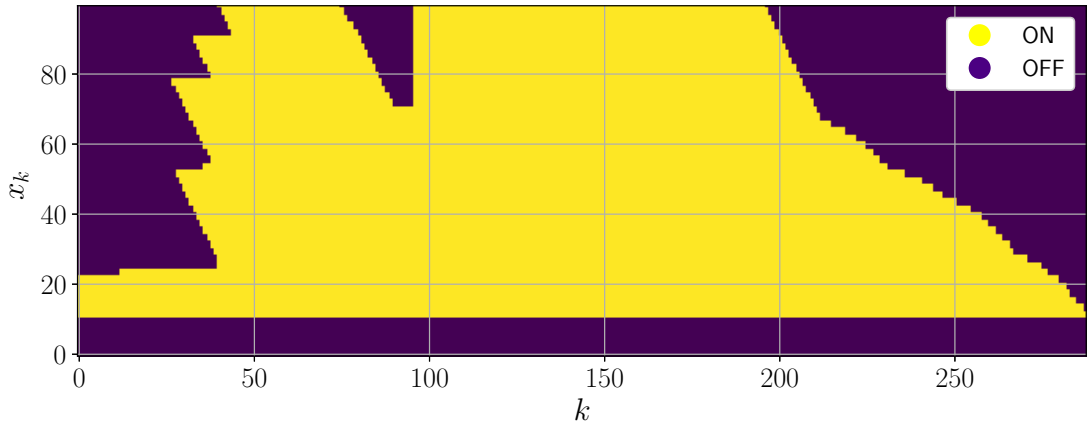


Figure 15: Control table obtained through Finite-Horizon DP Algorithm for the fifth most charged battery with $\alpha = 0.9$ and $\beta = 0.9$.

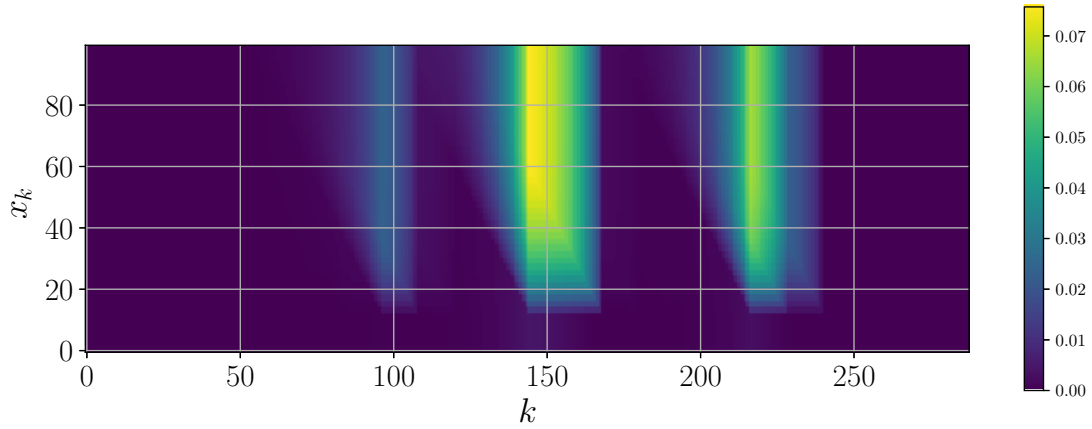


Figure 16: Cost-to-go values table obtained through Finite-Horizon DP Algorithm for the fifth most charged battery with $\alpha = 0.9$ and $\beta = 0.9$.

## 11.2 Impacts of $\alpha$ on DP tables

Hyper-parameter $\alpha$ as initially shown in Eq. (4.3.1), regulates the "visibility" of the DP algorithm. As $\alpha \to 1$ the algorithm weights more heavily step cost-function values for future time steps. Its impact can better be seen on the cost-to-go value table computed by DP as shown in Fig. 17. It can be seen how for great values of $\alpha$, the cost-to-go function assumes larger values (see the scale reference on each plot), and how accumulation starts at earlier values of time $k$. In this work it was selected to have a value of $\alpha = 0.9$, which gives a visibility for the algorithm (assuming that once $\alpha^k = 0.1$ it can be neglected) of about:

$$k_H = \log_\alpha (0.1) \approx 22 \tag{48}$$

with a time step of $\Delta t = 5\text{min}$, it means that the algorithm has a visibility of about $t_H = \Delta t \cdot k_H = 109\text{min}$, so nearly 2 hours of visibility, which is the recharging time for a battery.

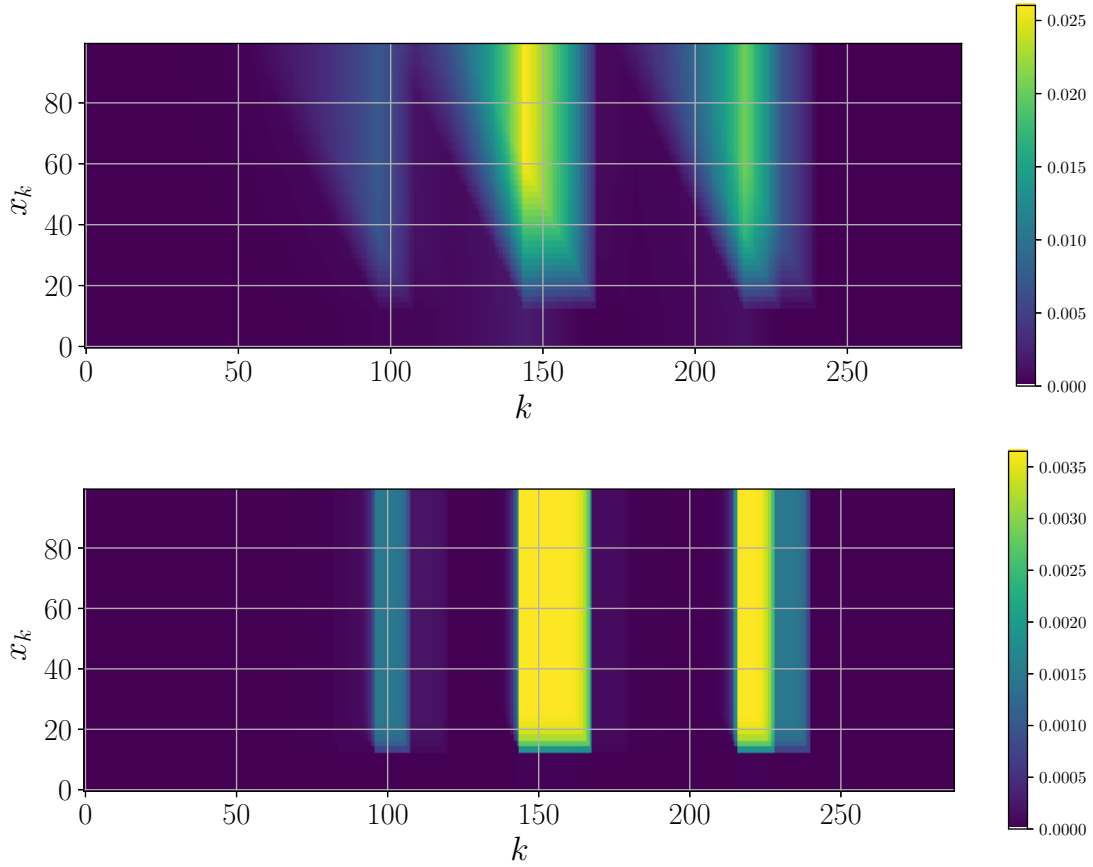


Figure 17: Cost-to-go tables obtained through Finite-Horizon DP Algorithm for different values of $\alpha$. From top to bottom $\alpha = 0.95$ and $\alpha = 0.5$. It can be seen how as $\alpha$ varies from 1 to 0, the visibility of the algorithm decreases and the cost-to-go function takes more concentrated values.

## 11.3 Impacts of $\beta$ on DP tables

Hyper-parameter $\beta$ regulates the trade-off between buying the needed electrical energy from the grid, and losing cars in the system. In Fig. 18 it can be seen how as $\beta \to 1$, on the cost function the probability of losing a car becomes heavier, so the algorithm tends to charge the battery mostly always, while as $\beta \to 0$ the algorithm prioritizes not buying energy from the grid, so it turns on the sockets only when the solar energy is available.

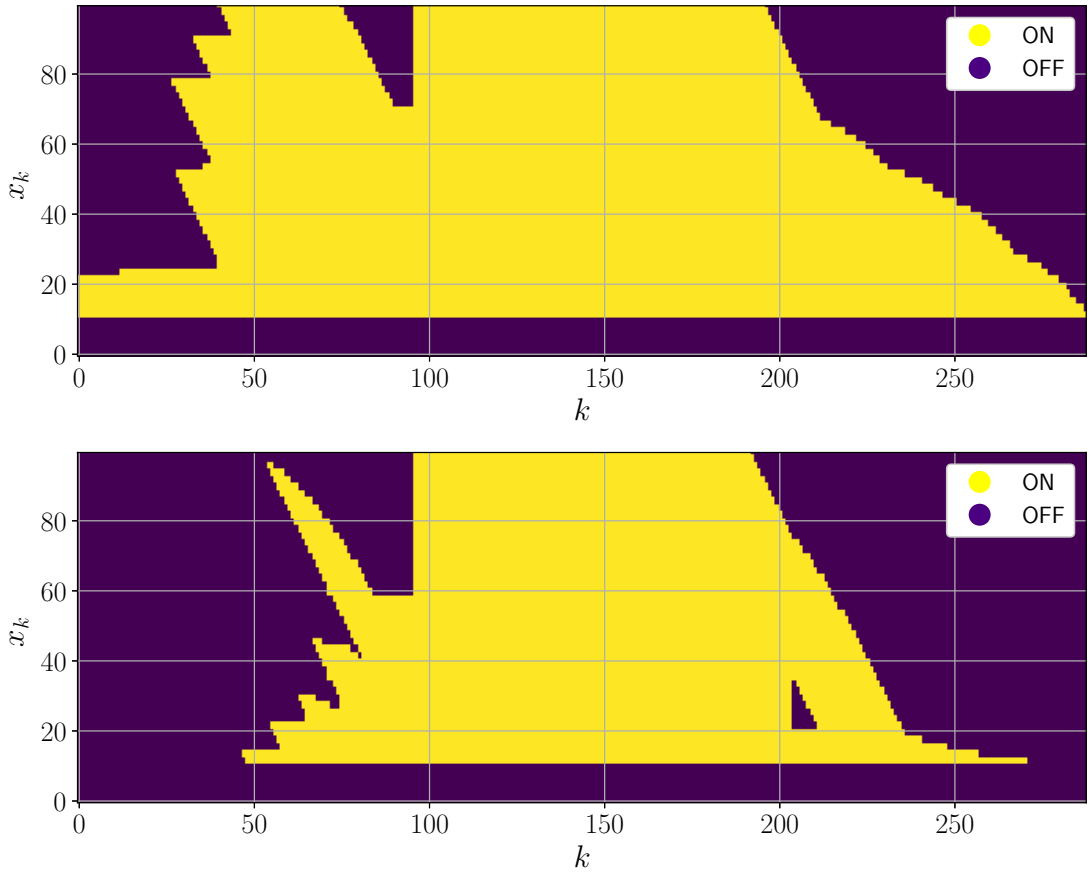


Figure 18: Control tables obtained through Finite-Horizon DP Algorithm for different values of $\beta$. From top to bottom $\beta = 0.9$ and $\beta = 0.2$. It can be seen how as $\beta$ varies from 0 to 1, the number of sates in which the socket is turned on decreases.

## 11.4 Problems with Finite-Horizon DP

The DP Algorithm solves the optimization problem starting at $k = N$ that in this case is the end of the day. In order to run the algorithm the starting condition which is $g_N(x_k)$ must be known for each possible final state $x_k$. In the system, it was set $g_N(x_k) = 0$ since it was not assigned a particular cost for the state of the battery by the end of the day, but this introduces some problems. Since the final cost is zero, as the end of the day approaches, some batteries will not be charged because they will not be able to arrive to $E_0$. This can be seen clearly on Fig. 19 in which output tables were generated with $\beta = 1$ which corresponds to the Always ON algorithm.

To overcome this problem, a value iteration approach was used and its impacts are shown in Sec. 11.5.

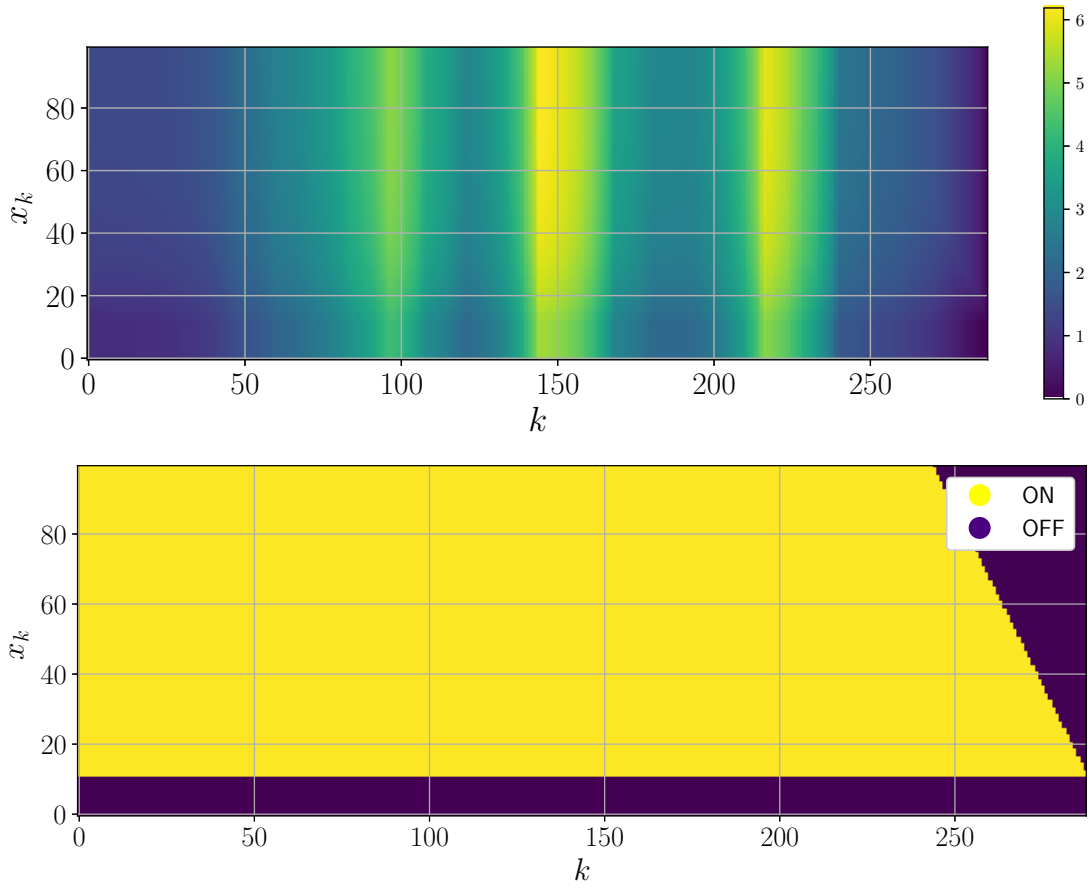


Figure 19: Cost-to-go and control table obtained through Finite-Horizon DP Algorithm for different values for $\alpha = 0.9$ and $\beta = 1$.

## 11.5 Value iteration tables

To overcome the boundary conditions effect of Finite-Horizon DP, value iteration was used. This also has an actual interpretation on the optimization problem, considering a time horizon on the system in senseless since it has to operate every day of the year. The BSS is a continuous system that has to operate every day, so using value iteration for computing DP control and cost-to-go tables is as solving the problem for a system in which every day has the same input curves. This is to say that at the beginning of each day, input curves are interpreted as stationary.

In Fig. 20 it can be observed how cost-to-go values are all non-zero, and that values at $k = 0$ are the same that are present at $k = N$. Also the control table does not present the boundary effect that was present in Fig. 19 for Finite-Horizon DP, and the control table presents also the same control policy at the beginning of the day and at the end of the day.

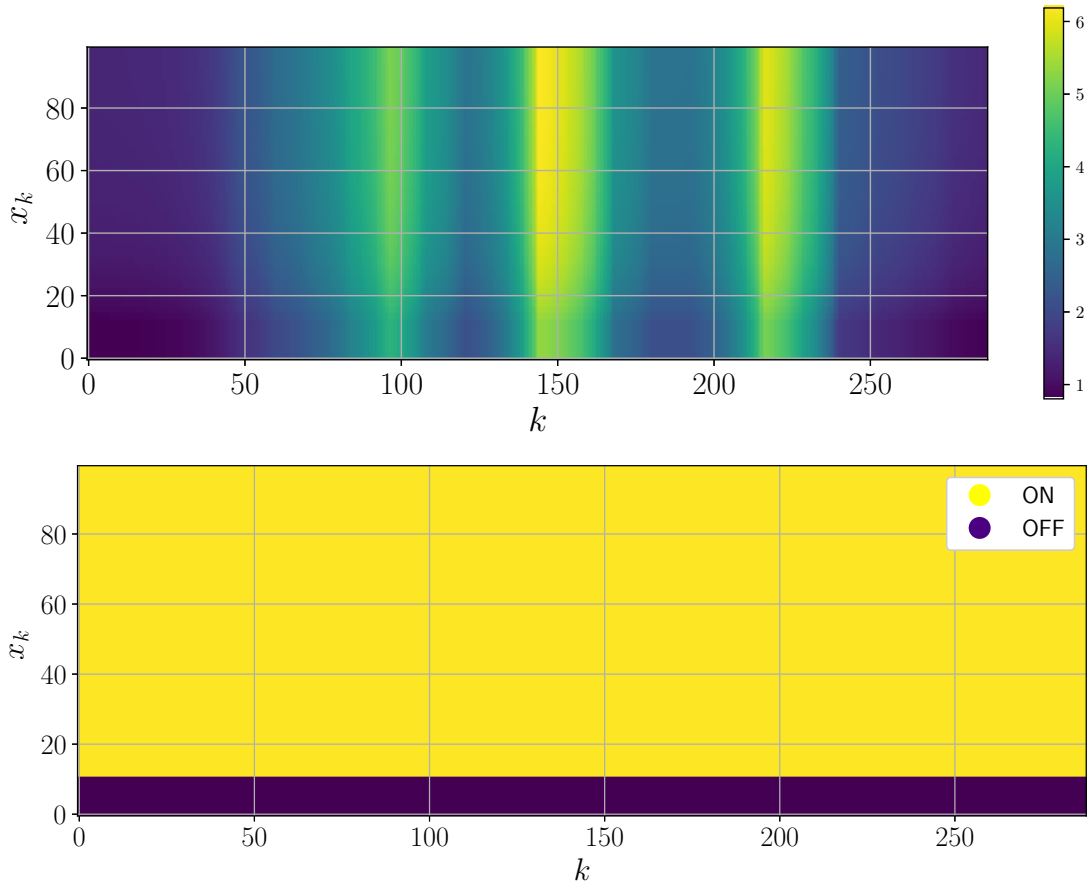


Figure 20: Cost-to-go and control table obtained through Infinite-Horizon DP Algorithm (through value iteration) for different values for $\alpha = 0.9$ and $\beta = 1$.

# 12 Impacts of $\beta$ on KPIs

Hyper-parameter $\beta$ allows to tune the weight to give to the trade-off between paying electricity from the grid, or accepting to lose client cars. Fig. 21 shows the normalization against Always ON and Always OFF Algorithms of the performance of the ADP algorithm by varying $\beta$. It is clear that when $\beta = 0$, the only cost that is taken into account is the electricity cost, so an Always OFF algorithm is applied, this minimizes the grid cost (it is actually zero), but maximizes the loss of cars. On the other hand, by setting $\beta = 1$, an Always ON algorithm is applied maximizing electricity costs and minimizing car losses. In the way between $\beta = 1$ and $\beta = 0$, the algorithm follows a trajectory that allows to choose the working point to use.

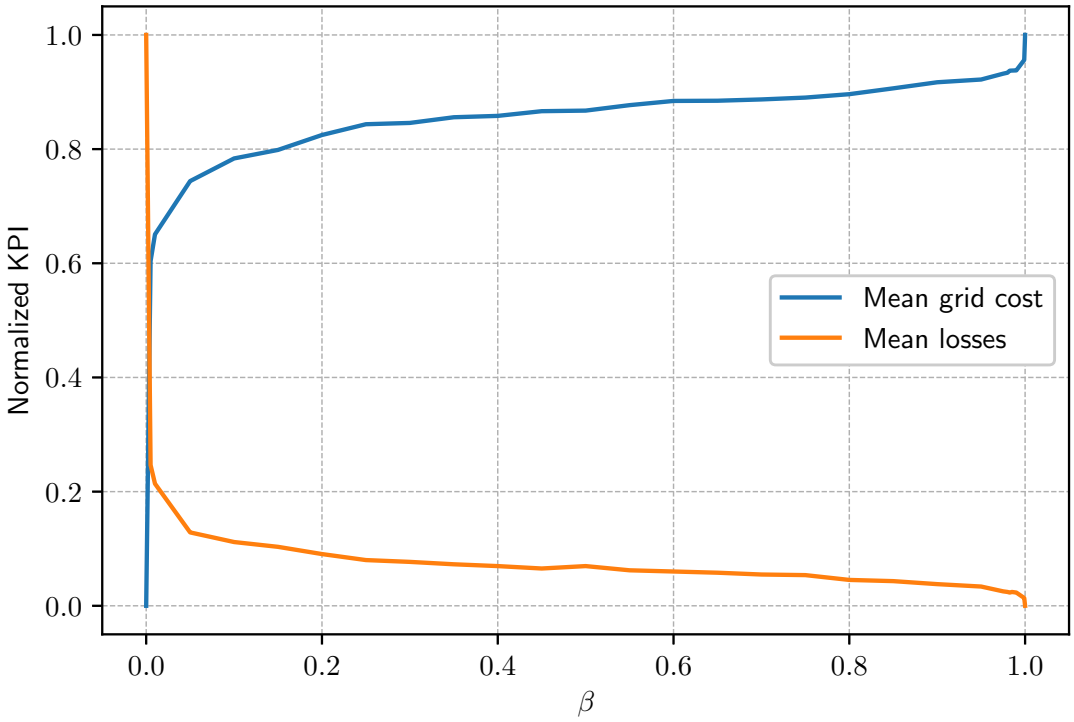


Figure 21: Infinite Horizon DP Algorithm performances by varying $\beta$. KPI are normalized with respect to the performances of Always ON and Always OFF algorithms.

## 12.1 Algorithms comparison

Since the interpretation and usage of $\beta$ is different on each algorithm, plotting different algorithms together as done in Fig. 21 is not useful. Instead, as described in Sec. 8.1, the best way to compare the performance of algorithms is by plotting the normalized cost-losses diagram. This diagram has two advantages, on one side it makes the representation of the performance of the algorithms independent from the particular values of $\beta$, while on the other side it allows to have a normalized representation of performances with respect to the quotas imposed by Always ON and Always OFF algorithm. These features make this diagram really useful when benchmarking different control algorithms.

In Fig. 22 the performance for the Heuristics, the ADP and the RL algorithm are presented.

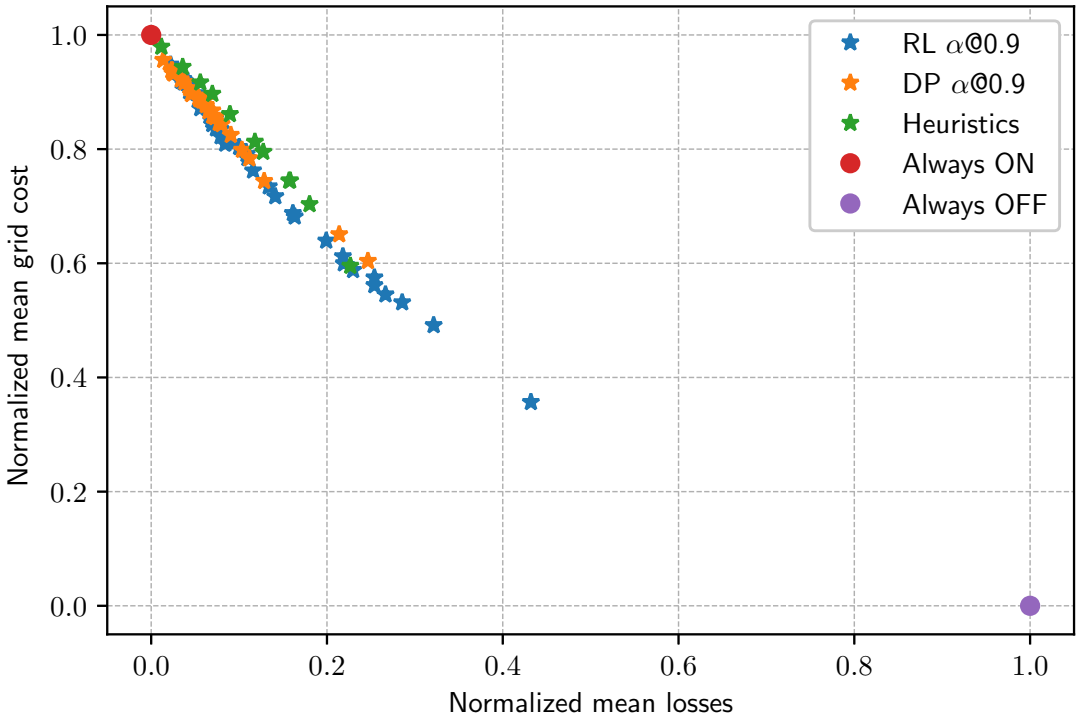


Figure 22: Performances of different algorithms by varying $\beta$.

Hyper-parameter $\beta$ can be set in any value from 0 to 1, but actually the most interesting working zone is the one near the Always ON performance. Even if there is a natural trade-off between electricity costs and car losses, the BSS must guarantee users that an charged battery is available almost always, otherwise the system would not be used by clients. This means that the algorithm will be set to work with values $\beta \sim 1$. If Fig. 22 is zoomed in near the point $(0, 1)$ which represents the Always ON performance, algorithms performance can be compared by analyzing the rate of change of the normalized electricity cost, with respect to the normalized car losses. In Fig. 23, algorithms performance are shown with the best fitting line passing through $(0, 1)$, showing that RL is the best algorithm to use.

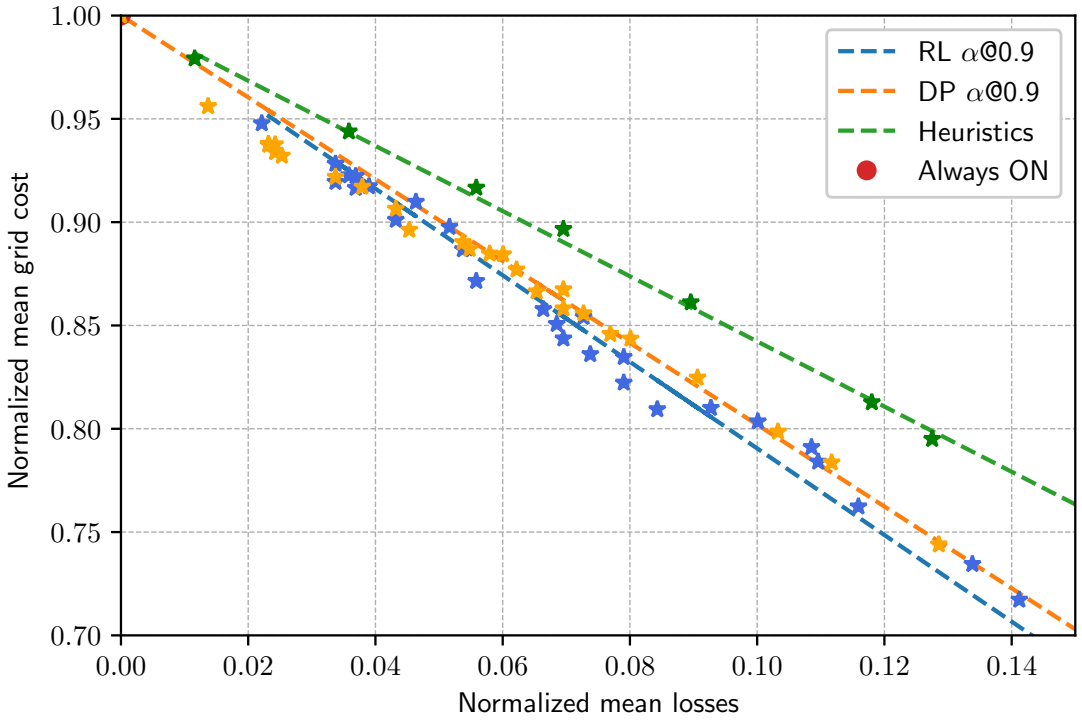


Figure 23: Performances by varying $\beta$ zoomed to the working zone of interest (near to Always ON performance).

Since as explained above algorithms can be compared through the best fitting line passing through $(0, 1)$, the following tables show the angular coefficient for each algorithm.

| **Control algorithm** | $m$ |
|---|---|
| Heuristics | $-1.58$ |
| Dynamic Programming | $-1.98$ |
| Reinforcement Learning | $-2.10$ |

Table 3: Angular coefficient for the best-fitting line passing through $(0, 1)$, to the performance points for different algorithms while varying their $\beta$ value.

# 13 Beyond KPIs

This sub-section introduces other performance indicators beyond the ones used by control algorithms. There is a price to pay in order to have a better performance system and for the Reinforcement Learning algorithm they turn to be the number of switching on and off of the sockets during the recharging cycle of batteries, and the amount of time needed to compute decisions.

## 13.1 Batteries time on the system

An interesting metric of the system is the mean time of a battery in the system. Since as ADP and RL algorithms order batteries according to the state of charge, one could ask if this incur on having batteries on the system for too much time. Fig. 24 shows the Probability Density Function (PDF) for the time on the system for each algorithm. It can be seen that actually RL Algorithm is the one that has the shortest tail, followed by the DP Algorithm and the Heuristics.

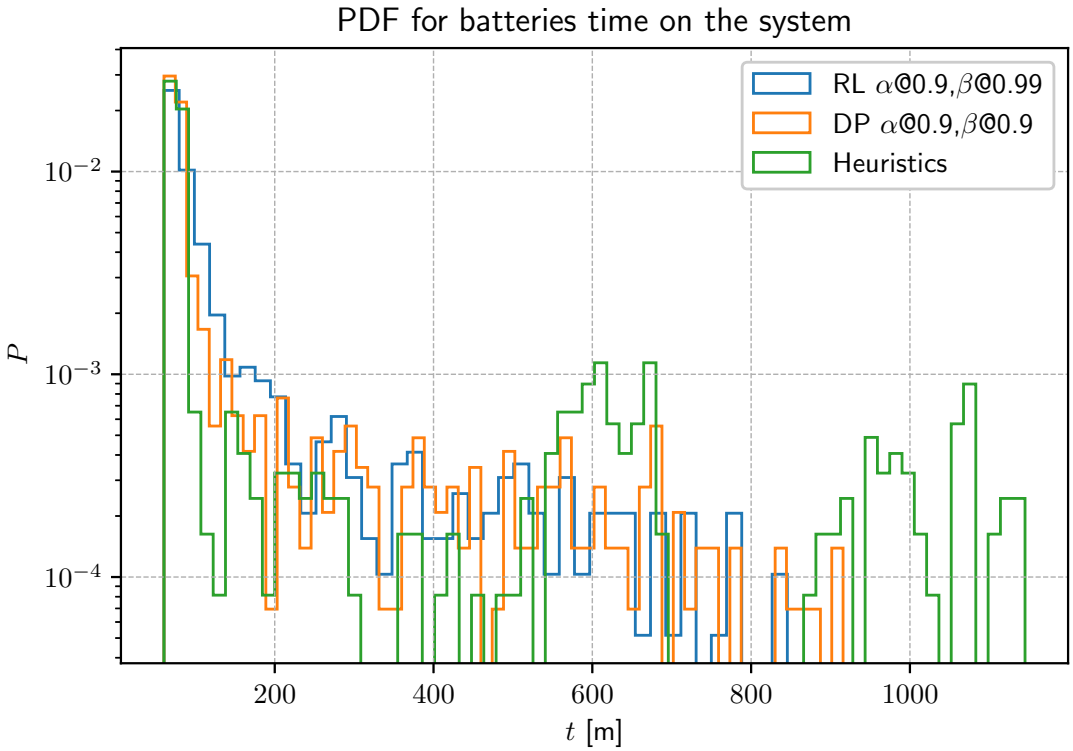


Figure 24: Probability Density Function (PDF) for the time of batteries on the system.

In Table 4 there are some statistics for the simulation performed. As expected from analyzing Fig. 24, the Reinforcement Learning is the one with shortest expected value for the times of batteries in the system, with a maximum value of 645 minutes (about 14 hours), while the Heuristic algorithm could take as long as 1145 minutes (19 hours) to fully recharge a battery.

| **Control algorithm** | $\mathbb{E}(T)$ [m] | $\max(T)$ [m] |
|---|---|---|
| Heuristics | 209 | 1145 |
| Dynamic Programming | 137 | 916 |
| Reinforcement Learning | 135 | 845 |

Table 4: Statistics for the time of batteries in the system.

## 13.2 Number of times charging is resumed

An important performance metric is the number of times that charging is resumed during the recharging cycle of a battery in the system. Switching off and on the socket during the battery recharging may affect and have negative impacts on the battery life. Fig. 25 shows the Probability Density Function for the number of time charging is resumed during the permanence of the battery in the system and it shows that the Heuristics algorithm has the less number of socket switching while RL is the one that can arrive to a maximum number of switchings of 8. This is mainly the price to pay in order to have a dynamic scheduling algorithm.

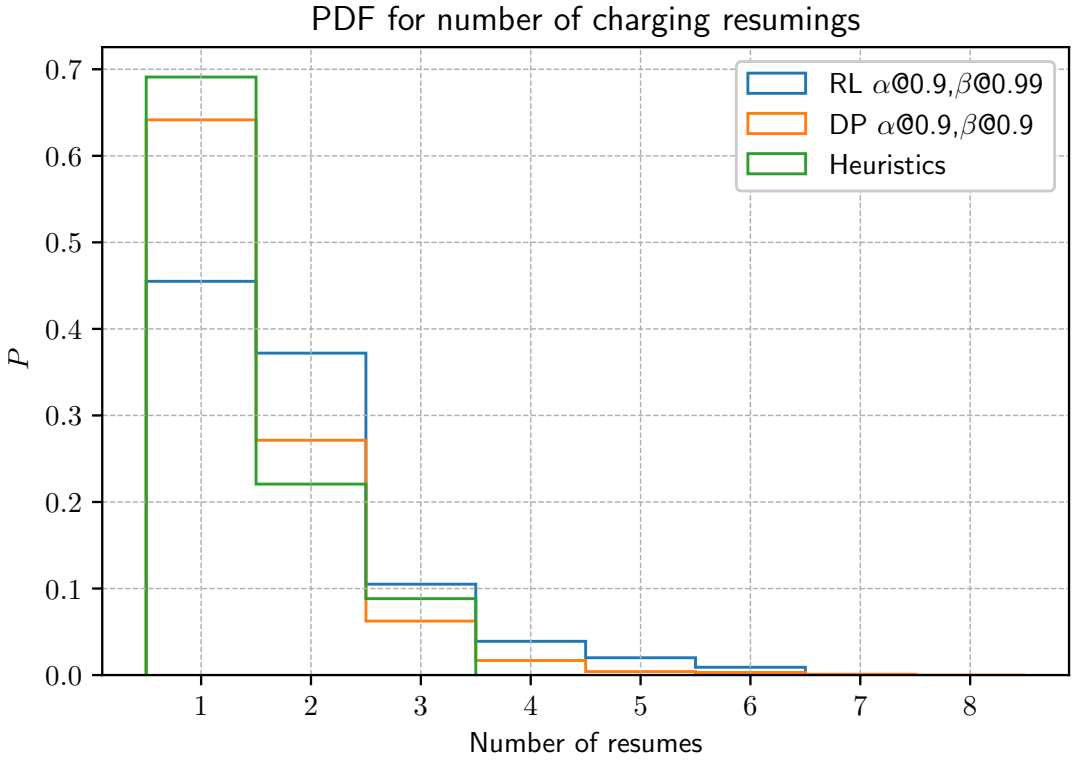


Figure 25: Probability Density Function (PDF) for the number of times that the charging is resumed/restarted during the time in which the battery is on the system.

In Table 5 there are some statistics for the simulation performed. As expected from analyzing Fig. 25, the Reinforcement Learning is the one with largest expected value for the number of charging times, with a maximum value of 9, while the Heuristic algorithm which is much less reactive has a maximum number of charging resumes of 3.

| **Control algorithm** | $\mathbb{E}(N_C)$ [m] | $\max(N_C)$ [m] |
|---|---|---|
| Heuristics | 1.40 | 3 |
| Dynamic Programming | 1.48 | 7 |
| Reinforcement Learning | 1.89 | 9 |

Table 5: Statistics for the time of batteries in the system.

### 13.3 Computational time

Computational time of complex algorithms tends to be greater with respect to simpler ones. In particular the computational complexity (worst case) was already computed for these algorithms in Sec. 10.4 showing that the time for RL is the greatest. In particular Table 6 shows the mean execution time for each one of the algorithms and it turns out that RL is the most time consuming one. Here however, the actual important aspect is the execution time to be less than $\Delta t = 5$min which is the discrete time step of the system. It is clear that even if RL is the most time consuming algorithm it is in any case usable in a real time system.

| **Control algorithm** | $\mathbb{E}(T)$ $[ms]$ |
|---|---|
| Heuristics | $21.0 \cdot 10^{-3}$ |
| Dynamic Programming | $33.7 \cdot 10^{-3}$ |
| Reinforcement Learning | 189.4 |

Table 6: Statistics for the decision making time for each algorithm during the online run.

## 14 A study case

A one year-long simulation was done in order to better compare the performance improvement introduced by the Reinforcement Learning algorithm with respect to the Heuristics one. In particular, for the Reinforcement Learning algorithms, hyper-parameters were tuned to be $\alpha = 0.9$ and $\beta = 0.99$. The control hyperparameters of the Heuristics algorithm were set to $F_{max} = 17$ (maximum number of concurrent postponable batteries), and $T_{max} = 40$ hours (maximum postponable time for batteries).

| Metric (Mean daily values) | Control algorithm | | Improvement |
|---|---|---|---|
| | **Heuristic** | **Reinforcement Learning** | |
| Arrivals [n] | 137.32 | 137.32 | N.A. |
| Losses [n] | 4.25 | 3.34 | -21.4% |
| Cost per service [€] | 0.411 | 0.391 | -4.86% |
| Grid consumption [Wh] | $8.64 \cdot 10^5$ | $8.62 \cdot 10^5$ | -0.23% |
| PV energy sold [€] | 15.85 | 16.01 | +1.01% |

Table 7: Statistics for the decision making time for each algorithm during the online run.

From Table 7, it can be seen how all the metrics are improved through Reinforcement Learning. The value of $\beta = 0.99$ was tuned to match the same level of electricity consumption from the grid of the Heuristic algorithm. For both the Heuristic and RL algorithms the losses are below 3%, the improvement introduced by the Reinforcement Learning is of about 21.4% for car losses and of 4.86% in terms of costs per service, showing that the algorithm is capable to use in a more intelligent way the electrical energy bought from the Smart Grid.

# Part V
# Conclusions

The previous parts have shown the historical context in which this research is being held, and presented the principle of work of a Battery Swapping Station, particularly focusing on a BSS embedded in a Smart Grid and provided with local solar energy generation. It was explained the mathematical background needed to design and implement control (scheduling) algorithms for the BSS. In particular, two algorithms based on Approximate Dynamic Programming and Reinforcement Learning were introduced, designed, implemented and tested. This Part will present the conclusions of the work, what has been done so far and what there is to do yet.

## 15 Improvements introduced by this work and next steps

Starting from a simulator that was already developed by previous researchers, this work aimed at designing, implementing, and testing control algorithms for the scheduling of a Battery Swapping Station. The initial objectives set at Sec. 3 were achieved and in particular this work contributed to this research on the following aspects:

- **Formalize the recharge scheduling of batteries as a mathematical optimization problem:** a Markovian Decision Problem for the $M$-socket station was introduced in Sec. 10. This general model could be also expanded introducing future considerations in the step cost-function, or in the evolution of the system.
- **Design and implement an adaptive algorithm:** two algorithms were proposed, implemented and tested based on Approximate Dynamic Programming (Sec. 10.2) and Reinforcement Learning (Sec. 10.3), being able to adapt to the stochastical nature of the system and to consider the forecasts for the changing electrical costs, cars arrival rates, and PV energy production.
- **Improvement of performance:** the proposed algorithms were implemented and tested, introducing an improvement in terms of Key Performance Indicator (KPI) with respect to the previous existing Heuristic algorithm (Sec. 14). It was shown that on a particular case the RL algorithm could improve losses of about 20% when the same electrical energy is being bought from the grid, and to improve the cost per service by about 5%.
- **Controlling the working point of the system through hyper-parameters:** the Markovian Decision Process that models the evolution of the system considers a step

cost-function in Eq. (36) that has an hyper-parameter $\beta$ that allows to weight differently electrical energy costs and losses, enabling engineers to fine-tune the algorithm according to business needs.

- **Introduce a benchmarking mechanism for future developments:** a method for benchmarking algorithms in terms of electrical cost and car losses was introduced, enabling future researchers to objectively compare the performance of different scheduling algorithms (Sec. 8.1 and Sec. 12.1).

## 15.1 Possible next steps

Even if several improvements were introduced, there are a lot of directions towards which this research may continue its path. Hereunder some weak points, and potential improvements are listed:

- **Use external estimations for solar energy, arrival rates, and electricity cost prices:** both ADP and RL algorithms are based on DP tables that are computed at the beginning of the day. When running the DP algorithm, the agent is using the exact same information that is used by the simulator to then run the system. This means that the Agent has perfect information of the stochastical nature of the external variables. One improvement for this research could be to give as input to algorithms information about the external variables from external sources of information to have more realistic results.

- **Reduce computational complexity of algorithms:** since the system is composed by $M$ sockets, at each time step $k$ of the day, the algorithm tries to optimize the control strategy for each of them with computational complexities that are expressed in Eq. (46) and Eq. (47). They may be too time consuming if one aims at comparing algorithms performances on a multi-year basis, so an improvement could be to explore different strategies for further reducing the exploration of the control space to reduce execution times.

- **Explore model-free approaches:** both ADP and RL algorithms are based on DP tables that are computed through a mathematical model of the system at the beginning of the day. Since it is not possible to compute a DP table for the whole system due to its complexity, in this work the problem was decomposed in $M$ simpler problems for which DP becomes feasible. In order to overcome this limitation, different model-free approaches could be used. Model-free approaches do not rely on a DP table that is precomputed based on a mathematical model, but instead, they automatically "learn" a model for the system by interacting directly with it and storing this information on $Q$-tables or parametrical models like Neural Networks.

- **Expanding cost function with new metrics:** instead of using the Key Performance Indicators (KPI) used in this work which were only electrical cost and car losses, the

cost function may be expanded to consider other indicators that may be of extreme importance, for example to preserve the health of batteries on the long run. In that case the cost function may have multiple hyper-parameters to construct a step-cost function that may be written as:

$$g_k = \sum_i \beta_i \cdot \text{Performance Indicator}_i \tag{49}$$

$$\sum_i \beta_i = 1 \tag{50}$$

- **Allow the BSS to modulate charging rates:** this work focused on just scheduling the charging of batteries, or in other words to simply apply an on/off control signal to the system. If there was the possibility to have smart sockets able to modulate the power sent to batteries then a more sophisticated optimization may be done, introducing the possibility to charge batteries at different rates. The DP algorithm implemented on this work is already predisposed to do such an optimization. In Fig. 26 it is possible to se how a control table for a system with 3 possible charging levels looks like.

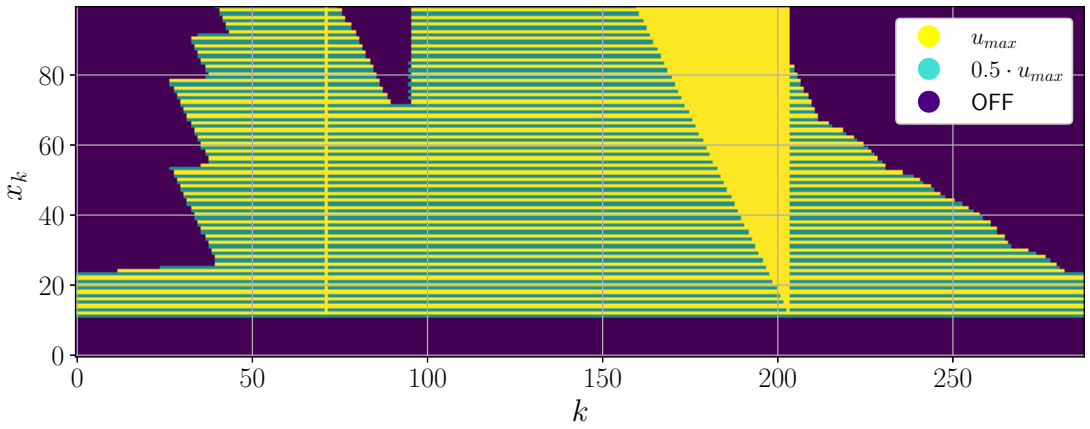


Figure 26: Control table obtained through Infinite-Horizon DP Algorithm for the fifth most charged battery with $\alpha = 0.9$ and $\beta = 0.9$ with two levels of charging power.

# 16 Final comments

As changes in the environment become more evident year after year, preoccupations and concerns about global warming are raising not only from scientific and economical actors but actually from the whole society. This work has presented different algorithms that may contribute to the usage of renewable energies within an Electric Vehicle recharging station. Starting from Dynamic Programming, different strategies were explored arriving to an optimal algorithm based on Reinforcement Learning. The results obtained are satisfactory since

they introduce an improvement in terms of performance of the system with respect to the previous controller. Also some ideas were introduced that may contribute to benchmarking future algorithms.

Reinforcement Learning is an emerging field that proposes different strategies for controlling stochastic and complex dynamical systems. Most approaches of RL are based on exploring the control space in search of the optimal control strategy, allowing the system to make decisions in a real-time basis without supervision. This implies a shift from a classical control theory in which a clear model of the system is available and solutions may be computed analytically, to a new Artificial Intelligence-driven one in which decisions are made based on cost functions without human intervention. About this last point a final reflection may be of interest, introduced by Prof. Dimitri Bertsekas in the Preface to his last book "Reinforcement Learning and Optimal Control" referring to the AI-driven chess player AlphaZero [14]:

*"What is frustrating about machine learning, however, is that the algorithms can't articulate what they're thinking. (...) AlphaZero gives every appearance of having discovered some important principles about chess, but it can't share that understanding with us. Not yet, at least. As human beings, we want more than answers. We want insight. This is going to be a source of tension in our interactions with computers from now on".*

# References


[1] *European response to climate change*. URL: https://www.europarl.europa.eu /news/en/headlines/society/20180703STO07129/eu-responses-to-climate-change

[2] *Warmest year record in Europe*. URL: https://climate.copernicus.eu/copernicus-2020-warmest-year-record-europe-globally-2020-ties-2016-warmest-year-recorded

[3] *Sources of Greenhouse Gas Emissions*. URL: https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions#:~:text=Greenhouse%20gas%20emissions%20from%20transportation,includes%20primarily%20gasoline%20and%20diesel

[4] *Total greenhouse gas emission trends and projections in Europe*. URL: https://www.eea.europa.eu/ims/total-greenhouse-gas-emission-trends

[5] *Greenhouse gases viewer*. URL: https://www.eea.europa.eu/data-and-maps/data/data-viewers/greenhouse-gases-viewer

[6] *European Climate Law*. URL: https://climate.ec.europa.eu/eu-action/european-green-deal/european-climate-law_en

[7] *Transport and the Green Deal*. URL: https://ec.europa.eu/info/strategy/priorities-2019-2024/european-green-deal/transport-and-green-deal_en#actions

[8] *EU approves effective ban on new fossil fuel cars from 2035*. URL: https://www.reuters.com/markets/europe/eu-approves-effective-ban-new-fossil-fuel-cars-2035-2022-10-27/

[9] *Horizon Europe*. URL: https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-europe_en

[10] MOVE, "Sustainable & Smart Mobility Strategy", European Comission, 2020.

[11] Ye Yan, Yi Qian, Hamid Sharif, and David Tipper, "A Survey on Smart Grid Communication Infrastructures: Motivations, Requirements and Challenges", IEEE Communications Surveys & Tutorials, Vol. 15, No. 1, 2013, DOI: 10.1109/SURV.2012.021312.00034.

[12] R. Deng, Z. Yang, M. -Y. Chow and J. Chen, "A Survey on Demand Response in Smart Grids: Mathematical Models and Approaches", IEEE Transactions on Industrial Informatics, vol. 11, no. 3, pp. 570-582, 2015, DOI: 10.1109/TII.2015.2414719.

[13] J. S. Vardakas, N. Zorba and C. V. Verikoukis, "A Survey on Demand Response Programs in Smart Grids: Pricing Methods and Optimization Algorithms" in IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 152-178, Firstquarter 2015, DOI: 10.1109/COMST.2014.2341586.

[14] D. Bertsekas, “Reinforcement Learning and Optimal Control”. United States: Athena Scientific, pp. 1-20, 2019, ISBN: 978-1-886529-39-7.

[15] D. Bertsekas, "Lessons from AlphaZero for optimal, model predictive, and adaptive control"”. United States: Athena Scientific, 2022, ISBN: 1-886529-17-5.

[16] *Gogoro: Swap&Go.* URL: https://www.gogoro.com/?utm_source=google&utm_medium=cpc&utm_campaign=bb_brand_generic&gclid=Cj0KCQiAg_KbBhDLARIsANx7wAxF8kIg6-YHOzvcZv22kVIBzuxUd8zWGkOyABOiEmgfgWorU_RCbB0aAiuVEALw_wcB

[17] K. Tahara, K. Ishikawa and M. Ishigaki, "Battery as a Service for Electric Vehicles: Design and Optimization of Partially Swappable and Shareable Battery System" 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pp. 1-8, 2020, DOI: 10.1109/ITSC45102.2020.9294582

[18] C. Zhu, J. Xu, K. Liu and X. Li, "Feasibility analysis of transportation battery second life used in backup power for communication base station" 2017 IEEE Transportation Electrification Conference and Expo, Asia-Pacific (ITEC Asia-Pacific), pp. 1-4, 2017, DOI: 10.1109/ITEC-AP.2017.8080810.

[19] E. Martinez-Laserna et al., "Technical Viability of Battery Second Life: A Study From the Ageing Perspective" in IEEE Transactions on Industry Applications, vol. 54, no. 3, pp. 2703-2713, 2018, DOI: 10.1109/TIA.2018.2801262.

[20] E. Hossain, D. Murtaugh, J. Mody, H. M. R. Faruque, M. S. Haque Sunny and N. Mohammad, "A Comprehensive Review on Second-Life Batteries: Current State, Manufacturing Considerations, Applications, Impacts, Barriers & Potential Solutions, Business Strategies, and Policies" in IEEE Access, vol. 7, pp. 73215-73252, 2019, DOI: 10.1109/ACCESS.2019.2917859.

[21] D. Bertsekas, “Dynamic Programming and Optimal Control”, Athena Scientific, volume I, pp. 12-14, 402-417, 2005, ISBN: 1-886529-26-4.

[22] V. François-Lavet, “An Introduction to Deep Reinforcement Learning”, NOW, vol. 1, no. 3-4, pp. 46-52, 2018, DOI: 10.1561/2200000071

[23] D. Bertsekas, "Multiagent Reinforcement Learning: Rollout and Policy Iteration" in IEEE/CAA Journal of Automatica Sinica, Vol. 8, pp. 249-271, 2021, DOI: 10.1109/JAS.2021.1003814.

[24] R. Monti, “Analysis and Dimensioning of Battery Switching Stations”, Politecnico di Torino, 2019, URI: http://webthesis.biblio.polito.it/id/eprint/13086

[25] G. Centonze, “Smart Recharging of Electrical Vehicles with Battery Switching Technology”, Politecnico di Torino, 2021, URI: http://webthesis.biblio.polito.it/id/eprint/21281