Analysis and Visualization of the Decisions of Autonomous Agents Acting in Uncertain Environments

Candidate: Giovanni Bagolin
Supervisor: Prof. Alessandro Farinelli
Co-Supervisor: Dott. Giulio Mazzi
Co-Supervisor: Dott. Alberto Castellini

Master Degree of Computer Engineering for Robotics and Smart Industries

Departement of Computer Science
Università degli Studi di Verona
Italy
2020/2021

Contents

Chapter 1

Introduction

Systems powered by artificial intelligence are now spread around the world. They can be found in our smartphones, for detecting the finger print, they can be used for face recognition, or natural language processing used also for voice recognition and speech understanding. Another key role of artificial intelligence is planning actions for an agent that acts in the world. An example of planning for an agent, is planning the speed of a mobile robot moving in an environment avoiding obstacles. The goal of the planning algorithm is to choose the highest speed possible for the robot to travel. However, moving fast increases the probability of colliding to an obstacle. Therefore the planning algorithm has to find the best trade-off between choosing an high speed and not colliding to obstacles. In this case the mobile robot is the agent. This example, is described in Section 4.1.2. A popular framework to model these problems is called *Markov Decision Process, (MDP)* described in Section 2.1. An MDP can model systems in which the actions have a stocastic outcome. A solution for the MDP is a policy, a function that maps each state to an action. In case of an MDP, a perfect policy can be found. However, if the agent does not have a perfect knowledge of the current state, we model the problem with an extension known as *Partially Observable Markov Decision Process, (POMDP)*. Unfortunately, the problem of finding a perfect policy when there are imperfect information in the problem is computationally intractable as shown in Section 2.2. For example, in the real world, the mobile robot does not know where the obstacles are, but it can obtain observations of where they are in a range of some meters using lasers for example. Unfortunately, the lasers used are noisy and therefore the information might be corrupted. This is an example of a decision making problem using imperfect information: the agent has to decide which speed to choose avoiding obstacles, but it is not certain of where the obstacles are in the environment. In this example, POMDP can be used to describe the problem, and an approximate

solution can be found using an online algorithm namely, *Partially Observable Monte Carlo Planning, (POMCP)*. This algorithm only finds an approximate solution but it has the advantage to work on very large problems (i.e., problems having very large state space). In this thesis we investigate specific cases in which POMCP selects unexpected or wrong (e.g., unsafe) actions. These might be situations where the action choosen is not the right one. This might happen due to wrong hyperparameter values in POMCP or for other reasons, for example noisy data due to a fault sensor. These situations are defined as anomalous, and might be dangerous for humans. In particular, we focus on a methodology called XPOMCP and presented in [21]. It aims to find anomalous behaviours in the policy computed by POMCP. The main contribution of the thesis are:

- The creation of a web-application to improve the analysis and the presentation of the results of this methodology.
- A comparison between XPOMCP and *Isolation Forest*, a common anomaly detection algorithm on a computationally tractable problem namely *Tiger*, so that a perfect solution can be found, and a real comparison can be done.
- A comparison of machine learning models and XPOMCP to approximate the policy computed by POMCP on another problem domain namely *Velocity Regulation*, for which a perfect solution cannot be found because of the large state space that makes the problem computationally intractable. Therefore, we tried to approximate POMCP, using two models: *Logistic Regression* and *Neural Network*, to verify whether these models are able to imitate POMCP and compare their interpretability with XPOMCP.

These are the main results we discovered:

- XPOMCP is able to identify anomalies better than common anomalies detection algorithms such as Isolation Forest, with an high accuracy up to 99%.
- XPOMCP is also able to approximate the policy computed by POMCP better than common machine learning algorithms (e.g., Logistic Regression, Neural Network).
- Comparing the interpretability of XPOMCP with respect to Logistic Regression and Deep Neural Network, we found out that XPOMCP is the most interpretable and accurate model. We also found out that the coefficients of the Logistic Regression model are interpretable, as described in Section 4.4.2. While we were not able to find a good interpretation of the Deep Neural Network weights, even though it is able to approximate POMCP with an high accuracy.

Chapter 2

Background

2.1 Markov Decision Process

Markov Decision Process, (MDP) is a framework widely used in Artificial Intelligence to formulate the problem of decision making under uncertainty. In AI, an agent can be seen as an autonomous entity performing actions in a well-defined environment. MDP works on discrete time systems, although it can be extended to work on continuos time systems. A discrete time systems evolves in time steps, starting from time step t_0 in the initial state s_0. In an MDP the outcome of an action is modeled as a probability distribution over the possible future states to formalize the intrinsic uncertainty of these systems. Whenever the agent acts, it receives a reward. In order to model an MDP, a common representation is the state transition graph. It is a graph based representation, where each node is a state, and each edge represents a possible outcome of an action. In Figure 2.1 the life cycle of an MDP is shown. In Subsection 2.1.1 a formal definition of the problem is given. While in section 2.1.2 some ideas and algorithms to solve MDP are presented. For a survey on MDP techniques, see [16].

2.1.1 Problem Formalization

Definition 1 *A Markov Decision Process (MDP), is a tuple* $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$*, where*

- $\mathcal{S}$ *is a finite set of states of the world.*
- $\mathcal{A}$ *is a finite set of actions.*
- $T : \mathcal{S}X\mathcal{A} \rightarrow \Pi(\mathcal{S})$ *is the transition function, it maps a state and an action to a probability distribution over the set of states.*
- $R : \mathcal{S}X\mathcal{A} \rightarrow \mathbb{R}$ *is the reward function, which defines the immediate reward achieved by selecting an action* $a \in \mathcal{A}$ *in state* $s \in \mathcal{S}$

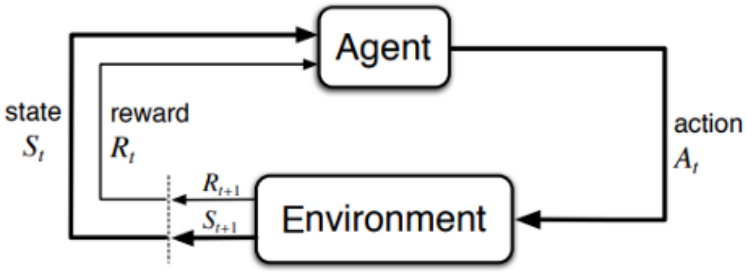


Figure 2.1: Representation of an MDP. (Reinforcement Learning with Exploration by Random Network Distillation, 2021)

Solving an MDP consists in finding an optimal policy, which is a mapping between state and actions, such that it maximizes the sum of the rewards achived by the agent during its execution. In particular, an agent that acts indefinitely must maximize its discounted return, defined as:

Definition 2 *The infinite horizon discounted return is the expected sum of reward during the agent's life.* $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t\right]$

The parameter $\gamma \in [0, 1)$ specifies how much value to give to the reward at each step. The larger γ is, the higher is the impact of future rewards on the current decision. A policy is defined as:

Definition 3 *A policy* $\pi : \mathcal{S} \to \mathcal{A}$ *is a mapping from the state space to the action space.*

There are two types of policies:

- Stationary policy π. It stays constant as system evolves.
- Non stationary policy π_t is a policy which changes as the system evolves.

As already defined in this Subsection, in order to solve an MDP we must compute an optimal policy π^*, that is a map between a state and the optimal action for that state. For this task, we define the value of a policy in a given state s as:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}(s') \tag{2.1}$$

In equation 2.1 the *Value Function* is defined, which is used to evalute a policy π used in a system from state s. The problem is to find an optimal policy given a Value Function V, defined as:

$$\pi^*(s) = \arg\max_a \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s') \right] \tag{2.2}$$

However, normally, the Value Function of the policy is not available, and needs to be computed. In every MDP problem there exists a policy $\pi^*(s)$ which is optimal for every starting state [13]. The Value Function associated to this optimal policy, is:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right] \tag{2.3}$$

2.1.2 Solution Techniques for MDPs

In order to solve an MDP, many algorithms are available, two popular algorithms are *Value Iteration* [6], and *Policy Iteration* [13]. The idea of Value Iteration is to use the *Bellman Optimality Equation* [6] which is defined as:

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(s',a')] \tag{2.4}$$

to update the Value Function at each step. Value Iteration computes the value for a given policy π, and makes the policy π greedy with respect to V^π. This method, is also known to be a *Dynamic Programming Algorithm*. Value Iteration, at each time step t, computes Q_t^a, which, given a state s and an action a, is the corresponding reward for taking action a in state s. Therefore, Q_s^a is the reward received at time t, summed with the future rewards the agent obtains for taking actions in the enviroment with the policy computed at step $t-1$. The algorithm terminates after k iterations, or if $|V_t^\pi(s) - V_{t-1}^\pi(s)| < \epsilon$ for each state s. The policy is then built using the greedy policy with respect to the Value Function returned by Value Iteration. Value Iteration always converges to the optimal Value Function. Complexity is $\mathbf{O}(s \cdot a)$ per iteration, while the convergence rate is linear. Policy Iteration [13] computes the policy and the utility values simultaneously. The algorithm is composed of two sub-algorithms, *Policy Evaluation* and *Policy Improvement*. It starts with a random policy π, Policy Evalutation returns the utility values $v(s)$ of the policy π and Policy Improvement updates π as if the values $v(s)$ were correct. Policy Iteration, iterates Policy Evaluation and Policy Improvement until no improvements are possible. Once no improvement is possible, the policy π is guaranteed to be the optimal policy π^*. Policy Iteration converges in few iterations, but each iteration is expensive, because Policy Evaluation must solve n equations, with n unknowns, $\mathbf{O}(n^3)$. Therefore, the solution is to use Value Iteration with a fixed π using the Value Function computed the last time Value Iteration has been executed [26]. Doing so, a Policy Evaluation step is performed. Then, Policy Improvement is used to update the policy π until convergence. This process is repeated until convergence. Normally this implementation, converges much faster than Policy Iteration or Value Iteration.

Algorithm 1: ValueIteration

Input: An MDP problem

Output: Value function V

$V_1(s) := 0$ for all $s \in S$;

$t := 0$;

repeat

  $t := t + 1$;

  for $s \in S$ **do**

    for $a \in A$ **do**

      $Q_t^a(s) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{t-1}(s')$;

    end

    $V_t(s) := \max_a Q_t^a(s)$;

  end

until $|V_t(s) - V_{t-1}(s)| < \epsilon$;

Algorithm 2: PolicyEvaluation

Input: An MDP problem, Value function V, Policy π, Threshold θ

Result: Utilities values V(s)

repeat

  $\Delta := 0$;

  for $s \in S$ **do**

    $V_{t-1} := V(s)$;

    $V(s) = \sum_{s' \in S} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V(s) \right]$;

    $\Delta := \max(\Delta, |V_{t-1} - V(s)|)$;

  end

until $\Delta < \theta$;

Algorithm 3: PolicyImprovement

Input: An MDP problem, Value function V, Policy π, policy_stable

Result: Policy π, policy_stable

$policy\_stable := true$;

for $s \in S$ **do**

  $\pi_{t-1} := \pi(s)$

  $\pi(s) := \arg\max_a \sum_{s \in S} T(s, a, s') \left[R(s, a, s') + \gamma V(s')\right]$;

  if $\pi_{t-1} \neq \pi(s)$ **then**

    $policy\_stable := false$;

  end

end

Algorithm 4: PolicyIteration

Input: An MDP problem, a threshold θ

Result: Optimal policy π^*

$V(s) \in \mathbb{R}$ arbitrarily for all $s \in S$;

$\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in S$;

while *policy_stable = false* **do**

  PolicyEvaluation(MDP, V, θ);

  PolicyImprovement(MDP, V, π, policy_stable) ;

end

2.2 Partially Observable Markov Decision Processes

Partially Observable Markov Decision Process, (POMDP) is an extension of MDP, used to model partially observable environments. The state of the system is not known, instead a belief is used to represent the probability distribution over possible states. While in MDP after performing an action, the agent knows the current state of the system, in POMDP, when the agent performs an action, the environment returns an observation based on the real (unknown) state of the world. The agent can use the observation obtained to estimate the current state. A naive approach to estimate the state would be to use only the last observation obtained, but this performes poorly [18]. In order to better estimate the state, the agent has to get as many observations as possible and keep them in memory, balancing the cost of performing actions in the world. Based on the context, actions are divided in two types: exploration and exploitation. Exploration actions, are used when the policy is still unknown so rarely used actions are explored. While exploitation actions follow the policy computed. In order to compute an optimal policy in POMDPs, both type of actions are required, as initally a random policy is used, then when the policy is mature enough, actions taken by the policy are used. However, a balancing between exploitation actions and exploration actions is needed, as when the policy is mature enough exploitation actions generally lead to a better result in terms of long range reward. We will see how to balance these two categories of actions while searching for the optimal policy in Section 2.3. Section 2.2.1 provides a formal description of a POMDP. While, in Section 2.2.2 a solution technique to solve exactly POMDP problems is presented.

2.2.1 Problem Formalization

Definition 4 *Partially Observable Markov Decision Process [18] is a tuple* $\langle \mathcal{S}, \mathcal{A}, T, R, \Omega, O \rangle$*, where*

- $\mathcal{S}, \mathcal{A}, T, R$ *are the elements of an MDP.*
- Ω *is a set of observations the agent can obtain from the environment.*
- $O : \mathcal{S}X\mathcal{A} \rightarrow \Pi(\Omega)$ *is the observation function, which given the resulting state and the action, returns the probability distribution of possible observations.*

A common notation for using the observation function is O(s',a,o), which is equivalent to $P(o|a, s')$, that is the probability of receiving observation o from the resulting

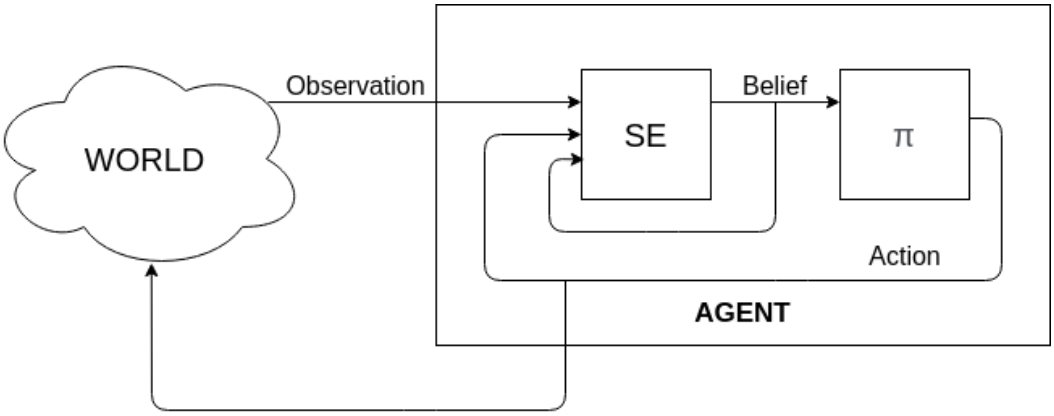


Figure 2.2: POMDP life cycle

state s', after taking action a, in state s. As MDP, the goal of POMDP is to find an optimal policy which maximizes the long range discounted reward. However, there is a further difficulty in POMDP, which consists in estimating the real current state given the observations obtained by the agent while acting in the real world. For this purpose, POMDP represents all the possible real current state as a probability distribution over all the states, called belief defined as:

Definition 5 *A belief state b of a POMDP $\langle \mathcal{S}, \mathcal{A}, T, R, \Omega, O \rangle$ is a probability distribution over $\mathcal{S}$, such that $0 \leq b(s) \leq 1$ and $\sum_{s \in S} b(s) = 1$.*

The belief is updated as the agent receives observations while acting in the environment. The belief is updated by the state estimator component in the POMDP life cycle, shown in Figure 2.2 The component state estimator takes in input the observation, the action, and the previous belief state. It returns a belief state as output. This probability distribution maps the uncertainity of what is the current state between all the possible states. Using probability distribution over other possible outputs, has the advantage of intrinsically modelling the past actions and observations without any need to memorize them. Specifically, no additional data of the past execution would help the agent to achive an higher reward, because the process is Markovian (thus the probability distribution on the possible future states depends only on the current state). In order to compute the new belief state, given the action, the current belief state and the observation, probability theory has to be

applied:

$$b(s') = Pr(s'|a,o,b) = \frac{Pr(o|s',a,b)Pr(s'|a,b)}{Pr(o|a,b)} = \frac{O(s',a,o)\sum_{s\in S} T(s,a,s')b(s)}{Pr(o|a,b)}. \tag{2.5}$$

Equation 5.2, shows how the new belief state can be computed by the state estimator.

2.2.2 Computing an Optimal Policy for POMDPs

The problem of finding a good policy given the current belief remains. Continuous MDPs are helpfull in this situation, because, thanks to them an exact policy for POMDP can be found.

Definition 6 *A continuous space MDP is defined as a tuple $\langle \mathcal{B}, \mathcal{A}, T, R\rangle$, where $\mathcal{B}$ is the set of belief states, $\mathcal{A}$ is the set of actions, τ is the transition function, and ρ is the reward function.*

The transition function $\tau(b,a,b')$ returns the probability of receiving the belief b' taking action a having the current belief b, mathematically $\tau(b,a,b') = Pr(b'|a,b)$. The reward function $\rho(b,a)$ is the reward obtained by the agent for taking action a given the belief b.
To find an optimal policy we focus on the finite horizon problem, as the case of infinite horizon can be deducted from the solution to the finite horizon. The idea is to find an approximation of the optimal Value Function using Value Iteration 1. Particularly Value Iteration, at each step finds the $t-step$ discounted Value Function over belief space. A t-step policy is represented as a tree, where each node represents an action, and each edge represents the observation the agent can receive given the belief b and action a. The root node is always an action, then depending on which observation the agent get, the next action is choosen following the edge representing the observation obtained.
Given this representation the Value Function for the policy π can be computed as:

$$\begin{aligned} V_p(s) = R(s,a(p)) + \gamma \sum_{s'\in S} Pr(s'|s,a(p)) \sum_{o_i\in\Omega} Pr(o_i|s',a(p)) V_{o_i(p)}(s') = \\ R(s,a(p)) + \gamma \sum_{s'\in S} T(s,a(p),s') \sum_{o_i\in\Omega} O(s',a(p),o_i) V_{o_i(p)}(s') \end{aligned} \tag{2.6}$$

The interesting part of equation 2.6 is how the expected value of future rewards is computed. First an expectation over all next states is found, (i.e

$\gamma \sum_{s' \in S} Pr(s'|s, a(p)) \sum_{o_i \in \Omega} Pr(o_i|s', a(p)) V_{o_i(p)}(s'))$ then the value for each next state s', which itself depends on the executed action is found. Note that, the action executed also depends on which observation the agent obtains. Thus, the value of state s' is found taking an expectation over all possible observations executing the subtree $o_i(p)$, starting from state s'. The value of executing a policy tree p can be found given the belief b by

$$V_p(b) = \sum_{s \in S} b(s) V_p(s) \tag{2.7}$$

In order to find the optimal value function for the t-step policy tree, a more compact form of equation 2.7 can be used:

$$\begin{aligned} \alpha_p &= \langle V_p(s_1) \dots V_p(s_n) \rangle [18] \\ V_p(b) &= b \cdot \alpha_p \end{aligned} \tag{2.8}$$

Equation 2.8 allows to find the optimal value function between all the policies. $\mathcal{P}$ is the set of all possible policies. Then, the t-step optimal value function is given by

$$V_t^*(b) = \max_{p \in P} b \cdot \alpha_p \tag{2.9}$$

It is possible that every policy tree in the belief space is the best policy for a belief b. For this reason, it is usefull to represent policy trees as set of vectors. However, it is unlikely that every policy tree is the best behaviour for a belief b. In fact, most of the time, many value functions related to their policy trees are dominated by other value functions. Hence, the dominated policy trees and their relative value functions can be discarded, keeping only the sub-optimal value functions, whose are the one contributing to compute the optimal value function. Using this idea, given a set of policy trees $\tilde{\mathcal{V}}$, a unique and minimal subset $\mathcal{V}$ can be found. It represents all the useful policy trees which contribute to compute the value function. Using this idea, a region $\mathcal{R}$, over belief space where the vector α dominates can be defined as:

$$\mathcal{R}(\alpha, \mathcal{V}) = \{b | b \cdot \alpha > b \cdot \tilde{\alpha} \text{ for each } \tilde{\alpha} \in \mathcal{V} - \alpha \text{ and } b \in \mathcal{B}\} \tag{2.10}$$

In order to compute the value function for a POMDP with finite horizon, value iteration can be used. The challenging aspect is to find a good minimal and unique subset $\mathcal{V}_{\text{t}}$ given the set of policy trees $\mathcal{V}_{\text{t-1}}$. Authors in [30] propose an algorithm to find the optimal subset $\mathcal{V}_{\text{t}}$. Their algorithm is composed of two parts: generation and pruning. The generation part consists in creating a superset $\mathcal{V}_{\text{t}}^+$ of policy trees from the optimal set $\mathcal{V}_{\text{t-1}}$ computed at step $t-1$. This superset contains all the possible policy trees which root node has action a and $|\Omega|$ subtrees. The number of elements in $\mathcal{V}_{\text{t}}^+$ is $|\mathcal{A}||\mathcal{V}_{\text{t-1}}|^{|\Omega|}$. So the complexity for the generation algorithm is

$\mathbf{O}(|\mathcal{A}||\mathcal{V}_{\text{t-1}}|^{|\Omega|})$. Instead, pruning, consists of finding the region $\mathcal{R}$ which can be efficiently computed using a linear programming approach. Since the pruning algorithm requires the execution of a linear program, the complexity class in unchanged. The complexity shows that if the number of elements of the observation space is high, this algorithm becomes computationally unfeasible. Another approach, the *witness algorithm* reduces the complexity of the generation algorithm. The main idea of reducing the complexity is to avoid creating the superset $\mathcal{V}_{\text{t}}^{+}$, creating directly the set $\mathcal{V}_{\text{t}}$. However, computing directly $\mathcal{V}_{\text{t}}$ is impracticle. The witness algorithm, instead, creates a set of policy trees $\mathcal{Q}_t^a$, which have as root node the action a. Then, $\mathcal{V}_t$ is just the union of all $\mathcal{Q}_t^a$ for each action. The witness algorithm is used to create the set $\mathcal{Q}_t^a$ polynomial in $|\mathcal{S}|$, $|\mathcal{A}|$, $|\Omega|$, $|\mathcal{V}_{t-1}|$, $|\mathcal{Q}_t^a|$. Specifically, the witness algorithm runs in polynomial time in the size of the input, the output, and the intermediate results it uses. $\mathcal{Q}_t^a$ can be found by:

$$\mathcal{Q}_t^a = \sum_{s \in \mathcal{S}} b(s) R(s,a) + \gamma \sum_{o \in \Omega} Pr(o|a,b) V_{t-1}(b'_o) \tag{2.11}$$

where b'_o is the belief after having taken action a and received observation o. The value function given the belief b and $\mathcal{Q}_t^a$ can be found by:

$$V_t(b) = \max_a Q_t^a(b) \tag{2.12}$$

Algoritmh 5 returns the optimal t-step policy tree. In Algorithm 5 the generation algorithm is not explicit, it is represented with *witness*$(\mathcal{V}_{t-1}, a)$ instead, and it is also called *witness inner loop*. Therefore, Algoritmh 5 is named after: *Witness Outer Loop*

Algorithm 5: WitnessOuterLoop

Input: POMDP, ϵ

Result: t-step Policy Tree

$V_1 := \{\langle 0, 0, \dots \rangle\}$;

$t := 1$;

repeat

  $t := t + 1$;

  for $a \in A$ **do**

    $Q_t^a := witness(\mathcal{V}_{t-1}, a)$;

  end

  prune $\bigcup_a Q_t^a$ to get $\mathcal{V}_t$;

until $\sup_b |V_t(b) - V_{t-1}(b)| < \epsilon$;

2.3 Monte Carlo Tree Search

Value Iteration for POMDP presented in Section 2.2 scales poorly as soon as the number of elements in the observation space $\mathcal{O}$ is large. This is due to the *curse of dimensionality* and *curse of history*. Curse of dimensionality is a common problematic which is present in many fields such as combinatory analysis, machine learning, statistics and others. In POMDP curse of dimensionality refers to the fact that the number of discrete beliefs grows exponentially with the number of states. While, curse of hystory is the difficulty of keeping in memory the number of action-observation under consideration, which grow exponentially with the planning horizon. Specifically a hystory is defined as:

Definition 7 *A hystory of a POMDP is a set* $(a_1, o_1, a_2, o_2, \dots a_t, o_t)$ *containing the action-observation hystory of the agent.*

The belief, is a sufficient statistic that contains all the information of the history. To overcome the limitation imposed by the curse of history and the curse of dimensionality, we consider approximate and online algorithms. In particular, in this work we focus on *Partially Observable Monte Carlo Planning, (POMCP) [29]*, an algorithm that employs *Monte Carlo Tree Search (MCTS)* to achieve high perfomance in many real world instances.

2.3.1 Monte Carlo Tree Search Definition

The hystory of actions and states, can be represented by a tree. There is one node for each state (a real state of the system). Each node contains three informations: $Q(s,a)$, $N(s,a)$, and $N(s)$. $Q(s,a)$ is the value of taking action a from state s. $N(s,a)$ is the counter visit which counts the number of times the simulation chooses action a from state s. Finally, $N(s)$ is the number of times state s has been visited, that is $N(s) = \sum_a N(s,a)$. The algorithm is divided in four phases ordered by execution: Selection, Expansion, Simulation, Backpropagation. The first phase, Selection, consists in following the policy computed till now, to choose actions from the root node, till reaching a leaf node. An example of a policy is to choose the action with the highest $Q(s,a)$ value. Once a leaf node has been reached, the Expansion phase starts. If the current leaf node has never been visited before, all the possible states reachable from the current leaf node are added to the tree. Then, the current state is set as the first child added, no matters what it is. From here the simulation phase starts. Simulation consists in simulating the behaviour of the agent using a random policy, from the current state, till a termination state is reached, then the

last phase can start. Backpropagation, consists in updating the values of the leaf parents, by the values found in the simulation. At the end of the process, each node will have a value $Q(s, a)$ which can be seen as the expected reward from that state choosing action a. The policy is then found by choosing action a from state s such that to maximize $Q(s, a)$. However, if $Q(s, a)$ is updated considering only the long term reward, only exploitation actions will be choosen. Instead, exploration actions might lead to a higher long term reward. Therefore, a better way to update the value $Q(s, a)$, is using a custom function with a parameter c, called exploration parameter which increases the value of actions that are rarely selected in a certain state s. The update is defined as:

$$Q^+(s, a) = Q(s, a) + c\sqrt{\frac{logN(s)}{N(s, a)}} \tag{2.13}$$

This update is called *Upper Confidence Trees, (UCT)*, [15] algorithm. The interesting part of this update is that the term under square root is used to make the tradeoff between exploration and exploitation. We want $N(s)$ to have a logarithmic grow, therefore $logN(s)$ is present at the numerator, while the term $N(s, a)$, is used to give value to actions that have never been choosen, or choosen less with respect to others. If an action has never been explored, the value $Q^+(s, a)$ tends to grow, so the algorimth chooses an action that has been choosen few or no times, exploration. While, when the term $N(s, a)$ is high, $Q^+(s, a)$ has a decreasing tendency, therefore, exploitation. Particularly the search tree strategy implemented by MCTS is UCT. UCT stands for *Upper Confidence Tree* , which expands the nodes doing a tradeoff between exploration and exploitation. Theoretically with a proper value of c, UCT is proven to converge to the opitmal policy, $Q(s, a) = Q^*(s, a)$.

2.3.2 Partially Observable Monte Carlo Tree Search

In order to get closer to our problem, which is find an approximal policy for a POMDP using MCTS, we must exend MCTS to handle partially observable environments. To begin, each node contains a history instead of a state. Each node contains similar information to the fully observable case: $N(h, a)$, $V(h)$. $N(h)$ is the counter of times the history h has been visited. While $V(h)$ is the value of history h, computed as the avarage reward of simulations starting from history h. The algorithm works as the fully observable case described in Subsection 2.3.1. The first phase is Simulation, which use again the UCT algorithm to expand more valuable nodes. UCT uses the *Upper Confidence Bound version 1, UCB1* formula, to

compute the value of a history followed by an action:

$$V^{+}(h,a) = V(h,a) + c\sqrt{\frac{logN(h)}{N(h,a)}} \tag{2.14}$$

In the second phase, simulation, a rollout policy is used (e.g random uniform policy). After simulation one new node, corresponding the the first history encountered during simulation is added to the tree. When the Backpropagation phase starts, the belief for each state needs to be updated. The update method using the Bayes' theorem could be use 5.2. However, if the state space is large, it is too computationally expensive. One way to solve this issue is to use a particle filter to represent the probability distribution and so to update the belief. Partially Observable Monte Carlo Tree Search uses an unweighted particle filter, because it can be implemented using a black box simulator. The belief of state s, given history h_t, is the sum of all particles, $\widehat{B}(s,h_t) = \frac{1}{K}\sum_{i=1}^{K}\delta_{sB_t^i}$. At the beginning K particles are sampled from the distribution. Once the real agent acts in the environment with an action a, and obtains an observation o the belief is updated. Particularly, a state s is sampled randomly from the belief B_t. This particle is passed in the black box simulator, to get observation o' and s'. Then the real agent acts in the world, and if it obtains an observation $o = o'$ then a new particle is added to the B_{t+1} because that is more probable to be the right state. This algorithm repeats until K particles have been added. Then an approximate belief is obtained. It is proven that if the number of particles $K \to \infty$ then, $\widehat{B}(s,h_t) = B(s,h_t)$ [29].

2.3.3 Partially Observable Monte Carlo Planning

This Subsection presents *Partially Observable Monte Carlo Planning, POMCP* [29]. POMCP uses all the techniques described in Subsection 2.3.2. Particularly, POMCP, uses the partially observable UCT search, with the hystory instead of belief as node in the tree, and the belief update using particle filter. In POMCP, each node contains three information: $N(h), V(h), B(h)$. The novel contribution is the added information $B(h)$, which contains the particles for history h. POMCP works as follows, a start state is sampled from the belief $B(h_t)$, then the simulation uses the UCT algorithm to exapand only valuable nodes. The belief $B(h)$ is updated for every history seen during the simulation. When the simulation reaches a terminal state, the action a with the highest $Q(s,a)$ value is choosen by the agent, which act in the world and obtain a real observation o_t. The node (h_t, a_t, o_t) becomes the new root, and the belief $B(h_tao)$ becomes the new belief state. Figure **??** shows how POMCP works in an environment. Particularly, the tree on the left, is called search

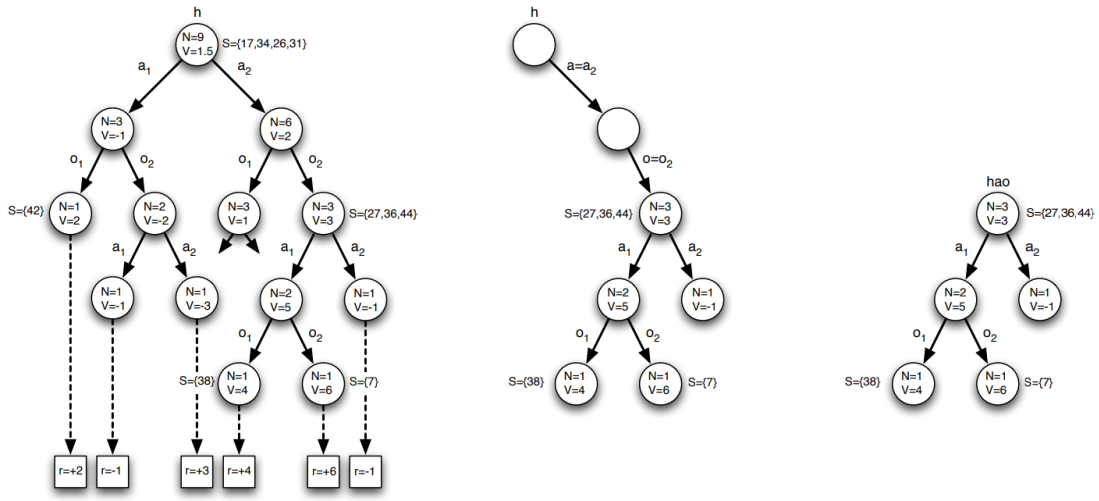


Figure 2.3: An example of a POMCP execution in an environment with 2 actions, 2 observations, and 50 states. [29]

tree and is constructed, doing many simulations by Algorithm 8. Then, the search tree is used to select a real action a, so the agent execute action a, obtains a real observation o and a real reward r. The tree is then pruned, and one new node is added to the tree, and a new search begins from the updated hystory *hao*, tree on the right.

Algorithm 6: Search

Input: The hystory h

Result: t-step Policy Tree

repeat

  if $h = empty$ **then**

    $s = \mathcal{I}$

  end

  else

    $s = \mathcal{B}(h)$

  end

  simulate(s,h,0);

until *timeout()*;

return $\arg\max_b V(hb)$;

Algorithm 7: Rollout

Input: State s, History h, Depth d **if** $\gamma^{depth} < \epsilon$ **then**

  return *0*

end

$a = \pi_{rollout}(h, \cdot)$;

$(s', o, r) = \mathcal{G}(s, a)$;

return $r + \gamma \cdot rollout(s', hao, depth + 1)$;

Algorithm 6 implements the POMCP algorithm. Particularly, it uses two algoritms, Algorithm 7 and Algorithm 8. Algorithm 7 is used to simulate the behaviour of an agent using a blackbox simulator, using a rollout policy, which might be a random uniform distribution between all actions. While 8 describe how to add nodes to the tree, adding node in the tree, calling the rollout algorithm, and backpropagating the result backward.

Algorithm 8: Simulate

Input: State s, History h, Depth d

if $\gamma^{depth} < \epsilon$ **then**
  return *0*
end
if $h \notin T$ **then**
  for $a \in \mathcal{A}$ **do**
    $T(ha) = (N_{init}(ha), V_{init}ha, 0)$
  end
  return *rollout(s,h,depth)*;
end
$a = \arg\max_b V(hb) + c\sqrt{\frac{logN(h)}{N(hb)}}$;
$(s', o, r) = \mathcal{G}(s, a)$;
$R = r + \gamma \cdot simulate(s', hao, depth + 1)$;
$B(h) = B(h) \bigcup s$;
$N(h) = N(h) + 1$;
$N(ha) = N(ha) + 1$;
$V(ha) = V(ha) + \frac{R - V(ha)}{N(ha)}$;
return R

2.4 Explainability

Decision making by artificial neural network, or complex planning algorithms might have an high accuracy, but often humans cannot understand the reason behind the decision process. [8]. However, a deep understanding of how decisions are made is crucial in many fields such as healthcare, finance, automotive. This Section presents some concepts about *EXplainable AI, XAI* and the main issues in this field.

2.4.1 EXplainable AI, Basic Concepts and Main Issues

For some sensitive domains, a detailed understanding of how the model works, and how it produces certain results is crucial. For example a deep learning model to classify lung cancer, must have an high accuracy of course, but also, its results must be interpretable, thus, we must be sure of how it predicts the area of the tumor if there is any. Another example is a self-driving car, driven by an agent executing a policy computed by a planning algorithm, we must ensure that each action planned in the policy is safe so that it does not lead humans to dangeruos situations. Although explainability is important in many domains, it should not incorporate all domains. There are fields such as aircraft collision avoidance, which works just fine without any human intervention, therefore, in that case no explaination is necessary. So when explainability is required? It is required when there is some degree of *incompleteness*. Incompleteness is often confused with *uncertainty*, but these are two different concepts. Uncertainty can be formalized by mathematical models. On the other hand, incompleteness, means that there is something about the problem that cannot be encoded into the model [11]. In the previous example of the self-driving car there is a certain incompleteness with regards to whether the test environment is a suitable model for the real world. Furthermore, decision making can determine which advertisement one sees, which job interview one is accepted to. Justify these decisions is important for the human, but difficult for the machine. Explainability is demanded whenever the goal for which the prediction model was constructed for differs from the actual usage when the model is being deployed [17]. In other words, explainability is necessary whenever there is a mismatch between what a model can explain, and what the human wants to know. Explainability is also necessary to explain why the model performs poorly with respect to some instances. European Union is also interested in explainibality, according to their renwed *EU General Data Protection Regulation (GDPR)*, it could require AI providers, to provide users explainations of the results of automated decision-making based on their personal data [8]. This could lead the industry to avoid the use of opaque systems, like com-

plex neural network, thus, resulting in the decreasing of accuracy, but increasing of interpretability. The *Association for Computing Machinery* published a paper [28] on algorithmic transparency and accountability. They underlined that automatic decision making can be harmful, and could lead to discrimination. To avoid this issue, they also published a list of principles to follow. Last but not least, the *DARPA* launched the program *XAI*, which stands for *EXplainable AI*, and is used to refer to system powered by AI, with an high accuracy, but that are also explainable. The reader might have noticed how the term *explainable, interpretable, and understandable* are used interchangeably. However, often the term *interpretability* refers to the comprehension of how the model works. While *explainability* is used when a model is also expected to give an explanation of its decisions although the model is incomprehensible. First, what an explanation is must be defined: an explanation is the answer to a why question [22] Moving on, among the reasons of why explainability is needed, the following can be found: *Trust, Casuality, Transferability, Informativeness, Fair and Ethical Decision Making, Accountability, Making adjustments.* In this work, they are defined as:

- **Trust**: Understand the strength and the weaknesses of the model is needed for the human to trust the model itself.
- **Causality**: It is needed for the human to understand the relationship between input-output.
- **Transferability**: The decision model needs to generalize well, so that the human is sure it will be good also with unseen data.
- **Informativeness**: It is necessary to know whether the system reaches its goal before deploying the system to the world.
- **Fair and Ethical Decision making**: The system needs to present results in a way such that the fair and ethical standard can be verified.
- **Accountability**: The decision making system must be responsible for its actions. In order to make it do so, make it explain its decisions and justify them is necessary.
- **Making adjustments**: Because the world is constaly changing, also decision making systems must be adapted. To know what needs to be changed in the system, the model needs to be interpretable.

How can we make the model explainable? Some approaches consists in replacing the inital model, with one more interpretable, such as decision tree [12]. Another approach would be to make the learning phase of the model more explainable, or just focus on explaining feature prediction [27]. There are many approaches for implementing interpretability: *ante-hoc, post-hoc, instance/local, model/global, specific,*

agnostic, data independent, data dependent. A brief definition for each of them is given below:

- **Ante-hoc**: Interpretability is built from the beginning of the data creation.
- **Post-hoc**: Interpretability is created after model creation
- **Instance/Local**: Interpretability only holds locally for a single data instance and its close vicinity
- **Model/Global**: Interpretability holds globally for the entire model
- **Specific**: The mechanism for gaining interpretability works only for a specific model class
- **Agnostic**: The mechanism for gaining interpretability is generally applicable for many or even all model classes
- **Data Independent**: The mechanism for gaining interpretability works without additional data
- **Data Dependent**: The mechanism for gaining interpretability requires data

These definitions have been reported because later, a novel approach for interpretability in planning is presented, therefore it is usefull to classify at which area it belongs to. Furthermore, this thesis also makes a comparison between the novel method based on rule, and linear regression methods used as approximation to the planning algorithm. Thereofore, a justification of why a linear model has been used have to be given. There are models, which are intrinsically interpretable, such as linear models, decision trees, k-nearest neighbours, k-means, and other machine learning modeld. Linear models are composed of input features, and weights for each of them. The weight is assigned to the feature such that to minimize an error function such as the *Root Mean Squared Deviation (RMSE), Mean Absolute Error (MAE)*. The prediction is compute by the sum of the feature and its weight for all features. The model is easy to interpret, because each weight tells the importance of the feature. The higher the weight, the more important the feature is. The opposite is also true. Thus linear models are interpretable by nature. Another machine learning algoritmh used as term of comparison is K-Means. K-means is a clustering algorithm whose task is to find group of similar points in the dataset. Prior information is used as the number of groups the algorithm needs to find, then a metric distance is used to classify the points as one belonging to certain groups or not. Also this algorithm is interpretable by design. In fact, it is clear how the groups are created. However, there are some algorithms like *Artificial neural networks ANN*, which by design are opaque, and therefore are used as black-box. A key challenge is to find methods to explain how they make decisions, when the model is too complex. Normally to a complex model corresponds an high accuracy, how-

ever, the more complex the model is the less an interpretation of its decision can be deducted. This is valid not only for ANN, but also to planning algorithms such as POMCP. In cases like these, where the model is not interpretable, the decisions the algorithm makes must be explained. . Furtheremore, the quality of an explanation depends on three different criteria: relevance, persuasiveness, and comprehensibility [14]. In fact, an explanation is good only if it responds to the question asked. But it should be short so that it does not provide additional information that the user might misunderstand. The explanation should be given in such a way that the user is able to comprehend it. Another characteristic is that it should be be simple, written in a language that the user understands. For this reason the explanation system has to be flexible, if the user does not understand an explanation, she/he might want to ask the system to explain it with different terms. There is a system able to generate a minimal explanation. The user can give a feedback to the system, positive or negative regarding the explanation comprehensibility. If it is negative, the system adds other information to the explanation with the hope to satisfy the user. This process repeats until the user is satisfied [14]. Furthermore, also the type of explanation given must be considered. The explanation type might be based on the type of data the algorithm is working. For example, working with images is different than working with tabular data. For this reason, also a study on what explanation has to be given is extremily important. Another key point is what to explain. A theory says that only abnormal decisions should be justified [17]. Some basic questions to pose when explainability have to be applied are: *What to explain*, *How to explain*, and finally, *to Whom is the explanation addressed*. Depending on what wants to be explained, different output the explanation system may return. The different type of explanation are present below, with a brief description.

- **Local Explanation**: Provide an explanation to a specific decision.
- **Counterfactual explanation**: Always provide an explanation to a decision, but with ground facts, in order for the user to know how to get a different decision based on different data the user has to produce.
- **Global explanation**: Focuses on returning an explanation of how the system works in general, what the system focuses on more.

There are also other type of explanations that in this thesis have not be considered. Moving on to the second question, *How?*, it is all matter of communication. It is important to understand what type of explaination suits the best to the problem. For example, describing using graphs instead of text, might not only be easier, because math is solvable by the algorithm, but also, the problem of dealing with *Natural Language Processing (NLP)* is avoided. There are three type of methods to

answer represent an explanation: *Textual Description, Graphics, Multimedia.*

- **Textual Description**: create a text explanation. This mimic humans behaviour to justify decisions. However, creating a good text explanation might be hard.
- **Graphics**: The use of graphs, or for example boxes in an image are used as explanation. This approch tries to explain what the model has learnt.

- **Multimedia**: This type is a mix of the previous types, textual description and graphics.

To answer the last question, to who, we can classify groups of people based on their experience in the field of AI. This allows to have a "custom" answer based on the prior knowledge of the user. For example, a non expert, cannot know what a reward is, or what a loss function is. Instead, an expert might be interested in those details, if they are helpfull to answer the questions. The following groups have been considered: *Non Expert, Domain Expert, System Developer, AI-Developer.*

- **Non Expert**: This type of user does not have neither any technical nor domanin knowledge. His goal is to receive a justification of the decisions.
- **Domain Expert**: This user has some prior knowledge about the application domain. His goal is to gain trust in the system by receiving a justification that agrees with his prior domain knowledge.
- **System developer**: It is a techinical user who also builds the system, but she/he does not have any domain knowledge about the problem. His goal is to understand how the algorithm works.
- **Ai-Developer**: She/he is the user who trains the model, he is a techincal expert and has knowledge about the domain.

2.4.2 Explainability in Planning

Experts have a deeper knowledge of what is happening when an agent makes decisions. However, we all posed the question *Is it safe?* for once, referring to a robot powerd by an intelligent agent. An agent for being safe would have to proove that actions taken are always safe, and do not lead damage to people or things. The problem is that prooving these facts is not easy. This section is more focused on explainibility for planning algorithms. As explained in Section 2.4.1, first we need to pose the basic questions: What, how, who.

- **What**: We want to understand the reason that leads an agent to select a certain action. All the actions are derived from a policy. There are two approach for this, explain single actions and explain the whole policy.

- **How**: Explainability in planning can be represented both using graphical tools such as plots, but also using text. An example of the explanation to the question *"Why are you going fast"* of a robot moving in a room is *"I am moving fast because the path is clear from obstacle"*.
- **Who**: The explanation of actions deriving from planning algorithms may be issued to domain knowledge expert, so that they can understand whether something in the agent is wrong, and to the AI-Expert.

An important topic of explainability in planning is *Contrastive Questions*. These are questions that may be posed to the agent in order to understand its behaviour. A Contrastive Question takes has the form of: *Why did you* $\boldsymbol{f}$ *rather than* $\boldsymbol{c}$*?*. Instead a "normal" question is *Why did you do f*. Using a contrastive question we can explain contrastive facts instead of plain facts [31]. If we think about this idea, explaining a decision using a contrastive fact is normal, as the user wants to trace relationships between facts. In planning, the use of Contrastive Questions to explain abnormal behaviours of the agent is common, although the answer the explanation system returns has to underline, in a clear fashion, the explanation, and it must be comprehensible by the human user. Some examples of Contrastive Questions are: *Why action* a_1 *instead of* a_2. These kind of questions, however, are difficult to answer, because the planner might respond: *Because with action* a_1 *the agent receives a reward of 5, instead, with action* a_2 *it receives a reward of 1.* This type of answer might not be acceptable, because it does not return any facts on where the difference of rewards comes.

2.4.3 Explainable Partially Observable Monte Carlo Planning

This thesis focuses on the identification of anomalous behaviours by an agent following a policy computed by POMCP. Understanding the reasons behind a decision taken by POMCP is hard, as it is an online algorithm. One approach is to use contrastive questions to interpret its behaviour. However, as already said in Section 2.4.2, answer to these questions in general is difficult. Specifically, being POMCP an online algorithm, there is not a complete representation of the policy. Instead, POMCP builds an approximate, partial policy. Furthermore there is the risk of receiving an answer which is not comprehensible for the human. In Section 3 a novel method created to analyze POMCP policies has been presented [21]. This approach is based on logic rules, used to integrate prior knowledge of the domain by the user, and define some behaviors that the user expects the agent to have. This approach

allows to find unexpected behaviours of the agent, which might be derived from an approximation of the logic rules or from a real anomaly of the agent. We explain all the details of this approach in Section 3.

2.5 Satisfiability Modulo Theroy

2.5.1 Satisfiability Boolean Problem or SAT

Satisfiability Boolean Problem (SAT), is the problem of finding if there exists a solution for a propositional formula. In order to solve this problem a SAT-solver can be used, which try to build a model (i.e., assignment that satisfies a propositional formula) or to prove that no solution exists. This is known to be a NP-complete problem, however, some heuristics can be used to tackle the problem, although heuristics do not always find a solution in util tyme, if one exists. *Systematic search* is a way to represent the problem using a tree, where each node corresponds to a variable whose value needs to be assigned, and each edge represents a possible assignment for that variable. The leaves of the tree are the possible assignment of variables, which make the formula satisfied. This way to solve the problem, however, is computataionally untractable given its NP-complete nature. There is an efficient algorithm to solve SAT, the *DPLL Algorithm*, which consists in three steps: decide, propagate, backtrack. If the search space is too large, thus if too many variables are present, the solver may use an heuristic to expand only the most valuable subtrees. One last aspect of the DPLL algoritmh, is that it expect the logic formula to be in normal form, such as *Conjunctive Normal Form (CNF), Disjunctive Normal Form (DNF)*. Specifically DPPL expect the formula to be in CNF, because in order for the formula to be satisfiable, all the clauses have to be SAT [24]. An example of CNF formula is $\neg p \wedge (p \vee q)$. The most popular procedure employed by state-of-the-art SAT solver is known as *Conflict Driven Clause Learning (CDCL)*, which is considered to be an improvement of DPPLL. When DPLL finds a conflict, it undo the last decision and assing the same literal to the opposite Boolean value. This could lead to poor performance, because a wrong decision made at the beginning of the search procedure is hard to detect in this way. Instead, when CDCL finds a conflict it builds a new clause that explains the reason of the conflict. This clause is used to perform a backtrack that can undo many decisions. In particular, it undo the assignment that was responsible for the specific conflict. Many problems can be formalized using propositional logic, and SAT-solvers can treat them and find a solution. However, certain problems require a more expressive language to be modeled. Take the formula: $p_0 > x1 \wedge p_2 < x2$, SAT solver would not be able to solve this formula, it contains arithmetic operation, that are needed in our case, as we will see in the next Chapter. Although certain mathematical problems are solvable with SAT-solver, for performance reasons we are interested in a more specific solver.

2.5.2 Satisfiability Modulo Theory Solver

Satisfiability Modulo Theory (SMT) [5] a logical formula that combines propositional logic and certain fragments of first order logic. First order logic is semi decidable, but in our case, we only take some fragments that are decidable. The solution for the first order logic can be found using ad-hoc solver such as the *Simplex Algorithm* used for linear artithmetics problems. In order for the solver to be able to find an assigment that satisfy the formula, the form of the formula has to be *differential arithmetic*. Differential arithmetic consists in having arithmetics formulas in the form: $t - s < c$, where t, s, c are free variables. If the formula is in this form, then it is relatively easy for a solver to find a set of values which satisfy the formula, looking for negative cycles in the weighted directed graph. In order for the solver to handle formulas that combines propositional logic and fragments of first-order logic, the solver employs a SAT-solver alongside specilized solvers (e.g., the simplex algorithm for linear real arithmetic). For example, given the logical formula: $\neg(a \geq 1) \wedge (a >= 1 \vee a >= 2)$ is translated into $\neg p_1 \wedge (p_1 \vee p_2)$. First, the SAT-solver tries to find a model for the propositional layer, the specialized modules check if this assigment is correct in their theory. If one of the module finds a conflict in the current assigment, it builds a new clause that explains the conflict and this clauses is then added to the propositional layer. If the SAT solver finds the formula unsatisfaiable then the problem is unsatisfiable. Otherwise it returns the combination of Boolean and first-order values that satisfy the formula

2.5.3 Maximum Satisfiability Problem or MAX-SMT

MAX-SMT is the problem of determining the maximum number of clauses that make the formula, composed of Boolean literals, satisfied. MAX-SMT is defined as a generalization of Boolean Satisfiability Problem, in fact, in case a formula cannot be satisfied, MAX-SMT returns the maximum number of clauses that can be made True by an assignment of Boolean values. In MAX-SMT there are two types of clauses:

- **hard-constraint**: It is a clause that must be satisfied. If it cannot be satisfied then also the formula cannot be satisfied.
- **soft-constraint**: It is a clause that should be satisfied by MAX-SMT. However, if it cannot be satisfied, MAX-SMT should try to satisfy all the other clauses.

It exists also a more general version of MAX-SMT called *Weighted MAX-SMT*, defined as a set of clauses and weights, where each clause has one non-negative

weight assigned. The goal is to find truth values for the variables such to maximizes also the weights. MAX-SMT is a specific version of Weighted MAX-SMT, where each clause has a weight with value 1.

2.6 Anomaly Detection Techniques

2.6.1 Isolation Forest

The other approach studied, is Isolation Forest (IF). IF is based on the fact that anomalies should be easier to isolate than the other data points. IF is an ensemble algorithm, so it creates a specified number of classifiers/regressors to classify/predict the final value. The algorithm recursively generates partitions by randomly selecting a feature, and then selecting a split value for the feature, which of course, must be within the feature range. At the end, all the points will be isolated by the parition, however the first points isolated, are more likely to be anomalies because as already said, it is easier to isolate anomalies, than normal points. When the algorithm ends, each point will be isolated in a partition. In order to classify a point as an anomaly, a score is used. The anomaly score, returns the probability for a point of being an anomaly, and is defined as:

$$s(x, m) = 2^{\frac{-E(h(x))}{c(m)}} \tag{2.15}$$

where m is the size of the sample set, $c(m)$ represents the avarage of $h(x)$ for the sample set, and $E(h(x))$ is the expected anomaly score for point x. In figure 2.4 the isolation forest partition is shown for two points, a normal point, picture to the left, and an anomaly, picture to the right. By this figure, we can deduce how a normal point needs a large number of partitions for being isolated, while an anomaly gets isolated sooner, with a smaller number of partitions.

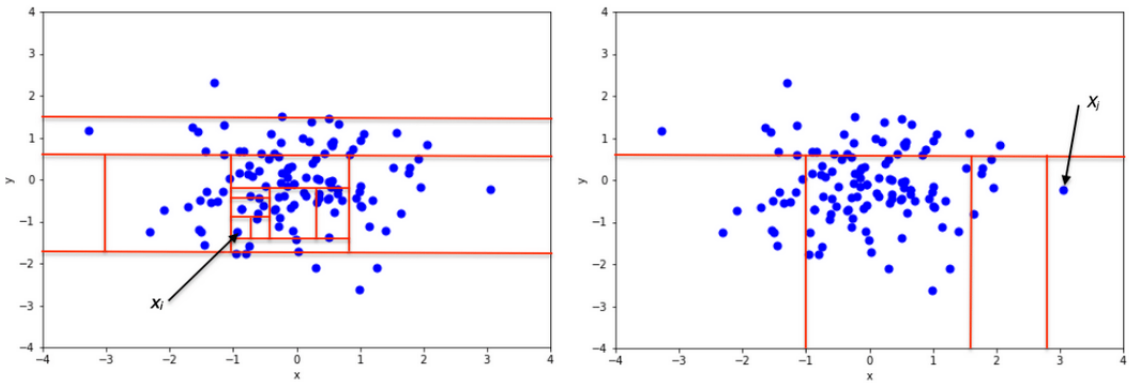


Figure 2.4: Isolation forest for anomaly identification. The picture on the right shows how easy it is to identify an anomaly by using a small number of partions. While the picture on the left shows how for isolating normal points a large number of partitions is required. "Isolation Forest for Anomaly Detection in Machine Learning." TechViz. 20 Apr. 2020. Web. 09 Sept. 2021.

Chapter 3

XPOMCP: A Rule Based Approach

POMCP's online nature of avoiding a complete policy representation leads to a lack of interpretability of its decisions. The result, is the use of POMCP as a black-box. In this thesis, I study *EXplainable POMCP (XPOMCP)* [21] which introduces a novel way of interpreting POMPC policies. XPOMCP analyzes traces which are sequences of belief-action pairs, looking for anomalous behaviours. It generates a logic-based compact representation of the behavior of the system called a *rule*, which allows to express the agent expected behaviour. The rules are generated by combining *rule templates*, logical formula that capture high-level ideas expressed by an expert, with the traces generated by a POMCP agent. Furthermore it is possible to integrate prior knowledge of the domain, into the rule template. Specifically, XPOMCP is able to find unexpected behaviour, namely *anomalies*, that are decisions taken by the policy that violate the high-level insight provided by the expert with the rule template. Further details are presented in Section 3.1 and Section 3.2. The results of anomaly detection of XPOMCP have been compared with a state of the art anomaly detection algorithm namely Isolation Forest. [19]. Furthermore, we also compared how well XPOMCP approximates POMCP with respect to two common machine learning algorithm: Logistic Regression and Neural Network. With Logistic Regression, we tried to Linearize the POMCP behaviour. While using a Neural Network, we tried to verify whether a non linear model could return better results than a linear model used to approximate POMCP. We decided to evalute these methods as we consider also interpretability a key point in deploying safe intelligent agent. Specifically, we compared their interpretability with respect to the instric interpretability of XPOMCP.

3.1 Rule Template Creation

The first step of XPOMCP is posing a question by means of a logic rule using propositional logic. The logic rule is used to express relationships between beliefs and actions. This logic representation is parametric, hence it contains free variables, whose values need to be assigned by a solver. A *rule template* is a logic rule with free variables, while a logic rule with all the parameters value assigned is called *rule*. In this thesis, also the concept of *action rule template* is used. *Action rule template* is a set of rule templates with one rule template per action. Action rule template are used to express constraints using the same variables. The synthetisis process is explained in Section 3.2, while in this Section, we focus our attention on the rule template creation. An expert creates a rule template based also on his prior knowledge of the problem domain. Note how the process is not automatic, the expert has to know the problem domain to pose good questions. This allows this methodology to outperform other anomaly detection algorithms, which do not allow to incorporate prior knowledge. Let's see how a rule is composed. In order for the solver to find the values of the free variables in the rule, rule templates need to follow a precise syntax, which in no way limit the expressiveness of the rule itself. An example of the rule template is:

$$\begin{aligned} \texttt{r:}\ \ & \texttt{select a when } (\vee_i \texttt{ subformula}) \\ & [\texttt{where } \wedge_j \texttt{ requirements}_j] \end{aligned} \tag{3.1}$$

where $\boldsymbol{a}$ is an action choosen among all the actions in the action space, and $\boldsymbol{subformula}$ represents some other logic formula connected in logic OR. $[\texttt{where } \wedge_j \texttt{ requirements}_j]$ is an hard constraint. In MAX-SMT, which is the solver used to assign values to free variables, every formula can contain two type of constraints, soft and hard. Hard constraints are clauses that must be satisfied to make the formula satisfied. Specifically, if even one hard-constraint is not satisfied the whole formula cannot be satisfied. While, soft constraints, are constraints that should be satisfied. A solution for a MAX-SMT formula satisfies as many soft clauses as possible. Going back to Formula 3.1, the part $[\texttt{where } \wedge_j \texttt{ requirements}_j]$ expresses an hard constraint, if there are multiple hard-constraints, they are connected using the AND logic connector. Hard constraints can be used to express prior knowledge, as we will see in the Velocity Regulation problem domain. Furthermore, hard constraints restrict the search space for the solver, this can lead to better performances. The second part which we can notice in the Formula 3.1 is $(\vee_i \texttt{ subformula})$. It is worth noticing what a subformula is. A subformula is a formula expressed using linear real arithmetic that follows a precise syntax: $p_s \approx \bar{\mathbf{x}}$, where p_s is the probability

for s to be the true state in the current belief, $\bar{\mathbf{x}}$ is a free variable, that MAX-SMT has to find the optimal value, and the symbol $\approx$ represents one arithmetic connector choosen among the set of possible connectors (i.e., $\approx \in \{<, \leq, \geq, >\}$). The current implementation supports only linear arithmetic operations that can be used to create logic rules, even though authors are already planning to extend the language for making it more expressible. Subformulas are connected using the OR logic connector, making the time needed to solve the problem exponential in time with respect to the number of literals, in the worst case. Finally, moving to the last part, `r: select a when` , very intuitevily is used to identify which action the agent should take when the conditions expressed in logic formula are met. This part is needed for two reasons: to identify steps of a trace which meet the conditions, but the agent choose another action, violating the rule. The second feature is identifying when the conditions are not met for a step, but the action choosen by the agent is the one analyzed in the rule. We will see later, that for the last case we can actually compute a metric, to quantify how far the conditions are from being met for an anomalous step.

Algorithm 9: RuleSynthesis

Input: A trace generated by POMCP ex, a rule template r

Result: an instantiation of r

solver ← probability constraint for threshold in r;

for *action rule* r_a *with* $a \in A$ **do**

  for *step t in ex* **do**

    build new dummy literal $l_{a,t}$;

    cost ← cost $\bigcup l_{a,t}$;

    compute $p_0^t, \ldots, p_n^t$ from t.particles;

    $r_{a,t}$ ← instatiate rule r_a using $p_0^t, \ldots, p_n^t$;

    if *t.action* $\neq a$ **then**

      $r_{a,t} \leftarrow \neg(r_{a,t})$;

    end

    solver.add($l_{a,t} \vee r_{a,t}$)

  end

end

solver.minimize(cost);

goodness ← 1 - *distance_to_observed_boundary*;

model ← solver.maximize(goodness);

return *model*;

3.2 Rule Synthesis

When the rule template is ready, hence the expert is satisfied with question she/he wants to pose, the next step is assigning a value to the free variables to generate a rule that explains as many of the decisions taken by the policy as possible. The process of finding an optimal assignment for the free variables in the formula is called *rule synthesis*. This process takes in input the trace the expert wants to analyze and the rule template. The algorithm uses a solver, specifically a MAX-SMT solver to find the values of the free variables in the rule template, which satisfy as many trace steps as possible. The solver used is Z3 [10], it is an efficient SMT solver. It has APIs for the most common programming languages such as Python, C++, Java. In our work we used it with Pyhton. The algorithm, which synthetizes the rule is Algorithm 9. First the solver is initialized, and all the hard constraints are added, to force variables to have values between 0 and 1, to make them be real probabilities. In line 2-10, we can notice, two for loops, the outer loop iterates over the actions analyzed in the rule template. Infact a rule template can contain many actions to analyze, and therefore, many rules, one rule template per action. While, the

inner loop iterates over the steps of the trace. In line 4 we generate a new dummy literal, which is a MAX-SMT variable, and in our case is used to satisfy steps that cannot be satisfied by the rule. We use the dummy literal to satisfay artificially the unexplainable step, thus we can converge to a solution even if the rule cannot explain everything. This is part of the MAX-SMT procedure expressed in line 10. We will come back to this later, when line 10 is explained. We need to use a cost function to tell the solver that the goal is satisfying as many steps as possible using the rule template, not using the variable. We will call a fake assignment the action of assigning *true* to the dummy literals. Then, the literal is added to the cost function, (line 5), which basically counts the number of fake assignments. Basically we want to minimize the number of fake assignments, therefore, maximizing the number of satisfied clauses in the rule template. POMCP uses a particle filter to represent the belief distribution states. Therefore, the actual belief needs to be computed, because traces contain only the number of particles for each state. Line 6, computes the belief values for the states from the particles in the trace. Line 7, assign the belief for that step (remember that these set of instructions are inside a for loop over step in traces) to the rule template, substituting the probability in the rule template with actual values that we have just computed. However, if the step action is different than the action which we are iterating over, the rule should not be satisfied, so the rule template is negated, line 8-9. This happens because, if the rule is satisfied for other actions, that is considered an anomaly. Imaging analyzing the behaviour of an agent, and expecting a certain action after some observations, if the action choosen by the agent is different, then, we could consider that action as anomalous. Finally, in line 10, the literal and the rule is added to the solver connected by an *OR*, so that if the rule is unsatisfiable, the literal is set to *true*, to make step satisfied. In line 11, the solver is called to minimize the cost function, which, again, tries to satisfy as many rules as possible, minimizing a cost function. Since there might be many assignments, which satisfy the rule, we need to tell the solver to find the assignment which gets closer to the POMCP behaviour. In line 12, the goodness of the model is defined as one minus the distance to the observed boundary, that we want to maximize. Finally, in line 13, the solver is called to maximize, the goodness, which again, is how good the assignment is close to the POMCP behaviour. Then the model is returned. The model contains the free variables assignments, of the final rule.

3.3 Hellinger Distance and Identification of Anomalous Decision

The last phase of the methodology after the solver assigned values to free variables, and find unsatisfiable steps, is to analyze which steps are not satisfied and why. In this case, we can classify the unsatisfiable steps into two groups: anomalies, and approximation errors. In the first group we can find steps in which the agent actually had an anomalous behaviour because of some untuned parameters, or other factors. In the second group we find the unsatisfied steps due to the rule approximation. It is worth remember that the rule has to be constructed such that to explain the POMCP behaviour, however, in early stage of the analysis, the expert might not know the agent behaviour, or the expert might not know all the detail about the problem domain, creating a rule which does not express all the constraints needed to approximate optimally the POMCP behaviour. We investigate this further in Chapter 4. Let's now focus on the first group of anomalies, which are due to actual anomalous behaviours by the agent. These by the authors of [20] are called unexpected decisions. They provide also a procedure to differentiate unexpected decisions from approximation errors. The procedure takes in input a rule r and the steps that violate the rule, which are all those steps which belief do not satisfy the rule synthetized, and a final argument is the threshold $\tau \in [0, 1]$. The threshold, as we will see is a hyperparameter, which is fundametal to distinguish approximation errors and unexpected decisions. Finally, the procedure returns the steps in which the agent took unexpected decisions. Let's see how it works. The first step, is to generate randomly w beliefs that satisfy the rule, $\bar{b}_1, \bar{b}_2, \dots \bar{b}_w$. Then, for each belief b_i in the rule, a distance is computed for each unsatisfiable step. Finally the minimum distance is computed between all b_i, and compared to the threshold τ, if $h_i > \tau$ then, b_i is considered an outlier, so an anomaly, because it is far from the other belief. The distance used is the *Hellinger Distance* [7] Formally the Hellinger Distance is:

$$H^2(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{k} (\sqrt{P_i} - \sqrt{Q_i})^2} \tag{3.2}$$

where P, and Q are probability distributions, and k is the discrete number of states in P and Q. It is used because it works with discrete variables as in our case. The Hellinger distance is also usefull, because its value range goes between [0,1], so it makes the tuning process on the threshold easier.

Chapter 4

Empirical Evaluation

4.1 Application domains

4.1.1 Tiger

Tiger, is a well known problem in planning, as an exact solution can be found. An agent stands behind two closed doors, behind a door there is a tiger, behind the other there is a treasure. The agent's goal is to open the door with the treausure behind it. The agent does not know prior, which door has the treasure behind. In fact, the agent can listen, to gain some information. The agent can hear a roar behind one of the two doors. However, listening is not accurate, the agent might hear the tiger behind the door on the left but the tiger is behind the door on the right. The states of the world are s_l, tiger on the left, and s_r tiger on the right. The possible actions are *Listen, Open Left, Open Right*. The agent obtains a reward $r = 10$ for opening the door with the treasure behind, while it obtains a penalty $r = -100$ for opening the door with the tiger behind. Furthermore, listening has a cost of $r = -1$. The possible observations are *Tiger Left, Tiger Right*. After the agent opens one of the door, it receivs the reward, and the problem resets, setting the tiger randomly behind one of the two doors.

4.1.2 Velocity Regulation

Velocity Regulation is a problem presented in [9], whose goal is to regulate the velocity of a robotic platform as a case study. A mobile robot travels on a defined path divided into different segments (i.e., four and eight in our examples). Each segment is divided into subsegments of different sizes. Figure 4.1 shows an example of map. Each segment has a partially observable difficulty. The difficulty can be one of the following:

- $f = 0$ *Clear.*
- $f = 1$ *Lightly obscured.*
- $f = 2$ *Heavily obscured.*

All the subgsegments in a segment have the same difficulty. For example if the subsegment 1.1 is in segment 1, and the segment has a difficulty of 1, also the subsegment will have a difficulty of 1. This map has 3^8 possible states, because there are three possible difficulties for each segment, and there are eight segments. The mobile robot has three possible speed levels:

- $a =$ slow
- $a =$ medium.
- $a =$ fast.

The robot's goal is to move in the defined path avoiding obstacles while travelling at the highest speed possible. The robot must decide which action (i.e., which speed level) to use at each subsegment. The robot also achives a reward for traversing the subsegment without colliding with obstacles. The higher the speed the higher the reward the robot gets. Specifically, the reward in this problem is defined as: $r = 1 + a$ where a is the action the robot choose. The higher the speed level, the higher the probability of colliding with an obstacle becomes. Particularly, Table 4.2 shows the probability $p(c = 1|f, a)$ of colliding with an obstacle given the current difficulty of the subsegment f and the velocity, a. However, the real difficulty of the segment/subsegment is unknown, but the robot receivs an observation o after it

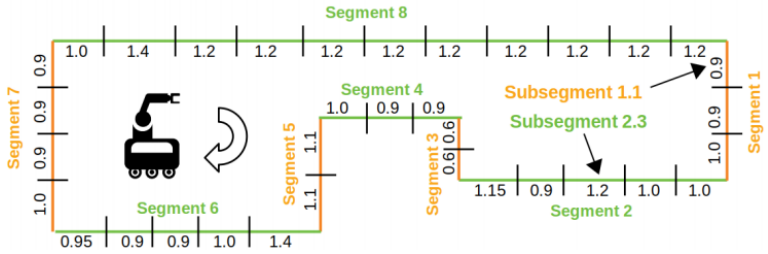


Figure 4.1: Velocity Regulation

crossed a subsegment which is defined as follows:

- $o = 0$ no obstacles.
- $o = 1$ obstacles.

The observation depends probabilistically on the difficuly of the subsegment as represented in Table 4.1. We encoded each state in the belief using the formula:

$$state = \frac{round(belief)}{3^{(num\_segments-1-segment)}} \tag{4.1}$$

f	$p(o = 1\|f)$
0	0.0
1	0.5
2	1

Table 4.1: Probability of receiving observation $o = 1$ given the subsegment difficulty f.

f	a	$p(c = 1\|f, a)$
0	0	0.0
0	1	0.0
0	2	0.028
1	0	0.0
1	1	0.056
1	2	0.11
2	0	0.0
2	1	0.14
2	0	0.25

Table 4.2: Probality of colliding with an obstacle given the difficulty of the subsegment and the action.

4.2 Results on the Detection of Anomalous Actions: Tiger

In Tiger, we can find an exact policy thanks to the *incremental pruning algorithm* implemented in [4]. The exact policy is then used as a term of comparison with XPOMCP. Specifically it is used to detect whether XPOMCP is able to approximate exactly POMCP. Particularly, we generated some traces with different values of *RewardRange*, which is the hyperparameter used in POMCP to set the tradeoff between exploration and exploitation. The exact value of RewardRange is 110, which comes from the absolute difference between the reward range. [-10, +100]. If this value is lower, then errors get injected during the execution of POMCP. In this problem domain, we show that XPOMCP is able to find anomalies (i.e., unexpected actions) with an high accuracy. Specifically, in this Section we present the results of XPOMCP with different traces generated with different values of RewardRange. Furthermore, we compare the result with a well known method for anomaly detection, Isolation Forest, already described in Section 2.6.1.

4.2.1 Results on Rule-Based Methodology

The first step to use XPOMCP is to create a rule template that encodes the high level insight of the expert. In this problem domain, we know that the agent should listen until it is confident of where the tiger is. Only then, it chooses which door to open based on the observations it obtained during the listening phase. Therefore, a good action rule template is:

$$\begin{aligned} &r_L : \texttt{select } \textit{Listen} \texttt{ when } (p(right) \leq x_1 \wedge p_{\text{left}} \leq x_2) \\ &r_{OR} : \texttt{select } \textit{Open Right} \texttt{ when } (p(right) \geq x_3) \\ &r_{OL} : \texttt{select } \textit{Open Left} \texttt{ when } (p(left) \geq x_4) \\ &\texttt{where}(x_1 = x_2) \wedge (x_3 = x_4) \wedge (x_3 > 0.9) \end{aligned} \tag{4.2}$$

In this situation, we created three action rule templates, one for each action. Rule template r_L describes when the agent should listen. Specifically, the agent should listen, if both the probabilities of having the tiger to the right, or to the left, is less or equal respectively to x_1 and x_2. This represents the fact that if the agent has not achived an high understanding of where the tiger is, then it should listen to obtain more observations, in order to increase its understanding. The rule template for action r_{OR} describes when the agent should open the right door, if the probability of the treasure being behind the right door is higher than a variable x_3.

This expresses the idea that we only want the agent to open the right door, if it thinks the tiger is behind the door on the left. The last rule template r_{OL} for action open left is symmetric to action open right. Finally, the `where` statement defines the hard constraints, which force the solver to find equal values of x_1 and x_2, and also for x_3, x_4. The last clause $x_3 > 0.9$ represents the fact the the agent should open the door only if the probability of finding the treasure is greater than 0.9. We evaluated this rule using different traces generated with different RewardRange values. Specifically, we known that the best RewardRange value is 110, so we expect XPOMCP not to find anomalies in the traces generated with that value. Then, we generated traces with a RewardRange $\in \{85, 65, 40\}$. In these traces, we expect XPOMCP to find some anomalies due to the injection of errors caused by a wrong RewardRange. In XPOMCP there is a threshold τ that needs to be tuned to consider a step anomalous. In order to tune correctly the parameter, two curves have been used: *Receiver Operating Characteristic (ROC)* curve and the *precision/recall* curve, and their relative performance measures: *Area Under Curve (AUC)* and *Avarage Precision (AP)*. Specifically, XPOMCP has been executed with 100 thresholds uniformly distributed in the interval $[0, 0.5]$. Table 4.3 shows a comparison of the performances of XPOMCP between AUC and AP.

W	**% errors**	**AUC**	**AP**
110	0.0 (0.0)	-	-
85	0.0004 ($\pm$ 0.0003)	0.993 ($\pm$ 0.041)	0.986 ($\pm$ 0.082)
65	0.0203 ($\pm$ 0.0021)	0.999 ($\pm$ 0.001)	0.999 ($\pm$ 0.002)
40	0.2374 ($\pm$ 0.0072)	0.995 ($\pm$ 0.034)	0.987 ($\pm$ 0.084)

Table 4.3: XPOMCP performances in terms of AUC and AP with respect to different values of W. [21]

Then, in order to find the best value of τ, the cross validation method has been used, training XPOMCP with 5 traces, and using 45 traces as validation, where each trace has 1000 run. As before, the parameter τ, has a valua range of: $[0, 0.5]$. In order to compare the performance of XPOMCP, the following metrics have been used: accuracy and F1 score. Accuracy is defined by the formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{TP + TN + FP + FN} \tag{4.3}$$

where:

- TP: is the number of elements classified as positive, which actually are positive.
- TN: is the number of elements classified as negative, which in fact are negative.

- FP: is the number of elements classified as positive but actually are negative.
- FN: is the number of elements classified as negative but actually are positive.

While F1-score, is defined using other two metrics namely: *precision* and *recall*, defined as:

$$\text{Precision} = \frac{\text{TP}}{TP + FP} \tag{4.4}$$

$$\text{Recall} = \frac{\text{TP}}{TP + FN} \tag{4.5}$$

Using these metrics, the F1-Score is defined as:

$$\text{F1-Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.6}$$

Using these metrics, we can represent how well a model performs in a binary classification problem, as in our case. Table 4.4 represents XPOMCP in terms of accuracy and F1-score and their standar deviations, considering only the best threshold, found by cross-validation. Particularly, the accuracy is high, in both situations, when few errors are present, and when the error rate is higher. Furthermore, the F1-score defines how well the model is able to generalize with new data. In this situation the F1-score, is high, showing that XPOMCP is able to generalize well with unseen data.

RewardRange	**Threshold**	**F1-score**	**Accuracy**	**time (s)**
85	0.061	0.979 ($\pm$ 0.081)	0.999 ($\pm$ 0.0001)	14.30 ($\pm$ 0.50)
65	0.064	0.999 ($\pm$ 0.002)	0.999 ($\pm$ 0.0001)	14.75 ($\pm$ 0.80)
40	0.045	0.980 ($\pm$ 0.072)	0.987 ($\pm$ 0.049)	12.78 ($\pm$ 0.83)

Table 4.4: Representation of XPOMPC in terms of Accuracy and F1-score

4.2.2 Results on IF

In this section the results of anomaly detection using Isolation Forest are presented. Isolation forest uses an hyperparameter namely *contamination*, which specify the expected number of anomalies in a range of $[0, 1]$. Therefore, different values of the contamination hyperparameter uniformely distributed in range [0, 0.5] have been tried. Then, the AUC, and Avarage Precision have been computed as already done for the rule-based approach. As Table 4.5 shows IF performs worse than rule-based. Specifically, the number of false positives is high leading to a degradation of performance regarding AP. XPOMCP has less false positives, thanks to the integration of prior knowledge in the rule template creation. While Table 4.6 shows how the

RewardRange	**% errors**	**AUC**	**AP**
110	0.0 (0.0)	-	-
85	0.0004 (± 0.0003)	0.964 (± 0.024)	0.057 (± 0.1076)
65	0.0203 (± 0.0021)	0.992 (± 0.001)	0.539 (± 0.0520)
40	0.2374 (± 0.0072)	0.675 (± 0.020)	0.333 (± 0.0153)

Table 4.5: Isolation forest AUC and AP with different values of RewardRange. [21]

W	**Threshold**	**F1-score**	**Accuracy**	**time (s)**
85	0.01	0.020 (± 0.033)	0.990 (± 0.001)	0.72 (± 0.013)
65	0.03	0.771 (± 0.044)	0.988 (± 0.001)	0.71 (± 0.010)
40	0.5	0.437 (± 0.035)	0.585 (± 0.026)	0.64 (± 0.037)

Table 4.6: Representation of IF in terms of Accuracy and F1-score

accuracy degrades as the percentage of errors increases. Furthermore, IF is not able able to generalize well, as the F1-score is much lower compared to XPOMCP. The conclusion of this analysis is that XPOMCP outperforms IF on all the datasets. The only drawback is the time needed for XPOMCP to synthetize the rule compared to the time IF takes to end. However, we think this is a good tradeoff between accuracy and time.

4.3 Results on the Detection of Anomalous Actions: Velocity Regulation

In this section, we present our work on the velocity regulation domain. Unfortunately, in velocity regulation a perfect policy cannot be computed, because the problem is intractable given the large state space. Therefore, in this problem domain, instead of studying in a quantitative manner how various anomaly detection algorithms perform, we ask ourself, what model can fit POMCP the best. Note that XPOMCP, is also able to approximate POMCP, by creating a rule. In fact, the anomalies resulting from the creation of the rule are also due to the model approximation, and not only due to bad decisions by POMCP. The specific question we pose is: "Is there a model which is more interpretable than the others?". To answer to this question we compared three models able to fit POMCP: *XPOMCP, Logistic Regression, Neural Network*. We compare the interpretability of these models, and their accuracy with respect to POMCP. This section is structured as follow: in Section 4.3.1 the result regarding the approximation of XPOMCP are given. Section 4.4.2 presents the results regarding the Logistic Regression approximation. Finally, in Section 4.3.3 the results regarding the POMCP approximation using a Neural Network are presented.

4.3.1 Results on Rule-Based Methodology

In this Section, we present the result of POMCP approximation using XPOMCP. Three action rules have been created, one per action. The action rules are:

- r_1: select action **fast** when:

$$\begin{gathered} easy \geq \mathbf{x_1} \vee \\ difficult \leq \mathbf{x_2} \vee \\ easy \geq \mathbf{x_3} \wedge intermediate \geq \mathbf{x_4} \end{gathered} \tag{4.7}$$

- r_2: select action **medium** when:

$$\begin{gathered} easy \geq \mathbf{y_1} \vee \\ difficult \geq \mathbf{y_2} \vee \\ easy \geq \mathbf{y_3} \wedge intermediate \geq \mathbf{y_4} \end{gathered} \tag{4.8}$$

- r_3: select action **slow** when:

$$\begin{gathered} easy \leq \mathbf{z_1} \vee \\ difficult \geq \mathbf{z_2} \vee \\ easy \leq \mathbf{z_3} \wedge intermediate \leq \mathbf{z_4} \end{gathered} \tag{4.9}$$

Rule r_1 represented in Equation 4.10 tells when intuitevely the agent should travel fast, that is when the segment is free, therefore the belief for state *easy* is high, or the belief for state $difficult$ is low, or when both the belief of states *easy* and *intermediate* are high. While, rule r_2 represents an intuitive approximation of when the agent should move at medium speed, that is when the segment is free, or when the segment is lightly obstructed. Finally, rule r_3 explains when the agent should move slow. Intuitively the agent should choose action *slow* when it is not safe enough to move at an higher speed, because of the presence of obstacles in the segment. Therefore, when the easy belief of state easy is low, or when the belief of state difficult is high. One last case is when the belief of state *easy* and of state *intermediate* is low. After the synthesis process ended, the free variables have been transformed in real probability values. These are the rule after the synthesis:

- r_1: select action **fast** when:

$$\begin{gathered} easy \geq \mathbf{0.87} \vee \\ difficult \leq \mathbf{0.02} \vee \\ easy \geq \mathbf{0.86} \wedge intermediate \geq \mathbf{0.10} \end{gathered} \tag{4.10}$$

- r_2: select action **medium** when:

$$\begin{gathered} easy \geq \mathbf{0.95} \vee \\ difficult \geq \mathbf{0.68} \vee \\ easy \geq \mathbf{0.80} \wedge intermediate \geq \mathbf{0.09} \end{gathered} \tag{4.11}$$

- r_3: select action **slow** when:

$$\begin{gathered} easy \leq \mathbf{0.79} \vee \\ difficult \geq \mathbf{0.09} \vee \\ easy \leq \mathbf{0.79} \wedge intermediate \leq \mathbf{0.66} \end{gathered} \tag{4.12}$$

We can now analyze what are the approximation errors for each rule. Specifically we want to verify what steps are considered anomalous by XPOMCP. It is worth remembering that the anomalous steps are due to the rule approximation or to real anomalous decisions made by POMCP. However, in this situation, the POMCP RewardRange parameter is set correctly. Therefore, we expect zero to few errors. The metric used to evalute XPOMCP is accuracy. In this list the accuracy for each rule is present:

- r_1: Accuracy: 99% (349/350 steps)
- r_2: Accuracy: 90% (317/350 steps)
- r_3: Accuracy: 92% (324/350 steps)

In conclusion, we can say that XPOMCP is able to approximate well the POMCP behaviour, with an high accuracy. The worst performance happens when the model has to predict action *medium*, as expected because it is in the middle between action *slow* and *fast*, therefore there is an overlap between *slow* and *medium*, and *medium* and *fast*. This concept is reccurent, and will come back even in the next models.

# Segment	**Accuracy**	**Precision**	**Recall**	**F1-score**
0	0.72	0.73	0.72	0.71
1	0.62	0.58	0.62	0.58
2	0.79	0.78	0.79	0.78
3	0.89	0.90	0.89	0.89

Table 4.7: Metrics regarding the Logistic regression model.

4.3.2 Results on Logistic Regression

Logistic Regression is a linear model used as a classification algorithm. The name might be misleading, as the term regression means to predict continuous values. Actually the name is not completly wrong. At the basis of logistic regression there is the standar linear regression algorithm, whose output is then processed using a sigmoid function, which returns a value in range $[0, 1]$, which is interpreted as a probability. Specifically if the classification problem is binary, then the output can be deducted as follows:

$$LogisticRegression = \begin{cases} 0 & \text{if } \sigma(\hat{x}) \leq 0.5 \\ 1 & \text{otherwise} \end{cases}$$

where $\hat{x}$ is the linear regression output, and σ is the sigmoid function. The goal of this work is to create a logistic regression model able to "imitate" the POMCP behaviour and verify its intepretability. In this Section we focus only on how well the model performs computing metrics such as the accuracy, the precision, and the recall. The model has to predict the action given the current belief. The environment used is composed of four segments, forming a rectangle of size 3x5 m, Figure 4.2. Specifically, we trained four logistic regression models (i.e one per segment) by giving in input the vector $\mathbf{X}$ of belief, and as label $\mathbf{y}$, the action. The inital dataset had only 3200 beliefs. However, the dataset was unbalanced regarding the action, the label the model has to predict. Therefore, a first step to balance the dataset using augmentation has been performed. The augmented dataset has 6654 beliefs obtained by copying the belief of actions that were in minority by sampling random indexes from a uniform distribution. Later the dataset has been divided in training and testing. Specifically, 20% of data has been used for testing, the remaining 80% has been used for training. The testing metrics are shown in Table 4.7. The results are promising, specifically remembering that the logistic regression model assumed the data model linear. In order to better represent these data, the confusion matrixes have also been plotted. Let's analyze the confusion matrix:

- Figure 4.3: This refers to the model of the first segment. From the confusion matrix, we can deduce that the model is able to classify quite well action 0 and action 2, where action 0, 1, 2 are respectively: slow, medium, fast. We note that the accuracy for action 1 is low. This is caused by the overlap between action 0 and 1, and between action 1-2.
- Figure 4.4: This refers to the model of the second segment. Also in this model, we can deduce that the model is able to classify quite well action 0 and action 2, where action 0, 1, 2 are respectively: slow, medium, fast. Unfortunately, action 1 is misclassified as action 0 or 2 most of the time.
- Figure 4.5: This refers to the model of the third segment. Also in this model, we can deduce that the model is able to classify quite well action 0 and action 2, where action 0, 1, 2 are respectively: slow, medium, fast. In this model, the action 1 is classified better then in the other cases. This probably happens because, as the agent moves on to the next segment, it implicitly acquires more knowledge as it gets more observations, therefore POMCP is able to make better decisions.
- Figure 4.6: This refers to the model of the fourth segment. Also in this model, we can deduce that the model is able to classify quite well action 0 and action 2, where action 0, 1, 2 are respectively: slow, medium, fast. This model is able to classify actio 1 perfectly. Probably the model overfitted action 1.

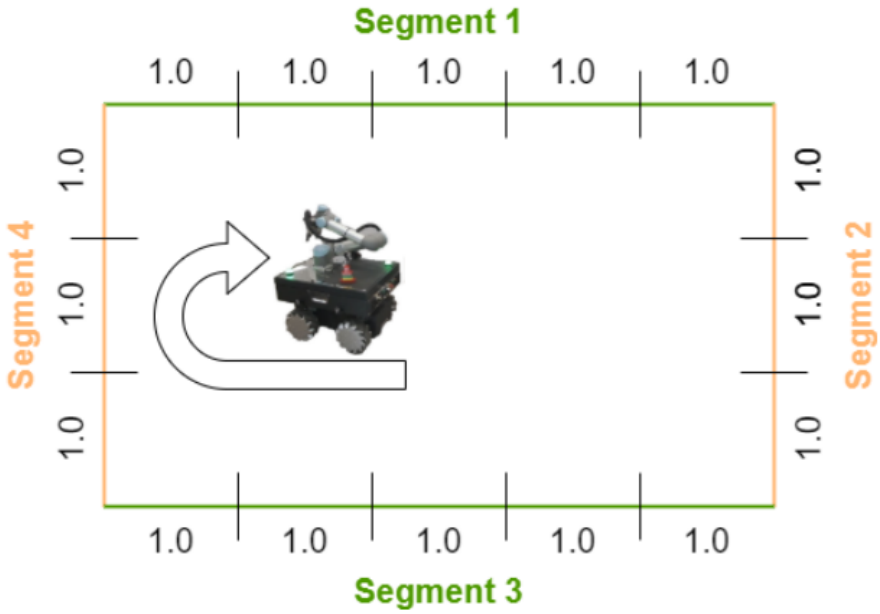


Figure 4.2: Velocity Regulation problem, four segments

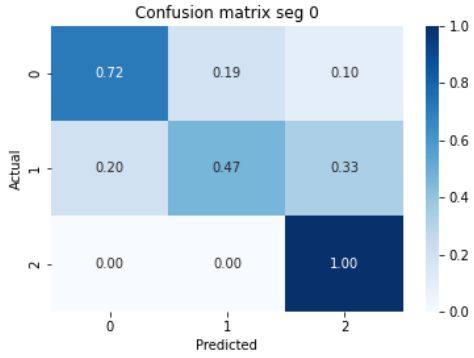


Figure 4.3: Confusion matrix logistic regression segment 0

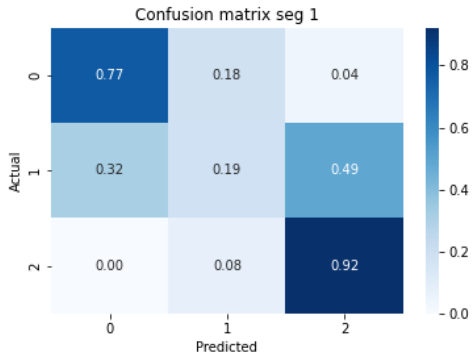


Figure 4.4: Confusion matrix logistic regression segment 1

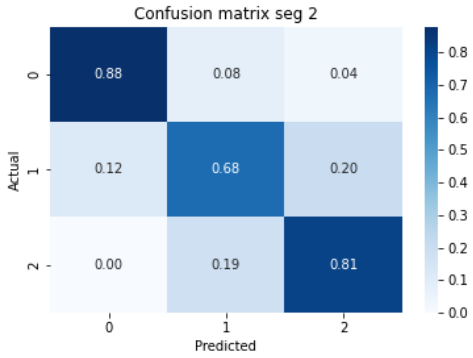


Figure 4.5: Confusion matrix logistic regression segment 2

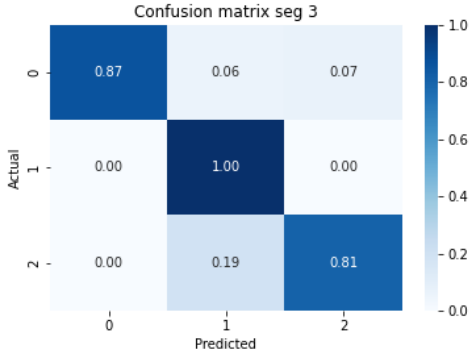


Figure 4.6: Confusion matrix logistic regression segment 3

4.3.3 Results on Neural Network

Given the good results of the model approximation using a logistic regression, we thought a neural network, that theoretically should be able to approximate exactly any function [3], could perform better, with the right number of neurons and layers. Neural networks are composed of neurons, where each neuron receives in input some connections from the input, or from other neurons. Each input in the neuron is multiplied by its respective weight, and then they are summed all together. This is the easiest type of neural network one could create. The problem is that till now this model is linear. In order to add non-linearity in the network, at least one neuron should have an activation function, which takes the sum of all inputs multiplied by their weights and returns a single value. Common activation functions are *ReLu* [25], *Sigmoid*, *Hyperbolic tangent* etc. Neural networks may have more than one layers, if so, they are called *deep neural network*. In general it is preferable to have a restrict number of neurons betweeen 2-32, and enough number of layers. A common way to train neural networks is the *stocastic gradient descent (SGD)* algorithm. What it does is looking for the global minimum computing the gradient of the loss function. For this reason, it is important for the loss function to be convex. The number of weights the neural network has depends on the layer type. If the layer is fully connected, therefore, each neuron at layer i is connected to all the neurons at layer $i+1$ then, there are in total $num\_neurons_i \cdot num\_neurons_{i+1}$ between layer i and layer $i+1$, where $num\_neurons_i$ is the number of neurons at layer i and $num\_neurons_{i+1}$ is the number of neurons at layer $i+1$. To be fair, there are also other type of layers, such as Convolutional layer, Recurrent Layer, which use sophisticated techniques to diminish the number of parameters to avoid overfit. However, for our case, we tested a "vanilla" neural network. Moving on the data preprocessing step, also in this situation, some data preprocessing operations have been done like in the Logistic Regression case. In fact, we augmented the dataset to make it balanced, and finally we divided the dataset in training and testing set. The best network we found is composed of 32 neurons, and 8 layers. The network architecture is standar, every layer contains one fully connected layer, followed by a batch-normalization layer. The final layer has only three neurons with the Softmax activation function. The loss function used is the sparse categorical cross-entropy. Also in this model, we printed the common metrics, such as the accuracy, precision, recall and f1-score. Table 4.8 shows these metrics. One consideration about these metrics, we see that in general the accuracy is better than the logistic regression model as expected. Also the other metrics, precision recall the f1-score got better. The confusion matrix have also been plotted for this model. The same considerations, as the logistic regression

# Segment	**Accuracy**	**Precision**	**Recall**	**F1-score**
0	0.7573	0.7839	0.7581	0.7354
1	0.8051	0.8262	0.8055	0.7889
2	0.9290	0.9341	0.9290	0.9286
3	0.9887	0.9888	0.9887	0.9887

Table 4.8: Metrics regarding the neural network.

model, can be made also for these confusion matrixes. Specifically we still note that the most problematic label to classify is action 1. The reason is always the same, it is in the middle of action 0 and 2.

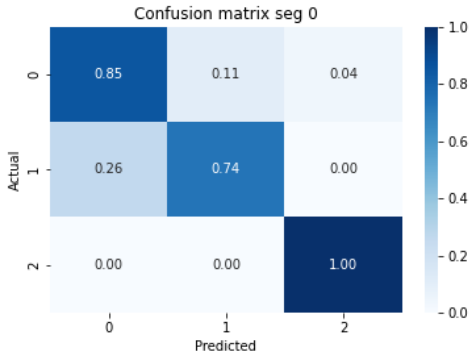


Figure 4.7: Confusion matrix neural network segment 0

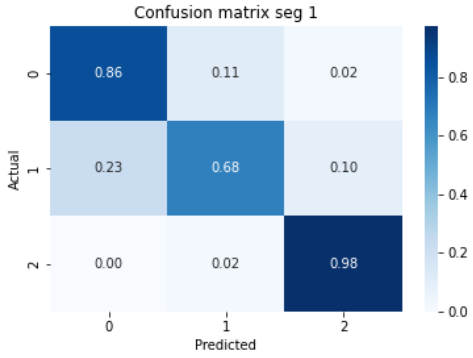


Figure 4.8: Confusion matrix neural network segment 1

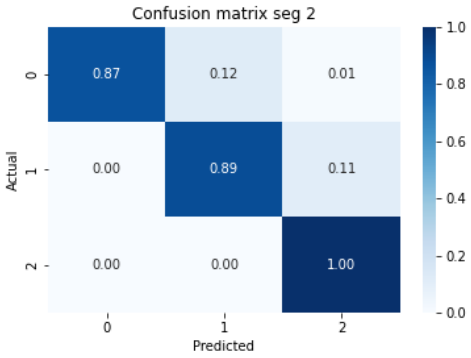


Figure 4.9: Confusion matrix neural network segment 2

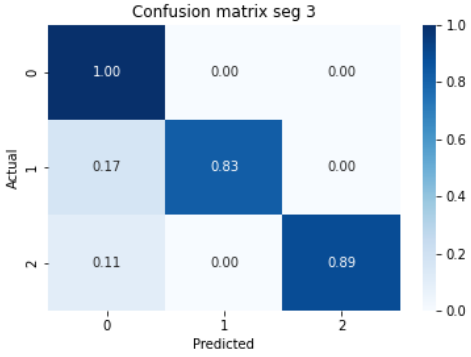


Figure 4.10: Confusion matrix neural network segment 3

4.4 Models Interpretation

In this Section we pose the question "Which model is best suited to present the results, and what is its interpretation?" Specifically, we focus on the Velocity Regulation problem domain, and the three models built to approximate POMCP: XPOMCP, Logistic Regression, and Neural Network. While in the previous Section, we saw how well these models perform in terms of metrics such as accuracy, precision, recall, and f1-score, in this Section, we would like to understand what these models learnt, and verify whether they are interpretable.

4.4.1 XPOMCP Interpretation

XPOMCP approximates POMCP using rule-based logic rules, and finding the free variable values using MAX-SMT to satisfy the highest number of steps, thus decisions POMCP made. The model is intrinsically interpretable, as the user creates a logic rule using almost plain english. The result of MAX-SMT is a logic rule which tries to describe the POMCP behaviour. An example of logic rule for the velocity regulation domain is:

$$\begin{gathered} r_1 : \text{select action } \textbf{fast} \text{ when :} \\ p(easy) \geq \mathbf{x_1} \vee \\ p(difficult) \leq \mathbf{x_2} \vee \\ p(easy) \geq \mathbf{x_3} \wedge p(intermediate) \geq \mathbf{x_4} \end{gathered} \tag{4.13}$$

Specifically, with this logic rule we are describing what the behaviour of POMCP should be regarding action **fast**. This rule, after the synthesis process will have values instead of free variables, it will be something like:

$$\begin{gathered} r_1 : \text{select action } \textbf{fast} \text{ when :} \\ p(easy) \geq \mathbf{0.80} \vee \\ p(difficult) \leq \mathbf{0.04} \vee \\ p(easy) \geq \mathbf{0.40} \wedge p(intermediate) \geq \mathbf{0.12} \end{gathered} \tag{4.14}$$

This shows the instrical interpretability of the model.

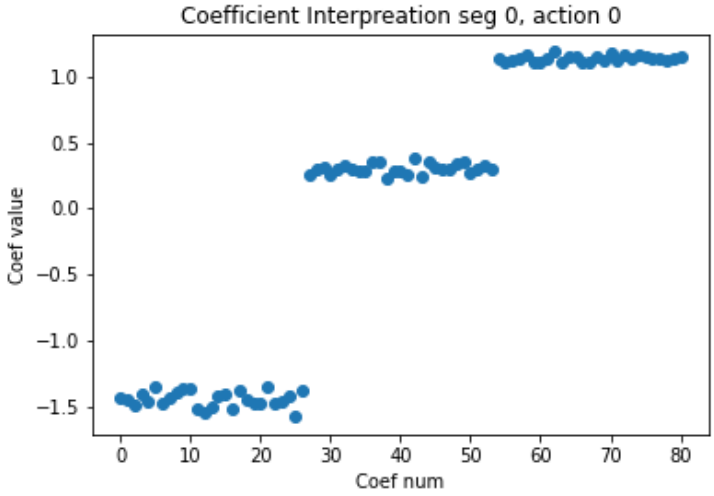
4.4.2 Logistic Regression Interpretation

The Logistic Regression model we built to approximate POMCP for the velocity regulation domain, has good performance, as we saw in Section . However, we would like to understand what the model learnt. Specifically, Logistic Regression

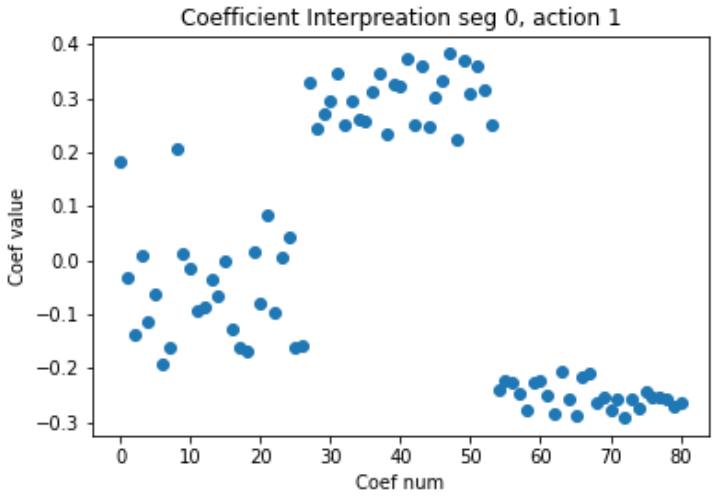
has n weights, where n is the input size, for each input instance. In our case the input is the belief, specifically there are 81 belief per step, because there are 3 possible states, and four segments therefore 3^4 states. Specifically we have created four models, one per segment. After having fitted the Logistic Regression models to the belief dataset, the weights learnt have been plotted. From Figure 4.11a to 4.14c the scatter plot of the weights the logistic regression model learnt are represented. Specifically, we divded the Figure in 4 rows, one per segment sorted by ascending order. Furthermore, the logistic regression model applies the one-versus-all classification when the problem is multiclass as in this situation, given that there are 3 possible class: *slow, medium, fast*. Note that in the Figure description the action are labelled as 0,1,2 which represent *slow, medium, fast* respectively. Note that based on the segment the single belief refers to different states. Remember that we can compute the state to which the belief refers using a formula which takes in input the segment number:

$$state = \frac{round(belief)}{3^{(3-segment)}} \tag{4.15}$$

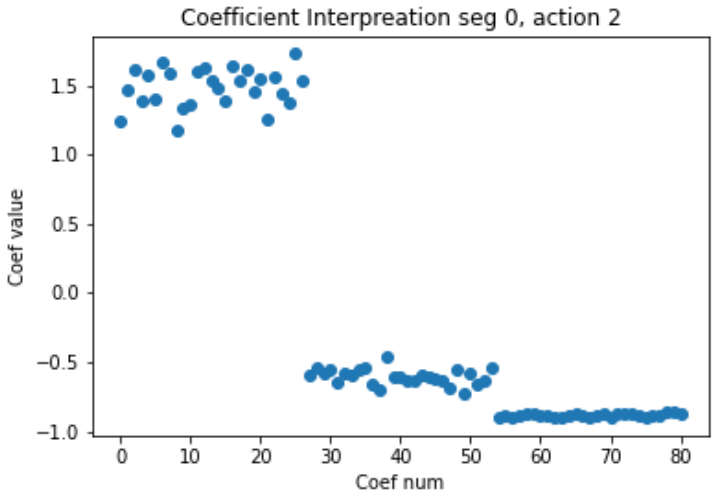
For example using these formula, we find out that the first 27 belief refers to state 0, from belief 28 to 54 refers to state 1, from belief 55 to 81 referst to state 2, where states 0,1,2 are decodified in *easy, medium, difficult*. Moving on to the model interpretation, we note that the model created to fit classify action 0 , Figure 4.11a has positive coefficients for the beliefs from 55 to 81, which represent state 2. The explanation can be deducted by how the logistic regression predicts the class: it returns 1 if the $\theta^{\mathbf{T}}\mathbf{X}$ is positive, returns 0 otherwise, where θ^T is the vector of weights. Therefore, this model has to predict action 0 when the state is 2, and thereofore the coefficients have to be positive, given the fact that the vector of belief $\mathbf{X}$ is never negative. We can give the same explanation regarding Figure 4.11c, which represents the model tuned to predict action 2 in segment 0. Specifically, the model should predict action 2 only when it is safe enough for the agent to go fast, therefore when the segment has not obstacles. For this reason only the first 27 coefficients are positive. While the other are negative. The same analysis can be done for the other segments, and the result is the same. Logistic Regression tries to approximate the POMCP behaviour using a linear model. Even though the accuracy could be better using other models, we found out that this model actually is interpretable.



(a) Segment 0 action 0

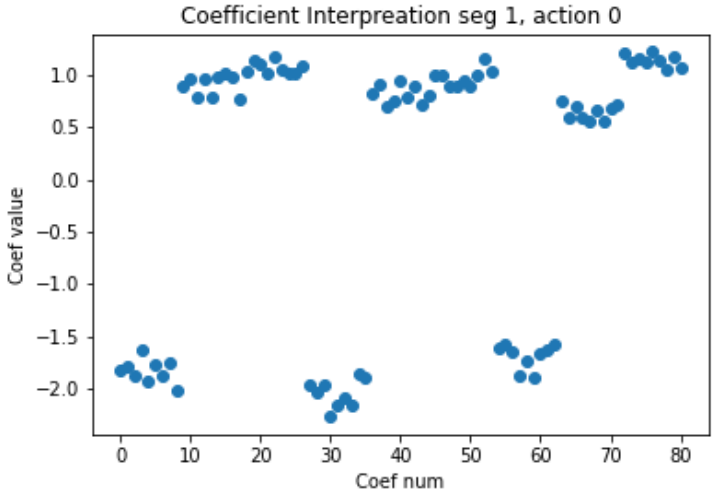


(b) Segment 0 action 1

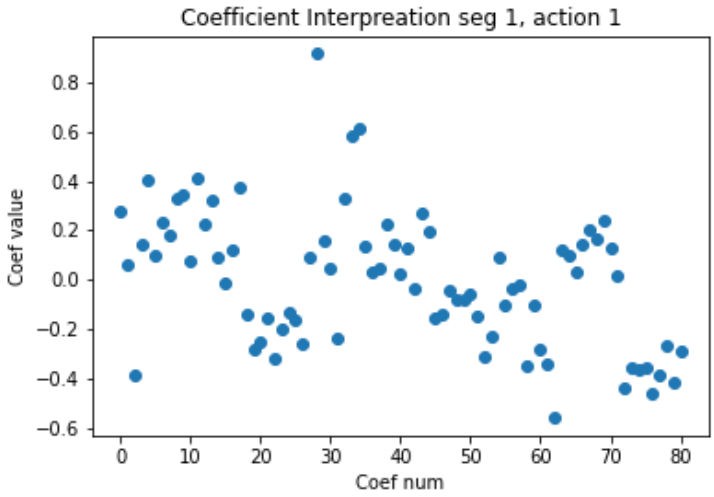


(c) Segment 0 action 2

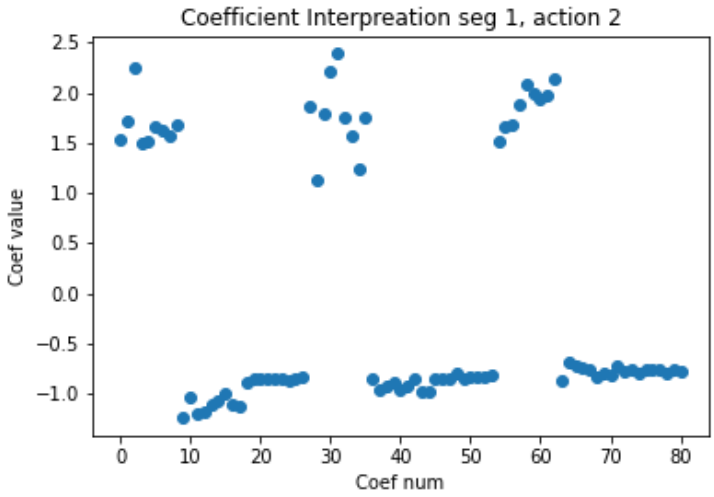
Figure 4.11: Scatter plot of the model coefficients fitted to belief of segment 0.



(a) Segment 1 action 0

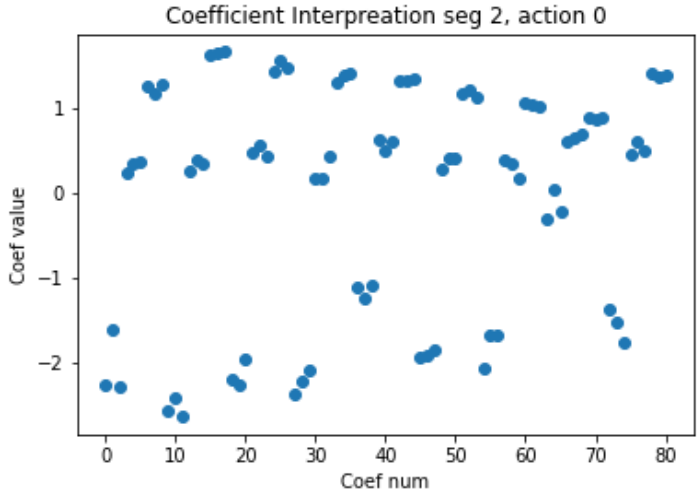


(b) Segment 1 action 1

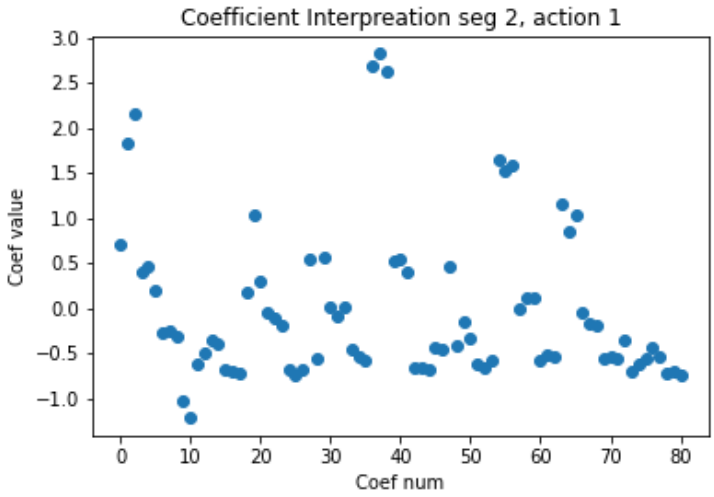


(c) Segment 1 action 2

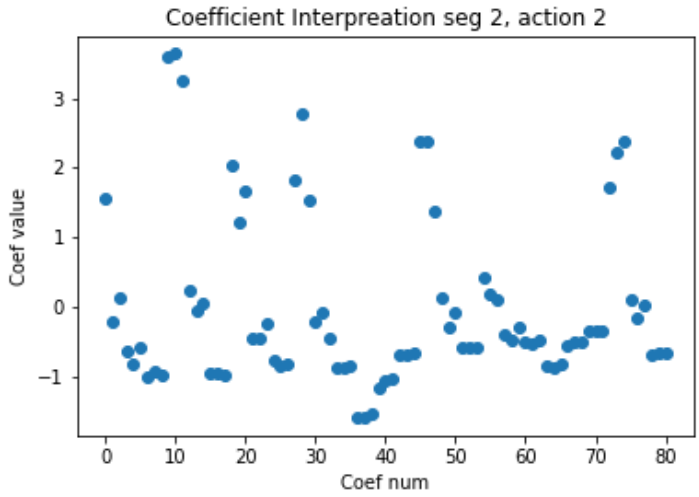
Figure 4.12: Scatter plot of the model coefficients fitted to belief of segment 1.



(a) Segment 2 action 0

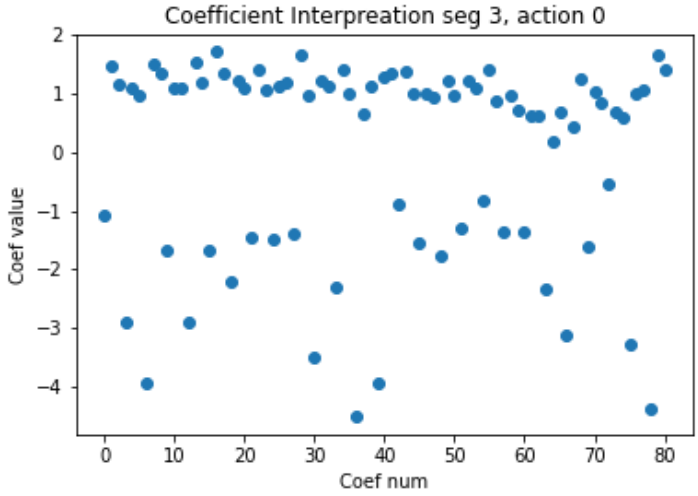


(b) Segment 2 action 1

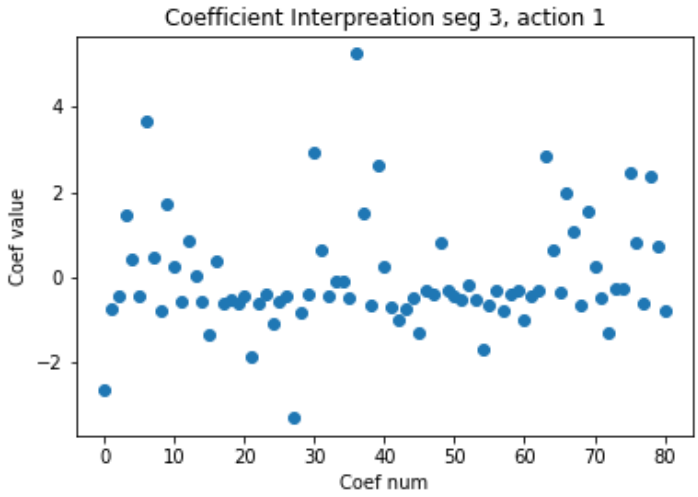


(c) Segment 2 action 2

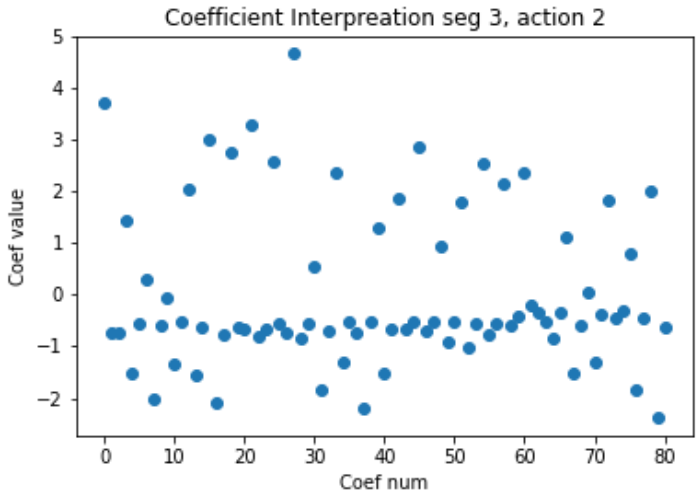
Figure 4.13: Scatter plot of the model coefficients fitted to belief of segment 2.



(a) Segment 3 action 0



(b) Segment 3 action 1



(c) Segment 3 action 2

Figure 4.14: Scatter plot of the model coefficients fitted to belief of segment 3.

4.4.3 Neural Network Interpretation

Neural Netoworks are a powerfull instrument used to approximate whatever function we need. However, because of the large number of layers, and neurons, unfortunately, interpret the weights the neural network learnt is not as easy as in the logistic regression model. Specifically, we were not able to find a good interpreation of the weights, given that in total there are $10,627$ trainable parameters.

Chapter 5

Interactive Visualizations of Anomalous Decisions

5.1 A Web App for the Visualization of Anomalous Decisions

In this chapter, we present a web application that can be used to analyze the anomalies using the methodology presented in [21]. The web application allows the user to use the novel method, without the need to hard-code the analysys in Python. In fact the web application *User Interface, UI* has been designed to be easy to use but also complete. Therefore, we designed this web application, which guides the user from the beginning to the end of the analysis. The application presents its explanation of the POMCP policy using both text output and graphics (e.g., bar plots and tables). The use of this application is divided in three phases sorted by execution order: *Rule creation, Rule synthesis, Analysis results.* These phases are representative of the methodology presented in Section 3, [21]. In Section 5.1.1 we present the frontend. While in Section 5.1.2 the backend is presented. In conclusion, the complete architecture of the web app, has been presented in Section 5.1.3.

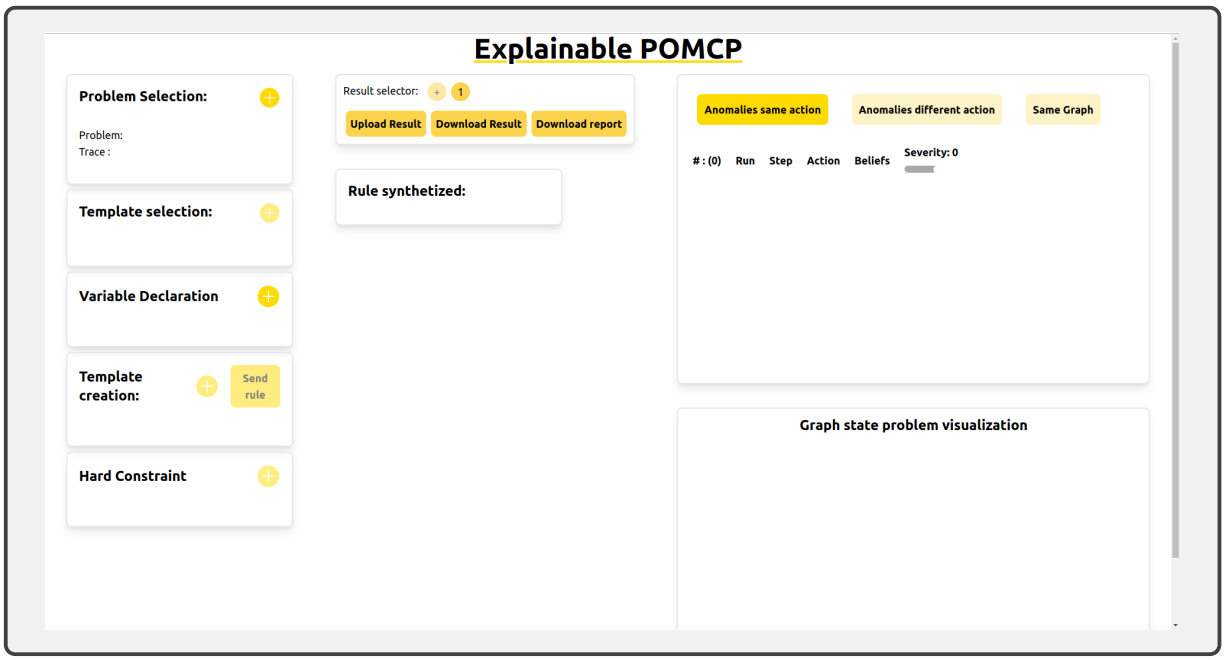


Figure 5.1: Web application namely "Explainable POMCP"

5.1.1 Frontend

The frontend of an application can be defined as the part of the application that is visible to the user. It is responsible for the acquisition of data from the user, as well as all the interaction with the user [23]. In this context, a web application has been created rather than other type of applications such as *Desktop app, Mobile app*, because we wanted it to be usable from any device, and especially easy to deploy. We based our application on a framework, namely React [1]. A framework is defined as a collection of tools and software that provides some basic architecture to create other software. There are also a number of no-code tool such as *Wordpress* that can be used to create website. However, in our case, we wanted to create a complete application, and at the same time have full control of all the components in the web-app, therefore, we decided to use a framework called *React* to create the frontend. *React* is a popular framerwork, written in Javascript, used for the creation of all sizes web applications. React uses the philosofy of dividing the app in components. Each component implements its own functionality. It is similar to the paradigm divide et impera. Communication between components is implemented using a custom library which implements the so called *Store*. It is defined as a memory object that stores information at runtime and mantain the status until any components set it to a new state [1]. Moving on to the application itself, Figure 5.1

[1]https://reactjs.org/

shows the Web Application. It is divided in three parts:

- **Rule Creation**: all the boxes on the left of the application are dedicated to the Rule Creation domain. The box *Problem Selection* consent to select a problem between the two problem domains available: Tiger and Velocity Regulation. Then a trace is required to be selected among the ones available that we created. Secondly, in the box *Template Creation* the action that the user wants to analyze, among the ones present in the problem domain, is selected. The user can create one rule template per action. When the action is changed, the rule for the action selected will be shown. Thirdly, in the box *Variable Declaration* the free variables for the rule have to be declared. In the box, there will be a textbox, in which the user is asked to insert the free variable name, and procede to complete the other fields, or insert another variable if needed. Moving on to the *Template Creation* box, this is the part where the rule is created. The user is guided to create the rule, so to respect the syntax, as explained in Section 3. Finally, the last box, *Hard Constraint*, allows the user to insert hard constraints if there are any.
- **Rule synthesis**: whenever the rule is ready, the user can press the button *Send Rule* present in the *Template Creation* box. Once the button is pressed, the rule template is sent to the backend, for the synthetization, which is the process of finding an assignment for the free variables. Once the process terminated, and the assignments are ready, the backend sends the result back, and the complete rule synthetized is shown in the box on the center namely *Rule synthetized*.
- **Analysis of the results**: Finally, the anomalies are displayed, in a bar plot below the *Rule synthetized* box, and listed in a compact table on the right. For each anomaly, some information are displayed: *Run, Step, Action, Beliefs, Severity*. *Run* and *Step* are needed to identify when the anomaly happend. While action is the action choosen by the agent, which is classified as anomalous. Belief is the belief that the POMCP had available to compute the optimal action. Finally, severity is the hellinger distance. There are two kind of anomalies namely: *Anomaly same action, Anomaly different action*. Anomaly same action, is a step in which the belief does not respect the rule synthetized, but POMCP selected the action analyzed anyway. While, anomaly different action, is a step which belief respects the rule synthetized, but the action choosen is different than the one analyzed. Finally, in the velocity regulation domain, it is possible to represent the problem as a graph, it is displayed in the box called *Graph state problem visualization*.

5.1.2 Backend

The application backend can be seen as the hidden logic of the system. In our case, the backend is responsible of managing the various requests coming from the frontend, such as getting the names of the problem domains, or getting all the traces of that problem. We can see the backend as the part of the application which answers to specific questions. We used *REST APIs* for the data exchange between the frontend and the backend. *Application program interface (API)* is defined as a set of definitions and protocols for building and integrating application software [2]. In other words, through an API, a protocol for the communication is defined so that the system can understand the request and fulfill it. *Representational state transfer (REST)* instead, is a software architectural style, that defines how the system should behave. REST is employed, as a higly accepted standard, to create stateless, reliable web-APIs. REST web-APIs are based on the HTTP methods such as *GET, POST, PUT, DELETE* protocol for the communication. In order to create these REST-full based APIs, many frameworks in different programming languages are available. We used *FastAPI*, which is a framework which allows to create RESTfull web-APIs in Python. We decided to use Python for this task, as the novel methodology has been coded in Python, and therefore, using another language would have meant create an inter-process communication system too. But in our case, FastAPI is sufficient for our purporses. Table 5.1 shows the APIs with their relative methods and a description of their usage.

API Name	**Method**	**Description**
get_traces_from_problem	POST	Returns the traces name related to a problem domain
get_all_problems	GET	Returns all the problem domain names
get_attributes_from_problem	POST	Returns attributes such as states and actions given a problem domain name
get_graph_from_trace	POST	Returns the graph representation of the problem domain, given its name, if it exists
send_file	POST	Returns a PDF report of the analysis given all the analysis attributes, such as problem, trace, anomalies.
send_rule	POST	Returns the synthetized rule with the anomalies given in input the rule template

Table 5.1: APIs description

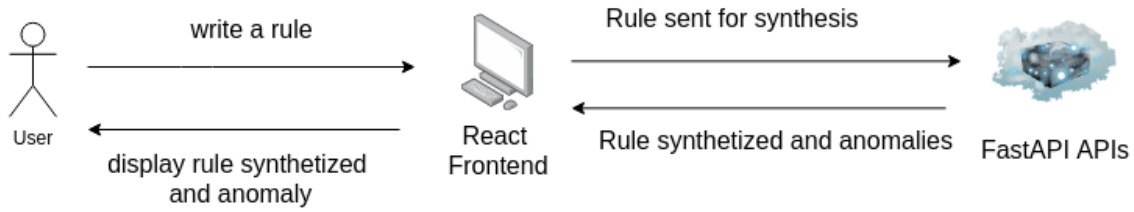


Figure 5.2: Web app architecture

5.1.3 Complete Architecture

Figure 5.2 shows the Web-App architecture. As represented in the figure, the user only directly interacts with the frontend, while the frontend interacts with the backend. Every time an information, that is specifically related to the problem domain (e.g., the traces, the problem domain, the states, etc.) is needed, a request is sent to the backend. For example, the first step for creating a rule is to select a problem domain. But in order to retrieve all the problem domains availables, a request to the backend, specifically to the API *get_all_problem_domain* is done. This way of structuring the application, thus not inserting specific information in the frontend about the problem we want to solve, is effective, because, whenever, a change is needed, for example, a new problem domain has to be added, only the backend is updated and deployed, while the frontend, does not need any change. Furthermore, this way of structuring the problem, has also the advantage of save the user computatation power, and use the computation power of the server, allowing to scale up or scale down the power when needed. For instance, there are rules, that with a big size trace might require a long time to be synthetized, in that case, we can save the user power, and use the server power, that usually is more powerful, and allows to obtain the result faster.

5.2 A Case Study on the Velocity Regulation Problem

5.2.1 Rule Template Creation

In this section, a case study on the *Velocity Regulation* problem domain presented in Section 4.1.2 has been performed. Specifically, we used the novel web-application to repeat the analysis performed in [21], to compare the analysis results, and to make a subjective comparison between the best result interpretability. Furthermore, this case study serves as a demonstration of how to use the web application. The first step of the analysis is the problem selection and the trace selection. Figure 5.3 shows how this step is done. From Figure 4.1 we can use the box on the top left called *Problem Selection* to select a problem and a trace. After the problem has been selected, the box will look like the picture on the bottom left in Figure 5.3.

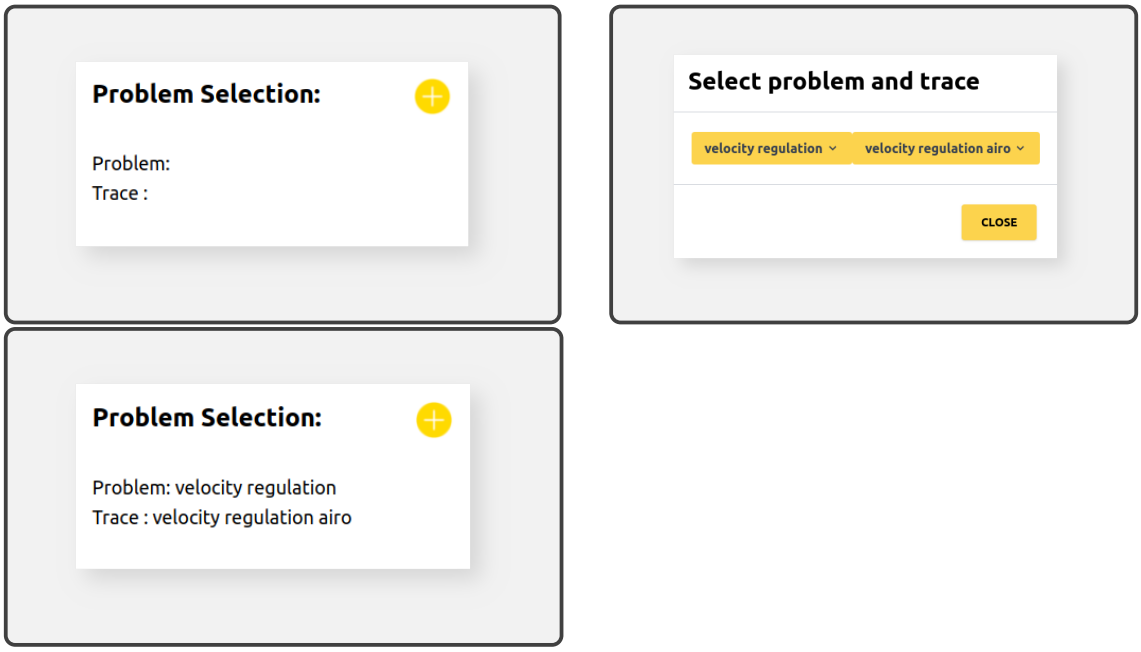


Figure 5.3: First step of the velocity regulation case study

The second step is adding a rule template. In this step, we select the action to analyze. In this case study, we want to write a rule to study the behaviour of a policy generated by POMCP, specifically we want to check anomalous decisions regarding the action *Fast*. Figure 5.4 represents the second step, adding a rule template, and selecting an action. Specifically, pressing the *plus* button on top right of the picture on the top left, a new box is opened, represented by the picture on the top right, which allows to select an action. After pressing the button *Add action*,

the action has been added, and is visible in the box, picture on the bottom left.

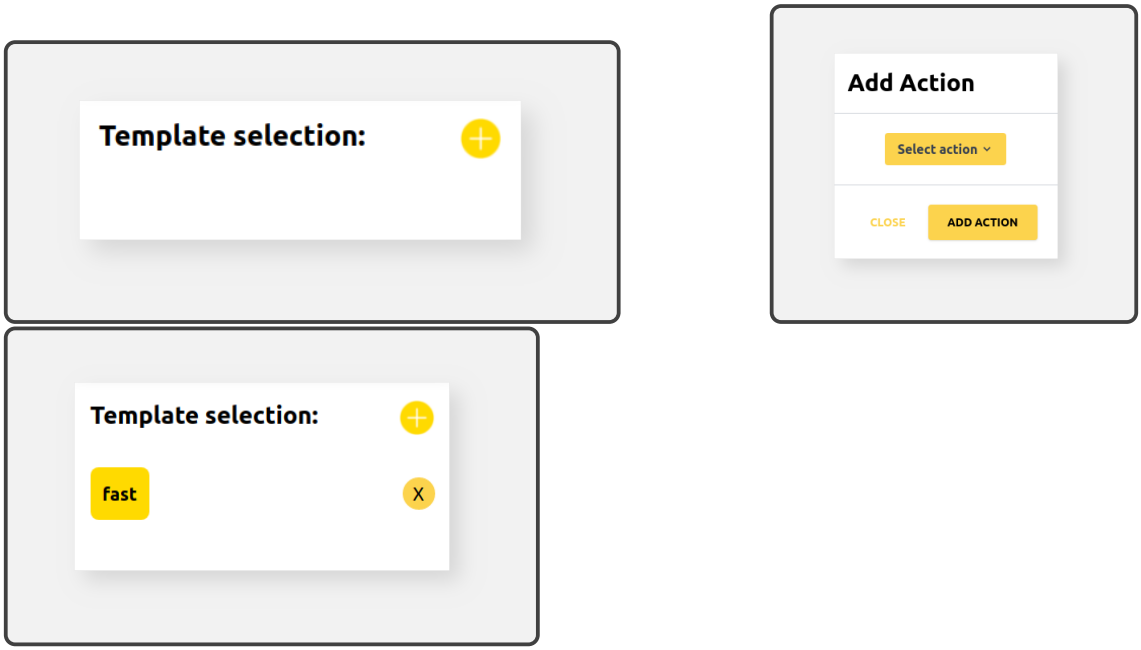


Figure 5.4: Second step, template creation and action selection

The third step is to create the free variables the rule will use. Figure 5.5 shows how free variables can be added to the rule. Free variables, can have custom names, and can also be deleted. In our first analysis, we need just two free variables, one for the state *easy*, the other for the state *difficult*.

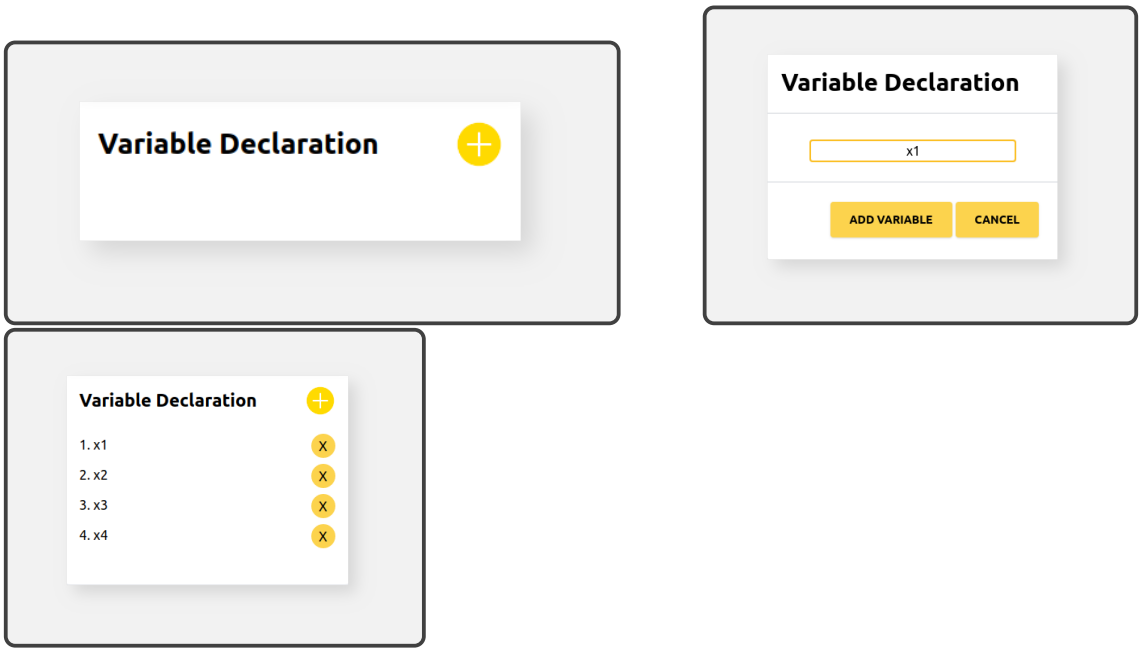


Figure 5.5: Third step, free variable declarations

Moving on to the template creation, Figure 5.6 represents the rule. Specifically, the image on the right, represents the complete rule, while the picture on the left, represents the box before the creation of the rule. The rule is defined to analyze the action *fast*. Specifically, with that rule, we are saying that POMCP should choose action *fast* if the probability of being in a subsegment having as state *easy* is greater or equal to a certain threshold, or the probability of being in subsegment with state *difficult* is less or equal to another threshold. The values of these thresholds, that are the free variables, are found using the procedure presented in Chapter 3, and this job is done server-side. Figure 5.7 shows, how the rule is created. Starting from the left, the state is choosen among the available problem states, then the algebric operator is choosen, picture in the middle, finally the variable is choosen among the free variables declared, picture on the top right. The last step is connecting this subrule using a logic operator, or press *Done* if no other subrules are required, picture bottom left.

The last step regarding the rule template creation, is to insert hard constraints if there are any. In our case, given our prior knowledge, we want the variable $\mathbf{x_1}$ such

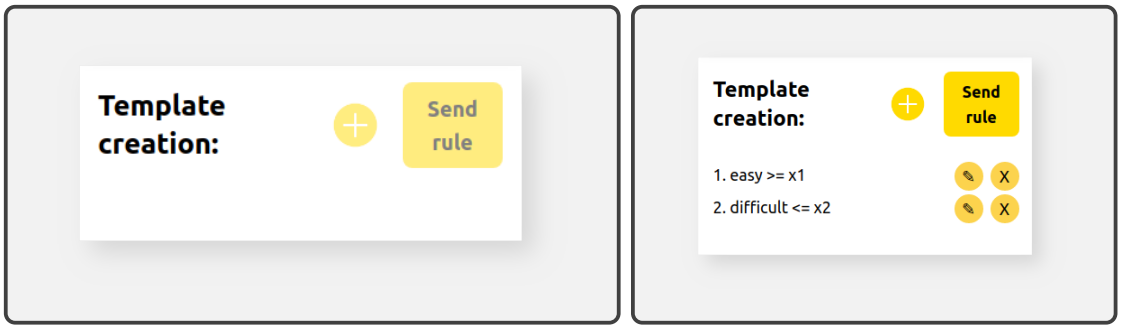


Figure 5.6: Fourths step, template creation

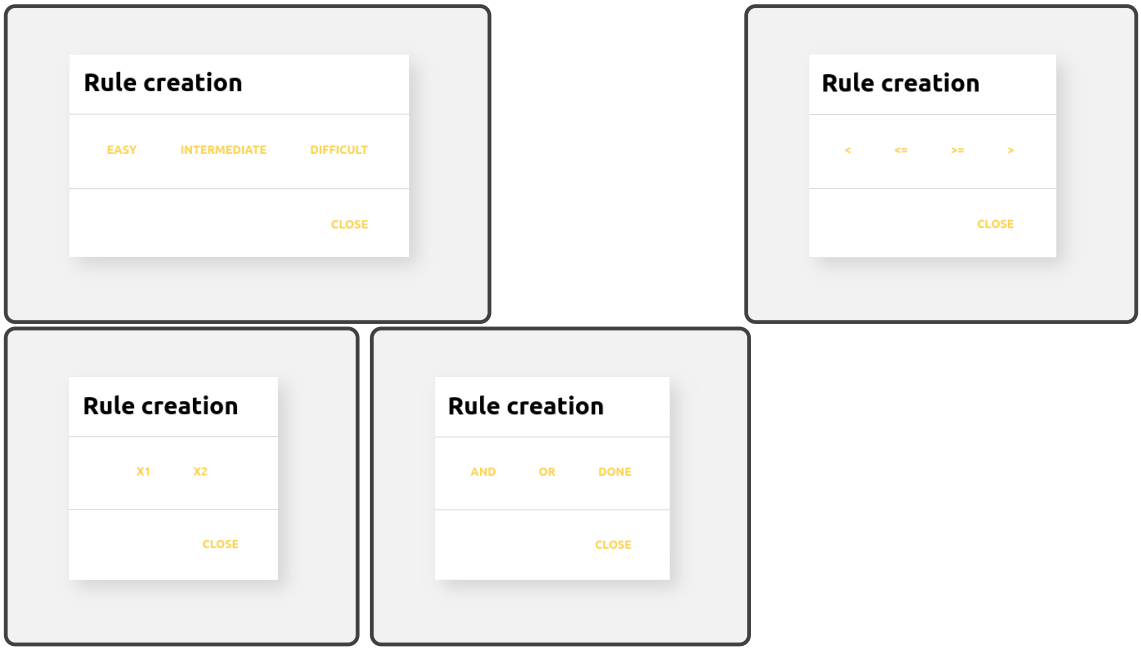


Figure 5.7: Fourths step, template creation

that $\mathbf{x_1} \geq 0.80$. To do so, we can use the apposite box *Hard Constraint*. Figure 5.8 represents the hard constraint creation. While, Figure 5.9 shows the producedure to create the hard constraint. As noted, first, the variable to constraint has to be selected, then, there is the selection of the algebric operator, finally, the numeric value to constraint the variable is asked to be inserted. To recap before proceeding to the rule synthesis, we created a rule to analyze the action *Fast*, particularly, using the prior knowledge of the domain, we know the robot should move fast if the probability of being in a subsegment free from obstacles is high, or if the probability of being in a subsegment with obstacles, is low. This rule is transformed in $easy >= \mathbf{x_1} \vee difficult <= \mathbf{x_2}$, where the value of the variables $\mathbf{x_1}$ and $\mathbf{x_2}$ have to be found using the approach described in Section 3.2.

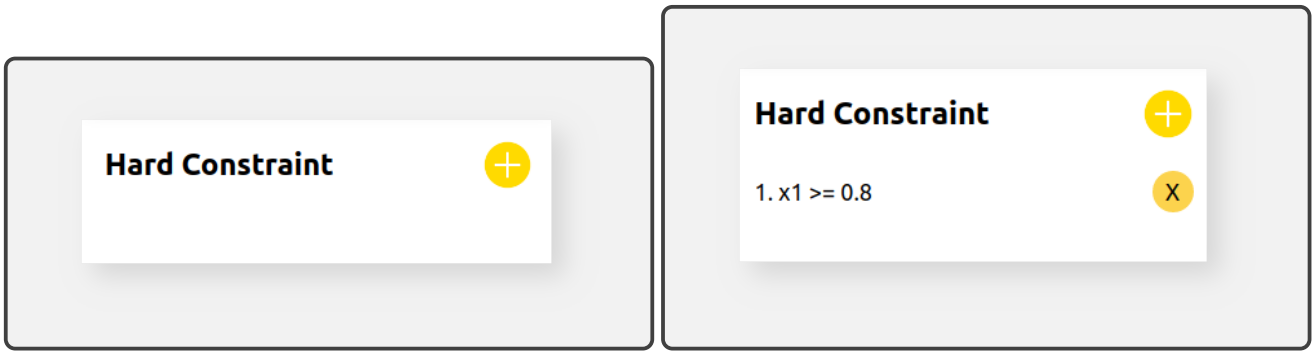


Figure 5.8: Fifth step, hard constraint creation

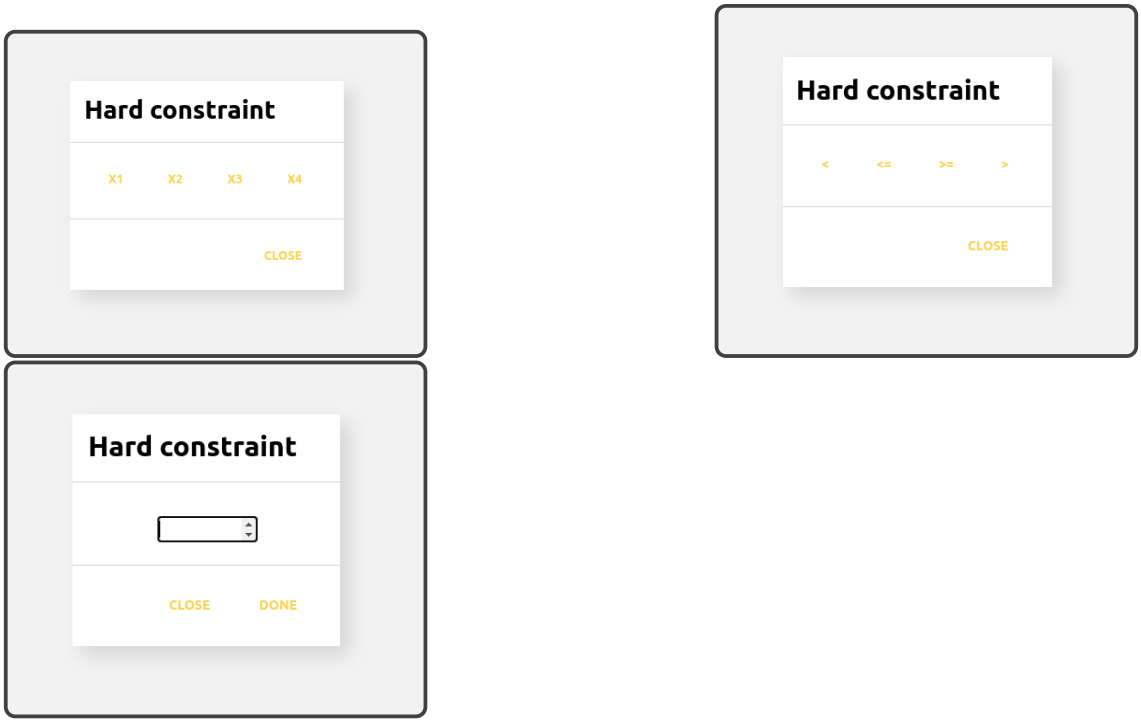


Figure 5.9: Fifth step, hard constraint creation

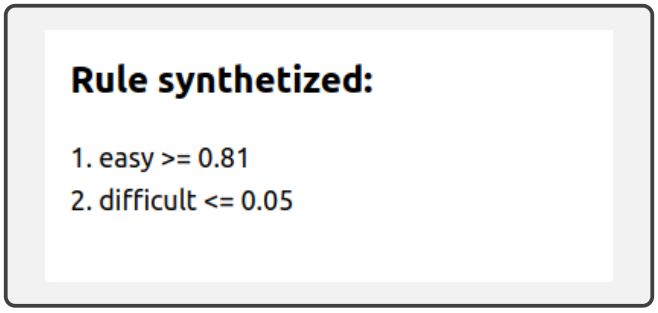


Figure 5.10: Rule synthetized

5.2.2 Rule Synthesis

After the rule template is ready, the *Send rule* button can be pressed, so to send the rule template to the backend and start the synthesis process. When the process ends, the backend returns the rule synthetized, with the free variables assigned. Figure 5.10 shows this. Specifically, we can notice, how the synthesis process assigned $\mathbf{x_1} = 0.81$ and $\mathbf{x_2} = 0.05$. These are the values that maximizes the number of satisfied steps in the trace.

5.2.3 Anomalies analysis

We can then analyze at what cost we approximated the policy, thus what anomalies XPOMCP found.

To do so, we have two instruments, a bar plot, which display the rule as a bar plot, and all anomalies represented as dots in the bar plot. Figure 5.11 represents the bar plot of the rule synthetized. We can notice there are two bar plots, infact there is one bar plot per subrule. In this case there are two subrules, connected using the $\vee$ logic connector, and therefore two subplots ordered by subrule. The first bar plot refers to the first subrule: $easy \geq \mathbf{x_1}$, which synthetized becomes $easy \geq 0.81$, to this corresponds the red area in the bar *easy* from 0 to 0.81. This red area represents all the probability values not acceptable as belief for the corresponding state *easy*. While the green area corresponds to all the acceptable belief for the action *fast*. Instead, the gray area represents the fact that it does not matter where the belief is, because the subrule is not considering that state. The second bar plot represents the second subrule: $difficult \leq \mathbf{x_2}$, which synthetized becomes $difficult \leq 0.05$. Therefore, the green area, from 0 to 0.05 represents all the belief values acceptable that satisfy the rule. While the red area, from 0.06 to 1 is the part of the state belief which should not be accepted by the rule for action *fast*. Moving on to the anomalies represented as dots in the bar plot, Figure 5.11 shows dots inside the red area in

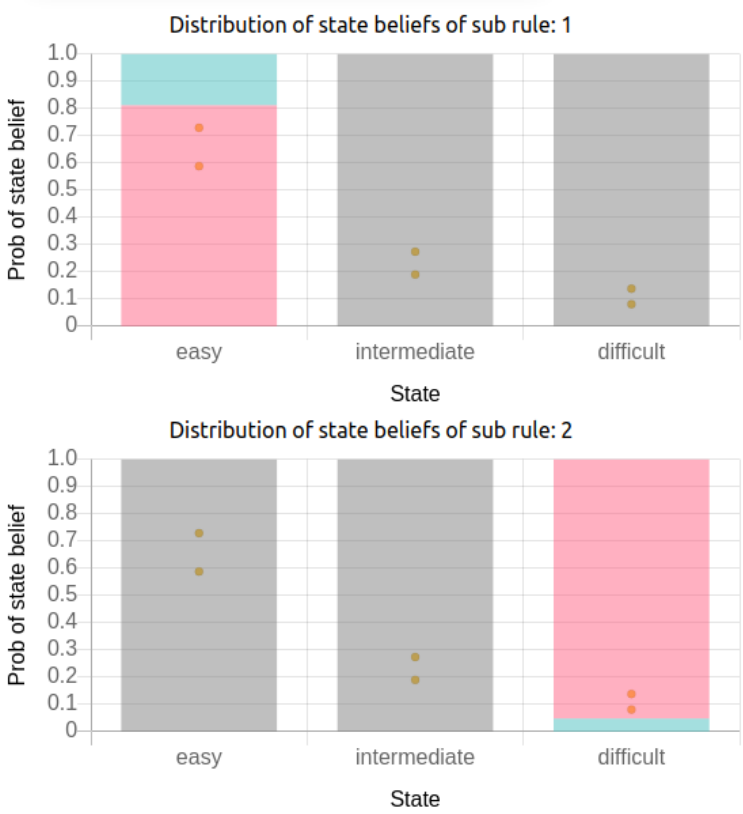


Figure 5.11: Rule bar plot and anomalies representation

the bar plots. These are all the steps of trace with a belief that does not satisfy the rule, but POMCP chooses the action analyzed, that in our case is *fast*. In our opinion these graphical way of representing anomalies, helps in the interpretation of the policy which XPOMCP tries to approximate. Specifically, from the dots in the bar plot, we can see how much the belief step is far from satisfying the rule. At first, all the anomalies are displayed. Then by click the button *run*, we can underline one anomaly displaying just its belief in the bar plot. In fact, a specific table, displaying all the anomalies is also shown. Figure 5.12 represents this table. As discussed in Section 5.1.1, the anomalies are divided in two type, anomalies same action, and anomalies different action. By default, in the table all the anomalies of type same actions are displayed. In case the other type of anomalies wants to be displayed, it can be done easily by clicking the button *Anomalies different action*. Moving on to the analysis, XPOMCP found two anomalies of type *same action*. For this anomaly type, the hellinger distance can be computed, and it is displayed in the column *Severity*. The first anomaly, has an hellinger distance of 0.12, in fact, comparing the belief of the step anomalous with the rule, we see that the step does not satisfy the rule because of the belief for state easy, with value 0.59, while it should have had a value greater or equal to 0.81. Also it does not satisfy the belief of state *difficult*,

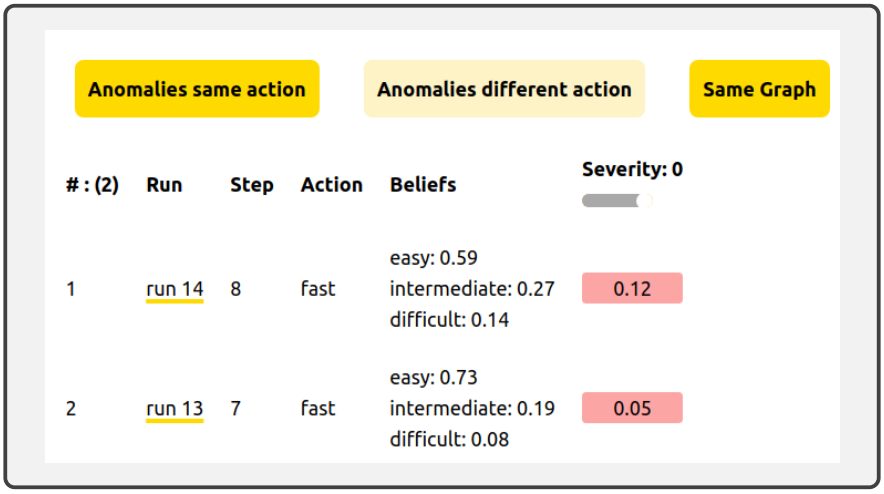


Figure 5.12: Table of anomalies with action equal to the action analyzed

with a step having a value of 0.14, while for being satisfied it should have had a belief less or equal of 0.05. The same analysis can be done for the second anomaly. It has an hellinger distance of 0.05, which means this step is slightly out from the rule being satisfied. Infact, comparing the belief, with the rule, we clearly see that the step has a probability of being in state easy of 0.73, while the rule computed the threshold for state easy at a value equal or greater to 0.81. Also the probability of being in state difficult is 0.08, while to satisfy the rule, the probability should have a value less or equal than 0.05. From this analysis we can conclude that these two anomalous steps found, actually are considered anomalies. Moving on to the second type of anomalies, anomalies different action, first, we note that XPOMCP found nine anomalies. Before analyzing these anomalies, we want to recall that this type of anomalies, are steps which should satisfy the rule because of the belief, but the action of the step is different than the action analyzed. In other words, POMCP should have choosen the action analyzed because in these steps the rule is satisfied. Focusing now on the analysis, from the first three anomalies displayed also in Figure 5.13, we can note that these probability distributions are really close to the rule, therefore we conclude that these are just rule approximation errors. In fact, there is an overlap between action *Fast* and action *Medium*. Note that, with XPOMCP we create an approximation which is explainable. As a consequence, some corner cases might not be considered. To sum up what has been done in this analysis till this point, we wanted to interpret the POMCP policy of the velocity regulation problem, by creating an approximation using a methodology presented in [21]. Thus, we created a rule template to analyze the action *fast*. Then the rule has been synthetized by a MAX-SMT procedure finding the value of the free

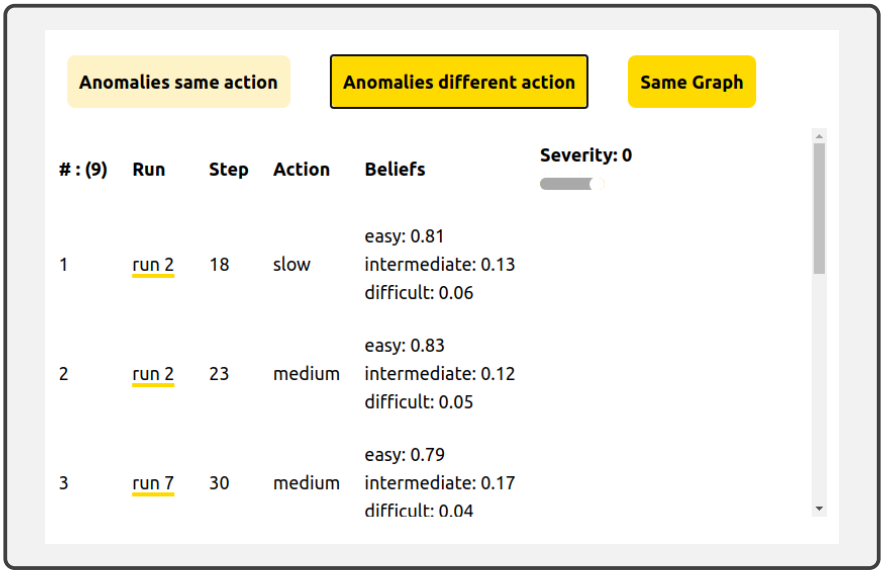


Figure 5.13: Table anomalies different action

variables. Finally, we analyzed the results, first analyzing the rule synthetized, and then, analyzing the anomalies XPOMCP found, concluding that some anomalies are due to the rule approximation. We wanted to try enriching the rule to take in consideration also the overlap between state *easy* and *medium*. Specifically the new rule template is:

$$\begin{gathered} r_2 : \text{select action fast when:} \\ easy \geq \mathbf{x_1} \vee \\ difficult \leq \mathbf{x_2} \vee \\ easy \geq \mathbf{x_3} \wedge intermediate \geq \mathbf{x_4} \end{gathered} \tag{5.1}$$

Which after synthesis became:

$$\begin{gathered} r_2 : \text{select action fast when:} \\ easy \geq \mathbf{0.83} \vee \\ difficult \leq \mathbf{0.04} \vee \\ easy \geq \mathbf{0.80} \wedge intermediate \geq \mathbf{0.12} \end{gathered} \tag{5.2}$$

Analyzing the anomalies, we see that the rule approximates better the POMCP behaviour, because only six steps are identified as anomalous. While before, the anomalous steps were nine. Thus, this new rule explained three anomalous steps. Now, analyzing the six anomalous steps, we see that three of them are in subsegment 8.12, which is critical for POMCP, because this subsegment is shorter than the other subsegments. If the problem can be represented as a graph, the web app display

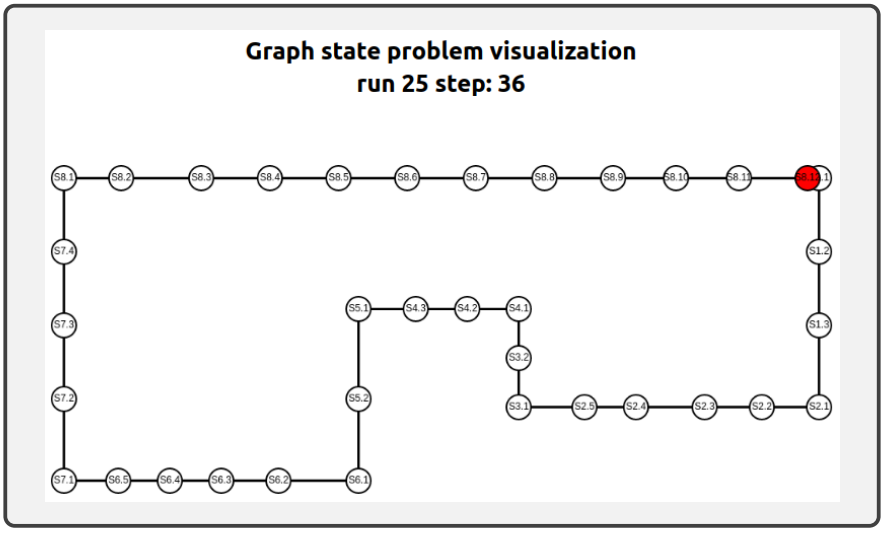


Figure 5.14: Velocity regulation graph representation

its representation in a specific box. Figure 5.14 shows the graph representation of the velocity regulation problem domain, after selecting the anomalous step. Infact, the graph is used also to identify in which node, which in this problem domain is the subsegment, POMCP made an anomalous decision. Going back to the rule interpretation, an idea to avoid those three anomalous steps, is to repeat the rule synthesis, but avoiding considering the subsegment 8.12. We can do this, using the API appositely created in the backend, which leads to only three anomalies, due to the rule approximation. Figure 5.15 shows the last three anomalies. For this example, this is a good compromise between policy approximation and policy interpretation. To recap what have been done:

- Firstly, we created a rule to interpret the decision of POMCP regarding the action *fast*. XPOMCP found out two anomalies with the action equal to the one analyzed (*fast*). After the analysis we concluded that the anomaly with the higher hellinger distance was actually an anomaly, while the other was just a rule approximation error. Then we verified which anomalies of type different action it found out. In total they were nine. But after an analysis we concluded that three of them were due to the rule approximation error. Therefore, we created another rule, to see whether a better result could be achieved.
- The second iteration consisted in the creation of a more sophisticated rule which is an enrichment of the first. Specifically, this rule takes in consideration also the state *intermediate*, and its overlap with state *easy*. XPOMCP found only six anomalies of type same action, which means that the rule actually

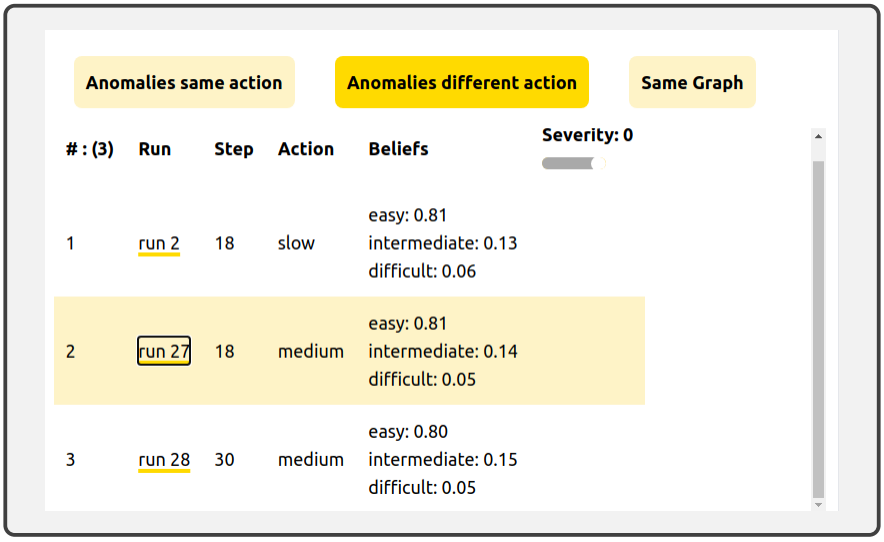


Figure 5.15: Table representing the last three anomalies due to rule approximation

approximates better the POMCP behaviour. Analyzing the anomalous steps, we found out that three of them happends in subsegment 8.12. This happens because subsegment 8.12 is shorter than the other, therefore POMCP is not confident enough in choosing action *fast* even though is mostly sure that the subsegment is free from obstacles. For this reason, we tried to create a third rule, which avoids to take in consideration subsegment 8.12.

- The third iteration, was trying to not considering subsegment 8.12 in the trace analysis, excluding it in the trace parsing process. XPOMCP found only three anomalies of type different action, which, after an analysis, we concluded are due to the rule approximation.

To sum up, what has been done in this Section, we showed how to use the web-based application created to make complex analysis using the XPOMCP methodology. We recreated the analysis presented in [21], which lead to the same results. We found out that this web-application has many advanted with respect to using the hard-coded XPOMCP methodology:

- It is able to presents results better, using interactive visualization systems, such as bar plots, interactive tables, graph for the problem representation.
- It has a feature namely *automatic report generation* which generates a PDF report after the analysis have been complete. This might be useful in case some formal results have to be presented.
- Lastly, there is another feature which allows the user to download the analysis, and eventually upload it again in the web-app. This might be usefull when there is a trace which size is big and XPOMCP takes long to end.

Chapter 6

Conclusion

In this thesis, we studied and compared the methodology to interpret and explain the decisions of POMCP, presented in [21]. Specifically we divided the work in two parts, first we compared XPOMCP with respect to other anomaly detection algorithm: Isolation Forest, on a problem domain in which the exact solution can be computed: Tiger. Secondly, we used another problem domain called velocity regulation, for which an exact policy cannot be computed because of the too large state space. Because in this problem domain an exact solution could not be found, we trained and compared different models to approximate the POMCP behaviour, comparing their accuracy and interpreation. Specifically we first tried to linearize POMCP using a Logistic Regression model, leading the approximation to a good accuracy. We were also able to find a good interpreation of the weights Logistic Regression used to approximate POMCP. We then trained a Neural Network, to verify whether a non linear model could be able to approximate better POMCP. In fact, it was, however, the interpretation of the neural network weights is still a topic under active research by all the machine learning community. We found that XPOMCP is able to do well two things: find anomalies in the POMCP decisions, approximate well the POMCP behaviour and give a clear explanation using propositional logic. To conclude, the methodology presented in [21] was hard-coded in Python, and thereofore, the learning curve to be able to use it could be steep. Therefore we created a web-application aimed to create any sort of analysis with XPOMCP. Specifically, this web-application guides the user to the creation of the rule-template. Once the rule is ready, the result of the approximation is displayed clearly. We used bar plots to plot the state probability distributions, and scatter plot to represent anomalies. We made this graph interactive so the user is able to identify easiliy when the anomaly happened. We used a table to represent all the anomalies, and a graph to represent the problem domain, when possible.

Bibliography

[1] Zustand documentation. *Zustand*.

[2] What is a rest api. *Red hat*, May 2020.

[3] Mark R. Baker and Rajendra B. Patil. Universal approximation theorem for interval neural networks. *Reliab. Comput.*, 4(3):235–239, 1998.

[4] Eugenio Bargiacchi, Diederik M Roijers, and Ann Nowé. Ai-toolbox: A c++ library for reinforcement learning and planning (with python bindings). *J. Mach. Learn. Res.*, 21:102–1, 2020.

[5] Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[6] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[7] Rudolf Beran. Minimum hellinger distance estimates for parametric models. *The Annals of Statistics*, 5(3):445–463, 1977.

[8] Nadia Burkart and Marco F Huber. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70:245–317, 2021.

[9] Alberto Castellini, Georgios Chalkiadakis, and Alessandro Farinelli. Influence of state-variable constraints on partially observable monte carlo planning. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5540–5546. ijcai.org, 2019.

[10] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[12] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[13] Ronald A Howard. Dynamic programming and markov processes. 1960.

[14] Robert Kass, Tim Finin, et al. The need for user models in generating expert system explanations. *International Journal of Expert Systems*, 1(4), 1988.

[15] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

[16] Andrey Kolobov. Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210, 2012.

[17] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[18] Michael L Littman. The witness algorithm: Solving partially observable markov decision processes. *Brown University, Providence, RI*, 1994.

[19] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 413–422. IEEE Computer Society, 2008.

[20] Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli. Identification of unexpected decisions in partially observable monte-carlo planning: A rule-based approach. *arXiv preprint arXiv:2012.12732*, 2020.

[21] Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli. Identification of unexpected decisions in partially observable monte-carlo planning: A rule-based approach. In Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé, editors, *AAMAS '21: 20th International Conference on Autonomous Agents*

and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021, pages 889–897. ACM, 2021.

[22] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[23] Tim Miller. Front end and back end. *Artificial intelligence*, 2021.

[24] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535, 2001.

[25] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814. Omnipress, 2010.

[26] Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[28] Heleen Rutjes, Martijn Willemsen, and Wijnand IJsselsteijn. Considerations on explainable ai and users' mental models. In *Where is the Human? Bridging the Gap Between AI and HCI*, United States, May 2019. Association for Computing Machinery, Inc. CHI 2019 Workshop : Where is the Human? Bridging the Gap Between AI and HCI ; Conference date: 04-05-2019 Through 04-05-2019.

[29] David Silver and Joel Veness. Monte-carlo planning in large pomdps. Neural Information Processing Systems, 2010.

[30] Edward Jay Sondik. *The optimal control of partially observable Markov processes*. Stanford University, 1971.

[31] Petri Ylikoski. The idea of contrastive explanandum. In *Rethinking explanation*, pages 27–42. Springer, 2007.