UNIVERSITÀ DEGLI STUDI DI VERONA

DIPARTIMENTO DI INFORMATICA

DOCTORAL THESIS

Rule-Based Policy Interpretation and Shielding for Partially Observable Monte Carlo Planning

---

Author:
Giulio Mazzi

Advisor:
Alessandro Farinelli
Co-advisor
Alberto Castellini

June 10, 2022

UNIVERSITÀ DEGLI STUDI DI VERONA

Abstract

Dipartimento di Informatica

Doctor of Philosophy

Rule-Based Policy Interpretation and Shielding for Partially Observable Monte Carlo Planning

by Giulio Mazzi

Partially Observable Monte Carlo Planning (POMCP) is a powerful online algorithm that can generate approximate policies for large Partially Observable Markov Decision Processes. The online nature of this method supports scalability by avoiding complete policy representation. However, the lack of an explicit representation of the policy hinders interpretability. In this thesis, we propose a methodology based on Maximum Satisfiability Modulo Theory (MAX-SMT) for analyzing POMCP policies by inspecting their traces, namely, sequences of belief-action pairs generated by the algorithm. The proposed method explores local properties of the policy to build a compact and informative summary of the policy behaviour. This representation exploits a high-level description encoded using logical formulas that domain experts can provide. The final formula can be used to identify unexpected decisions, namely, decisions that violate the expert indications. We show that this identification process can be used offline (to improve the explainability of the policy and to identify anomalous behaviours) or online (to shield the decisions of the POMCP algorithm). We also present an active methodology that can effectively query a POMCP policy to build more reliable descriptions quickly. We extensively evaluate our methodologies on two standard benchmarks for POMDPs, namely, *tiger* and *rocksample*, and on a problem related to velocity regulation in mobile robot navigation. Results show that our approach achieves good performance due to its capability to exploit experts' knowledge of the domains. Specifically, our approach can be used both to identify anomalous behaviours in faulty POMCPs and to improve the performance of the system by using the shielding mechanism. In the first case, we test the methodology against a state-of-the-art anomaly detection algorithm, while in the second, we compared the performance of shielded and unshielded POMCPs. We implemented our methodology in C++, and the code is open-source and available at https://github.com/GiuMaz/XPOMCP.

Acknowledgements

Firstly, I want to offer my special thanks to Prof. Alessandro Farinelli and Dr Alberto Castellini for the continuous support during my PhD and for providing me with the opportunity to prove myself even during one of the hardest periods of my life. Besides my advisor and co-advisor, I would like to thank Prof. Mohan Sridharan and Dr Andrea Orlandini for reviewing my thesis and providing their useful and insightful comments, and Prof. Ferdinando Cicalese and Dr Pietro Sala for giving helpful advice during the development of my PhD.

I thank all the members and ex-members of the ISLa group, specifically Davide, Enrico, and Maddalena, for the stimulating discussions and all the fun we have had in the last years. I will always hold the ISLa's mug as one of my most precious possession! I would also like to thank all the other fellow PhD students that accompanied me during this journey, specifically Luca and Samuele, for making the experience richer and deeper.

Finally, I thank my family and my friends for all the support that they provided during these hard pandemic years. Specifically, I thank my uncle Giampaolo Rossignati, that sadly passed away in March 2021, with all my heart. Your contribution to my personal and intellectual growth is invaluable and will never be forgotten.

Contents

List of Figures

List of Tables

List of Abbreviations

MAX-SMT	**Max**imum **S**atisfiability **M**odulo **T**heory
MDP	**M**arkov **D**ecision **P**rocess
POMCP	**P**artially **O**bservable **M**onte **C**arlo **P**lanning
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
SAT	Boolean **Sat**isfiability
SMT	**S**atisfiability **M**odulo **T**heories
XAI	E**X**plainable **A**rtificial **I**intelligence
XAIP	E**X**plainable **P**lanning
XPOMCP	E**X**plainable **P**artially **O**bservable **M**onte **C**arlo **P**lanning

In memoria di Gianpaolo Rossignati

Chapter 1

Introduction

Artificial Intelligence (AI) methodologies have recently achieved impressive success in several areas such as robotics, speech recognition, computer vision, autonomous driving, and recommendation systems. AI is now largely considered an enabling technology and can dramatically impact production systems and society. A key component of an AI system is the ability to learn. One of the main paradigms of machine learning is *Reinforcement learning (RL)* (Sutton and Barto 1998) that is the problem of learning a strategy that maximizes a cumulative reward, i.e. the sequence of actions that results in the best outcome for an agent that operates in an environment. RL systems have overcome significant challenges in the last decade. The most popular success has been recently obtained with AlphaGo, which reached superhuman performance in the game of Go (Silver, A. Huang, et al. 2016; Silver, Schrittwieser, et al. 2017). Some current research in artificial intelligence aims to transfer these strong results from games, such as Go, Chess, and Atari, in which the dynamics of the environment are entirely known, and the uncertainty comes from the other player to real-world problems in which the environment is very complex and only partially known. Consider, for instance, a robot controlled by an RL-based algorithm that moves in a building, a warehouse, or farmland, where humans and other robots are present. Uncertainty about the environment, in this case, comes from robot-human interaction, weather, light conditions, and the effect of other robots on the environment.

Therefore, applying RL to real-world problems poses new exciting challenges, one of which is *safety*. The robots mentioned above, for instance, should avoid performing actions that make them collide with humans, even if they lack complete knowledge of the surrounding environment. The safety and trustworthiness of AI systems are, for this reason, critical topics of recent research in AI (High-Level Expert Group on AI 2019). In this thesis, we present a methodology that can describe policies generated by a specific reinforcement learning algorithm called Partially Observable Monte Carlo Planning. The proposed approach generates rules that combine high-level insight expressed by a human expert with an analysis of the real execution of the algorithm. These descriptions provide a compact and human-readable representation of the behaviour of a policy. They can automatically identify anomalous decisions in the policy, both offline and online. We use the offline mode method

to detect bugs and improve the policy under investigation. The online mode allows us to develop shields that can be used in real-time to improve the safety of the policy.

1.1 Planning and reinforcement learning in partially observable environments

Planning in partially observable environments is an important problem in artificial intelligence and reinforcement learning. One of the most popular frameworks for modelling decision-making under uncertainty is Partially Observable Markov Decision Processes (POMDPs) (Cassandra, Littman, and N. L. Zhang 1997). POMDPs encode dynamical systems having partially observable states, where the hidden part of the state can only be estimated from observations with some degree of uncertainty. However, computing exact optimal policies for POMDPs is very complex and unfeasible for large state spaces. Specifically, the problem is PSPACE-hard when the horizon is finite and undecidable when the horizon is infinite (Papadimitriou and Tsitsiklis 1987). However, several approximate and online methods have been proposed to handle real-world instances because of the importance of sequential planning under uncertainty. A pioneering algorithm for this purpose is Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness 2010) which uses Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári 2006) to compute the policy online. POMCP uses a black-box simulator to decide which action to take in a certain belief. This is an instance of *model-based reinforcement learning*, i.e. a RL problem in which the reward function is known (unlike *model-free RL*, in which the function is unknown and must be estimated). This approach is sampling-based and only computes the policy for the specific states the agent reaches while operating. POMCP uses a particle filter to represent the probability distribution over (hidden) states, called belief, given a history of observations. We are interested in this method because *i)* it allows us to deal with partially observable environments and uncertainty, *ii)* it uses an explicit (possibly approximated and learnable) model of the environment, which can speed up policy synthesis compared to model-free RL methods, *iii)* it is online, and therefore it scales to very large state spaces, which are common in real-world problems. A key challenge for POMCP is to identify the rationale behind the selection of each action. The reason is twofold. First, the policy for a specific belief the agent achieves is not available before reaching the belief itself. It is unfeasible to compute the policy for each possible belief reachable by the agent because infinite beliefs exist. Second, the policy is computed in a simulation-based fashion and represented by an MCTS, which is difficult to interpret.

1.2 Explainable AI

The widespread use of AI techniques resulted in deploying AI systems in production environments (e.g., smart industries) and safety-critical scenarios (e.g., autonomous

driving). In these scenarios, the AI systems need to cooperate with humans and take decisions that can have dramatic effects. A key element to allowing the use of AI in these scenarios is the ability to explain their decisions. Recently, this requirement was framed as the field of *Explainable AI (XAI)* (Gunning and Aha 2019). Due to the human need to understand why complex autonomous agents take specific decisions, this novel field is growing in popularity. In this thesis, we consider the problem of *Explainable planning (XAIP)* (Fox, Long, and Magazzeni 2017; Cashmore, Collins, et al. 2019), which focuses on explainability in planning methods. Wrong or unexpected decisions in policies automatically synthesized by planning algorithms can have disastrous impacts if they are used to control, for instance, cyber-physical or robotic systems interacting with humans. Detecting policy flaws that can potentially have harmful effects is fundamental for the safe deployment of autonomous agents.

Explainability is particularly important for POMCP because its approximate and online nature, which is crucial to scale to real-world problems, makes the policy represented by the algorithm very difficult to analyze (Castellini, Marchesini, et al. 2020), with consequent repercussions on the safety of the policy. To address this issue, in this thesis, we focus on analyzing the policies generated by POMCP using a logic-based approach.

1.3 Contributions of this thesis

This thesis proposes a methodology called *Explainable POMCP (XPOMCP)*. It builds rules that can be used to interpret POMCP policies, detect unexpected decisions, and shield the algorithm's execution from unwanted actions.

1.3.1 Rule-based description of POMCP policies

The first contribution of this thesis is to develop a methodology for generating a rule-based description of a POMCP generated policy. In particular, experts are required to provide qualitative information about expected system behaviours as logical templates with free variables, and XPOMCP returns quantitative details about those behaviours based on evidence observed in belief-action traces previously generated by the policy. For instance, in a logistic application for smart-manufacturing (e.g., in (Jansen et al. 2020)), an expert could define a logical rule saying that "the robot should move fast if it is highly confident that the aisle in which it is moving is not cluttered". After analyzing some traces produced by the policy, XPOMCP could explain that the robot, controlled by POMCP, moves fast if the probability of being in a cluttered segment is lower than 3%. The expert can use this information to refine his/her knowledge of the policy and synthesize shields that guarantee some safety properties expected by the expert.

Experts are usually interested in identifying situations where the policy does not satisfy their assumptions and expectations. In the context of the warehouse domain,

a possible question is: "Is there a situation in which the robot moves at high speed even if it is likely that the environment is cluttered?".

XPOMCP requires the expert to define a *rule template* representing a question. The template is a set of logical formulas expressing partially defined assumptions. Parameters of rule templates are then computed from traces by a Satisfiability Modulo Theory (SMT) solver (C. W. Barrett and Tinelli 2018). In particular, we formalize the parameter computation problem as a *MAX-SMT* problem. This encoding allows to express logical formulas using an expressive formalism and to compute optimal assignments when the template is not fully satisfiable, which is very common in analyzing real policies. To facilitate the usage of our methodology, we also introduce a rich and flexible language, based on SMT-LIB (C. Barrett, Fontaine, and Tinelli 2016), that allows us to define rule templates expressing high-level concepts about the expected behaviour of the policy.

1.3.2 Rule-based anomaly detection and shielding

The explainability achieved by XPOMCP using logical formulas to represent policy properties can be used for two main purposes, namely, to identify anomalous decisions taken by the policy (called *anomaly detection* in the following) and to prevent such anomalous decisions during policy execution (called real-time *shielding*). Anomaly detection concerns identifying decisions that violate logical rules representing experts' indications. To perform anomaly detection, we first compute approximated decision boundaries of the expected policy using logical rules. Then, we mark the decisions that do not satisfy these boundaries as unexpected. For instance, a decision to move fast in an aisle having a high probability of being highly cluttered does not satisfy the decision boundaries of the logical rule for action "move fast". Hence, this will be detected as an outlier, i.e., an anomaly. On the other hand, shielding (Bloem et al. 2015; Alshiekh et al. 2018) refers to the capability of XPOMCP to use logical rules to check in real-time the actions selected by POMCP and to block unsafe actions that do not satisfy the indications provided by the expert for the current situation faced by the agent. Specifically, XPOMCP builds a pre-shield that, given a certain belief, returns a list of legal actions. This approach allows combining the safety guarantees specified by the shield with the performance of POMCP in selecting the best action (among the legal ones) using Monte Carlo techniques. We assume that representing properties related to safety is simpler than representing the entire policy. Hence, human-understandable models can represent safety properties, i.e., logical rules in our approach that bridge symbolic and sub-symbolic methods.

1.3.3 Active rule syntehsis

The proposed rule synthesis procedure scales to large instances but requires significant data to perform well. Some common beliefs appear multiple times in many domains, while others' reachable beliefs are significantly more uncommon. These

rare scenarios often provide useful insight into the behaviour of the policy in particular situations. Thus it is relevant to collect them efficiently. To achieve this goal, we present an extension of the methodology, called *Active XPOMCP*, that actively uses the POMCP policy to improve and refine the boundaries of a rule generated with XPOMCP. The active approach leads POMCP toward exploring significant beliefs that were never visited before. This procedure respects the online nature of POMCP, a crucial requirement to scale toward large instances with many possible scenarios (of which only a small fraction is reachable).

1.3.4 Empirical evaluation

We empirically evaluate the proposed methods on an experimental setting composed of two standard benchmark domains for POMDPs, namely, *tiger* and *rocksample*, and a real-world domain involving mobile robot navigation, called *velocity regulation*. To evaluate the ability of XPOMCP to identify anomalous behaviours and preserve the safety of the policy, we consider a POMCP implementation that suffers from two common types of bugs related to an incorrect setting of some POMCP parameters. In one case, the parameter that is wrongly set is the constant c of the Upper Confidence bound for Trees (UCT) method, see (Kocsis and Szepesvári 2006). This parameter has a very subtle effect on the decisions taken by POMCP because a wrong c parameter produces a non-deterministic imbalance of the MCTS used to evaluate action values, which generates wrong decisions from time to time. This is a realistic case study for a wrong usage of POMCP that is challenging to identify. In the other case, we consider using a number of simulations in the Monte Carlo sampling that is too small to achieve good performance. This problem could happen in real-world applications due to limitations in computation time. In our test, we compare the capability to identify anomalous actions of XPOMCP with that of *Isolation Forest* (Liu, Ting, and Zhou 2008), a state-of-the-art anomaly detection algorithm, and show that our methodology can outperform isolation forest in the identification of unexpected decisions. Specifically, our approach improves the Area Under Curve (AUC) by up to 47%. This metric is important because it measures the quality of an approach independently by how its parameters are set. Moreover, the shielding mechanism allows to preserve safety and improves the performance of POMCP when the policy takes unexpected decisions. The shield POMCP improves the average discounted return up to 188.71% compared to the original algorithm. We also present results on the active approach, showing that active XPOMCP outperforms the non-active strategy on two challenging experimental domains, namely, rocksample and velocity regulation. XPOMCP uses a predefined trace generated by executing POMCP without any strategy to search for informative beliefs. We show that *Active XPOMCP* manages to reduce the uncertainty interval (i.e., the reachable beliefs that are not already described by a rule) using less data than XPOMCP and, consequently, it generates accurate rules using a much smaller number of runs than XPOMCP. Furthermore, in the velocity regulation domain, we also show that the

rules generated by *Active XPOMCP* can be more accurate than those produced by XPOMCP. This increase in accuracy results in up to 264% performance improvement when the rules generated with the two methodologies are used to shield the behaviour of POMCP. This happens because the belief space is large, and the predefined trace used by XPOMCP may sample only partially the belief space in the area close to the decision boundary of the policy. Instead, the strategy used by *Active XPOMCP* actively searches the decision boundary and accurately describes it using the logical rule.

1.3.5 Summary of contributions

To summarize, this thesis makes the following contribution to state of the art:

- We provide a methodology, *XPOMCP*, to build logic-based representations of POMCP policies. These descriptions merge expert knowledge and information gathered from observed belief-action traces.
- We define an anomaly detection approach in which the rules generated by XPOMCP are used to identify unexpected decisions taken by POMCP.
- We introduce a shielding procedure for POMCP based on the anomaly detection method, which prevents the planner from selecting actions that do not satisfy the expert's indications.
- We present an active approach that leads the exploration of POMCP toward significant beliefs to make the rule synthesis procedure faster and more precise.
- We empirically evaluate the performance of the proposed methods on an extensive experimental setting concerning three benchmark domains, namely, *tiger*, *rocksample* and a problem of *velocity regulation* for mobile robots.

1.4 Organization of the thesis

The rest of this thesis is organized as follows:

- Chapter 2 presents the related work. It focuses on four main areas: planning with POMCP, explainable AI, safety, and shielding.
- Chapter 3 presents the theoretical notions used in the thesis. It describes the planning problem, particularly for systems modelled as MDPs or POMDPs. It also provides an in-deep description of the POMCP algorithm. The SAT and the SMT problems are described alongside a discussion of the key elements of state-of-the-art SMT solvers. Finally, the benchmarks used to test the presented methodologies are described.
- Chapter 4 describes the rule generation procedure of XPOMCP. The language for writing rule templates is also introduced.

- Chapter 5 explains how the rules generated by XPOMCP can be used to identify anomalous decisions in POMCP policies. It also provides experimental results by comparing the rule-based anomaly detection methodology to an unsupervised state-of-the-art anomaly detection algorithm, i.e. *isolation forest*. Results show that XPOMCP outperforms isolation forests by exploiting the high-level information provided by the expert.
- Chapter 6 presents how the rules generated by XPOMCP can be used to build a shielding mechanism that blocks unexpected decisions in real-time systems. The chapter also presents experimental results for the shielding methodology.
- Chapter 7 presents an extension of XPOMCP, called *Active XPOMCP*, that uses the POMCP algorithm and actively explores an important part of the policy. The chapter presents experimental results that show that this approach is significantly faster than standard (passive) XPOMCP in generating strict rules.
- Finally, Chapter 8 draws conclusions and presents possible future research directions.

1.5 Publications

Most of the work presented in this thesis has been published in top-ranked international conferences, as AAMAS and ICAPS. Specifically, the results presented in Chapter 4 and Chapter 5 has been published in (Mazzi, Castellini, and Farinelli 2021a). An in-deep case study of this methodology has been presented in (Mazzi, Castellini, and Farinelli 2020). The shielding mechanism presented in Chapter 6 is described in (Mazzi, Castellini, and Farinelli 2021c). A combination of the results presented in chapters 4, 5, and 6 with extended experiments is under review as a journal paper for a top-ranked journal. The results presented in Chapter 7 were presented as an extended abstract at AAMAS 2022(Mazzi, Castellini, and Farinelli 2022); an extension of this approach is under review as a journal paper. The results were also presented in workshops, specifically at AIRO (Mazzi, Castellini, and Farinelli 2020), ARMS (as part of AAMS 2021), PLANROB (as part of ICAPS 2021), and OVERLAY (Mazzi, Castellini, and Farinelli 2021b). Finally, some preliminary analysis on the explainability of POMCP that leads to the methodologies developed in this thesis was presented in (Castellini, Marchesini, et al. 2020). The aforementioned publications are listed in the following:

1. Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli (2022). "Active Generation of Logical Rules for POMCP Shielding". In: *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 1696–1698
2. Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli (2021a). "Identification of Unexpected Decisions in Partially Observable Monte Carlo Planning:

A Rule-Based Approach". In: *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*. Ed. by Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé. ACM, pp. 889–897

3. Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli (2020). "Policy Interpretation for Partially Observable Monte-Carlo Planning: a Rule-based Approach". In: *Proceedings of the 7th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2020@AI*IA2020)*. Vol. 2806. CEUR Workshop Proceedings. CEUR-WS.org, pp. 44–48

4. Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli (2021c). "Rule-based Shielding for Partially Observable Monte-Carlo Planning". In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*. Ed. by Susanne Biundo, Minh Do, Robert Goldman, Michael Katz, Qiang Yang, and Hankz Hankui Zhuo. AAAI Press, pp. 243–251

5. Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli (2021b). "Rule-based Shield Synthesis for Partially Observable Monte Carlo Planning". In: *Proceedings of the 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis hosted by the Twelfth International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2021), Padua, Italy, September 22, 2021*. Ed. by Dario Della Monica, Gian Luca Pozzato, and Enrico Scala. Vol. 2987. CEUR Workshop Proceedings. CEUR-WS.org, pp. 19–23

6. Alberto Castellini, Enrico Marchesini, Giulio Mazzi, and Alessandro Farinelli (2020). "Explaining the Influence of Prior Knowledge on POMCP Policies". In: *Multi-Agent Systems and Agreement Technologies - 17th European Conference, EUMAS 2020, and 7th International Conference, AT 2020, Thessaloniki, Greece, September 14-15, 2020, Revised Selected Papers*. Ed. by Nick Bassiliades, Georgios Chalkiadakis, and Dave de Jonge. Vol. 12520. Lecture Notes in Computer Science. Springer, pp. 261–276

Chapter 2

Related Work

We discuss four main research areas related to this thesis: POMCP-based planning, explainable planning, safety in AI systems, and shielding. Notice that a general discussion on planning and satisfiability Module Theories is presented in Chapter 3 as background material, while this chapter focuses on analyzing the approaches in state-of-the-art literature having larger similarities with XPOMCP. We also explain the main features that differentiate XPOMCP from the other approaches.

2.1 Planning with POMCP

POMCP has been first introduced in 2010 (Silver and Veness 2010) as an extension of the UCT algorithm (Kocsis and Szepesvári 2006) to partially observable environments. Both UCT and POMCP aim at solving problems in large state spaces, and they employ MCTS (Coulom 2006) for this purpose. UCT works on Markov decision processes (MDPs) (Russell and Norvig 2020) and POMCP on POMDPs (Smallwood and Sondik 1973; Cassandra, Littman, and N. L. Zhang 1997). We present a detailed description of the algorithm in Section 3.1.4. Several extensions of POMCP have been realized since then, and many applications have also been developed. To mention a few of these methodologies, we remember an algorithm called BA-POMCP (Katt, Oliehoek, and Amato 2017) which extends POMCP to Bayesian Adaptive POMDPs, allowing the model of the environment to be learned during execution. A version of POMCP for scalable planning in multiagent POMDPs (Amato and Oliehoek 2015); a scalable extension of POMCP for dealing with cost constraints (Lee et al. 2018); another extension, presented in (Castellini, Chalkiadakis, and Farinelli 2019), exploits prior knowledge on state-variable relationships. Regarding applications of POMCP, some examples come from the exploration of partially known environments with robots (Lauri and Ritala 2016) and active visual search in indoor environments (Yiming Wang et al. 2020), but also other applications are available in the literature (Goldhoorn et al. 2014). MCTS-based approaches have been recently used also for developing agents with superhuman performance in the game of Go (Silver, A. Huang, et al. 2016; Silver, Hubert, et al. 2018).

The goal of our work is not to propose a methodological extension or a new application of POMCP. Instead, we propose a symbolic approach based on logic

rules for explaining the (non-symbolic) POMCP-based policies that are difficult to explain because they are generated online by a complex stochastic process based on MCTS. Moreover, the logic rules generated by XPOMCP are also used to guarantee POMCP policies' safety through a tool for identifying anomalous (i.e., unexpected) decisions and a shielding approach for substituting in real-time anomalous decisions with decisions expected by experts.

2.2 Explainability

Explainable Artificial Intelligence (XAI) (Gunning and Aha 2019) is a rapidly growing research field (Burkart and Huber 2021) focusing on human interpretability and understanding of artificial intelligence (AI) systems. In particular, Explainable planning (XAIP) (Cashmore, Collins, et al. 2019; Fox, Long, and Magazzeni 2017; Anjomshoae et al. 2019) aims at developing planning tools that come with justifications for the decisions they make. These explanations are particularly important when agents and humans interact (Chakraborti et al. 2017; Y. Zhang et al. 2017; Langley et al. 2017). Our methodology supports this interaction by using the expert's insight as a basis for the explanation. This approach leads to compact explanations that can also be used to highlight when the policy and the expert's expectations are different.

A particularly interesting class of questions analyzed in XAIP is known as *contrastive question* (Fox, Long, and Magazzeni 2017; Brandão et al. 2021; Krarup et al. 2021), in which a user can ask an agent why it takes a certain decision instead of another one that the expert believes to be better. The system then should answer by motivating the advantage of its choice over the alternative one. Contrastive questions are difficult to answer in online frameworks like POMCP because the information required is often unavailable to the agent. We, therefore, do not use contrastive questions but ground the interaction between humans and planners on logical formulas. These formulas frame the expert's insight and can be used to identify the decisions in contrast with the expert's expectations. While it is possible to compute metrics that capture interesting properties of POMCP's generated policies (Castellini, Marchesini, et al. 2020), the knowledge provided by the expert proves to be crucial for proper computing representations. In particular, our approach can be used as an iterative process in which the expert can refine these logical rules to acquire a new understanding of specific properties of the policy by analyzing the execution traces generated by a POMCP agent. A detailed example of iterative refinement of logic rules performed by XPOMCP has been recently presented (Mazzi, Castellini, and Farinelli 2020).

A methodology that builds explanations based on logical formulas by analyzing a series of events is presented in (Costa and Dasgupta 2021). The approach is based on temporal logic and decision trees and builds sequential explanations (i.e., a relation between an event and past events in the sequence). This is different from our approach because a rule generated by XPOMCP is a compact representation of

a policy (i.e., a function that maps beliefs to actions) based on MAX-SMT formulas, not a sequential explanation for events in a time series based on temporal logic. In particular, we build a compact representation of the policy by combining the analysis of multiple execution traces into a single rule to highlight the important aspects of the policy and not a justification for the decisions in a specific trace. Another logic-based methodology is presented (Mota and Sridharan 2021; Mota, Sridharan, and Leonardis 2021; Sridharan and Mota 2020). It presents an architecture that combines non-monotonic logical reasoning and incomplete knowledge bases to build explanations for the decisions taken by a complex robotic system. This is different from our approach for two main reasons. First, XPOMCP generates rules that present a compact summary of the policy behaviour, not on-demand explanations of actions based on a knowledge base. Second, we base our method on MAX-SMT, not non-monotonic logic. This distinction is important because it leads to handling the domains' uncertainty in different ways. Specifically, we consider uncertainty as a probability distribution over possible domains, while the non-monotonic logic is based on defeasible inferences. However, extending XPOMCP to other logic is an interesting research direction, as discussed in Chapter 8.

2.3 Safety in AI Systems

Safety of artificial intelligence methods has recently become a central research topic (Seshia and Sadigh 2016; McAllister et al. 2017; P. S. Thomas et al. 2019). Logic-based approaches have been developed, for instance, to verify the safety of neural networks (Zakrzewski 2001; Cheng, Nührenberg, and Ruess 2017; X. Huang et al. 2017; Katz et al. 2017; Narodytska et al. 2018; Bunel et al. 2018; Gehr et al. 2018; Jia and Rinard 2020) motivated by the need to use these tools in safety-critical applications, such as autonomous driving. These methodologies mainly attempt to solve the verification problem on neural networks, which are highly non-linear because of the activation functions employed. Hence standard verification methods are not applicable. The approaches cited above then adapt verification methods to work on specific classes of neural networks or approximate neural networks to allow the applicability of standard verification methods. In both cases, safety requirements must be mathematically specified in advance by logical formulas. Verification methodologies confirm that a network satisfies those requirements or provide counterexamples showing cases where the SMT formulas do not satisfy safety requirements. Our methodology differs from these because our goal is not to verify predefined safety requirements written with logical rules but to use logical rules to support the interaction between the human and the planner. Namely, we use these formulas as rule templates describing the properties of the policy we want to investigate. XPOMCP then uses a MAX-SMT-based procedure to instantiate the rule parameters according to the observed traces and provides the list of observations that do not satisfy the instantiated rule. In this way, XPOMCP can provide novel insight to the domain

expert. This process can be iterated since the human can change the rule template and check different aspects of the policy in this way until an explanation of which the human is confident is achieved. The final explanation can be used as a shield to prevent unsafe decisions of the policy, or it can be used to identify the reason for the wrong behaviour of the policy (e.g., a wrong parameter, a small number of simulations performed by the MCTS, or a wrong model of the environment).

The literature also provides various approaches specifically focused on the synthesis of safety policies and verifying safety properties in policies for reinforcement learning and POMDP. In these cases, temporal logic is often used to represent safety requirements since the involved models are sequential. SMT-based approaches are used, for instance, in (Norman, Parker, and Zou 2017; Yue Wang, Chaudhuri, and Kavraki 2018; Bastani, Pu, and Solar-Lezama 2018; Verma et al. 2018; Cashmore, Magazzeni, and Zehtabi 2020). These frameworks use formal approaches to synthesize a policy that satisfies predefined properties, an approach that presents scalability problems and has never been applied to online and approximate solvers such as POMCP. In contrast, we use a MAX-SMT solver to generate a human-readable representation of pre-computed policies that summarizes properties of interest. This enhances policy explainability without altering the policy itself and thus can be used to analyze complex policies generated online. This makes it possible to deal with very large state spaces. This explanation can then be used to build a shielding mechanism that blocks decisions that do not respect the desired properties. To the best of our knowledge, there is no approach equivalent to XPOMCP in the literature that can build a mechanism that checks when high-level properties hold on POMCP.

A related problem is *safe reinforcement learning* (Junges, Jansen, Dehnert, et al. 2016; Fulton and Platzer 2018; Hasanbeig, Abate, and Kroening 2020), in which an agent must develop a policy for model-free systems (i.e., systems for which the reward function is unknown) while respecting some safety constraints. Our problem differs because, in POMCP, we have a black-box simulator that can be used as a model. We exploit this element to build effective shields, specifically in the active approach presented in Chapter 7.

The methodology recently presented in (Junges, Jansen, and Seshia 2021) can be used to compute policies for POMDPs that satisfies safety requirements. Specifically, The approach uses a SAT solver to compute *winning regions*, a subset of beliefs for which a policy never reaches a forbidden state exists. This approach provides strong guarantees but cannot scale to large instances of POMDPs. Instead, our approach exploits the online nature of POMCP to handle large problems.

Another approach for verifying properties in probabilistic systems is statistical model checking (SMC), as discussed in (Klauck et al. 2020). The approach is applied to POMCP in (Newaz, Chaudhuri, and Kavraki 2019) where SMC is used to verify that qualitative objectives, specified on possible states of the environment (i.e., safe reachability), are satisfied with a certain confidence level. This is different from our methodology since we specify properties on beliefs instead of on states, and we use

them to build a shield that blocks undesired behaviours. SMC also requires a large number of particles to generate reliable results, while our methodology does not have such a requirement.

2.4 Shielding

Shields have been used to verify critical properties of a complex hardware design in cases when traditional verification methods are unusable because of scalability issues (Bloem et al. 2015; Knighofer et al. 2017). A shielding mechanism for reinforcement learning agents is presented in (Alshiekh et al. 2018). This work presents two kinds of shielding mechanisms: a preemptive shield that selects a priori to which actions are allowed and a post-posed shield that intervenes only after the reinforcement learning algorithm selects an action. These shields enforce safety constraints expressed in temporal logic. An extension of this work is presented in (Jansen et al. 2020) where the shield is specialized to prevent unsafe actions due to the agent's exploration of reinforcement learning-based policies. Our shielding mechanism, firstly introduced in (Mazzi, Castellini, and Farinelli 2021c), works as a preemptive shield, but our methodology has two important differences from the abovementioned methodology. Namely, we focus on partially observable problems. Thus our approach must deal with uncertainty, and we build the shield from a high-level representation. Specifically, the user does not have to specify the details of the shielding mechanism because XPOMCP computes them by analyzing execution traces.

A recent approach (Zhu et al. 2019) presents a shielding mechanism based on a simplified policy representation. The methodology verifies properties related to the safety of a fully observable system modelled by Markov Decision Processes (MDPs). The approach works on a pre-trained neural network representing a black-box policy. It is composed of three main stages: *synthesis*, in which the complex neural policy is approximated using linear formulas that behave as close as possible to the original neural policy while being also easy to analyze; *verification*, in which an off-the-shelf SMT solver verifies the safety of the approximated policy; *shielding*, in which the behaviour of the neural policy is monitored in real-time and the synthesized (approximated) policy is used as a shield to substitute unsafe actions suggested by the neural policy. This differs from our work for two main reasons: first, we work on partially observable environments instead of completely observable environments, and our logical formulas work on beliefs instead of states (i.e., the formulas that we have to use to describe the policy are computationally more expensive and cannot be used in the same way); second, our approach does not require a full description of the behaviour of the policy and can be used to focus the investigation on the most critical properties of a policy. Furthermore, in (Zhu et al. 2019) the logical formulas are used as an input to the verification tool that uses them to verify safety properties. In contrast, we use logical formulas directly on the POMCP policy to generate

human-readable representations of some of its properties. These logical formulas are then used to detect anomalous actions and perform a shield.

Chapter 3

Background

This chapter provides background for *planning* and *Satisfiability Modulo Theories (SMT)*, which are the two main problems addressed in this thesis. Furthermore, it describes the benchmarks used in the experiments. The methodologies presented in this thesis aim to solve the problems of planning in partially observable environments using the theoretical framework of *Partially Observable Markov Decision Processes (POMDP)*. In this context, the correct state of the system is unknown, but it can be estimated using imperfect observations received from the environment. We use a logic-based approach to generate rules that compactly describe the behaviour of a POMDP agent. SMT formulas are used to encode the rule that describes which actions the agent can take based on the currently available information on the actual state of the system.

3.1 Planning in partially observable domains

Artificial Intelligence is the study of rational actions (Russell and Norvig 2020). A central topic n this context is *AI planning* which concerns devising strategies for executing optimal sequences of actions. Planning systems must decide which action to take given the information collected until the current time instant. Actions have an impact on the state of the system. Thus, it is paramount to consider both the action that must be taken immediately and the impact that this action has on future decisions. In this thesis, we focus on planning problems in which the actions are stochastic (i.e., the outcome of the actions is not always guaranteed) and the environment surrounding the agent is only partially observable (i.e., the agent must reason about imperfect information). In particular, we use *Partially Observable Markov Decision Processes (POMDPs)* to consider the uncertainty in the planning problem. POMDPs are an elegant mathematical framework that can model complex systems for which the *Markov Property* holds, namely, the effect of the previous actions is wholly encoded in the current state. Therefore the agent can reason on the state only, disregarding the previous states. Computing optimal plans for a POMDP is a computationally challenging problem, and most of the state-of-the-art algorithms provide approximate and online solutions (i.e., solutions that consider only a limited part of the solution space). An algorithm that proved to be particularly effective for planning

in large POMDP instances is *Partially Observable Monte-Carlo Planning (POMCP)* (Silver and Veness 2010). This thesis proposes a methodology for better understanding the online policies generated by the algorithm to make them more understandable to human experts. These explanations are critical to building systems that avoid risky decisions during their execution. This section provides the theoretical background about Markov Decisions Processes and Partially Observable MDP. We also present an in-depth description of the POMCP algorithm.

3.1.1 Markov Decision Processes

A *Markov Decision Process (MDP)* is a mathematical framework used to model decision processes with stochastic actions. An MDP is defined as a tuple (S, A, T, R, γ) where:

- S is a set of states.
- A is a set of actions.
- $T: S \times A \to \Pi(S)$ is the *state-transition model*.
- $R: S \times A \to \Pi(\mathbb{R})$ is the immediate *reward function*.
- $\gamma \in [0, 1]$ is a *discount factor*.

In a *Finite MDP* sets S and A have a finite number of elements. With $\Pi(S)$ ($\Pi(\mathbb{R})$) we define a probability distribution over the set of states S (the real numbers $\mathbb{R}$). To present explicit distribution, we use $\Pi(x_1 : p_1, \ldots, x_k : p_k)$ to define a distribution over k values where each value i has probability p_i. Function T is used to specify the dynamics of the system. Specifically, equation

$$T(s'|s, a) = Pr(s_t = s'|s_{t-1} = s, a_{t-1} = a) \tag{3.1}$$

specifies the probability of transitioning toward a certain state s' if action a is selected in state s. A state has the *Markov Property* if it contains all the information required to express the dynamics of the system. The goal of an MDP agent is to maximize the expected discounted return

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]. \tag{3.2}$$

To describe the behaviour of a system, we use a *policy* $\pi: S \times A$, which is a function that maps states to actions. We compute the optimal policy (i.e., the policy that maximizes the expected discounted return) by computing an *optimal value function*. A value function $v_\pi(s)$ is a mapping that specifies the expected return of each state s given a policy π. A useful formulation of the value function, known as the *Bellman Equation* is defined as follow:

$$v_\pi(s) = \sum_a \pi(s) \cdot \sum_{s',r} T(s'|s, a)[R(r|s, a) + \gamma v_\pi(s')], \text{ for all } s \in S \tag{3.3}$$

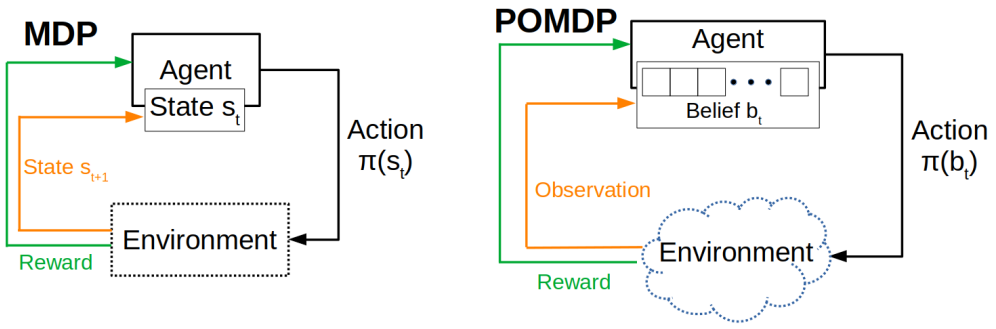


FIGURE 3.1: Interaction between the agent and the environment in an MDP (left) and in a POMDP (right)

This equation divides the expected return in two components, namely, the immediate rewards $R(r|s,a)$, received by selecting action a in state s, and the future reward $v_\pi(s')$ computed using the value function in the next state s'.

3.1.2 Partially Observable Markov Decision Processes

In this thesis, we focus on an extension of MDP called *Partially Observable Markov Decision Process (POMDP)* (Smallwood and Sondik 1973; Kaelbling, Littman, and Cassandra 1998). POMDP can be used to model systems in which the agent cannot directly observe the actual state of the system, but it must estimate it by using (imperfect) observations received from the environment. A POMDP is defined as a tuple $(S, A, O, T, Z, R, \gamma)$ where:

- S is a set of partially observable *states*.
- A is a set of *actions*.
- Z is a finite set of *observations*.
- T: $S \times A \to \Pi(S)$ is the *state-transition model* .
- O: $S \times A \to \Pi(Z)$ is the *observation model* (with $\Pi(Z)$ space of probability distributions over observations).
- R: $S \times A \to \Pi(\mathbb{R})$ is the *reward function* (where $\Pi(\mathbb{R})$ is a probability distribution over real numbers).
- $\gamma \in [0, 1]$ is a *discount factor*.

In which S, A, T, R and γ are defined as in standard MDP, and Z, O are added to handle the partial observability of the environment. Figure 3.1 shows a schematic of the interaction between agent and environment for MDPs and POMDPs. An *history* h is a sequence of actions and observations. We denote $h_t = \{a_1, o_1, \ldots, a_t, o_t\}$ as the history from the beginning to the discrete time-step t. With hao, we express that the agent selects action a after history h, and it receives an observation o from

the environment, leading it toward a new belief $\mathcal{B}$. For each story h, we can define a *belief* $\mathcal{B}(s,h) = Pr(s_t = s|h_t = h)$, which is a probability distribution over the possible states. The real initial state of the system $s_0 \in S$ is one of the possible states contained in the initial probability distribution $\mathcal{B}_0$.

A solution for a POMDP is a *policy*, namely a function π: $B \rightarrow A$ that maps beliefs into actions (where B represents the belief-space). Similarly to MDP, this policy must maximize the *discounted return* defined as

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]. \tag{3.4}$$

The value function $V^{\pi}(h)$ for a history h is defined as the expected return achieved by applying policy π from history h. Thus, an optimal policy π^* requires an *optimal value function* $V^* = \max_{\pi} V^{\pi}(h)$.

3.1.3 POMDP solvers

Computing an optimal policy for a POMDP is a challenging theoretical computing problem. Specifically, it proves to be PSPACE-hard in the cases of finite horizons and undecidable when the horizon is infinite (Papadimitriou and Tsitsiklis 1987). Two elements, namely, the *curse of dimensionality* and the *curse of history*, are the main reasons for this complexity (Pineau, Gordon, and Thrun 2006). The curse of dimensionality explains that the complexity of optimally computing returns for the value function increases exponentially with the number of states. The curse of history expresses that the number of histories that must be evaluated grows exponentially in the size of the horizon. Thus, exact complete algorithms, such as value iteration (Kaelbling, Littman, and Cassandra 1998), does not scale to large instances.

To overcome these limitations, it is possible to compute *online* policies, which are approaches that avoid full-width exploration by computing the policy only from the current history and forward. A popular approach is to use value iteration, like in (Pineau, Gordon, and Thrun 2006) and (Ross et al. 2008), to compute local approximations for the value function. These approaches require an explicit model of the POMDP, and they construct a search tree using a best-first approach.

A different approach to online policy generation is to use Monte Carlo techniques (Bertsekas and Castañón 1999; Kearns, Mansour, and Ng 1999). These methods use a black-box simulator as a generative model to sample successor states, observation and reward, given the current state and an action. Thus, an explicit model of the POMDP is not required. These approaches are limited to fixed horizons and sparse samplings. A pioneering approach that overcomes these shortcomings is *Partially Observable Monte Carlo Planning* (Silver and Veness 2010). This algorithm is the main focus of the thesis. Therefore we provide a detailed description of the methodology in the next section.

Algorithm 1: Search

Input: History: h
Output: action: a

while $\neg Timeout()$ **do**
  if $h = \emptyset$ **then**
    $pt \sim \mathcal{B}_0$;
  else
    $pt \sim B(h)$;
  $Simulate(pt, h, 0)$;
return $\arg\max_a V(ha)$;

Algorithm 2: Rollout

Input: Particle: pt, History: h, Depth: dp
Output: Reward: rw

if $\gamma^{dp} < \epsilon$ **then**
  return 0;
$a \leftarrow \pi_{rollout}(h)$;
$[pt', o, rw] \sim Step(pt, a)$;
return $rw + \gamma \cdot Rollout(pt', hao, dp + 1)$;

Algorithm 3: Simulate

Input: Particle: pt, History: h, Depth: dp
Output: Reward: rw

if $\gamma^{dp} < \epsilon$ **then**
  return 0;
if $h \notin Tree$ **then**
  forall $a \in \mathcal{A}$ **do**
    $Tree(ha) \leftarrow (N_{init}(ha), V_{init}(ha), \emptyset)$;
  return $Rollout(pt, h, dp)$;
$a \leftarrow \arg\max_b V(hb) + c \cdot \sqrt{\frac{\log N(h)}{N(hb)}}$;
$[pt', o, rw] \sim Step(pt, a)$;
$rw \leftarrow rw + \gamma \cdot Simulate(pt', hao, dp + 1)$;
$B(h) \leftarrow B(h) \cup \{pt\}$;
$N(h) \leftarrow N(h) + 1$;
$N(ha) \leftarrow N(ha) + 1$;
$V(ha) \leftarrow V(ha) + \frac{rw - V(ha)}{N(ha)}$;
return rw;

FIGURE 3.2: Main elements of the POMCP algorithm.

3.1.4 Partially Observable Monte Carlo Planning

In this thesis, we focus on analyzing the *Partially Observable Monte Carlo Planning (POMCP)* (Silver and Veness 2010) to solve POMDPs. POMCP is an *online* algorithm that solves POMDPs by using Monte Carlo sampling techniques. One of the strengths of POMCP is that it does not require an explicit definition of the transition model, observation model, and reward. Instead, it uses a black-box simulator to simulate the environment and estimate the effectiveness of each action. POMCP uses a *Monte Carlo Tree Search* (MCTS) at each time step to explore the belief space and select the best action. By employing MCTS, POMCP can break both the curse of history and dimensionality. The algorithm employs *Upper Confidence Bound for Trees (UCT)* (Kocsis and Szepesvári 2006) as a search strategy to select the subtrees to explore and to balance exploration and exploitation during the simulation. The belief is implemented as a *particle filter*, which is a sampling over the possible states updated at every step.

Particle filter

While it is possible to compute a belief state update exactly using the Bayes Theorem (Ross et al. 2008), this proves to be intractable in real-world instances. POMCP

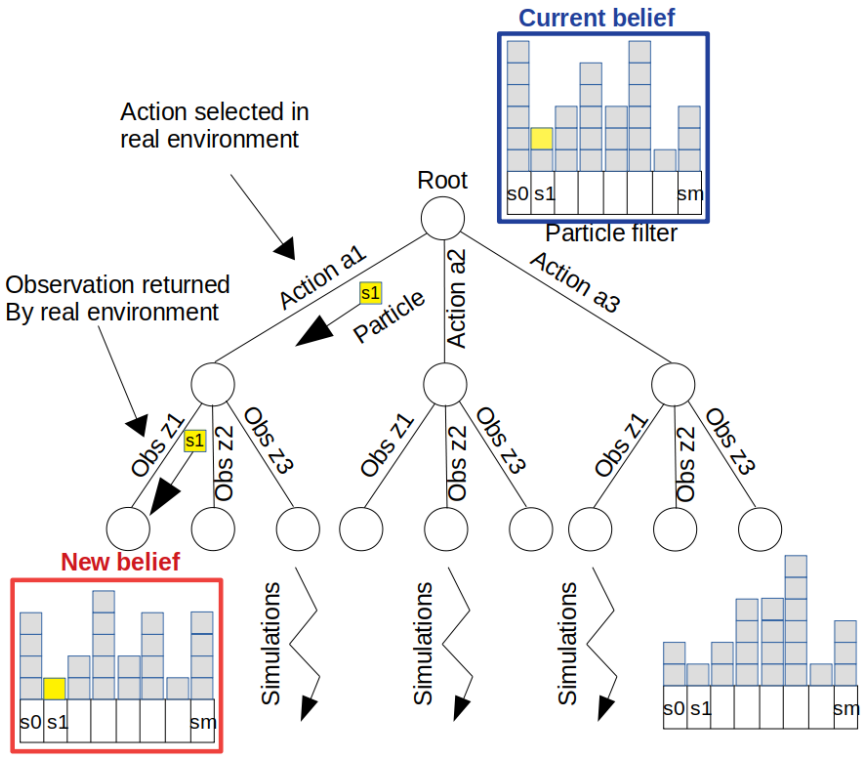


FIGURE 3.3: POMCP Belief Update

uses an unweighted particle filter to approximate a belief to overcome this limitation. Each particle encodes a possible state in the current belief, and it is updated with a Monte-Carlo procedure. The update uses a black box that simulates the system's behaviour and returns an observation, a reward, and a state transition. In this way, it is possible to perform the simulation without requiring a complete explicit model for the POMDP.

At each time step a particle is selected from the filter (Algorithm 1 lines 2–5). In the first step performed by the agent (i.e., step 0), the procedure selects a particle from an initial distribution $\mathcal{B}_0$. The particle is selected from the updated particle filter $B(h)$ in the subsequent steps. The state of the particle taken from the particle filter is used as an initial point to perform a simulation in the Monte Carlo tree (Algorithm 1 line 6). Each simulation is a sequence of action-observation pairs which collects a discounted return. Algorithm 3 presents the pseudocode for the searching procedure of POMCP. If the simulation encounters a new history (Algorithm 3 line 3), it estimates the value of an action in the history using a Rollout policy (Algorithm 2). Otherwise, the simulation selects an action and updates the current node's values according to the results provided by the black box simulator. Specifically, the expected value of each action is approximated by averaging the discounted returns of all simulations starting with that action, according to a backtracking procedure (Algorithm 3 line 13).

At the end of the simulation time (see Algorithm 1 line 1), POMCP uses the best action according to the value function V in the current history. The particle filter is

then updated after performing a step in the real environment, considering the particles in the branch of the tree related to the observation received by that environment. Figure 3.3 shows a graphical representation of this procedure. If required, POMCP can use a particle reinvigoration procedure to generate new particles from the current belief. The reinvigoration step is helpful to avoid particle depletion in which the algorithm does not have enough particles to approximate the true belief.

Partially observable UCT

To decide which action should be selected during the simulation, POMCP uses an extension of the UCT algorithm (Kocsis and Szepesvári 2006) to balance exploration and exploitation. This extension, called *Partialy Observable UCT (PO-UCT)* (Silver and Veness 2010), uses histories h instead of states to handle partially observable environments. Specifically, PO-UCT uses a tree T in which each node stores the expected value function $V(h)$ and the number $N(h)$ of simulations performed for each specific history h. Since POMCP is an online algorithm, these nodes are built when the simulation step explores them, and in general, they are a small subset of all the possible histories. The value function is estimated as the mean return of all the simulations performed from history h. The action selected for the simulation is the one that maximizes the function

$$V_{UCT}(ha) = V(ha) + c \cdot \sqrt{\frac{\log N(h)}{N(ha)}} \tag{3.5}$$

where $V(ha)$ is the current expected mean value achieved by selecting action a in history h, $N(h)$ is the total number of simulations for the current history, and $N(ha)$ is the number of simulations performed with action a in h (Algorithm 3 line 7). The second element of the formula (i.e., $c \cdot \sqrt{\frac{\log N(h)}{N(ha)}}$) is used to encourage the exploration of different actions, and it is higher for rarely used actions (i.e., actions with a low value of $N(ha)$). This element increases as the logarithmic of the total number of simulations $N(h)$, thus its value is higher and more relevant when the number of simulations is low. When both $N(h)$ and $N(ha)$ increase, the value of the denominator $\log N(h)$ dominates, thus the value tends to zero. On the other hand, the first element of the equation (i.e., $V(ha)$) is used to exploit the information currently available to the system since it is higher for actions that prove to be effective. The *exploration constant* c, also known as the reward range, regulates the balance between exploration and exploitation in the system. In particular, if $c = 0$, the algorithm acts greedy on the available information. This value is important to achieve good performance, and it must be hand-tuned for each application domain.

3.2 Satisfiability Modulo Theory

This thesis proposes a logic-based methodology used to build rules that describe the behaviour of a POMCP system. We express these rules using *Satisfiability Modulo Theories (SMT)*, which is the problem of reasoning on the satisfiability of formulas involving propositional logic and specific quantifier-free fragments of first-order logic, called *Background Theories*, that fix the interpretation of certain function symbols. In this thesis, we formulate the problem of generating rules describing POMCP behaviour as a *Maximum Satisfiability Modulo Theory (MAX-SMT)* problem, a formalism allowing us to encode rules that are abstract and approximate. This section describes the main elements of an SMT solver and the MAX-SMT problem used for the rule synthesis procedure.

3.2.1 Satisfiability Modulo Theory solvers

Although it is possible to use a generic first-order theorem prover to solve an SMT formula, this is known to be computationally intractable in practice. General solvers reason on logical formula without considering specific theory. However, we want to reason with fixed interpretation for predicates and functions symbols in most instances. Thus it is beneficial to use specialized theory solvers. For example, consider the formula

$$(\neg p \vee (x > y)) \wedge (x + y - z \leq 5). \tag{3.6}$$

It contains a mix of propositional logic (i.e., $\vee, \wedge, \neg$) and linear arithmetic functions (i.e., $+, -, \leq, >$), and predicates (i.e., 5). We are interested in a solution that uses a standard interpretation for the arithmetic terms, not a general first-order procedure that could apply non-standard interpretation to some symbols (e.g., by considering $>$ as a $\leq$ comparison). While it is possible to force first-order theorem prover to only interpret symbols in accordance to a background theory $\mathcal{T}$, this proves to be impractical. On the other hand, SMT solvers provide specialized procedures that handle these symbols efficiently (e.g., by employing the simplex algorithm to solve linear real arithmetic).

Another critical advantage of State-of-the-art SMT-solvers is that they are built on top of efficient SAT solvers. They combine the SAT solver with specialized modules that treat the theory fragments. Modern SAT-solvers can handle an impressive amount of practical problems given the theoretical complexity of SAT, a notable NP-Complete problem (Cook 1971). These solvers are based on the *conflict driven clause learning* (CDCL) scheme (Moskewicz et al. 2001). This algorithm is a combination of a research process for a satisfying assignment with a deduction system (based on Boolean resolution) that fires only when the current assignment conflicts with the clauses to be satisfied, from which the notion of *conflict driven reasoning*. Since many SMT-solvers are built directly over an efficient SAT solver, they indirectly benefit from the efficiency offered by CDCL.

The most popular architecture used in state-of-the-art SMT solvers is known as DPLL($\mathcal{T}$) (Ganzinger et al. 2004). It is an extension of the *Davis Putnam Logemann Loveland (DPLL)* (Davis and Putnam 1960) procedure for solving SAT problems because it employs the SAT solver to enumerate assignments on the Boolean abstraction of first-order formulas. A theory solvers can then intervene when the current assignments either entail another assignment in the theory (e.g., by deduction that if the literal $x > y$ is true, then the literal $y > x$ must be false), or cause a conflict in the theory fragment (e.g., by detecting that $x + y > 1$ and $x + y < 0$ cannot be both true, a conflict that the SAT solver cannot identify). For example, to solve the formula

$$(p \vee (x + y > 2)) \wedge ((x > 2) \vee \neg(x < 0) \vee \neg q)$$

DPLL($\mathcal{T}$) builds a new literal for each first order term

$$(p \vee \underbrace{(x + y > 2)}_{l_1}) \wedge (\underbrace{(x > 2)}_{l_2} \vee \neg \underbrace{(x < 0)}_{l_3} \vee \neg q)$$

where $l_1 : x + y > 2, l_2 : x > 2$, and $l_3 : x < 0$. The SAT solver tries to assign a truth value to these literals without considering any first-order theory. The specialized modules are then used to check if this assignment is correct. Otherwise, they build a new propositional clause to "explain" to the SAT solver why this assignment is not possible. For example, if the SAT solver decide to assign both l_2 and l_3 to true (i.e., $x > 2$ and $x < 0$ must both be true) the arithemtic module should identify the conflict, and build the new clause $(\neg l_2 \vee \neg l_3)$, to force at least one of the two value to be false. The propositional component then restarts its search with the extra clause, and it could not generate the same error again.

If more than one theory is involved, it is required to use a framework to combine the theories, the most popular being the Nelson-Oppen combination scheme (Nelson and Oppen 1979). This scheme requires the theories to be disjoint and stably infinite. The theory solvers must be equipped with procedures that interact by exchanging only disjunctions of equalities between variables shared by the theories. The disjunctions are handled by case analysis in the SAT-solver.

In this thesis, we primarily use propositional logic and the theory of *linear real arithmetic* (noted as *LRA* in the solver) to encode the rules that describe the behaviour of policies. We use the Z3 solver (Moura and Bjørner 2008), and its extension for the MAX-SMT problem (Bjørner, Phan, and Fleckenstein 2015), to compute the optimal variable assignment of the rules.

3.2.2 Maximum satisfiability

In a classical SMT problem, the solver must prove that the problem is satisfiable by building a *model*, an assignment that satisfies the formula, or to prove that no model exists (i.e., that the formula is *unsatisfiable*). In general, a satisfiable formula can have many models, and in many instances, we are not interested in a generic solution, but

in a solution that optimizes certain *objective function* (e.g., by maximizing the value of an arithmetic equation). In this thesis, we encode the rule generation procedure as a *Maximum Satisfiability Modulo Theory (MAX-SMT)* problem. In MAX-SMT, there are two kinds of clauses, namely, *hard* clauses that must be satisfied, and *soft* clauses that can be satisfied. A model of the MAX-SMT problem hence satisfies all the hard clauses and as many soft clauses as possible, and it is unsatisfiable only if no assignment satisfies all the hard clauses. This is performed by adding a dummy literal l_d for each soft clause and building a pseudo-boolean objective function obj that sum all the dummy literals. A dummy literal can be used to satisfy a clause artificially, and to avoid using too many dummy literals, we minimize the objective function obj that counts them. We use the algorithms provided by Z3 (Bjørner, Phan, and Fleckenstein 2015) to perform this search efficiently. Since we use this formulation to describe as many of the decisions taken by POMCP as possible, MAX-SMT provides a perfect formalism to encode this procedure. Notice that the formulation of rule generation for POMCP as a MAX-SMT problem is one of the main contributions of this work.

Our algorithms also perform target functions optimization in the arithmetic module. Specifically, when we build a model that satisfies the maximum number of clauses, we then maximize an objective function that defines how many beliefs are described by our rule. It is important to note that this is not a MAX-SMT problem, but it is a maximization performed on the arithmetic module over a set of arithmetic variables.

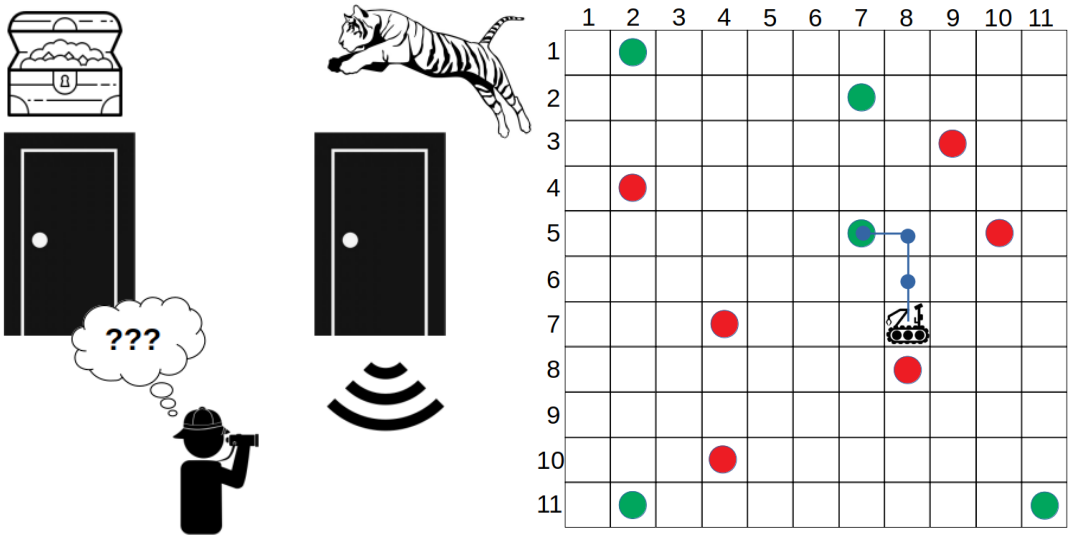
3.3 Benchmarks

This section presents the three banchemarks used during the experiments in Chapters 5, 6, and 7.

3.3.1 Tiger

Tiger is a well-known problem (Kaelbling, Littman, and Cassandra 1998) in which an agent has to choose which door to open among two doors, one hiding a treasure and the other hiding a tiger. Discovering the treasure yields a positive reward of $+10$ while finding the tiger yields a negative reward of -100. The agent can also listen (by paying a small penalty of -1) to gain new information. However, listening is inaccurate since there is a 0.15 probability of hearing a roar from the wrong door. A visual representation of the domain is presented in Figure 3.4a. In details, the POMDP model $(S, A, O, T, Z, R, \gamma)$ of this domain is:

- $S = \{Tiger_L, Tiger_R\}$, the states encode the position of the tiger.
- $A = \{Listen, Open_L, Open_R\}$. The three actions are listening or opening one of the two doors.



(A) Tiger.

(B) Rocksample With 11 rocks.

FIGURE 3.4: Visual representation of the tiger and the rocksample domains.

- $Z = \{None, Roar_L, Roar_R\}$. *None* is used only after opening a door, while the *Listen* action yields one of the other two.
- $T(s, Listen) \rightarrow \Pi(s : 1.0), \forall s,\ T(s, Open_L) \rightarrow \Pi(Tiger_L : 0.5, Tiger_R : 0.5)$, $T(s, Open_R) \rightarrow \Pi(Tiger_L : 0.5, Tiger_R : 0.5)$. Listening does not alter the current state an while the transition function shuffle the state after an opening.
- $O(Tiger_L, Listen) \rightarrow \Pi(Roar_L : 0.85, Roar_R : 0.15)$, $O(Tiger_R, Listen) \rightarrow \Pi(Roar_L : 0.15, Roar_R : 0.85)$. There is only an 85% chance of hearing the roar of the tiger coming from the correct door.
- $R(Tiger_L, Open_L), R(Tiger_L, Open_R) = +10, R(Tiger_R, Open_L) = +10$, $R(Tiger_R, Open_R) = -100$, $R(s, Listen) = -1, \forall s$. The reward funtion.
- We use a discounted factor $\gamma = 0.95$ for this domain.

Tiger is an interesting benchmark due to its tractability. The belief space is small, and it is possible to compute an exact solution using complete methods, like *incremental pruning* (Cassandra, Littman, and N. L. Zhang 1997). Therefore, we use tiger as a baseline to test the capabilities of XPOMCP in detecting anomalous behaviour and in generating successful shields.

3.3.2 Rocksample

In the well-known Rocksample domain Smith and Simmons 2004 a robot moves in a grid to collect valuable rocks. The rocks can be valuable or valueless (see green and red circles in the grid of Figure 3.4b). The true value of the rock is unknown, but the robot can measure it using a noisy sensor whose precision depends on the

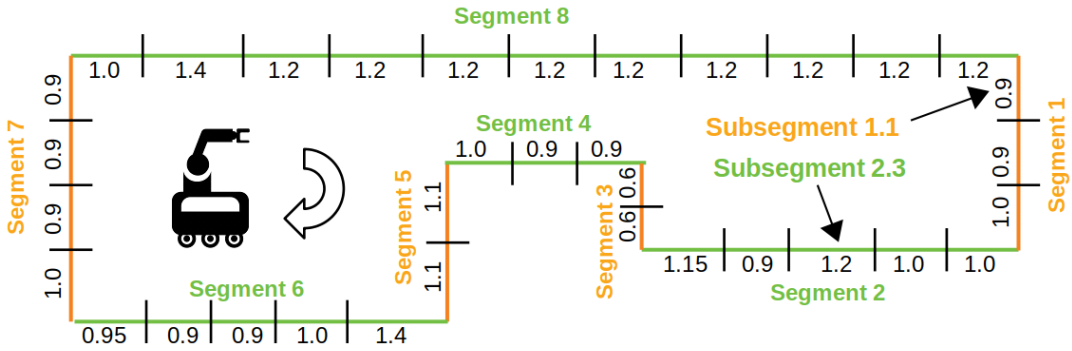
distance between the agent and the rock. The available actions are *i)* movements (North, South, East, West) of one cell, which have a negative reward of -1; *ii)* check a specific rock, which returns the true value of the rock with probability

$$obs = \frac{1}{2} \cdot (1 + 2^{-\frac{d}{\alpha}}) \tag{3.7}$$

where d is the distance between the robot and the rock and α is the efficiency of the sensor. The reward of this action is zero; *iii)* sample the rock in the same position of the robot, which has a reward 10 if the rock is valuable and -10 if it is valueless. The run ends when the robot exits from the east border of the board. This operation also yields a reward of 10. The transition model is deterministic. Namely, when the robot performs a movement, it moves according to the selected action. Our experiments consider 11 rocks in a 11×11 board setup. Specifically, the POMDP model is:

- S contains 2^{11} possible states that are a distribution over the possible truth values for the rocks, which can be valuable or valueless. It also contains an observable component, the robot's x, y position.
- $A = \{North, South, East, West, Sample, Check_{1,\dots,11}\}$. The formulation has four actions for the movement, one for sample, and eleven checks actions, one for each specific rock.
- $Z = \{None, Valuable, Valueless\}$. When the agent uses a $Check_i$ action, the system returns a noisy observation on the value of i.
- The transition function T is deterministic for the four movements and the sampling action. Checking does not change the state.
- The observation function O returns $Valuable$ of $Valueless$ after a $Check_i$ action, based on the value of rock i and the distance between i and the agent according to Equation 3.7. It returns $None$ for all the other actions.
- The reward function R returns zero for $Checks_i$, 10 for sampling a valuable rock, -100 for sampling a valueless rock, and -1 for the movements unless this leads the robot out of the map. In this case, R returns 10 when the agent exits from the East side, and -100 otherwise. In any case, this also terminates the run.
- We use $\gamma = 0.95$ for this domain.

Rocksample is a challenging problem for two main reasons. First, it has a large state space where online approaches are crucial in achieving acceptable performances. Second, POMCP performs well in the domain, given an adequate number of particles. Nonetheless, we observed that the agent sometimes samples a rock with low confidence in its true value using standard parameter setting. In our experiments, we characterize these cases and prevent them online by employing the shielding mechanism.



(a)

f	$p(o = Obstructed \mid f)$
0	0.44
1	0.79
2	0.86

(b)

f	a	$p(c = 1 \mid f, a)$
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	2	0.25

(c)

FIGURE 3.5: Main elements of the POMDP model for the velocity regulation problem. (a) Path map. The map presents the length (in meters) for each subsegment. (b) Occupancy model $p(o \mid f)$: probability of observing a subsegment occupancy given segment difficulty. (c) Collision model $p(c \mid f, a)$: collision probability given segment difficulty and action

3.3.3 Velocity regulation

In the *velocity regulation* domain, a robot must travel on a predefined path as fast as possible while avoiding collisions. The difficulty of each segment is unknown to the robot, but the agent receives (noisy) information on the real difficulty of the segment after each move. The path is divided into eight *segments* which are in turn divided into *subsegments* of different sizes, as shown in Figure 3.5.a. Each segment has a (hidden) difficulty value among *clear* ($f = 0$, where f is used to identify the difficulty), *lightly obstructed* ($f = 1$) or *heavily obstructed* ($f = 2$). All the subsegments in a segment share the same difficulty value. Hence, the hidden state-space has $3^8 = 6561$ states. The robot's goal is to travel on this path as fast as possible while avoiding collisions. In each subsegment, the robot must decide a *speed level* a (i.e., action). We consider three different speed levels, namely 0 (slow), 1 (medium speed), and 2 (fast). The reward received for traversing a subsegment is equal to the length of the subsegment multiplied by $1 + a$, where a is the agent's speed, namely the action that it selects. The higher the speed, the higher the reward, but a higher speed suffers a greater risk of collision (see the collision probability table $p(c = 1 \mid f, a)$ in Figure 3.5.c). The real difficulty of each segment is unknown to the robot (i.e., hidden part of the state), but in each subsegment, the robot receives an observation,

which is 0 (no obstacles) or 1 (obstacles) with a probability depending on segment difficulty (see Figure 3.5.b). The state of the problem contains a hidden variable (i.e., the difficulty of each segment) and three observable variables (current segment, subsegment, and time elapsed since the beginning). The POMDP formulation of the problem is:

- S contains 6561 possible states that specify which one of the three difficulties values is used in each of the eight segments. It also contains an observable component, the current segment *seg* and the current subsegment *subseg* of the agent.
- $A = \{Slow, Medium, Fast\}$. The three speed levels.
- $Z = \{Clear, Obstructed\}$. The two possible observations. It is important to notice that this does not distinguish between slightly obstructed and heavily obstructed segments.
- The transition function T is deterministic since the robot always moves to the next segment.
- The observation function O returns *Clear* of *Obstructed* based on the real difficulty of the segment, as described in Figure 3.5.b.
- The reward function R returns a positive value related to the speed level (i.e., 1 for *Slow*, 2 for *Medium*, 3 for *Fast*, multiplied by the length of the subsegment) and a possible negative value of -100 if the robot collides. Collisions happen with a probability that depends on speed and difficulty, as described in Figure 3.5.c.
- We use $\gamma = 0.95$ for this domain.

In our experiments, we build a rule describing when the robot travels at maximum speed (i.e., $a = Fast$). We expect the robot to move at that speed only if it is confident enough to be in an easy-to-navigate segment (i.e., in a segment with low difficulty). However, this level of confidence varies slightly from segment to segment (due to the length of the segments, the elapsed times, or the relative difficulty of the current segment compared to the others). To obtain a compact but informative rule, we want the rule to be a local approximation of the robot's behaviour. Thus we only focus on the current segment without considering the path as a whole when we write this rule.

Chapter 4

Synthesis of Logic-based Rules for Partially Observable Monte Carlo Planning

In this chapter, we describe the proposed *rule synthesis methodology*. To better explain the approach, we use a running example based on the velocity regulation domain (as presented in Section 3.3.3). We also introduce a formal language that can be used to express templates.

4.1 Overview

The methodology proposed in this thesis, called *XPOMCP*, is summarized in Figure 4.1. It leverages the expressiveness of logical formulas to represent specific properties of the investigated policy. As a first step, a logical formula with free variables is defined (see box 2 in Figure 4.1) to describe a property of interest of the policy under investigation. This formula, called *rule template*, defines a relationship between some properties of the belief (e.g., the probability of being in a specific state) and an action. Free variables in the formula allow the expert to avoid quantifying the limits of this relationship. These limits are then determined by analyzing a set of observed traces (see box 1 in Figure 4.1). For instance, a template saying "Do this when the probability of avoiding collisions is at least x", with x free variable, is transformed into "Do this when the probability of avoiding collisions is at least 0.85". In the rule template, the expert provides useful prior knowledge about the structure of the investigated property. The rule template defines the *question* asked by the expert. The *answer* to this question is provided by the SMT solver (see box 3 in Figure 4.1), which computes optimal values for the free variables to allow the formula to explain as many actions as possible in the observed traces.

The rule (see box 4) provides a human-readable local representation of the policy function that incorporates the prior knowledge specified by the expert. It allows splitting trace steps into two classes: those satisfying the rule and those not satisfying it. The approach, therefore, allows identifying unexpected decisions (see box 6) related to actions that violate the logical rule (i.e., that do not verify the expert's

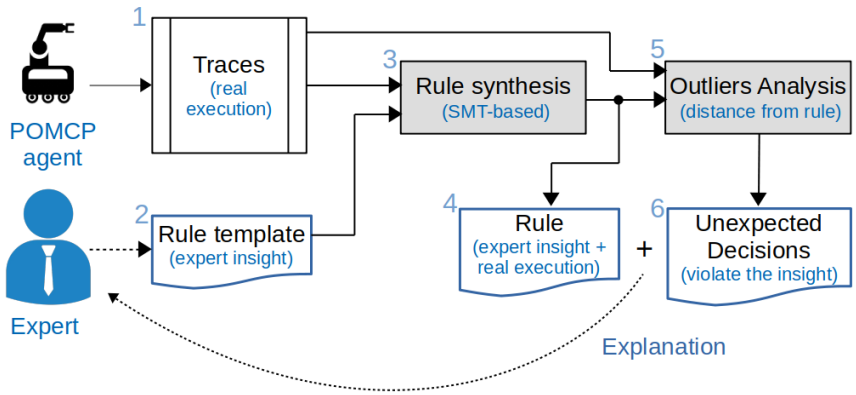


FIGURE 4.1: Overview of the XPOMCP approach

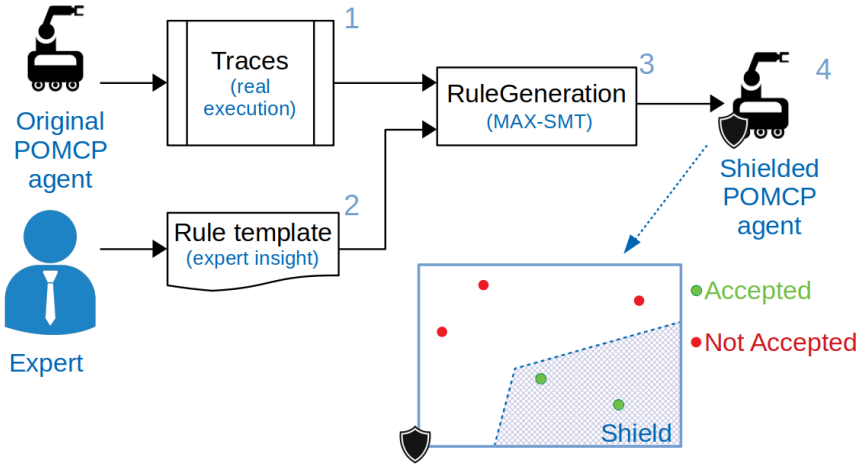


FIGURE 4.2: Shielding

assumption). The quantification of the severity of the violation (i.e., the distance between the rule boundary and the violation) also supports the analysis because it provides a clear way to explain the violations themselves, which could be completely unexpected due to expert imprecise knowledge or policy errors.

This analysis can be integrated into POMCP as a *shield* to block unexpected decisions proactively. Figure 4.2 shows an overview of the shielding mechanism. The XPOMCP methodology is used to build a rule from a rule template and a set of traces (boxes 1, 2, and 3 of Figure 4.2). The result is used as a base to build a shield. This shield is then integrated into POMCP (see box 4 in Figure 4.2) to filter in real-time the unexpected actions in the current belief. Chapter 6 presents the shielding mechanism in detail.

4.2 A language for expressing rule templates

To facilitate the usage of the methodology, we introduce a language to write *rule templates*, namely, logical formulas that express high-level expert insight. In particular, the language specifies the elements of a POMCP execution we want to use in

our rule templates (i.e., actions, beliefs, and problem-specific information) and how these elements should be combined. The syntax is based on SMT-LIB (C. Barrett, Fontaine, and Tinelli 2016), a standardized language that can be used to write SMT problems. Unlike SMT-LIB, we use a more readable infix notation for the operators. For example, we rewrite the SMT-LIB expression $(+\ a\ b)$ (easier to parse for computers) as $a + b$, which is more readable for humans. We present the constructs of the language alongside its application to the *velocity regulation* domain (Section 3.3.3). Chapters 5 and 6 present the usage of this language in other domains.

4.2.1 Header

To write a valid rule template, we need to specify which elements of the problem we want to describe. The elements of POMCP executions are stored in *traces*. A trace contains one (or more) execution of the POMCP algorithm, also called a *run*, and each run is a sequence of *steps*. A step corresponds to a belief-action pair where POMCP selects the action according to the belief, possibly containing other information (e.g., the position of a robot on a grid). Traces are stored using *Extensible Event Stream (XES)* (Acampora et al. 2017), a standard format used to collect execution traces of programs. This information is collected in the header of a rule template and are organized as follow:

$$\texttt{actions} = \{id, \ldots, id\}\ type;$$
$$\texttt{belief} = type;$$
$$\texttt{problemInfo} = \{id\ type, \ldots, id\ type\};$$
$$\texttt{runInfo} = \{id\ type, \ldots, id\ type\};$$
$$\texttt{stepInfo} = \{id\ type, \ldots, id\ type\};$$

The *actions* keyword is used to specify the list of actions available to the agent and thus the actions that can be used in the rule templates. All these actions are of the same type (e.g., string or integers). Note that the type is usually specified after the identifier of a variable, as in SMT-LIB. We support most of the type provided by XES. In particular, *bool*, *int*, *real*, and *string* as simple types, and *list* and *map* as aggregate types. Since it is very common to work with probability, we add the *prob* type, which is a real number in the interval $[0, 1]$. The keyword `belief` is used to specify the type used to store the states of the belief (usually encoded as simple integers, used as a unique identifier). The belief is a map that contains the number of particles assigned at each state in the current belief. Finally, *problemInfo*, *runInfo* and *stepInfo* are used to define extra information specific to the domain. With *problemInfo*, it is possible to specify generic information valid for all the runs, for example, the size of a grid or the length of a certain segment in a map. They are expressed as a list of identifiers (a uniquely defined string) with an associate type. Similarly, *runInfo* (*stepInfo*) is used to add information specific to a certain run (step). For example, for the velocity

regulation problem we specify the header:

$$\texttt{actions} = \{Speed_L, Speed_M, Speed_F\}\ int;$$
$$\texttt{belief} = int;$$
$$\texttt{problemInfo} = \{segments\ int, subsegments\ list[int]\};$$
$$\texttt{runInfo} = \{\};$$
$$\texttt{stepInfo} = \{seg\ int, subseg\ int\};$$

It contains three actions, namely, $Speed_L$, $Speed_M$ and $Speed_F$ (i.e., slow, medium, and fast speed). In *problemInfo*, the variable *segments* stores the total number of segments and the variable *subsegments* stores the number of subsegments in each segment. Each belief state is stored as an integer that encodes the expected distribution of difficulties among the segments. In particular, the number is computed by raising the expected difficulty of each segment (i.e., 0, 1, or 2) by the id of the segment (i.e., $0, \ldots, subsegments - 1$). We also store the current segment (i.e., *seg*) and subsegment (i.e., *subseg*), that is the visible part of the state, for each step using *stepInfo*. To simplify the usage of these information in our template, a *run* (*step*) variable is always defined, it is used to retrieve information encoded in *runInfo* (*stepInfo*). We also store the number of the *run* and *step* under investigation retrievable as *run.num* and *step.num*.

4.2.2 Variable and function declaration

Rule templates capture high-level knowledge of the POMCP-generated policy. To express them, we use the information stored in the trace alongside free variables that the methodology must instantiate. Free variable are declared using the syntax:

$$\texttt{declare-var}\ id, \ldots, id\ type;$$

It is also possible to declare simple functions to be used in rule templates:

`define-fun` *id (id type, . . . , id type) type* {formula};

the parentheses include the list of the parameters, while last *type* is the type of the function. The *formula* is an SMT expression based on SMT-LIB usage. When used in a rule template, these functions are instantiated using the proper values and variables at each step. *Functions* are useful to encode information compactly. In particular, we introduce a function p that takes a state and returns the probability in the belief for that state, useful for working with probabilities (i.e., a number in $[0.0, 1.0]$) instead of the number of particles (e.g., $p(segment_1 hard)$ return 0.1 if there are 100 out of the 1000 total particles in states that consider the first segment as difficult to navigate).

For example, to explain the behaviour of the agent in the velocity regulation problem we need to reason on the expected difficulty of the current segment. To express this concept in a compact way, we introduce the `diff` function which takes

a belief `b`, a segment `s`, and a difficulty value `d` as input and returns the probability that segment `s` has difficulty `d` in the belief `b`, which is the sum of probabilities of all states having difficulty d in segment s in the belief.

4.2.3 Rule templates

It is possible to specify one (or more) rule template as:

```
declare-rule
  action a_1 rel formula_1;
  ...
  action a_n rel formula_n;
[where requirements;]
```

for each *action* statement, we specify the action involved in the rule using a_i. It can be one of the actions specified in the *actions* declaration or a combination of multiple actions grouped using an $\vee$ operator (e.g., to express the idea that two actions have an equivalent effect under certain circumstances). The action and the formula are combined using $rel \in \{ \implies , \impliedby , \iff \}$. With the $\implies$ operator, we specify that an action is performed only when the formula is satisfied. With the $\impliedby$ operator, we specify that we must select that action when the formula is satisfied. Finally, using $\iff$, we specify that both requirements must be satisfied. A $formula_i$ is an SMT formula that combines free variables and information stored in the trace. Our language is based on SMT-LIB. Thus we support all the mathematical operators included in the language for writing these formulas. A copy of the formula is created for every single step. The methodology tries to satisfy as many of these formulas as possible (as described in Section 4.3). The (optional) `where` statement can be used to specify a set of hard requirements that the final rule must satisfy, such as the definition of a minimum value for a free variable (e.g., $x_0 \geq 0.9$). These are useful to define prior knowledge on the domain, which the rule synthesis algorithm uses to compute optimal parameter values (e.g., equality between two free variables belonging to different rules can be used to encode the idea that two rules are symmetrical).

For example, to express the idea that the robot in the velocity regulation problem should move at high speed only if it is confident that the current segment is clear we can write the rule template:

```
declare-var x_1, x_2 prob;
declare-rule
  action Speed_F ⟺ diff(belief, seg, 0) ≥ x_1 ∨ diff(belief, seg, 2) ≤ x_2
where x_1 ≥ 0.9
```

Algorithm 4: RuleSynthesis

Input: Trace ex, Rule template r
Output: instantiation of r

```
1  solver ← probability axioms;
2  foreach action rule r_a with a ∈ A do
3      foreach step t in ex do
4          build new dummy literal l_{a,t};
5          cost ← cost ∪ l_{a,t};
6          compute p_0^t, ..., p_n^t from t.particles;
7          r_{a,t} ← instantiate rule r_a using p_0^t, ..., p_n^t;
8          if t.action ≠ a then
9              r_{a,t} ← ¬(r_{a,t});
10         solver.add(l_{a,t} ∨ r_{a,t});
11 solver.minimize(cost);
12 fitness ← 1 − distance_to_observed_boundary;
13 model ← solver.maximize(fitness);
14 return model
```

The first part of the `action` statement specifies that we select action $Speed_F$ (i.e., high speed) if the probability to be in a segment seg with low difficulty (i.e., $\texttt{diff}(belief, seg, 0)$) is greater than a certain threshold x_1. The second literal specifies that we select action $Speed_F$ if the probability of being in a segment with high difficulty (i.e., $\texttt{diff}(belief, seg, 2)$) is less than a certain threshold x_2. Since the two literals are combined with an $\vee$, we select $Speed_F$ if and only if at least one of them is true. The *where* clause contains an information that we expect to be true, the expression $x_1 \geq 0.9$. Namely, we force the probability of the current segment to be low difficulty to be greater than 0.9 in the final rule.

4.2.4 Traces

A trace is a sequence of $(belief, action)$ pairs generated by POMCP. A trace contains one or more than one run, which are complete executions of the POMCP algorithm. We store these traces using *Extensible Event Stream (XES)* (Acampora et al. 2017), a standard format used to collect execution traces of programs. This format uses an XML scheme. Thus it is easy to use in other tools and programs.

4.3 Rule generation algorithm

Rule synthesis is the core of the methodology presented in this thesis. It works as a basis for the mechanisms presented in Chapters 5 and 6. In Chapter 7 we present an extension of this approach that generates the data actively instead of relying on a predefined trace.

This methodology generates *rules* from *rule templates* by instantiating free variables to explain as many as possible of the decisions taken by a POMCP policy in

the set of observed traces. The implementation, presented in Algorithm 4, takes as input a trace *ex*, generated by POMCP and stored in XES format and a rule template *r*, as explained in Section 4.2. The output is the rule *r* with all free variables instantiated. The *solver* (we use Z3 (Moura and Bjørner 2008)) is first initialized and hard constraints are added in line 1 of Algorithm 4 to force all variable of type *prop* in the template to satisfy the probability axioms (i.e., to have value in range $[0, 1]$). Then in the *foreach* loop in lines 2–10 the algorithm maximizes the number of steps that satisfy the rule template *r*. In particular, for each action rule r_a, where *a* is an action, and for each step *t* in the trace *ex* the algorithm first generates a literal $l_{a,t}$ (line 4) which is a dummy variable used by MAX-SMT to satisfy clauses that are not satisfiable by a free variable assignment. This literal is then added to the *cost* objective function (line 5) which is a pseudo-boolean function collecting all literals. This function counts the number of fake assignments that correspond to unsatisfied clauses. The belief state probabilities are collected from the particle filter and used to instantiate the action rule template r_a (line 7) by substituting their probability variables p_i with observed belief probabilities. This generates a new clause $r_{a,t}$ which represents the constraint for step t. This constraint is considered in its negated form $\neg(r_{a,t})$ if the step action $t.action$ is different from *a* (line 9) because the clause $r_{a,t}$ should not be true.

The set of logical formulas of the solver is then updated by adding the clause $l_{a,t} \vee r_{a,t}$. In this formulation, the added clause can be satisfied in two ways, namely, by finding an assignment of the free variables that makes the clause $r_{a,t}$ true (the expected behaviour) or by assigning a true value to the literal $l_{a,t}$ (unexpected behaviour). However, the second kind of assignment has a cost since the dummy variables have been introduced only to allow partial satisfiability of the rules. In line 11, the solver is asked to find an assignment of free variable, which minimizes the cost function, considering the number of dummy variables assigned to true. This minimization is a typical MAX-SMT problem in which an assignment maximizing the number of satisfied clauses is found. Since there can be more than a single assignment of free variables that achieves the MAX-SMT goal, the last step of the synthesis algorithm concerns the identification of the assignment, which is closer to the behaviour observed in the trace. This problem is solved by maximizing a fitness function that moves the free variables assignment as close as possible to the numbers observed in the trace without altering the truth assignment of the dummy literals. This problem concerns the optimization of real variables, and the linear arithmetic module solves it.

4.3.1 Complexity

Notice that, even if MAX-SMT is an NP-hard problem, in practice, Z3 can solve our instances in a reasonable time (as shown in Section 5.3) due to the limited number of variables involved and the structure of our instances. Specifically, the variables used in the SMT problem are the free variables specified in the template (a constant number, usually small) and the dummy literals that are linear on the size of the trace

because the algorithm builds a clause for each step, and each clause introduces a new dummy literal. Z3 employs numerous heuristics to achieve high performance. In particular, by employing subsumption heuristics, it is possible to remove many redundant steps (i.e., steps generated from similar beliefs) from the problem and thus reduce significantly the number of dummy literals involved.

Chapter 5

Rule-based Anomaly Detection

In this chapter, we present a methodology that uses the rules generated by XPOMCP, as described in Chapter 4, to detect decisions that violate the behaviour described by the expert with the rule templates. These decisions, called *unexpected decisions* in the following, are notable steps in the trace and must be identified to provide an accurate description of the policy. This chapter also presents an in-deep case study of XPOMCP, alongside an extensive experimental evaluation of the methodology in different domains.

5.1 Identification of unexpected decisions

A key element of XPOMCP concerns the characterization of steps that fail to satisfy the rule. They can provide useful information for policy interpretation. We define two important classes of exceptions: those related to the approximation made by the logical formula and those actually due to unexpected policy behaviour (e.g., an error in the POMCP algorithm or a decision that cannot be described with only local information). Exceptions in the first class fall quite close to the rule boundary, while exceptions in the second class are usually more distant from the boundary. In the following, we call the second kind of exceptions *unexpected decisions* since their behaviour is unexpected compared to the expert knowledge on the policy. Unexpected decisions are particularly important when the rule template and the policy behaviour are very different. This could happen when the expert provides an erroneous rule template (e.g., because she/he does not properly understand the system) or when there is a bug in the POMCP-generated policy (e.g., because one of its parameters is wrongly set). These cases present, in general, significant and numerous unexpected decisions that highlight the conflict between templates. For example, consider a template that specifies that a robot should move fast only when its confidence of being in a *cluttered* segment is above a threshold (instead of below). XPOMCP builds a rule that tries to satisfy as many steps as possible but will report many unexpected decisions. In this case, an unexpected decision would be: "in step 10 the robot decides to move fast even if its confidence of being in a cluttered segment is 0%", a statement that highlights what is wrong in our template, i.e., it points out that a situation that should be acceptable is anomalous regarding our template.

Algorithm 5: UnexpectedDecisionIdentification

Input: rule r, the set of steps *steps* that violate the rule
Output: a set $unexp$ of unexpected decision steps

```
1  Init: n = 0, points = ∅, out = ∅;
2  while n < N_p do
3      x ← random point in the belief space;
4      if x ∈ r then
5          n ← n + 1;
6          add x to points;
7  foreach Step s in steps do
8      lower ← ∞;
9      foreach Step p in points do
10         distance ← H²(p, s);
11         if distance < lower then
12             lower ← distance;
13     if lower ≥ threshold then
14         out ← out ⋃ s;
15 return out
```

However, if we use a correct template (i.e., "I expect the robot to move fast when the segment is uncluttered") but the POMCP has a wrongly set parameter, sometimes it takes risky decisions. For instance, one unexpected decision can be: "In step 6, the robot decides to move fast when its confidence in being in a cluttered segment is 85%", which highlights a genuine error in the POMCP agent that requires further investigation from the designer.

We provide a procedure to identify unexpected decisions in Algorithm 5. Its input is composed of a learned rule r, a set of steps (called *steps*) that violate the rule, and a threshold $\tau \in [0, 1]$. The algorithm's output is a set of steps related to unexpected decisions. The procedure first randomly generates N_p beliefs $\bar{b}_j, j = 1, \ldots, N_p$, that satisfy the rule (see lines 2–6 in Algorithm 5). Specifically, we use $N_p = 1000$ in our experiments. Then, for each belief b_i in *steps* a distance measure is computed between b_i and all $\bar{b}_j, j = 1, \ldots, N_p$ (see Algorithm 5 lines 7–12). The minimum distance h_i is finally computed for each b_i and compared to a threshold τ. If $h_i \geq \tau$ (Algorithm 5 line 13) then b_i is considered an outlier because its distance from the rule boundary is high.

5.1.1 Distance between beliefs

Since beliefs are discrete probability distributions, we use a specific measure dealing with these kinds of elements called the *discrete Hellinger distance* (Hellinger 1909).

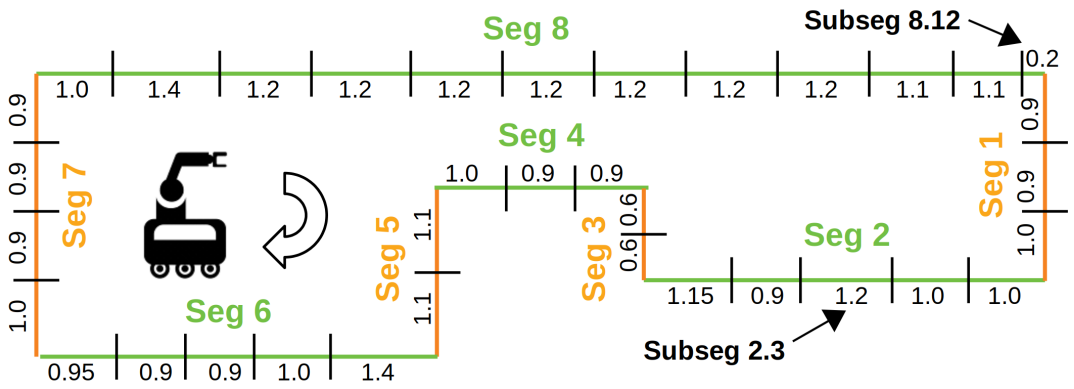


FIGURE 5.1: Modified version of the velocity regulation map, in which one of the segment is shorter than the others

This distance is defined as follows:

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

where P, Q are probability distributions and k is the discrete number of states in P and Q. An interesting property of H^2 is that it is bounded between 0 and 1, which simplify the task of identifying a meaningul threshold for the procedure. In Section 5.3 we discuss how we set this threshold in practical cases. However, a good rule of thumb is to use a threshold $\tau = 0.1$.

5.2 Iterative rule synthesis example

This section presents an in-deep case study of the rule synthesis process. We define a rule template by analyzing previous iterations of the rules and the anomalous actions detected with them. To obtain rules that are compact and informative, we want them to be a local approximation of the behaviour of the robot in the *velocity regulation* domain (as described in Section 3.3.3. We consider a slightly different version of the standard map, presented in Figure 5.1, in which subsegment 8.12 is significantly shorter than the others. This peculiar subsegment highlights an interesting error case that is not easy to generate in the original version of the map.

Iteration 1

We start with a rule describing when the robot travels at maximum speed (i.e., $a = 2$). We expect the robot to move at that speed only if it is confident enough to be in

an easy-to-navigate segment. We express this with the template:

$$\begin{aligned}&\texttt{declare-var } x_1, x_2 \; prob;\\ &\texttt{declare-rule}\\ &\quad \texttt{action } Speed_F \iff p_0 \geq x_1 \vee p_2 \leq x_2\\ &\texttt{where } x_1 \geq 0.8\end{aligned}$$

This template can be satisfied if the probability of being in a clear segment p_0 is above a certain threshold x_1 or the probability of being in a heavily obstructed segment p_2 is below another threshold x_2. We expect x_1 to be above 0.8 in this instance, thus we add this information in the `where` statement. Our methodology provides the rule:

$$\texttt{action } Speed_F \iff p_0 \geq 0.858 \vee p_2 \leq 0.004$$

that fails to satisify 6 out of 370 steps.

Iteration 2

By analyzing the unsatisfiable steps, we notice that three of them are in subsegment 8.12 (the robot moves at low speed with beliefs, respectively, $[p_0 = 0.895, p_1 = 0.102, p_2 = 0.003]$, $[p_0 = 0.955, p_1 = 0.045, p_2 = 0.0]$, $[p_0 = 0.879$, and $p_1 = 0.120, p_2 = 0.002]$). Figure 5.1 shows that this subsegment is the shortest on the map. Our template is approximate and does not consider the length of the subsegment. This local rule cannot describe the behaviour of the policy in segment 8.12, it is too short, and POMCP decides that it is better to move slowly even if it is nearly certain that the subsegment is safe. This is because the reward depends on the length of the subsegment. Hence, we want to exclude the subsegment from the rule. Finally, by analyzing the other three of the six unsatisfiable steps, we notice that they are close to the rule, but cannot be described with this simple template Specifically, in these steps, the robot move a speed 2 with belief $[p_0 = 0.789, p_1 = 0.181, p_2 = 0.031]$, $[p_0 = 0.819, p_1 = 0.164, p_2 = 0.017]$, and $[p_0 = 0.828, p_1 = 0.162, p_2 = 0.010]$. To improve the template, we add a more complex literal ($p_0 \geq x_3 \wedge p_1 \geq x_4$), that use both difficulty 0 (clear) and 1 (lightly obstructed) to describe the behaviour of the policy. We obtain the template:

$$\begin{aligned}&\texttt{declare-var } x_1, x_2, x_3, x_4 \; prob;\\ &\texttt{declare-rule}\\ &\quad \texttt{action } Speed_F \iff p_0 \geq x_1 \vee p_2 \leq x_2 \vee (p_0 \geq x_3 \wedge p_1 \geq x_4))\\ &\texttt{where } x_1 \geq 0.8 \wedge (seg \neq 8 \vee subseg \neq 12)\end{aligned}$$

and the rule:

$$\texttt{action } Speed_F \iff p_0 \geq 0.841 \vee p_2 \leq 0.004 \vee (p_0 \geq 0.789 \wedge p_1 \geq 0.156)$$

that only fails to satisfy 2 steps (speed 1 with belief $[p_0 = 0.801, p_1 = 0.190, p_2 = 0.009]$, and speed 0 with belief $[p_0 = 0.826, p_1 = 0.162, p_2 = 0.013]$). These steps were satisfied by the first iteration of the template, but now we have a stricter rule that describes more steps. We further refine the template, but this result is a good compromise between simplicity and correctness.

Iteration 3

We write a template to describe when the robot moves slowly. We identify three critical situations that can lead the robot to move at a slow speed. Namely, *i)* the robot is uncertain about the current difficulty (the belief is close to a uniform distribution); *ii)* the robot knows that the current segment is hard; *iii)* the robot is in the short subsegment 8.12. We try to use $p_1 \geq y_1$ and $p_2 \geq y_2$ to describe the first two situations. The template is the following:

$$
\begin{aligned}
&\texttt{declare-var } x_1, x_2, x_3, x_4 \; prob;\\
&\texttt{declare-var } y_1, y_2 \; prob;\\
&\texttt{declare-rule}\\
&\quad \texttt{action } Speed_F \iff p_0 \geq x_1 \vee p_2 \leq x_2 \vee (p_0 \geq x_3 \wedge p_1 \geq x_4))\\
&\quad \texttt{action } Speed_S \iff p_1 \geq y_1 \vee p_2 \geq y_2\\
&\texttt{where } x_1 \geq 0.8 \wedge (seg \neq 8 \vee subseg \neq 12)
\end{aligned}
$$

that yields the rule:

$$
\begin{aligned}
&\texttt{action } Speed_F \iff p_0 \geq 0.841 \vee p_2 \leq 0.004 \vee (p_0 \geq 0.789 \wedge p_1 \geq 0.156)\\
&\texttt{action } Speed_S \iff p_1 \geq 0.244 \vee p_2 \geq 0.024
\end{aligned}
$$

which fail to satisfy 38 out of 370 steps. Notice that the low value for y_1, y_2 (i.e., $0.244, 0.024$) describes all the belief close to the uniform distribution. By analyzing the 38 unsatisfiable steps, we notice that 35 of them are situations in which the robot decides to move at speed $Speed_M$ even if the condition for moving at $Speed_S$ are satisfied. In particular, three of these steps have belief $[p_0 = 0.319, p_1 = 0.342, p_2 = 0.338]$, $[p_0 = 0.345, p_1 = 0.337, p_2 = 0.318]$, and $[p_0 = 0.335, p_1 = 0.333, p_2 = 0.332]$ respectively. This analysis tells us that POMCP considers a worthy risk to move at medium speed even if it does not understand the current difficulty strongly. If we consider this a non-acceptable risk, we should modify the design of POMCP, e.g., by increasing the number of particles used in the simulation. This is a case in which the explainability capabilities of XPOMCP are used to understand a POMCP policy that would be otherwise difficult to understand. The understanding can be then used to improve the policy.

5.3 Results on rule-based anomaly detection

We test our methodology in the three domains described in Section 3.3. In Section 5.3.5, we use XPOMCP to detect unexpected decisions in the same three domains. For *tiger*, we present a comparison between our methodology and a state-of-the-art anomaly detection algorithm called *Isolation Forest* (Liu, Ting, and Zhou 2008) which uses an exact policy as the ground truth. For *velocity regulation* and *rocksample* it is not possible to compute an exact policy. Thus we only provide an analysis of the results. In Section 6.2, we present results for the shielding mechanism in the three domains. We evaluate this approach's effectiveness by comparing the original (unshielded) implementation of POMCP with that of the shielded POMCP.

5.3.1 Experimental setting

We implemented two of the domains, namely *Tiger* and *velocity regulation*, as black-box simulators in the original C++ version of POMCP (Silver and Veness 2010). For *rocksample*, we used the implementation provided in (Silver and Veness 2010), and we extended it to allow the collection of traces in the XES format. We extend the implementation to collect *traces* in XES format. The *RuleSynthesis* algorithm (i.e., Algorithm 8) and the procedure for identifying *unexpected decisions* (see Section 5.1) have been developed in Python. The Python binding of Z3 (Moura and Bjørner 2008) has been used to solve the SMT formulas. The shielding mechanism is implemented in C++ as an extension of the original POMCP implementation. Experiments have been performed on a notebook with Intel Core i7-6700HQ and 16GB RAM. An implementation of the XPOMCP methodology and the shielding mechanism is available at https://github.com/GiuMaz/XPOMCP.

5.3.2 Error injection

To quantify the capability of the proposed method to identify policy errors and perform shielding, we introduce errors in the parameters tuning of POMCP. Specifically, we consider the *reward range* (called c in the following, using the notation of (Silver and Veness 2010)) and the number of particles n_p, because these parameters must be hand-tuned, and setting them to a wrong value is a frequent mistake that can happen in practice.

The parameter c defines the maximum difference between the lowest and the highest possible reward, and UCT uses it to balance exploration and exploitation. If this value is lower than the correct one, the algorithm could find a reward that exceeds the maximum expected value leading to a wrong state. Namely, the agent believes to have identified the best possible action, and it stops exploring new actions, even though the selected action is not the best one. This creeping error randomly

affects the exploration-exploitation trade-off, making POMCP incorrect in some situations. We use this kind of error since parameter c must be set by hand in POMCP, and it requires specific values that are not always easy to devise.

The parameter n_p specifies how many particles to use during the simulation. The number of particles also corresponds to the number of simulations performed to generate the lookahead tree in POMCP. More particles lead to better performance, but more computing power is required to handle them. Thus it is important to find the best trade-off between computational time and achieved performance.

5.3.3 Exact solution

We use the *incremental pruning algorithm* (Cassandra, Littman, and N. L. Zhang 1997) implemented in (Bargiacchi, Roijers, and Nowé 2020) to compute an exact policy for *tiger*. This is used as ground truth for evaluating the performance of our method in detecting wrong actions. Unfortunately, we cannot compute the exact policy for *velocity regulation* and *rocksample*, since their size makes the computation intractable. However, we use these domains to evaluate the applicability of our method to larger problems.

5.3.4 Baseline method for detecting unexpected decisions

Isolation forest (IF) (Liu, Ting, and Zhou 2008) is an anomaly detection algorithm that we use as a benchmark for evaluating the performance of our procedure in identifying unexpected decisions. It assumes anomalies as rare events and can be applied to a training set containing both nominal and anomalous samples. Hence it is a good candidate for comparison with XPOMCP. We use the Python implementation of IF provided in *scikit-learn* (Pedregosa et al. 2011) and consider each step of a trace (i.e., a pair *belief, action*) as a sample (notice that the action is not used as a label). The algorithm uses the *contamination* parameter (i.e., the expected percentage of anomalies in the dataset) to set the threshold used to identify which points are anomalies.

5.3.5 Detection of unexpected decisions

We focus on the *tiger* problem to test the capabilities of our methodology in detecting unexpected decisions. It is possible to compute a complete policy that works as a baseline for evaluating performance in this domain. We also provide an analysis of the performance of unexpected decision detection for *velocity regulation* and *rocksample*. These use cases show the capability of XPOMCP to scale to larger instances. However, we cannot provide the accuracy in these domains because it is not feasible to compute an exact policy (to be used as ground truth) for them.

5.4 Comparison with unsupervised anomaly detection algorithms

In this section, we use XPOMCP and IF to detect anomalous decisions injected into POMCP traces.

5.4.1 Unexpected decisions in tiger

A successful policy for *tiger* should listen until enough information is collected about the position of the tiger and then open a door when the agent is reasonably certain to find the treasure behind it. However, from the analysis of the observation model and reward function, it is not immediate to define what "reasonably certain" means. To investigate it, we create a rule template specifying a relationship between the confidence (in the belief) over the treasure position and the related opening action. Then we learn the rule parameters from a set of traces performed using POMCP. Finally, by analyzing the trained rule, we understand the minimum confidence required by the policy to open a door. The correct value of c is 110 (the reward interval is $[-100, 10]$). For each value of c in $\{110, 85, 65, 40\}$, we generate 50 traces with 1000 runs each, using different seeds for the pseudo-random algorithm in every trace. For each run, we use 2^{15} particles and a maximum of 10 steps per episode. Lower values of c produce a higher number of errors, as show in Table 5.1 (see column % *errors*).

Rule synthesis

To formalize the *tiger* problem within our language, we use the header:

$$\texttt{actions} = \{Listen, Open_R, Open_L\}\ int;$$
$$\texttt{belief} = Bool;$$
$$\texttt{problemInfo} = \{\};$$
$$\texttt{runInfo} = \{\};$$
$$\texttt{stepInfo} = \{\};$$

The header presents three actions (i.e., $Listen, Open_R, Open_L$) and a Boolean belief (i.e., true when we believe the tiger is behind the left door, false otherwise). No extra information is required in this domain. To formalize the property that the agent has to gather enough confidence on the tiger position before opening a door we use the

following rule template:

$$
\begin{aligned}
&\texttt{declare-var } x_1, x_2, x_3, x_4 \; prob;\\
&\texttt{declare-rule}\\
&\quad \texttt{action } Listen \iff (p(right) \leq x_1 \wedge p(left) \leq x_2);\\
&\quad \texttt{action } Open_R \iff p(right) \geq x_3;\\
&\quad \texttt{action } Open_L \iff p(left) \geq x_4;\\
&\texttt{where } (x_1 = x_2) \wedge (x_3 = x_4) \wedge (x_3 > 0.9);
\end{aligned}
$$

The action rule template for action *Listen* describes when the agent should listen. Similarly, template action rules for $Open_R$ and $Open_L$ describe when the agent should open the right and left door, respectively. To make the formula more readable, we use $p(left)$ instead of $p(true)$ to refer to the true part of the Boolean belief and, similarly, $p(right)$ instead of $p(false)$. Some hard clauses are also added (in the bottom) to state that *i)* the problem is expected to be symmetric (i.e. the thresholds used to decide when to listen and when to open are the same for both doors, namely $x_1 = x_2$ and $x_3 = x_4$), *ii)* minimum confidence is required to open the door (namely, $x_3 > 0.9$, hence the door should be opened only if the agent is at least 90% sure to find the tiger behind it).

Performance evaluation across different thresholds

Both XPOMCP and IF use a threshold to identify anomalous points. We use the *Receiver Operating Characteristic (ROC) curve* and the *precision/recall curve* of the two methods to compare the performance across thresholds. The ROC curve considers the relationship between the true positive rate (tpr) and the false positive rate (fpr) at different thresholds. As a performance measure, we use the *Area Under Curve* (AUC). Similarly, the precision/recall curve considers the relationship between precision and recall at different thresholds, and we use the *Average Precision* (AP) as a performance measure. Performance are compared on traces generated using $c \in \{85, 65, 40\}$. We do not evaluate the methods in the case with $c = 110$; it is error-free; thus, it is impossible to have any true positive (AUC and AP are 0).

In Table 5.1 we compare the average performance of the two methods. We test XPOMCP with a uniform sampling of 100 thresholds in the interval $[0, 0.5]$. Similarly, we use IF with 100 different values for the contamination parameter uniformly distributed in the interval $[0, 0.5]$. XPOMCP outperforms IF in nearly every instance in both AUC and AP. For both AUC and AP, the difference is high with *c=40*. This is because XPOMCP effectively exploits the information in the template to avoid being influenced by the number of errors. IF performs poorly also with *c=85* because it exhibits a large number of false-positive that leads to very low precision and value of AP. Finally, with *c=65* the difference between the two methods is smaller. In this dataset, both methods achieve their best performance. IF is more effective

- *true positive (tp)*: a result that correctly indicates the presence of a condition.
- *false positive (fp)*: a result that wrongly indicates the presence of a condition.
- *true negative (tn)*: a result that correctly indicates the absence of a condition.
- *false negative (fn)*: a result that wrongly indicates the absence of a condition.
- *true positive rate (fpr)* (also known as sensitivity or recall): the fraction of positive instances that was correctly identified. $tpr = \frac{tp}{tp+fn}$
- *false positive rate (tpr)*: the fraction of negative instances that was wrongly identified. $fpr = \frac{fp}{fp+tn}$
- *Precision (ppv)*: the fraction of positive instances correctly identified among all the positive tests. $ppv = \frac{tp}{tp+fp}$
- *Receiver operating characteristic (ROC)*: a graphical plot that shows how *fpr* and *tpr* varies when the threshold of a binary classifier changes. It is used to show how dependent is the methodology from its parameters.
- *precision/recall curve*: a graphical plot similar to ROC that use *precision* and *recall* instead of *fpr* and *tpr*.
- *Area under the curve (AUC)*: is the area covered by a ROC curve. An higher value is better.
- *Average precision (AP)*: is the area covered by a precision/recal curve. An higher value is better.
- *F1 score*: measure of the accuracy of a binary classifier that combines precision and recal. Higher is better. $F_1 = 2 \cdot \frac{ppv \cdot tpr}{ppv+tpr}$
- *Accuracy (acc)*: measure of the accuracy of a binary classifier that present the fraction of correct tests over the total number of tests. $acc = \frac{tp+tn}{tp+tn+fp+fn}$

FIGURE 5.2: Summary of the used metrics.

c	% errors	$\mathbf{AUC_{XPOMCP}}$	$\mathbf{AUC_{IF}}$	$\mathbf{AP_{XPOMCP}}$	$\mathbf{AP_{IF}}$
110	0.0($\pm$0.0)	–	–	–	–
85	0.0004($\pm$0.0003)	**0.993**($\pm$0.041)	0.964($\pm$0.024)	**0.986**($\pm$0.082)	0.057($\pm$0.1076)
65	0.0203($\pm$0.0021)	**0.999**($\pm$0.001)	0.992($\pm$0.001)	**0.999**($\pm$0.002)	0.539($\pm$0.0520)
40	0.2374($\pm$0.0072)	**0.995**($\pm$0.034)	0.675($\pm$0.020)	**0.987**($\pm$0.084)	0.333($\pm$0.0153)

TABLE 5.1: Comparison between the performance of XPOMCP and that of Isolation Forest in terms of Area Under Curve (AUC) and Average Precision (AP) in tests performed with different reward range (c) and related error rate (*%errors*). Standard deviations are in parenthesis and the best results are highlighted in bold

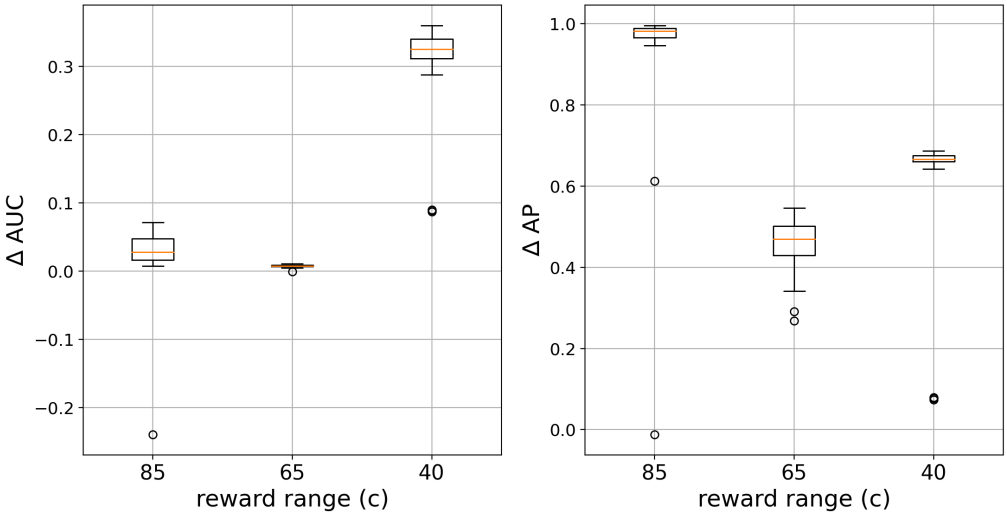


FIGURE 5.3: Box-plots of AUC and AP (considering 100 different parameters) with different values of c

in identifying true error compared to the case of $c = 85$, but it still generates more false-positive than XPOMCP. Figure 5.3 displays two boxplots that show how the AUC and the AP varies for each execution trace. We compute these values as:

$$\Delta AUC = AUC_{XPOMCP} - AUC_{IF}$$
$$\Delta AP = AP_{XPOMCP} - AP_{IF}$$

Since a positive value in the box-plot means that XPOMCP outperforms IF the plot shows that our algorithm is consistently better than IF except for an outlier whit $c = 85$.

Performance evaluation with optimal thresholds

To provide further details on the performance of XPOMCP, here we show its performance with optimal threshold τ (see Section 5.1). To compute the value of τ, we performed cross-validation by training XPOMCP on five traces and testing it on 45 traces. The F1 score for identifying unexpected decisions was computed on the test set using 100 threshold values uniformly distributed in $[0, 0.5]$, and the test with

(A) XPOMCP

c	% errors	τ	F1	Accuracy	time (s)
85	0.0004($\pm$0.0003)	0.061	**0.979**($\pm$0.081)	**0.999**($\pm$0.0001)	14.30($\pm$0.50)
65	0.0203($\pm$0.0021)	0.064	**0.999**($\pm$0.002)	**0.999**($\pm$0.0001)	14.75($\pm$0.80)
40	0.2374($\pm$0.0072)	0.045	**0.980**($\pm$0.072)	**0.987**($\pm$0.049)	12.78($\pm$0.83)

(B) Isolation Forest

c	% errors	Cont.	F1	Accuracy	time (s)
85	0.0004($\pm$0.0003)	0.01	0.020($\pm$0.033)	0.990($\pm$0.001)	**0.72**($\pm$0.013)
65	0.0203($\pm$0.0021)	0.03	0.771($\pm$0.044)	0.988($\pm$0.001)	**0.71**($\pm$0.010)
40	0.2374($\pm$0.0072)	0.5	0.437($\pm$0.035)	0.585($\pm$0.026)	**0.64**($\pm$0.037)

TABLE 5.2: Comparison between the performance of XPOMCP and that of Isolation Forest in terms of F1-score, accuracy and time in tests performed using the best threshold τ across different reward ranges (c) and related error rate(%*errors*). Standard deviations are in parenthesis, and the best results are highlighted in bold

the best F1 score was selected. The results of this test are presented in Table 5.2.a. Column τ contains values of threshold used and columns *accuracy* and *F1* show the related performance values on the test set, and column *time* shows the average elapsed time (in second). We used the same procedure to tune the *contamination* parameter of IF (*Cont*. in Table 5.2.b). Figure 5.4 compares the average F1-score and accuracy achieved by the two approaches in each test (the value in parenthesis presents the standard deviation). This comparison shows that with optimal parameters, XPOMCP always outperforms IF. Both methods achieve high accuracy due to the high number of non-anomalous samples in the dataset (anomaly and non-anomaly classes are unbalanced), and both methods compute several true negatives. However, the F1 score is very different. IF achieves a low score in this metric because it cannot identify some true positives and generates many more false positives than XPOMCP. In general, IF is faster than XPOMCP by order of magnitude, but the performance of our methodology is acceptable since it takes POMCP an average of 158 seconds to generate a *tiger* trace with 1000 runs, and XPOMCP analyze it in less than 15 seconds.

Analysis of a specific trace

To complete our analysis on *tiger*, we show the rule generated by XPOMCP on the analysis of a specific trace generated by POMCP using a wrong value of c, namely, $c = 40$. The rule generated by the MAX-SMT solver from this trace is:

$$\begin{aligned} &\texttt{action}\ Listen \iff (p(right) \leq 0.847 \wedge p(left) \leq 0.847); \\ &\texttt{action}\ Open_R \iff p(right) \geq 0.966; \\ &\texttt{action}\ Open_L \iff p(left) \geq 0.966; \end{aligned} \tag{5.1}$$

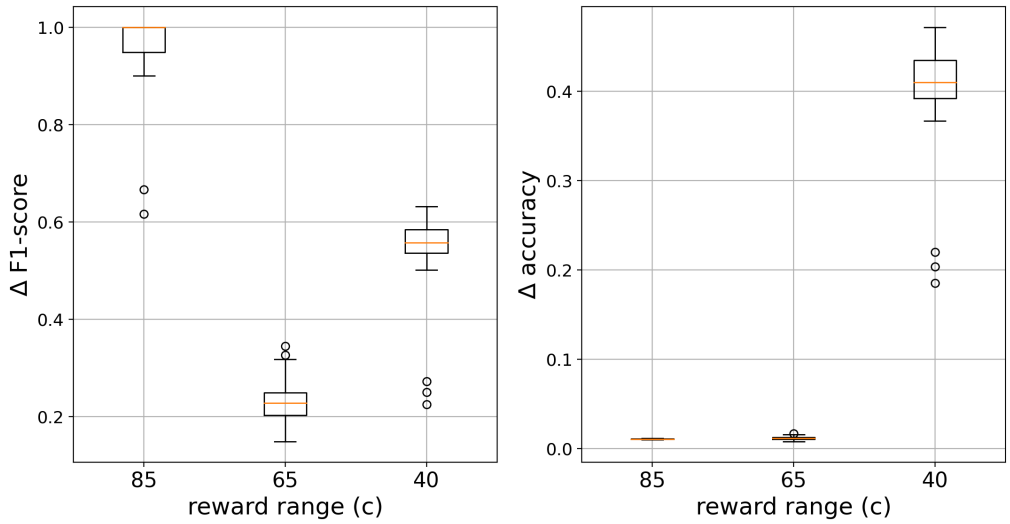


FIGURE 5.4: Box-plots of Δ F1-score and Δ accuracy using the optimal thresholds with different values of c

It is a compact summary of the policy that highlights the important details in a structured way. Notice that this result leverage the information expressed with the hard constraint $x_3 > 0.9$ (i.e., $x_3 = 0.966$ for action $Open_R$). There is a gap between the value of the rule for *Listen* (i.e., 0.8638) and the rules for $Open_R$ and $Open_L$ (i.e., 0.9644). This is because the trace does not contain any belief in this gap, and XPOMCP cannot build a rule to describe how to act in these beliefs.

Among the total 2659 trace steps, 1601 satisfies the rule, and 1058 does not satisfy it. We computed their Hellinger distance for all steps not satisfying the rule using the procedure described in Section 5.1. To classify unexpected decisions, we use the optimal threshold of $\tau = 0.045$ (Table 5.2). This procedure identifies 637 of the 1058 unsatisfiable steps as anomalous (i.e., their H^2 is above threshold τ). In this case, we achieve an F1-score of 1.0 and an accuracy of 1.0 using the proposed approach, and it takes XPOMCP $12,509s$ to compute this solution. This confirms the ability to detect unexpected decisions even when state-of-the-art anomaly detection methods have degraded performance (i.e., IF reaches an F1-score of 0.59 in this test, but it only takes $0,791s$ to do so).

5.4.2 Unexpected decisions in velocity regulation

To evaluate XPOMCP on the *velocity regulation* problem, we analyze one-by-one the decisions marked as unexpected by XPOMCP. (we recall that the exact policy cannot be computed for this problem because the state space is too large). The template

used to describe when the robot must move at high speed is the following:

```
declare-var x1, x2, x3, x4 prob;
declare-rule
  action Speed_F  ⟺  p0 ≥ x1 ∨ p2 ≤ x2  ∨ (p0 ≥ x3 ∧ p1 ≥ x4)
where x1 ≥ 0.9
```

where x_1, x_2, x_3, x_4 are free variables and p_0, p_1, p_2 are defined using the `diff` function presented in Section 4.2. The first two constraints of the rule are identical to the running example, but we add a third constraint (i.e., $p_0 \geq x_3 \wedge p_1 \geq x_4$) that combines p_0 and p_1 to describe when the robot must move at high speed. The correct value of c for *velocity regulation* is 103 (i.e., the difference between moving at speed 1 in a short segment and collide vs. going fast in a long subsegment without collisions, i.e., $0.6 \cdot 2 - 100, 1.4 \cdot 3$), but we set it to 90 to generate some errors. We run XPOMCP on a trace of 100 runs and XPOMCP returns the following rule:

$$\texttt{action } Speed_F \iff p_0 \geq 0.910 \vee p_2 \leq 0.013 \vee (p_0 \geq 0.838 \wedge p_1 \geq 0.132)$$

It takes 69.53 seconds to analyze the trace. This rule falls on 33 out of 3500 decisions, but only 4 are marked by XPOMCP as unexpected using threshold $\tau = 0.1$, which we select empirically by analyzing the values of H^2 of unexpected decisions. Table 5.3 shows some of the most notable steps that do not satisfy the rule (which are not necessarily unexpected decisions) in decreasing order of H^2. Column *id* shows an identification number for the step, columns p_0, p_1, p_2 show the probabilities in the beliefs of unexpected actions, column H^2 shows the Hellinger distance of the steps with unexpected actions, and we write in bold the Hellinger distance values greater than the threshold $\tau = 0.1$. Steps 1 and 2 are unexpected behaviours since POMCP decided to move at high speed even if it had poor information on the difficulty of the segment (p_0, p_1, p_2 are close to a uniform distribution). Steps number 3 and 4 are also unexpected. While they are closer to our rule because p_0 is the dominant value in the belief, they are significantly distant from the boundary of the rule and the decision taken by POMCP. Steps from 5 to 33 have beliefs that only slightly violate the rules of the related actions; these steps are not marked as unexpected due to the approximate nature of the rule.

To visualize the result, we show a *T-distributed Stochastic Neighbor Embedding (t-SNE) projection* (Maaten and Hinton 2008) of the beliefs (which are vectors in 3^8 dimensions). The belief of each step is used to compute point coordinates, and different colours represent the action taken by POMCP (see Figure 5.5). In particular, green, blue, and orange points represent steps in the traces in which POMCP selected, respectively, a low speed, a medium speed, and a high speed. While the rule generated by XPOMCP for action 2 (i.e., $Speed_F$) presents a clear and compact boundary, there are no obvious separations between the points of the three speed

id	p_0	p_1	p_2	H^2
1	0.335	0.331	0.334	**0.3526**
2	0.261	0.461	0.278	**0.3090**
3	0.671	0.198	0.131	**0.1717**
4	0.678	0.228	0.094	**0.1389**
5	0.775	0.196	0.029	0.0411
6	0.832	0.127	0.041	0.0347
32	0.853	0.126	0.021	0.0109
33	0.826	0.160	0.014	0.0105

TABLE 5.3: Notable unsatisfiable steps in velocity regulation ($c = 90$). The table present the identification (*id*), the belief (p_0, p_1, p_2), and the Hellinger distance (H^2) of each step.

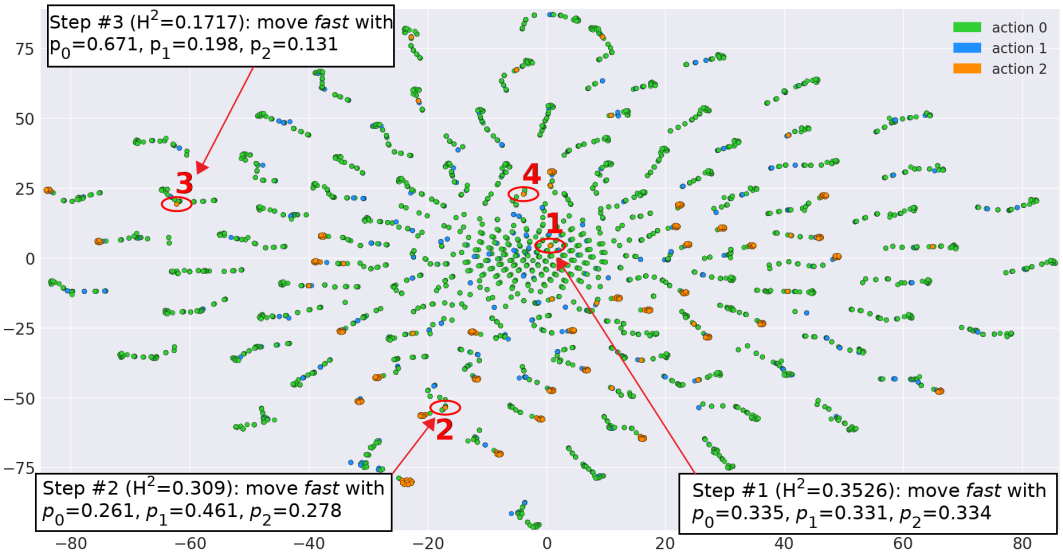


FIGURE 5.5: t-SNE of n-uples (belief, action) for the *velocity regulation* with $c = 90$. Our algorithm identify 4 anomalies, they are circled in red

values in the t-SNE chart. Most orange (i.e., high speed) points are grouped in small clusters spread around the graph, but some isolated orange points are also present. This kind of distribution is hard to analyze using standard anomaly detection algorithms like isolation forests. The four steps that are classified as unexpected decisions are circled in red. Points 1, 3 are isolated and far from any small cluster of orange points, while points 2 and 4 are close to one of the clusters. Note that not all isolated points are marked as unexpected. XPOMCP identifies the unexpected points not only by using their belief but also by the insight provided by the expert.

5.4.3 Unexpected decisions in rocksample

Rocksample is a challenging problem since its state space grows quickly as the dimension of the board and the number of rock grows. For instance, *Rocksample(11,11)*,

which uses 11 rocks in a 11×11 board has 247,808 states samples (Silver and Veness 2010). To achieve good performance in such large problems without using too many particles, POMCP uses some hacks to simplify the computation. In particular, it identifies a subset of interesting actions at each step and increases their starting values in the MCTS (see the C++ implementation of (Silver and Veness 2010)). This improves the performance of UCT regarding the exploration-exploitation trade-off since the most promising actions are favoured. We are interested in applying our methodology in this challenging domain. We inject errors by changing the number of the particles n_p used for the simulation, and we use XPOMCP to quantify the impact of this parameter. As in *velocity regulation*, it is not possible to compute an optimal policy for *rocksample* as a ground truth. Instead, we provide an in-depth analysis of the traces using the methodology and evaluate the results. The rules formulated in this section are then used as a base to test the shielding mechanism in Section 6.2.

We write rule templates about two important aspects of the policy. In the first analysis, we are interested in understanding the confidence level required before selecting the *sampling* action (i.e., the action in which the agent collects the rock and gets a related reward). This is a crucial decision with a big impact on the performance. In our second analysis, we investigate the impact of the distance between the agent and the rock on the *check* action (i.e., the agent should decide whether to check a rock that is at a certain distance or not). In particular, we want to understand if the policy decides to move on the North-South axis to perform better checks.

To describe the problem, we use the header

$$\texttt{actions} = \{North, East, South, West, sample, check_{1,\dots,11}\}\ int;$$
$$\texttt{belief} = int;$$
$$\texttt{problemInfo} = \{\};$$
$$\texttt{runInfo} = \{Rocks\ list[position]\};$$
$$\texttt{stepInfo} = \{pos\ position\};$$

There are four actions for movement, one for sampling, and eleven for checking. The states used in the belief are integers that encode the true value of the eleven rocks. At each step we store the (x, y) position of the agent (i.e., the *pos* value, $pos.x$, $pos.y$ can be used to specify the x and y coordinates). Finally, we store the predetermined position of the rocks in the *runInfo* field. To properly describe the robot's behaviour, it is important to keep track of which rocks were sampled and which ones were not. To do that, we introduce the `collected(r, n)` function, it takes a rock r from *runInfo* and a number of steps n as input and returns *true* if and only if r was sampled at a

step lower than n. We can write it as

$$\texttt{define-fun collected}(r\ rock, n\ step)\ bool$$
$$\exists\, n' \ s.t.\ (n'.run = n.run) \wedge (n'.num < n.num) \wedge$$
$$(n'.action = sample) \wedge (n'.x = r.x) \wedge (n'.y = r.y);$$

This function controls whether it exists a previous step in the same run that performed a sample action on the same rock.

We compute rules from different traces using different n_p, namely, 2^i particles with $i \in \{11, \ldots, 18\}$. The same number of simulations is performed. We select these parameters because 2^{18} provides great performance, as shown in (Silver and Veness 2010), and 2^{11} is the bare minimum to represent the possible states of the 11 rocks.

Analysis of sampling actions

We can now create a rule that explains when the agent considers worthy to sample a rock:

$$\texttt{declare-var}\ u\ prob;$$
$$\texttt{declare-rule}$$
$$\texttt{action}\ sample \iff$$
$$\bigvee_{r \in \{rocks\}} (pos = r.pos \wedge \neg \texttt{collected}(r, step) \wedge p(r.valuable) \geq u);$$

The above rule specifies that we should use the sample action if and only if we are in the position of a rock (i.e., $pos = r.pos$), the rock was not collected in a previous step (i.e., $\neg\texttt{collected}(r, step)$), and the probability that the rock in this position is valuable is above a certain unknown threshold (i.e., $p(r.valuable) \geq u$). The $\bigvee$ is used to compactly represent an *or* condition over the various rocks. This rule is very important because sampling is the most dangerous action, as it can lead to great rewards if used correctly or penalties otherwise.

Table 5.4 shows the results of our analysis. Column n_p shows the number of particles used in POMCP. Column u reports the value computed for u by XPOMCP. Columns # *unsafe sample* and # *unnecessary check* show the two kinds of unexpected decisions that can happen, namely, the samples made with confidence below u and the checks made on a rock with a confidence above u. As presented in column u, a higher number of particles leads to requiring a higher level of confidence before sampling a rock. This improves performance since the agent avoids sampling too many worthless rocks. It is important to note that some steps do not satisfy the rule, and this happens for two reasons. Namely, *i)* the agent samples a rock that is unlikely to be valuable (i.e., # *unsafe sample*), *ii)* the agent checks the value of a rock even if the confidence for the rock to be valuable is above the threshold u (i.e.,

n_p	u	# unsafe sample	# unnecessary check
2^{18}	0.835	0	1
2^{17}	0.818	0	4
2^{16}	0.827	0	3
2^{15}	0.79	0	8
2^{14}	0.797	5	2
2^{13}	0.708	7	7
2^{12}	0.620	58	5
2^{11}	0.627	70	39

TABLE 5.4: Rule for action *sample* of *rocksample*. For each number of particle n_p the table the *prob u* computed by XPOMCP. Columns # *unsafe sample* and # *unnecessary check* show the two kinds of unexpected decisions for action *sample*

column # *unnecessary check*, that is never zero). The first kind of unexpected decision is the most important since it can lead to poor performance. As shown in Table 5.4, with n_p greater than 2^{15} no unsafe sampling is performed.

Analysis of checks and movements

The implementation of *rocksample* provided in (Silver and Veness 2010) uses a heuristic to favour check actions in certain circumstances. It considers checking a rock based on the agent's history. In particular, it considers the number of previous checks performed on the rock and the ratio between positive and negative checks (if any). Notice that it is possible to check a rock from every position of the board, but the precision of these checks decreases when the distance between the agent and the rock increases. However, the POMCP heuristic does not consider the distance between the agent and the rock. After too many checks, the heuristic stops measuring a rock (i.e., only the number of checks is considered, not their quality). We want to investigate the strategy used by POMCP and, specifically, to understand whether POMCP decides to approach a rock before checking it. We express this idea using two rules, namely, one that describes when to use the various check actions and one that explains when to move on the North-South axis. *Rocksample* uses a specific action for checking each rock (i.e., $check_1$ for checking rock 1,..., $check_n$ for checking rock n, with $n = 11$ in our case study). Checking is particularly difficult to capture with a rule, mostly because it is a free action (in terms of reward); thus, the POMCP uses several checks in different scenarios. We assume that the agent checks rocks that are not too far and only if the confidence that the rock is valuable is inside a certain interval, the robot should not check a rock when the confidence that it is

valuable/worthless is high enough. For action $check_i$, we write the rule:

`declare-var` v_1, v_2 *real*;

`declare-var` w_1, w_2 *prob*;

`declare-rule`

$\texttt{action}\ check_i \implies v_1 < \texttt{distance}(pos, rock_i) < v_2 \wedge w_1 < p(rock_i.valuable) < w_2$;

`where` $v_1 < v_2 \wedge w_1 < w_2$;

The function *distance* computes the Euclidean distance between the agent's position and the position of the rock (a real number). Notice that the rule uses a simple implication and not a $\iff$, this means that when the robot performs a check, the requirement must be satisfied, but the robot is not forced to select this action when the formula is true. This is important to describe the behaviour of the policy for the various $check_i$ actions since it is common to have multiple valid checks at a single point, and we cannot do all of them at the same time. Finally, we want a rule that describes when the robot should move on the North-South axis. It is important to understand if the North (South) action is taken only when there is at least one valuable rock North (South) of the agent or if it is worth moving on the North-South axis to get closer to the rock before checking it. We write the template:

`declare-var` q *real*;

`declare-rule`

$\texttt{action}\ North \implies \exists r\ s.t.\ \neg\texttt{collected}(r) \wedge r.y < pos.y \wedge p(r.valuable) \geq q$;

$\texttt{action}\ South \implies \exists r\ s.t.\ \neg\texttt{collected}(r) \wedge r.y > pos.y \wedge p(r.valuable) \geq q$;

`where` $q \geq u$;

It expresses the idea that the agent should only move on the North-South axis when there is at least one uncollected rock with a value that is greater than u (i.e., the threshold that describes when it is valuable to sample a rock). We are interested in detecting unexpected decisions for this rule to understand if the policy decides to take these actions in different circumstances (i.e., for a check).

Table 5.5 shows the results on these two rules. Column n_p shows the number of particles used in POMCP. Column *check distance* shows the interval of the acceptable distance between the agent and the rock before using a check. Column *check confidence* shows the level of confidence in the value of a rock before performing a check. Finally, column *North-South* shows the level of confidence required on the fact that a valuable rock is situated North (South) of the agent before moving in that direction.

These rules do not generate any unexpected decisions and can describe all the actions taken by the policy. This answers our doubt on the influence of the distance on the check actions: moving on to a new position to improve the quality of a check is never worth it, and the agent considers a good strategy to check rocks far

n_p	**check distance** $([v_1, v_2])$	**check confidence** $[w_1, w_2]$	**North-South**(q)
2^{18}	[0.000, 12.728]	[0.025, 0.946]	0.835
2^{17}	[0.000, 12.042]	[0.037, 0.947]	0.818
2^{16}	[0.000, 12.042]	[0.029, 0.951]	0.962
2^{15}	[0.000, 12.042]	[0.033, 0.968]	0.79
2^{14}	[0.000, 12.042]	[0.035, 0.959]	0.797
2^{13}	[0.000, 12.042]	[0.027, 0.974]	0.708
2^{12}	[0.000, 12.042]	[0.024, 0.986]	0.972
2^{11}	[0.000, 12.042]	[0.008, 0.983]	0.627

TABLE 5.5: Rules for actions $check_{1,\dots,n}$, *North*, and *South* of *rocksample*. The tables shows the values for the variables v_1, v_2, w_1, w_2, q computed by XPOMCP for different number of particles n_p

away from their position. This is shown in columns *check distance* and *check confidence*, in which values remain mostly similar regardless of the number of particles. This happens because these checks are free. Thus, the best policy is to use many of them, disregarding the single checks' quality. The fact that the rule for moving on the North-South axis never generates unexpected decisions using this rule is particularly important. It means that these actions are only taken to move toward a rock that we already know to be valuable (i.e., we already have checked them, and we have good confidence in their real value) and not get closer before a check.

Chapter 6

Rule-based Shielding

In this chapter, we present a methodology that uses the rules generated by XPOMCP (as described in Chapter 4) to build a shield mechanism. As in the anomaly detection algorithm presented in Chapter 6, this methodology identifies unexpected decisions of the policy. However, the shield is used online, on top of the policy is used by the agent. Notice also that the shield is not employed to identify anomalies in pre-generated traces but to prevent unwanted actions during future executions of POMCP.

6.1 XPOMCP-based shielding

It is possible to use the logical rules generated by XPOMCP as a shield to prevent undesired actions. We integrate the shield into POMCP to preemptively (i.e., before executing them) prune undesired actions considering the current belief. It includes a set of rules trained as explained in Section 4.3 and a set of representative beliefs generated as described in Section 5.1. We use the procedure presented in Algorithm 6 to shield the actions. We start with an empty set of *legal actions* $\mathcal{L}$ (line 1). We add an action a to $\mathcal{L}$ if it satisfies at least one of three possible conditions, namely, *i)* there is no rule for the action in the shield (line 3), *ii)* the current belief b satisfies the constraints defined by the rule for a (line 5, where we use the `test_constraints` function to check the constraints), *iii)* the current belief b does not satisfy the rule, but the distance between b and the boundaries defined by the rule for a are below the threshold τ defined in the shield Sh (line 7). These conditions could result in an empty set of legal actions $\mathcal{L}$ (i.e., if the rules are very strict). Hence, for each belief, we define a default safe action called a_{safe}, which is used when no other action is permitted. We force POMCP only to consider legal actions in the first step of the simulation. After a legal action is selected, the Monte Carlo Tree Search is performed as usual. Notice that when the original implementation of POMCP (Silver and Veness 2010) selects a particle in the simulation process, it assumes that the state encoded by the particle is the current state of the system (which for a POMDP is not observable). Therefore the belief can only be considered in the first step. With this mechanism, we can ensure that the rule of the shield is respected, but we do not force POMCP to select a specific

Algorithm 6: Shielding procedure

Input: a belief b, a shield Sh, safe action a_{safe}
Output: set of legal actions $\mathcal{L}$

$\mathcal{L} \leftarrow \emptyset$;
foreach *action* $a \in \mathcal{A}$ **do**
  if $a \notin Sh$ **then**
    $\mathcal{L} \leftarrow \mathcal{L} \cup a$;
  else if $Sh.\texttt{test\_constraints}(b)$ **then**
    $\mathcal{L} \leftarrow \mathcal{L} \cup a$;
  else if $\exists r \in Sh.Repr : H^2(b, r) < Sh.\tau$ **then**
    $\mathcal{L} \leftarrow \mathcal{L} \cup a$;
if $\mathcal{L} = \emptyset$ **then**
  $\mathcal{L} \leftarrow \{a_{safe}\}$;
return $\mathcal{L}$;

action. The best action is still decided using the regular POMCP search considering only legal actions.

The computation of legal actions is performed only once per step since the current belief does not change during the simulation. Checking that the belief satisfies the constraints (by the `test_constraints` function) has a fixed cost. Checking the H^2 of the representative beliefs increases linearly with the number of beliefs. As shown in Section 5.3, this is a negligible cost, and the reduced number of actions that must be tested (because not all actions are now legal) can also slightly reduce the execution time.

6.2 Results on rule-based shielding

We test our shielding mechanism on *tiger*, *velocity regulation*, and *rocksample* that are described in Section 3.3.

6.2.1 Experimental setting

In our experimental setup, we consider several values of parameters c and n_p. We create a trace for each domain and parameter value using the unshielded POMCP. Then we train a rule to be used as a shield using XPOMCP. Notice that these traces could contain errors. Finally, we test the shielded POMCP using the same parameters. We evaluate the methodology's performance by comparing the average discounted return achieved by the two versions of POMCP (original and shielded). We consider 1000 runs for *tiger* and 100 runs for *velocity regulation* and *rocksample*. We generate 1000 representative beliefs in all cases and use $\tau = 0.10$ as a threshold to identify anomalies.

	No Shield		Shield			
c	return	time (s)	return	RI	time (s)	#SA
110	3.702($\pm$0.623)	0.066($\pm$0.027)	3.702($\pm$0.623)	0.00%	0.065($\pm$0.029)	0
80	3.593($\pm$0.632)	0.067($\pm$0.030)	**3.702**($\pm$**0.623**)	**3.03**%	0.061($\pm$0.027)	4
60	3.088($\pm$0.673)	0.060($\pm$0.025)	**3.702**($\pm$**0.623**)	**19.88**%	0.061($\pm$0.027)	121
40	$-$4.173($\pm$1.101)	0.035($\pm$0.017)	**3.702**($\pm$**0.623**)	**188.71**%	0.052($\pm$0.023)	647

TABLE 6.1: Comparison between shielded and unshielded POMCP for *tiger*. The table shown the average discounted return and the average execution time. Standard deviations are in parenthesis and the best results are highlighted in bold. Column RI shows the relative increase and column #SA shows the number of shielded actions

6.2.2 Shielding for tiger

We base our shield on the same rule presented in Section 5.4.1. We learn the rule parameters from a POMCP trace and create a shield from this rule. Since this shield gives a rule for all possible actions, it is important to provide a safe action for each belief, as explained in Section 6.1. For this domain, we use $a_{safe} = Listen$. The correct value of c is 110 because the reward interval is $[-100, 10]$. For each value of c in $\{110, 80, 60, 40\}$, we generate a trace with 1000 runs each, using a fixed seed for the pseudo-random algorithm. In each case, we use 2^{15} particles and a maximum of 10 steps. POMCP with a correct value of c produces the optimal policy (we tested that by comparing the decisions taken by POMCP with an exact policy computed using *incremental pruning* (Cassandra, Littman, and N. L. Zhang 1997)). The results are presented in Table 6.1. The first column shows the different values of c. The second (third) column shows the average return (time) achieved by the original POMCP and the relative standard deviation. The *Shield* section shows the average return and time achieved by POMCP using a shield (columns four and six). Column RI shows the relative increase in performance between the two original and shielded POMCP. Finally, column #SA shows how often the shield alters the decision during the execution.

As shown in Table 6.1, in the first row of column #SA, the shield does not interfere with the correct policy. Lower values of c produce a lower average discounted return, as shown in column *return* of the *No Shield* section. In the *Shield* section we present the relative increase (computed as $RI = \frac{shielded - original}{|original|} \cdot 100$) between the original and the shielded version of the POMCP (see column RI). This column shows that the benefit of using a shield increases when the number of errors increases. For the return column, we report in bold values whose difference from their no-shield counterpart is statistically significant according to a paired t-test with 95% confidence. While there is no difference between the two cases in the first row, the difference is statistically significant in the other three rows.

The average return achieved using the shield is the same in all four cases, and this is also identical to the return achieved by the correct policy. This is because in *tiger* we can write a shield that perfectly recreates the behaviour of the correct policy

	No Shield		Shield			
c	return	time (s)	return	RI	time (s)	#SA
103	24.716($\pm$3.497)	10.166($\pm$0.682)	**26.045 ($\pm$ 3.640)**	**5.38%**	10.118($\pm$0.238)	7
90	18.030($\pm$3.794)	10.173($\pm$0.234)	**22.680 ($\pm$ 3.524)**	**25.79%**	10.166($\pm$0.241)	12
70	4.943($\pm$5.260)	10.278($\pm$0.234)	**8.970 ($\pm$ 4.556)**	**81.46%**	10.377($\pm$0.230)	51
50	0.692($\pm$5.051)	10.374($\pm$0.230)	1.638($\pm$4.525)	136.53%	10.435($\pm$0.336)	171

TABLE 6.2: Comparison between shielded and unshielded POMCP for *velocity regulation*. The table shown the average discounted return and the average execution time. Standard deviations are in parenthesis and the best results are highlighted in bold. Column RI shows the relative increase and column $\#SA$ shows the number of shielded actions

(as shown in Section 5.3.5), a goal that is difficult to achieve in real-world problems. This is particularly interesting because the shields in the cases of $c \in \{80, 60, 40\}$ are obtained by using traces generated with a POMCP implementation that does make some mistakes. As a consequence, the execution traces contain wrong decisions. However, the combination of expert insight and MAX-SMT-based analysis of the traces results in a shield with extremely good performance. As shown in the *time* column, in general, the presence of the shield does not noticeably impact in terms of run-time. The shield generation algorithm takes between 10 and 12 seconds to generate the shield in this case. In the last row, the original POMCP is particularly fast. This happens since the erroneous POMCP opens many doors as fast as possible without listening. This leads to runs that achieve very low average return but ends quickly.

6.2.3 Shielding for velocity regulation

We use the same rule template of Section 5.4.2 to describe when the robot moves at maximum speed (i.e., action 2). This is the most dangerous action because there is always a risk of collision involved, as explained in Table 3.5.c. We use a trace with 100 runs to train and test the shield. The shield generation takes approximately 50 seconds to generate velocity regulation shields. Table 6.2 shows the result of the experiment. The first column shows the different values of c. The second (third) column shows the average return (time) achieved by the original POMCP and the relative standard deviation. The *Shield* section shows the average return and time achieved by POMCP using a shield (columns four and six). Column RI shows the relative increase in performance between the two original and shielded POMCP. Finally, column $\#SA$ shows how often the shield alters the decision during the execution. As in *tiger*, a lower value of c produces a lower return. In this case, the best c is 103 (the difference between a collision when we move slowly in the shortest segment and a fast movement in the longest segment without a collision). The first row of the table shows that, unlike *tiger*, the usage of a shield can improve the performance even when c is correct. In this case, the shield intervenes only 7 times

	No Shield		Shield			
n_p	return	time (s)	return	RI	time (s)	#SA
2^{14}	13.01($\pm$0.57)	14.29($\pm$5.0)	13.14($\pm$0.56)	0.99%	17.11($\pm$6.98)	5
2^{13}	12.32($\pm$0.58)	8.05($\pm$2.51)	12.39 ($\pm$ 0.55)	0.05%	8.68($\pm$2.95)	7
2^{12}	10.21($\pm$0.77)	4.12($\pm$1.37)	**11.98 ($\pm$ 0.54)**	17.20%	4.64($\pm$1.60)	52
2^{11}	8.83($\pm$0.80)	2.13($\pm$0.56)	**10.25 ($\pm$ 0.58)**	16.01%	2.43($\pm$0.75)	65

TABLE 6.3: Comparison between shielded and unshielded POMCP for *rocksample*. The table shown the average discounted return and the average execution time. Standard deviations are in parenthesis and the best results are highlighted in bold. Column *RI* shows the relative increase and column #*SA* shows the number of shielded actions.

(over the 3500 analyzed steps), yielding a 5.38% increment in the return. This happens because the shield blocks the rare cases in which the POMCP simulations are not enough to properly assess the risk of moving at high speed. In fact, we use 2^{15} particles as in *tiger*. This yields acceptable performance in general, but sometimes the simulations are not good enough, and the robot takes a decision that the expert considers too risky.

When c decreases, the shield intervenes more often (see column #*SA*) since the error due to the limited number of simulations is combined with the errors generated by an incorrect value of c. Table 6.2 also shows that a higher number of interventions leads to a bigger relative increase in the performance (column *RI*). The difference is statistically significant in the case of $c \in \{103, 90, 70\}$, and shows that the shield provides up to 81% performance improvement. This happens even in cases where the shield is trained using traces generated by a POMCP process that makes some mistakes. In the case of $c = 50$, the return increases, but the difference is not statistically significant because the standard deviation is very high. The shield intervenes 171 times by blocking risky high-speed moves, but unlike *tiger*, in which we use a rule for every possible action, here POMCP made many wrong decisions when it moves at low or medium speed (for example, by moving slowly when the path is clear). As in *tiger*, the usage of the shield does not significantly increase the time required to perform the simulations.

6.2.4 Shielding for rocksample

We test *rocksample* with a shield for the sampling action. It blocks the sampling of rocks with low confidence of being valuable in the current belief. As shown in Section 5.4.3, the agent performs unsafe sampling only when it uses $n_p = 2^{14}$ or less. Thus we focus our analysis on these cases. We use a trace with 100 runs to train and test the shield. The shield generation takes 23 seconds.

Results are presented in Table 6.3. The first column shows the number of particles used by POMCP. The second (third) column shows the average return (time) achieved by the original POMCP and the relative standard deviation. The *Shield* section shows the average return and time achieved by POMCP using a shield (columns four and six). Column *RI* shows the relative increase in performance between the

two original and shielded POMCP. Finally, column #*SA* shows how often the shield alters the decision during the execution. *Rocksample* is a harder problem than *tiger* and *velocity regulation*, and the RI achieved is lower in this case. The difference is significant only when we use 2^{11} or 2^{12} particles. In these cases, the shield provides roughly a 15% increase in performance because the agent does not sample rocks for which it has low confidence. In the cases of 2^{13} and 2^{14} particles, the difference is minimal since unsafe samplings are rare. Notice that in these cases, the agent never performs an unsafe action thanks to the shield, but this is not enough to build a better policy in this case. For example, the agent does not perform unsafe sampling. Instead, it simply ignores some potentially valuable rocks. The policies generated using more particles perform extra checks in the same context. Due to the iterative nature of our methodology, it is possible to investigate these aspects using new rules, thus generating a new shield to handle these situations.

Chapter 7

Active Rule Synthesis

In this section, we present an extension of the XPOMCP methodology, called *Active XPOMCP*, that inspects the POMCP policy actively. Instead of relying on a pre-generated trace, the active approach uses XPOMCP generated' rules as guidance to reach new unexplored beliefs that provide meaningful information. We show that this approach can outperform the passive approach, achieving more precise rules using fewer (more informative) belief-action pairs.

7.1 Active XPOMCP

The XPOMCP methodology builds logical formulas that describe the properties of POMCP policies in a human-comprehensible way. The structure of these formulas is, in fact, designed by human experts. Hence they can embed assumptions about the policy, i.e., human expectations about the action the policy should select in specific conditions. The methodology uses a set of traces generated by a POMCP agent to achieve this. The main difference between the passive approach presented in Chapter 4 and *Active XPOMCP* here introduced is that XPOMCP produces rules that describe a set of executions produced by POMCP, while *Active XPOMCP* produces rules that describe the actual POMCP policy. To reach this goal *Active XPOMCP* actively queries a POMCP instance providing beliefs and receiving actions to minimize the number of queries to make the rule generation process as fast and accurate as possible.

Since a rule can only describe the policy according to the beliefs sampled by *Active XPOMCP*, we represent the *uncertainty* of a rule as a set of unexplored beliefs. The rule uncertainty is a key concept in our methodology, which guides two processes: identifying the most informative beliefs and the termination of the rule improvement when the rule convergences. The rule uncertainty depends on the fact that given a rule template and a trace, multiple rules can be generated because the trace could contain only partial information about the policy's property expressed by the rule. In more detail, a rule is located in a position that satisfies the majority of the beliefs in the current trace. Hence, several positions are available if the beliefs are far from the decision boundary of the policy (i.e., the hyperplane of beliefs in which the policy changes its action). XPOMCP locates the rule as close as possible to

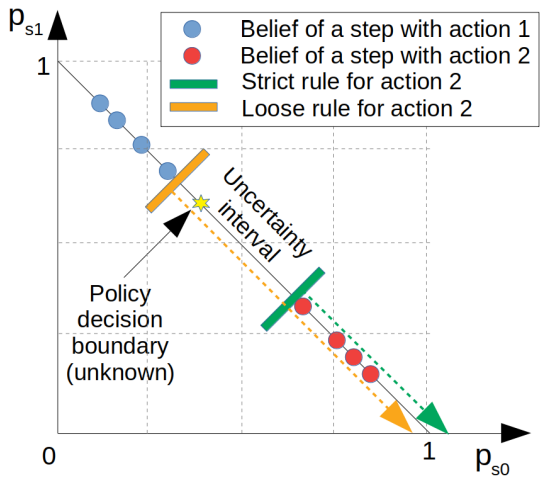


FIGURE 7.1: Graphical representation of the main elements involved in the *Active XPOMCP* algorithm.

the beliefs (in the trace) related to the rule action. At the same time, *Active XPOMCP* reduces the uncertainty interval to locate the rule as close as possible to the decision boundary.

7.1.1 Strict and loose rules

Figure 7.1 graphically depicts the main elements of the *Active XPOMCP* algorithm. It computes two rules (yellow and green lines in the picture) to describe the behaviour of the policy on a specific action (action 2 in the picture). The first rule (green line) is called *strict rule* and is located as close as possible to the observed beliefs related to that action (i.e., red points). The second rule (yellow line) is called *loose rule*, and it is located as far as possible from the red points without including any blue point (i.e., points that represent action 1). In summary, the strict rule behaves conservatively. It describes only beliefs collected in the trace about the action of interest. The loose rule captures all the beliefs (explored or not) that do not explicitly violate the requirement expressed by the expert in the rule template. *Active XPOMCP* reduces the distance between the two rules by selecting informative beliefs, namely beliefs in the uncertainty interval that contribute to reducing the distance between loose and the strict rule until the two converge. This is non-trivial in real problems because the belief space is highly multidimensional and therefore huge. For this reason, *Active XPOMCP* introduces another constraint, namely, *reachability*. In fact, only a small part of the belief space is reachable from the initial belief b_0, hence considering only reachable beliefs we reduce the search space and make *Active XPOMCP* scalable.

7.1.2 Active analysis of POMCP policies

The methodology proposed in this chapter is implemented by the *Active XPOMCP* procedure displayed in Algorithm 7. It internally calls two procedures, *RuleSynthesis*

Algorithm 7: *Active XPOMCP*

Input: POMCP instance: *POMCP*; Rule template: rt; Initial belief: b_0; Runs in first trace: N; Uncertainty thrs: ϵ
Output: Instantiated rule: r

```
1  t=∅; // Trace init.
2  for i = 1, ··· , N do
3  |  t ← t ∪ POMCP (b_0);
4  [st, lo] ← RuleSynthesis(t, rt); // Strict/loose init
5  while lo - st < ε do
6  |  t_a ← ActiveTraceGeneration(POMCP,lo, st);
7  |  t ← t ∪ t_a; // Trace set update
8  |  [st, lo] ← RuleSynthesis(t, rt); // st/lo rule update
9  r ← st ;
10 return r
```

(displayed in Algorithm 8) which generates the loose and strict rules given the current trace, and *ActiveTraceGeneration* (shown in Algorithm 9) which generates a new informative trace given the current loose and strict rules. *Active XPOMCP* (Algorithm 7) receives in input a POMCP instance, a rule template rt, the number of runs N in the initial trace, an initial belief b_0, and the threshold ϵ on the maximum distance between loose rule and strict rules required for stopping the update process. The procedure first generates a trace t containing a small set of runs (we usually use only five runs to start *Active XPOMCP*) by launching the POMCP instance from a starting belief b_0 (lines 1-3, usually a uniform distribution over all possible states). Empirically, this is useful to find approximate values for the strict and loose rules. Then, it initializes the loose and strict rules (line 4) by passing the trace t and the rule template rt to the *RuleSynthesis* procedure, explained in Subsection 7.2. We notice that standard XPOMCP returns the strict rule here since the methodology considers only the solution that is as close as possible to the beliefs related to the action of the rule template (see Figure 7.1). Instead, *Active XPOMCP* starts an iterative process of rule update (lines 5-8) in which, first, it extends the trace with a new run generated by the *ActiveTraceGeneration* (line 6), explained in Subsection 4.2.4, aiming to reduce the uncertainty interval between the loose and the strict rules; second, new strict and loose rules are generated by the *RuleSynthesis* procedure (line 8) using the old set of runs together with the newly generated one. This process ends when the difference between the loose and the strict rules becomes smaller than ϵ (see line 5). If the template has a single free variable, the distance between loose and strict rules is computed as a simple difference between the variable's strict and loose values. Otherwise, we consider each free variable independently, stopping when the largest difference is below ϵ. When this process stops, the strict rule is returned.

Algorithm 8: RuleSynthesis

Input: Rule template: rt; Trace: t
Output: Strict rule: st, Loose rule: lo

```
1  solver.initialize();
2  foreach step s in t do
3      r_s ← [rt]_{s.belief}; // instantiate variables in rt
4      if s.action ≠ rt.action then
5          r_s ← ¬ r_s;
6      l_s ← new literal;
7      cost ← cost ∪ l_s;
8      solver.add(l_s ∨ r_s);
9  maxsat ← solver.minimize(cost); // satisfy steps
10 solver.add(maxsat);
11 slack ← |rt.variables − t.belief|;
12 st ← solver.minimize(slack); // build st
13 lo ← solver.maximize(slack); // build lo
14 return [st, lo]
```

7.2 Strict and loose rule syntehsis

Active XPOMCP uses a rule template and a trace to build two rules that describe the system. The procedure is presented in Algorithm 8. The algorithm builds a clause for every step in the trace. In line 3, we instantiate the probability variables of the rule template t with the values store in step s. For example, if we have a rule template "`select` a `when` $p > x$" we substitute p with the values stored in the trace. We aim to find a value for x (and the other free variables) that satisfies as many of these clauses as possible. If the step uses an action that is different from the action described by the template, we negate the formula (line 5) because the clause must be false. We then build a dummy literal l_s (line 6) for each step s. The MAX-SMT procedure uses this to satisfy the steps that cannot be satisfied using the rules. We want to use as few of these literal as possible. Thus we collect them into a *cost* function that must be minimized. Finally, in line 8 we insert a combination of the clause and the dummy literal into the solver. After this step, we run a MAX-SMT solver to find a solution that minimizes the cost of using dummy literals. In general, there could be many solutions that satisfy these requirements, and we are particularly interested in two of them, namely, the loose and the strict solution. We force the solver to satisfy the bound computed in the MAX-SMT step (line 10), and we create a value that measures how far our solution is from the steps observed in the trace. We do this by computing the shortest Euclidean distance between the values of the free variables of a solution and the closer belief collected into the trace (i.e., the information that we have stored in r_s in line 3). The strict rule searches for a solution that minimizes this value, while the loose rule maximizes it while respecting the MAX-SMT constraints. Notice that the maximization procedure used in line 9 is a MAX-SMT procedure (i.e., it tries to maximize the satisfiable clauses in

Algorithm 9: ActiveTraceGeneration

Input: POMCP inst.: $POMCP$; Loose rule: lo; Strict rule: st; Initial belief: b_0
Output: trace: t

```
1  b ← b_0;
2  tr ← ∅;
3  while ¬ POMCP.terminal() do
4      a ← POMCP.select_action(b);
5      t ← t ∪ ⟨b, a⟩;
6      o ← POMCP.apply(a);
7      for o' ∈ POMCP.observations do
8          b' ← bao';
9          if b' ∉ st ∧ b' ∈ lo then
10             o ← o';
11     b ← bao;
12 return tr
```

the Boolean layer) while the procedure used in line 12 and 13 are maximizations on real variables performed by the arithmetic module.

7.2.1 Heuristics for active trace generation

The goal of the *ActiveTraceGeneration* procedure (Algorithm 9) is to collect new beliefs that can be used by *RuleSynthesis* to make the strict and loose rules converge to the true policy decision boundary. These beliefs should have two main features: to be located in the uncertainty interval of the rule and to be reachable from the initial belief. Specifically, by reachable beliefs, we mean beliefs that POMCP can generate in one or more steps from the current belief. We use a modified black-box simulator as the environment that returns observations to POMCP, instead of the standard environment, to guide the exploration towards informative beliefs. This is because, given a belief, the new beliefs reachable from POMCP depend on the selected action and the received observation (returned by the environment). The action is selected by POMCP using the policy that we are describing with the rule; thus, we do not alter it. The observation works as a filter of particles in the belief update process performed by POMCP. It is selected by the (simulated) environment that we modify to explore the reachable states better. In other words, we generate a fake environment by modifying the black box simulator of POMCP to address the exploration of reachable beliefs not yet explored. The fake environment provides, for instance, also observations that are unlikely (but not impossible) to be obtained. In this way, we generate several reachable beliefs. Since the number of reachable beliefs can be very large, we then select the best observations returned by the environment by exploring two main elements, namely, *i)* the Monte Carlo Tree generated by POMCP, *ii)* the strict and loose rules built by analyzing previous executions (i.e., a trace).

In particular, *ActiveTraceGeneration* searches for beliefs that have not been explored yet and are located in the uncertainty interval (i.e., the space between the loose and the strict rule). The procedure starts from an initial belief b_0 (line 1). It builds a trace (i.e., t) containing a single run. It uses POMCP to compute the best action in the current belief (line 4) and it adds the current belief-action pair to the trace (line 5). The procedure must then select which reachable belief is the most significant given the current loose and strict rules. First, it computes the observation POMCP would have received using its black box simulator (line 6). Then it tries to compute an alternative observation that would lead the research toward unexplored beliefs (line 7–10). We compute the future belief as $b' \leftarrow bao'$ (i.e., by applying action a to belief b and receiving observation o'). We then select o' if b' does not satisfy the strict rule but it also satisfies the loose rule (line 9). This belief is a valuable addition to the trace t because it represents a belief not currently described by the rule (i.e., it does not satisfy the strict rule) but for which we do not have any explicit confirmation that the rule should not explain it (i.e., it satisfies the loose rule). Notice that if an observation is not possible in a given belief, this observation is never considered because b' is an empty belief that does not satisfy any rule (i.e., line 9 is always false). We generate the future belief using the best action and the selected observation (line 11), and we iterate the procedure until it convergences to a final state. When the procedure ends, we return the newly generated trace t.

7.3 Experimental results

This section tests the shielding capabilities of the rules generated using *Active XPOMCP*. We consider *rocksample* and *velocity regulation* (as described in Section 3.3), and we compare the performance of standard XPOMCP and Active XPOMCP. We do not consider the *tiger* problem since it is too easy, and the active approach does not present any advantage over the passive strategy.

7.3.1 Active rule synthesis in rocksample

We consider a rule that describes when the agent should sample a rock. Sampling is the most critical action because it yields the biggest difference in reward. To develop a successful sampling strategy, the agent must exploit the noisy observations received from the sensor to update its belief about the true value of the rock until it becomes confident enough about this value. Only then should it decide to sample or not to sample the rock. Excessive use of the sensor leads to a lower return (i.e., cumulative reward), but sampling valueless rocks yields a strongly negative reward (i.e., -10). Hence the agent must balance sensing and sampling to reduce the risk to optimal levels.

Experimental setting

We are interested in understanding when the agent should sample a rock. In particular, the value of probability above which the POMCP policy selects to sample the rock. To this end we generate the following rule template:

`declare-var` *u prob*;

`declare-rule`

`action` *sample* $\iff$

$$\bigvee_{r \in \{rocks\}} (pos = r.pos \wedge \neg \texttt{collected}(r, step) \wedge p(r.valuable) \geq u);$$

It specifies that we expect the agent to sample a rock only if it believes that it is valuable with a probability (i.e., $p(r.valuable)$) greater than a certain value u. However, the value u is a free variable in the template because we do not know it. We use *Active XPOMCP* to discover this value by analyzing the behaviour of the policy is highly informative (and reachable) situations selected using our active strategy. The template also states that the robot should not sample rocks that have been already sampled (i.e., $\neg\texttt{collected}(rock, step)$). Furthermore, it is worth noting that the robot can successfully sample a rock only if it is in the same position as the rock itself. Thus, if the robot is on an empty cell, the value $p(rock.valuable)$ is always set to zero.

performance measure

We perform two experiments to measure the performance of *Active XPOMCP*. In the *first*, we generate rules using different numbers of runs N ($N = 5, 10, \ldots, 50$). For each value of N we compute a logical rule using both XPOMCP and *Active XPOMCP*. Notice that *Active XPOMCP* starts from $N = 5$ and automatically increases the number of runs until convergence, while XPOMCP works with a single trace. Hence we run it several times using traces with different numbers of sets. The test is repeated ten times, changing the seed at each test. The measures of interest for this test are two: the size of the uncertainty interval, which measures the uncertainty of the rule, and the F_1 score, which measures the capability of the logical rule to describe the property of the POMCP policy.

In the *second* experiment, we compute logical rules from traces with $N = \{5,$ 10, 15, $20\}$ runs, and we use these rules to shield future execution of POMCP. We build this shield with the methodology presented in Chapter 6. It takes a logical rule generated and builds a shield that guarantees that the rule is satisfied during the execution. The maximum N we use is 20 because *Active XPOMCP* converges with that number of runs. Rules are always computed using $NS_{gen} = 2^{15}$ simulations. Then we use the two rules (for each N) to shield an instance of POMCP, which uses a smaller number of simulations, namely, $NS_{shield} = 2^{13}$. This instance of POMCP is,

of course, more imprecise but also nearly four times faster than the instance using 2^{15} (it performs 75% simulations less). The measures in which we are interested are the average discounted return of *i)* the standard POMCP, *ii)* the shielded POMCP using the logical rule computed by XPOMCP, *iii)* the shielded POMCP using the logical rule computed by *Active XPOMCP*. In all cases POMCP uses $NS_{shield} = 2^{13}$ simulations. Notice that the average discounted return is computed over 100 runs in this case.

Results on the first experiment

Figures 7.2.a,b,c show the results of the first experiment. In particular, the mean uncertainty interval, measuring the difference between the strict and the loose value of x, is displayed in Figure 7.2.a. The interval values decrease both for rules computed by XPOMCP (blue line) and for rules computed by *Active XPOMCP* (red line), but the decrease of the rules computed by *Active XPOMCP* is faster and the difference is quite large on rules computed using 10 runs (i.e., $\Delta\mathcal{U} = 0.086$) and 15 runs (i.e., $\Delta\mathcal{U} = 0.098$). Then this difference decreases with 20 runs, and with 30 runs *Active XPOMCP* ends because it meets its stopping criterion. Interestingly, the standard deviation represented by the shadow areas in the chart is much smaller for *Active XPOMCP* than for XPOMCP, showing that the active strategy converges quickly to the decision boundary.

The difference in the uncertainty interval of the rules produced by XPOMCP and *Active XPOMCP* directly affects the capability of these rules to describe the property of the POMCP policy investigated by the rule template. This is shown in Figure 7.2.b, which displays the F_1 score obtained comparing the actions selected by POMCP with those selected by the generated rules on 100 runs. Starting from the trace with 15 runs, the F_1 score of the rules computed by *Active XPOMCP* is larger than that of the rules computed by XPOMCP, showing that the rules generated by *Active XPOMCP* are more accurate in those cases.

Results on the second experiment

The final goal of these experiments is not only to show that *Active XPOMCP* can generate more accurate rules than XPOMCP with a smaller number of runs but also to generate better shields than XPOMCP. In this regard, Figure 7.2.c shows the results of the second experiment. Namely, the average discounted return of the shielded POMCP using the rules generated by *Active XPOMCP* (orange line) is larger than that of the shielded POMCP using the rules generated by XPOMCP (blue line) for 10 and 15 runs, yielding a relative increase in performance of 4.5% and 1.7% respectively. As shown in the results presented in Chapters 5 and 6, rocksample is a challenging problem because standard POMCP already provide good results except for some very rare but significant errors. The discounted return of XPOMCP and *Active XPOMCP* on five runs are equal because these runs are the same for the two

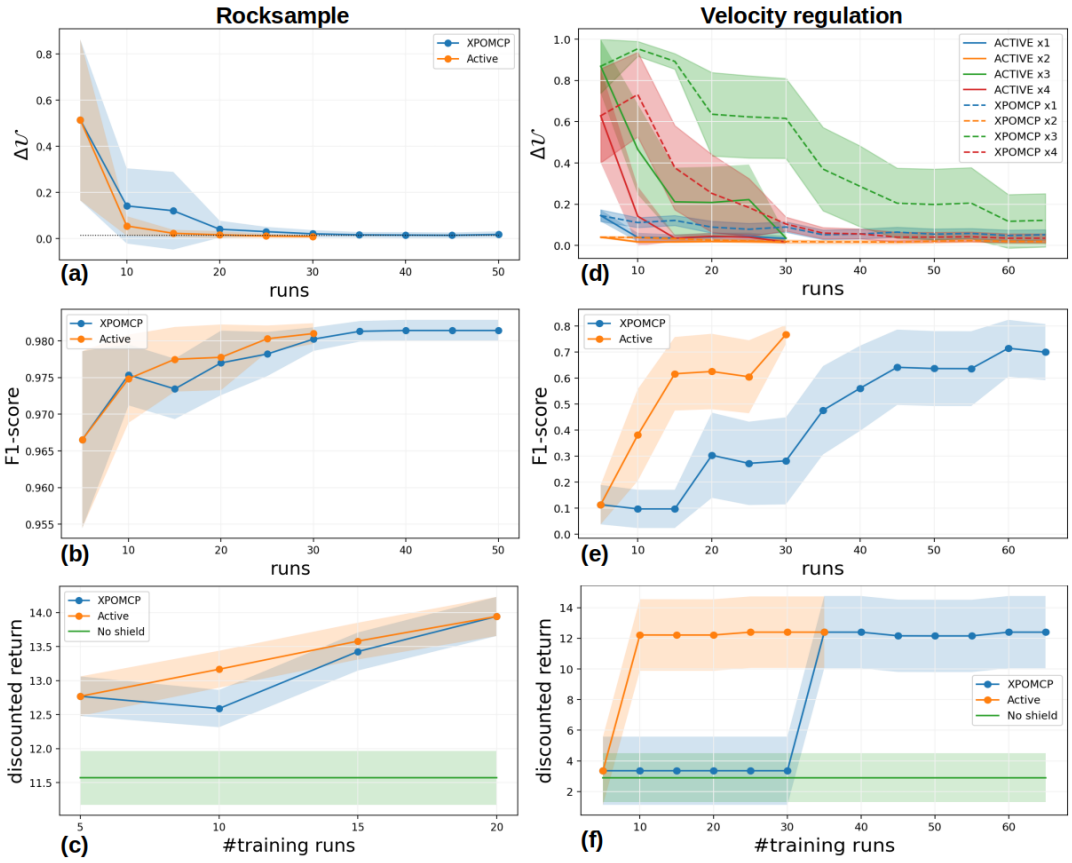


FIGURE 7.2: Results on rocksample *a.* mean uncertainty interval $\Delta\mathcal{U}$; *b.* mean F_1 score; *c.* average discounted return (with and without shielding). Results on velocity regulation *d.* mean uncertainty interval $\delta\mathcal{U}$; *e.* mean F_1 score; *f.* average discounted return (with and without shielding).

methods, and it is the same also for 20 runs because, in this example, XPOMCP gets a good rule with this number of runs. Interestingly, the performance of all the shielded POMCP (using XPOMCP and *Active XPOMCP*) are better than that of the standard POMCP. This points out the importance of the shields to improve the performance of POMCP.

7.3.2 Active rule synthesis in velocity regulation

We consider a rule that describes when the robot should move at high speed. As in rocksample, we describe the most dangerous action (i.e., the one that incurs the highest risk of collision). We expect that the robot should move fast only when it is reasonably certain that the current segment is easy to navigate, and we use the same rule templates as in Chapter 5.

performance measure

As in the case of rocksample, we perform two experiments to compare the performance of *Active XPOMCP* with that of XPOMCP. In the *first* experiment, we analyze how the uncertainty interval $\Delta\mathcal{U}$ depends on the number of runs, using *Active XPOMCP* and XPOMCP. In the *second* experiment, we measure the shielding performance of the rules generated with the two methodologies. Results show again that *Active XPOMCP* generates more accurate rules using fewer runs. In the first experiment, we consider the rules generated using $N = 5, 10, \ldots, 65$ runs in the trace. For each value of N we compute a logical rule using XPOMCP and another rule using *Active XPOMCP*. In the second case, the algorithm starts from $N = 5$, and it converges when the uncertainty interval of each variable is lower than a threshold $\epsilon = 0.02$. The test is repeated ten times, changing the seed at each test. The measures we analyze are the size of the uncertainty interval and the F_1 score. In the second experiment, we evaluate the shielding performance of the rules generated using the two methodologies. As in the first experiment, we consider traces with $N = 5, 10, \ldots 65$ runs, and we build the shields using the methodology presented in chapter 6. We compute the rules using 2^{15} simulations. For each N, we use the rules to shield an instance of POMCP, which uses fewer simulations, namely, $NS_{shield} = 2^{13}$.

Results on the first experiment

Figure 7.2.d presents the size of uncertainty interval $\Delta\mathcal{U}$ of each variable x_1, x_2, x_3, x_4, separately. The interval sizes decrease both for rules computed by XPOMCP and for rules computed by *Active XPOMCP*, but the decrease of the rules computed by *Active XPOMCP* is significantly faster. In particular, the difference is large for variable x_3 (green lines) and x_4 (red lines). This is important because a poor instantiation of these variables leads to poor shielding performance. As shown in Figure 7.2.e, the F_1 score is heavily impacted by the change in $\Delta\mathcal{U}$. XPOMCP requires 60 or more runs, on average, to achieve performances comparable to that reached by *Active XPOMCP* with ten runs. The rules generated by *Active XPOMCP* reach a maximum F_1 score of 0.79, which is higher than the F_1 score reached by rules generated by XPOMCP (i.e., 0.71) but lower than the maximum value 1.0. The rule generates some false negatives (i.e., the rule predicts that the robot should move fast, but the agent selects a lower speed). A more complex template can improve the F_1 score, but it is unlikely that this will improve the shielding performance of the rule because the shield already blocks dangerous decisions.

Results on the second experiment

Finally, we compare the shielding performance. Figure 7.2.f shows the average discounted return of the shielded POMCP using the rules generated by *Active XPOMCP*

(orange line) and XPOMCP (blue line). *Active XPOMCP* achieves the maximum performance using only ten runs, while XPOMCP reaches this value only after 35 runs. Between 10 and 30 runs, *Active XPOMCP*achieves a relative performance increase of up to 264% compared to XPOMCP. This is because the last part of the rule (i.e., $p(0) \geq x_3 \wedge p(1) \geq x_4$) is hard to tune. This condition contains two free-variable and describes some rare but important beliefs that must be properly characterized in the rule to describe the agent's behaviour. The passive approach struggles to collect relevant beliefs for this part of the rule, as shown in the red and green lines of Figure 7.2.d, while *Active XPOMCP* collects them after 5 or 10 runs.

Chapter 8

Conclusion and Future Work

This thesis presents a methodology that combines high-level indications provided by a human expert with a set of execution traces generated by a POMCP agent. We exploit such rules offline by detecting unexpected decisions in the traces and online by shielding actions related to unexpected decisions of the policy. We show that our methodology outperforms a state-of-the-art anomaly detection algorithm in detecting anomalous decisions. The shielding mechanism also improves the performance of POMCP policies in the three application domains we considered. Two are standard benchmarks, and one is a realistic scenario of mobile robot navigation. We also present an extension of the base methodology that actively uses the POMCP policy to generate relevant data to increase the beliefs that the rule can describe correctly. We show that this approach requires significantly less data than the passive approach to generate meaningful rules. This chapter presents some final considerations on the methodologies developed during this PhD and discusses possible future research directions.

8.1 Explainability for online planning algorithms

Explainability in AI is a novel and challenging topic. Even if many methodologies presented themselves as *general* explainability methodology (i.e., explanations that work independently of algorithms and domains), there is currently no consensus on how to build a *general* explanation for the behaviour of a system. A common approach, also employed in the proposed methodologies, is to present a simplified version of the systems' behaviour or map it toward a formalism that is already considered easy to understand. In our case, we are mapping a POMCP policy into compact logical formulas that capture the important aspects of the decision procedure. The procedure to automatically generate these formulas uses a template that captures high-level intuitions. Thus explanations are always framed in an intuitive context for the human interacting with the system. However, this is not enough to achieve explainability. This is why we also present an analysis of the unexpected decisions, which are the decisions in which this approximated representation fails to properly describe the reality of the policy. Thus, this thesis's important "take-home message"

regarding explainability is that it is not enough to provide an abstract representation to a human expert. However, it is crucial to point out when this abstraction *is not* a good abstraction. Our experiments consider POMCP instances that contain errors against "good" rules that should not generate unexpected decisions. Thus, the anomalies are always bugs in the policy. We decided to follow this line of work because it provides measurable metrics that can be used to assess the quality of the methodology. However, this is not always the case. It could be possible that POMCP develops complex strategies in solving problems that come out as a complete surprise to the expert, not because they are wrong, but because they are different (and possibly better) than the strategies envisioned by the expert. In this case, analysing the unexpected decision is crucial, and it should point out the differences between reality and expectations to make the system more predictable. This is, however, hard to measure, and we are actively trying to identify interesting scenarios that will let us experiment on this aspect of the proposed approach.

An important design choice we followed in our line of work is focusing on the POMCP algorithm. The basic XPOMCP methodology can be used on any POMDP policy, assuming it generates a trace of belief-action pairs. However, it is important to notice that our approach is particularly relevant for online methodologies because they must rely on traces since they do not build complete representations of the policy. A possible direction is to extend the basic XPOMCP towards other online approaches, e.g. the DESPOT algorithm (Ye et al. 2017). Another possible direction is to focus future studies on increasing the interaction between the rule synthesis procedure and the POMCP algorithm. The first result achieved in this line of work is the active approach presented in chapter 7. This approach improves performance over the basic approach that relies on traces, thanks to integrating the POMCP algorithm and the rule generation procedure. The next step that we plan to follow in this direction is to improve this integration further, to provide formal guarantees on the completeness of this representation. Right now, the active approach improves the rule synthesis procedure by leading POMCP towards unexplored beliefs that provide new useful information. It stops when the uncertainty in the rules (i.e., the difference between strict and loose rules) is lower than a small threshold. However, this does not guarantee that the policy could present unexpected behaviours in unrelated scenarios not described by the rule. In theory, it is possible to overcome this limitation by using a complete and correct implementation of the observation function of the POMDP model. The POMCP algorithm does not require this function because it only relies on a black-box simulator. However, having the observation function as an additional requirement is not uncommon, as it is used in DESPOT and some POMCP extensions (for example, in the extension of POMCP to handle $\rho-$POMDPs, as presented in (V. Thomas, Hutin, and Buffet 2020)). With this extra element, it is possible to use automated reasoning techniques to identify important unexplored beliefs and thus prove the approach's completeness (while still respecting the online nature of POMCP).

8.2 Extending XPOMCP to different logics

The main advantage of using a logic-based formalism is that rules and rule templates are both formal and easy to read for a human expert. First-order logic is widely used to express understandable but formal requirements. By restricting the general case of first-order logic to SMT, it is possible to achieve higher performance while maintaining a rich language to express high-level ideas. In the proposed methodologies, we always use SMT as the formalism to encode the indications of the expert, and our experiments show that a state-of-the-art SMT-solver can handle meaningful POMCP instances. However, we aim to improve the expressiveness of the logical formulas used in these rules by employing temporals and non-monotonic logics. Different kinds of logic can further improve the benefit of a logic-based formulation. For example, extending this approach to temporal logic could make it easier to express requirements and expectations on future events of a run. For instance, it is harder to formulate questions such as "The robot should arrive at the destination before a certain time frame; thus, it should move fast when it can" because this template has requirements on future beliefs (i.e., the confidence of reaching the end of a run in a certain timeframe), not only in the current one. This extended formalism could improve the readability of the rules and make it possible to express more complex safety constraints on the expected behaviour. Specifically, formal requirements for AI systems are often expressed as temporal logic requirements. In this formulation, the requirements specify a set of forbidden states that must be avoided and a reachability requirement (i.e., a guarantee that the system will reach a specific final state). Supporting these specifications as requirements for a shield could extend the usefulness of XPOMCP. However, the extension of the MAX-SMT-based methodology toward temporal logic is not simple since this logic raises different theoretical and computational challenges. In particular, it is crucial to encode the rule synthesis as MAX-SMT and not simply SMT to allow high-level rules that admit some small errors, but the extension of this approach to other kinds of logic is not straightforward. Specifically, it is important to encode high-level requirements as *soft* clauses in MAX-SMT. We plan to investigate how temporal requirements could be handled similarly to preserve flexibility in composing XPOMCP templates. However, the extension is non-trivial because it is not clear how to handle this kind of uncertainty on temporal operators (e.g., *finally* and *until*).

8.3 Rule-based safety for POMCP agents

In this thesis, we present a methodology that uses the rules generated by XPOMCP as a basis to build a shielding mechanism that prevents unwanted action on a POMCP agent. This is done by expressing meaningful high-level requirements that are fine-tuned by the rule synthesis method. Shielding approaches are popular because they are lightweight mechanisms that work online alongside the system after building

them offline from formal specifications. Thus, in general, they scale well to large instances. In the proposed methodology, XPOMCP builds a rule (from a set of traces, or actively as shown in Chapter 7) that is then embedded in the shield, and the rule is used during the execution. An interesting research direction, explored in (Pranger et al. 2021), is to refine this shield during the agent's execution. This can be achieved by leveraging the effectiveness of the active approach, but with the aim of safety, not completeness, i.e., without pushing POMCP towards unexplored beliefs, but by optimising the need of recomputing the parameters of the shield. Specifically, an improved active methodology should balance the cost of rebuilding the shield while acting in the environment. This can make the shield more robust in handling unexplored situations during the rule synthesis step.

Recent results (Junges, Jansen, and Seshia 2021) show that it is possible to prove that a POMDP system verifies a safety requirement formally. However, this approach cannot provide formal guarantees in certain scenarios. Specifically, if two beliefs are identical, but one could lead to a forbidden future state, and one is safe, it is impossible to provide formal guarantees on the system. In this context, it is useful to shift the focus from *risk-free* systems to *risk aware* systems. Using the rules generated by XPOMCP makes it possible to characterise the risk involved in selecting a certain action properly. This could be the basis for building shields that help overcome these limitations.

8.4 Multi agent XPOMCP

All the methodologies proposed in this thesis work for a single agent that takes action in a partially observable environment. However, it is common to consider systems involving multiple agents that act together in the same environment. While POMCP is meant to consider only single agents, there are interesting extensions of POMCP to multi-agent systems (Amato and Oliehoek 2015). The methodology relies on a factorised value function representation to avoid the exponential explosion in actions and observations typical of multi-agent systems. We plan to extend XPOMCP to multi-agent agent POMCP. To achieve this goal, it is important to build rules that can describe the behaviour of each agent independently and incorporate features that regulate the interplay between the agents. In particular, a successful explanation should properly describe the interaction between different agents because this is usually the most crucial aspect in these domains. This line of work should emphasise the description of these interactions. However, this is a challenging extension because XPOMCP is meant to explain traces generated by a single agent, in which each one of the belief-action pairs is a direct consequence of the POMCP policy. Multi-agent XPOMCP should identify the elements in the traces that characterise the interaction between robots and use them to build a rule-based description of the agents and their interactions. This improved description could prove useful in discovering bugs, as in the anomaly detection procedure described in Chapter 5.

We also plan to employ the rule-based shielding mechanism to improve the performance in the multi-agent problem by regulating the interactions between agents.

8.5 Application of XPOMCP

Finally, an important research direction is to extend XPOMCP to new application domains. A particularly interesting domain is co-operative robotics (Pellegrinelli et al. 2017), i.e. domains in which a robot and a human should act together to achieve their goals. The agent's behaviour should be understandable and predictable for a human to trust a robot. An approach similar to XPOMCP could prove to be effective in these cases since it can be used both to improve the understanding of the robot's behaviour and shield unwanted actions in future interactions. These extensions present many exciting challenges because several aspects of the rule synthesis steps, currently executed offline, should work online and act reactively to human requests.

Another important topic is to assess the impact of explainability in AI systems. For these experiments, the results should be evaluated using user studies. In general, the goal should not measure "how explainable" a rule is but prove that an explanation system could improve the performance of a system. For instance, consider a problem in which a human cooperates with a robotic arm guided by a well-performing, but complex, POMCP algorithm. A good experiment should prove that the results achieved by a human are higher when the robot is equipped with a good explanation system, i.e., a system that can justify the decisions taken by the arm. However, these experiments are complex because they require transversal and multi-disciplinary skills, e.g., robotics and psychology. In this thesis, we focused our effort on the computational aspect of XPOMCP. We are looking to collaborate with researchers in different areas of expertise to extend the usage of the proposed methodologies.

8.6 Conclusion

In conclusion, achieving explainability and safety in complex AI systems is clearly an important challenge that must be addressed to further the adoption of AI in real-life scenarios. We believe that this thesis provides an important contribution in this direction. The provided methodologies also serve as the first step in several significant research lines.

Bibliography

Acampora, Giovanni et al. (2017). "IEEE 1849: The XES Standard: The Second IEEE Standard Sponsored by IEEE Computational Intelligence Society [Society Briefs]". In: *IEEE Computational Intelligence Magazine* 12.2, pp. 4–8.

Alshiekh, Mohammed et al. (2018). "Safe Reinforcement Learning via Shielding". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 2669–2678.

Amato, Christopher and Frans A. Oliehoek (2015). "Scalable Planning and Learning for Multiagent POMDPs". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, pp. 1995–2002.

Anjomshoae, Sule et al. (2019). "Explainable Agents and Robots: Results from a Systematic Literature Review". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. Ed. by Edith Elkind et al. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1078–1088.

Bargiacchi, Eugenio, Diederik M. Roijers, and Ann Nowé (2020). "AI-Toolbox: A C++ library for Reinforcement Learning and Planning (with Python Bindings)". In: *Journal of Machine Learning Research* 21, 102:1–102:12.

Barrett, Clark, Pascal Fontaine, and Cesare Tinelli (2016). *The Satisfiability Modulo Theories Library (SMT-LIB)*. `www.SMT-LIB.org`.

Barrett, Clark W. and Cesare Tinelli (2018). "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke et al. Springer, pp. 305–343.

Bastani, Osbert, Yewen Pu, and Armando Solar-Lezama (2018). "Verifiable Reinforcement Learning via Policy Extraction". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al., pp. 2499–2509.

Bertsekas, Dimitri P. and David A. Castañón (1999). "Rollout Algorithms for Stochastic Scheduling Problems". In: *J. Heuristics* 5.1, pp. 89–108.

Bjørner, Nikolaj, Anh-Dung Phan, and Lars Fleckenstein (2015). "vZ - An Optimizing SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015.*

Proceedings. Ed. by Christel Baier and Cesare Tinelli. Vol. 9035. Lecture Notes in Computer Science. Springer, pp. 194–199.

Bloem, Roderick et al. (2015). "Shield Synthesis: Runtime Enforcement for Reactive Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. Ed. by Christel Baier and Cesare Tinelli. Vol. 9035. Lecture Notes in Computer Science. Springer, pp. 533–548.

Brandão, Martim et al. (2021). "Towards providing explanations for robot motion planning". In: *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*. IEEE, pp. 3927–3933.

Bunel, Rudy et al. (2018). "A Unified View of Piecewise Linear Neural Network Verification". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al., pp. 4795–4804.

Burkart, Nadia and Marco F. Huber (2021). "A Survey on the Explainability of Supervised Machine Learning". In: *Journal of Artificial Intelligence Research* 70, pp. 245–317.

Cashmore, Michael, Anna Collins, et al. (2019). "Towards Explainable AI Planning as a Service". In: *CoRR* abs/1908.05059. arXiv: 1908.05059.

Cashmore, Michael, Daniele Magazzeni, and Parisa Zehtabi (2020). "Planning for Hybrid Systems via Satisfiability Modulo Theories". In: *Journal of Artificial Intelligence Research* 67, pp. 235–283.

Cassandra, Anthony R., Michael L. Littman, and Nevin Lianwen Zhang (1997). "Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes". In: *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Brown University, Providence, Rhode Island, USA, August 1-3, 1997*. Ed. by Dan Geiger and Prakash P. Shenoy. Morgan Kaufmann, pp. 54–61.

Castellini, Alberto, Georgios Chalkiadakis, and Alessandro Farinelli (2019). "Influence of State-Variable Constraints on Partially Observable Monte Carlo Planning". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. Ed. by Sarit Kraus. ijcai.org, pp. 5540–5546.

Castellini, Alberto, Enrico Marchesini, et al. (2020). "Explaining the Influence of Prior Knowledge on POMCP Policies". In: *Multi-Agent Systems and Agreement Technologies - 17th European Conference, EUMAS 2020, and 7th International Conference, AT 2020, Thessaloniki, Greece, September 14-15, 2020, Revised Selected Papers*. Ed. by Nick Bassiliades, Georgios Chalkiadakis, and Dave de Jonge. Vol. 12520. Lecture Notes in Computer Science. Springer, pp. 261–276.

Chakraborti, Tathagata et al. (2017). "Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy". In: *Proceedings of the Twenty-Sixth*

International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. Ed. by Carles Sierra. ijcai.org, pp. 156–163.

Cheng, Chih-Hong, Georg Nührenberg, and Harald Ruess (2017). "Verification of Binarized Neural Networks". In: *CoRR* abs/1710.03107. arXiv: 1710.03107.

Cook, Stephen A. (1971). "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. ACM, pp. 151–158.

Costa, Antonio Anastasio Bruto da and Pallab Dasgupta (2021). "Learning Temporal Causal Sequence Relationships from Real-Time Time-Series". In: *Journal of Artificial Intelligence Research* 70, pp. 205–243.

Coulom, Rémi (2006). "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers. Vol. 4630. Lecture Notes in Computer Science. Springer, pp. 72–83.

Davis, Martin and Hilary Putnam (1960). "A Computing Procedure for Quantification Theory". In: *ACM Journals* 7.3, pp. 201–215.

Fox, Maria, Derek Long, and Daniele Magazzeni (2017). "Explainable Planning". In: *CoRR* abs/1709.10256. arXiv: 1709.10256.

Fulton, Nathan and André Platzer (2018). "Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 6485–6492.

Ganzinger, Harald et al. (2004). "DPLL(T): Fast Decision Procedures". In: *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*. Ed. by Rajeev Alur and Doron A. Peled. Vol. 3114. Lecture Notes in Computer Science. Springer, pp. 175–188.

Gehr, Timon et al. (2018). "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation". In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, pp. 3–18.

Goldhoorn, Alex et al. (2014). "Continuous real time POMCP to find-and-follow people by a humanoid service robot". In: *14th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2014, Madrid, Spain, November 18-20, 2014*. IEEE, pp. 741–747.

Gunning, David and David W. Aha (2019). "DARPA's Explainable Artificial Intelligence (XAI) Program". In: *AI Magazine* 40.2, pp. 44–58.

Hasanbeig, Mohammadhosein, Alessandro Abate, and Daniel Kroening (2020). "Cautious Reinforcement Learning with Logical Constraints". In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*. Ed. by Amal El Fallah Seghrouchni et al. International Foundation for Autonomous Agents and Multiagent Systems, pp. 483–491.

Hellinger, Ernst (1909). "Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen." In: *Journal für die reine und angewandte Mathematik* 136, pp. 210–271.

High-Level Expert Group on AI (2019). *Ethics guidelines for trustworthy AI*. Report. European Commission. URL: `https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai`.

Huang, Xiaowei et al. (2017). "Safety Verification of Deep Neural Networks". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, pp. 3–29.

Jansen, Nils et al. (2020). "Safe Reinforcement Learning Using Probabilistic Shields (Invited Paper)". In: *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*. Ed. by Igor Konnov and Laura Kovács. Vol. 171. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:16.

Jia, Kai and Martin Rinard (2020). "Efficient Exact Verification of Binarized Neural Networks". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al.

Junges, Sebastian, Nils Jansen, Christian Dehnert, et al. (2016). "Safety-Constrained Reinforcement Learning for MDPs". In: *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. Ed. by Marsha Chechik and Jean-François Raskin. Vol. 9636. Lecture Notes in Computer Science. Springer, pp. 130–146.

Junges, Sebastian, Nils Jansen, and Sanjit A. Seshia (2021). "Enforcing Almost-Sure Reachability in POMDPs". In: *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II*. Ed. by Alexandra Silva and K. Rustan M. Leino. Vol. 12760. Lecture Notes in Computer Science. Springer, pp. 602–625.

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). "Planning and Acting in Partially Observable Stochastic Domains". In: *Artificial Intelligence* 101.1-2, pp. 99–134.

Katt, Sammie, Frans A. Oliehoek, and Christopher Amato (2017). "Learning in POMDPs with Monte Carlo Tree Search". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1819–1827.

Katz, Guy et al. (2017). "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, pp. 97–117.

Kearns, Michael J., Yishay Mansour, and Andrew Y. Ng (1999). "Approximate Planning in Large POMDPs via Reusable Trajectories". In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller. The MIT Press, pp. 1001–1007.

Klauck, Michaela et al. (2020). "Bridging the Gap Between Probabilistic Model Checking and Probabilistic Planning: Survey, Compilations, and Empirical Comparison". In: *Journal of Artificial Intelligence Research* 68, pp. 247–310.

Knighofer, Bettina et al. (2017). "Shield synthesis". In: *Formal Methods Syst. Des.* 51.2, pp. 332–361.

Kocsis, Levente and Csaba Szepesvári (2006). "Bandit Based Monte-Carlo Planning". In: *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Vol. 4212. Lecture Notes in Computer Science. Springer, pp. 282–293.

Krarup, Benjamin et al. (2021). "Contrastive Explanations of Plans Through Model Restrictions". In: *CoRR* abs/2103.15575. arXiv: 2103.15575.

Langley, Pat et al. (2017). "Explainable Agency for Intelligent Autonomous Systems". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by Satinder P. Singh and Shaul Markovitch. AAAI Press, pp. 4762–4764.

Lauri, Mikko and Risto Ritala (2016). "Planning for robotic exploration based on forward simulation". In: *Robotics and Autonomous Systems* 83, pp. 15–31.

Lee, Jongmin et al. (2018). "Monte-Carlo Tree Search for Constrained POMDPs". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al., pp. 7934–7943.

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

Maaten, Laurens van der and Geoffrey Hinton (2008). "Viualizing data using t-SNE". In: *Journal of Machine Learning Research* 9, pp. 2579–2605.

Mazzi, Giulio, Alberto Castellini, and Alessandro Farinelli (2020). "Policy Interpretation for Partially Observable Monte-Carlo Planning: a Rule-based Approach". In: *Proceedings of the 7th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2020@AI*IA2020)*. Vol. 2806. CEUR Workshop Proceedings. CEUR-WS.org, pp. 44–48.

— (2021a). "Identification of Unexpected Decisions in Partially Observable Monte Carlo Planning: A Rule-Based Approach". In: *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*. Ed. by Frank Dignum et al. ACM, pp. 889–897.

— (2021b). "Rule-based Shield Synthesis for Partially Observable Monte Carlo Planning". In: *Proceedings of the 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis hosted by the Twelfth International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2021), Padua, Italy, September 22, 2021*. Ed. by Dario Della Monica, Gian Luca Pozzato, and Enrico Scala. Vol. 2987. CEUR Workshop Proceedings. CEUR-WS.org, pp. 19–23.

— (2021c). "Rule-based Shielding for Partially Observable Monte-Carlo Planning". In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*. Ed. by Susanne Biundo et al. AAAI Press, pp. 243–251.

— (2022). "Active Generation of Logical Rules for POMCP Shielding". In: *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 1696–1698.

McAllister, Rowan et al. (2017). "Concrete Problems for Autonomous Vehicle Safety: Advantages of Bayesian Deep Learning". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by Carles Sierra. ijcai.org, pp. 4745–4753.

Moskewicz, Matthew W. et al. (2001). "Chaff: Engineering an Efficient SAT Solver". In: *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*. ACM, pp. 530–535.

Mota, Tiago and Mohan Sridharan (2021). "Answer me this: Constructing Disambiguation Queries for Explanation Generation in Robotics". In: *IEEE International Conference on Development and Learning, ICDL 2021, Beijing, China, August 23-26, 2021*. IEEE, pp. 1–8.

Mota, Tiago, Mohan Sridharan, and Ales Leonardis (2021). "Integrated Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics". In: *SN Comput. Sci.* 2.4, p. 242.

Moura, Leonardo Mendonça de and Nikolaj Bjørner (2008). "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer, pp. 337–340.

Narodytska, Nina et al. (2018). "Verifying Properties of Binarized Deep Neural Networks". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 6615–6624.

Nelson, Greg and Derek C. Oppen (1979). "Simplification by Cooperating Decision Procedures". In: *ACM Trans. Program. Lang. Syst.* 1.2, pp. 245–257.

Newaz, Abdullah Al Redwan, Swarat Chaudhuri, and Lydia E. Kavraki (2019). "Monte-Carlo Policy Synthesis in POMDPs with Quantitative and Qualitative Objectives". In: *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*. Ed. by Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson.

Norman, Gethin, David Parker, and Xueyi Zou (2017). "Verification and control of partially observable probabilistic systems". In: *Real-Time Systems* 53.3, pp. 354–402.

Papadimitriou, Christos H. and John N. Tsitsiklis (1987). "The Complexity of Markov Decision Processes". In: *Mathematics of Operations Research* 12.3, pp. 441–450.

Pedregosa, Fabian et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pellegrinelli, Stefania et al. (2017). "Motion planning and scheduling for human and industrial-robot collaboration". In: *CIRP Annals* 66.1, pp. 1–4.

Pineau, Joelle, Geoffrey J. Gordon, and Sebastian Thrun (2006). "Anytime Point-Based Approximations for Large POMDPs". In: *Journal of Artificial Intelligence Research* 27, pp. 335–380.

Pranger, Stefan et al. (2021). "Adaptive Shielding under Uncertainty". In: *2021 American Control Conference, ACC 2021, New Orleans, LA, USA, May 25-28, 2021*. IEEE, pp. 3467–3474.

Ross, Stéphane et al. (2008). "Online Planning Algorithms for POMDPs". In: *Journal of Artificial Intelligence Research* 32, pp. 663–704.

Russell, Stuart J. and Peter Norvig (2020). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

Seshia, Sanjit A. and Dorsa Sadigh (2016). "Towards Verified Artificial Intelligence". In: *CoRR* abs/1606.08514. arXiv: 1606.08514.

Silver, David, Aja Huang, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Silver, David, Thomas Hubert, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144.

Silver, David, Julian Schrittwieser, et al. (2017). "Mastering the game of Go without human knowledge". In: *Nature* 550.7676, pp. 354–359.

Silver, David and Joel Veness (2010). "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*. Ed. by John D. Lafferty et al. Curran Associates, Inc., pp. 2164–2172.

Smallwood, Richard D. and Edward J. Sondik (1973). "The Optimal Control of Partially Observable Markov Processes over a Finite Horizon". In: *Oper. Res.* 21.5, pp. 1071–1088.

Smith, Trey and Reid G. Simmons (2004). "Heuristic Search Value Iteration for POMDPs". In: *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, Banff, Canada, July 7-11, 2004*. Ed. by David Maxwell Chickering and Joseph Y. Halpern. AUAI Press, pp. 520–527.

Sridharan, Mohan and Tiago Mota (2020). "Commonsense Reasoning to Guide Deep Learning for Scene Understanding (Extended Abstract)". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. Ed. by Christian Bessiere. ijcai.org, pp. 4760–4764.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press.

Thomas, Philip S. et al. (2019). "Preventing Undesirable Behavior of Intelligent Machines". In: *Science* 366.6468, pp. 999–1004.

Thomas, Vincent, Gérémy Hutin, and Olivier Buffet (2020). "Monte Carlo Information-Oriented Planning". In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by Giuseppe De Giacomo et al. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 2378–2385.

Verma, Abhinav et al. (2018). "Programmatically Interpretable Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 5052–5061.

Wang, Yiming et al. (2020). "POMP: Pomcp-based Online Motion Planning for active visual search in indoor environments". In: *31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020*. BMVA Press.

Wang, Yue, Swarat Chaudhuri, and Lydia E. Kavraki (2018). "Bounded Policy Synthesis for POMDPs with Safe-Reachability Objectives". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*. Ed. by Elisabeth André et al. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, pp. 238–246.

Ye, Nan et al. (2017). "DESPOT: Online POMDP Planning with Regularization". In: *Journal of Artificial Intelligence Research* 58, pp. 231–266.

Zakrzewski, R. R. (2001). "Verification of a trained neural network accuracy". In: *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*. Vol. 3, pp. 1657–1662.

Zhang, Yu et al. (2017). "Plan explicability and predictability for robot task planning". In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. IEEE, pp. 1313–1320.

Zhu, He et al. (2019). "An Inductive Synthesis Framework for Verifiable Reinforcement Learning". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*. Ed. by Kathryn S. McKinley and Kathleen Fisher. ACM, pp. 686–701.