Identification of Unexpected Decisions in Partially Observable Monte-Carlo Planning: A Rule-Based Approach

Giulio Mazzi
Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy
giulio.mazzi@univr.it

Alberto Castellini
Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy
alberto.castellini@univr.it

Alessandro Farinelli
Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy
alessandro.farinelli@univr.it

ABSTRACT

Partially Observable Monte-Carlo Planning (POMCP) is a powerful online algorithm able to generate approximate policies for large Partially Observable Markov Decision Processes. The online nature of this method supports scalability by avoiding complete policy representation. The lack of an explicit representation however hinders interpretability. In this work, we propose a methodology based on Satisfiability Modulo Theory (SMT) for analyzing POMCP policies by inspecting their traces, namely sequences of belief-action-observation triplets generated by the algorithm. The proposed method explores local properties of policy behavior to identify unexpected decisions. We propose an iterative process of trace analysis consisting of three main steps, i) the definition of a *question* by means of a parametric logical formula describing (probabilistic) relationships between beliefs and actions, ii) the generation of an *answer* by computing the parameters of the logical formula that maximize the number of satisfied clauses (solving a MAX-SMT problem), iii) the analysis of the generated logical formula and the related decision boundaries for identifying unexpected decisions made by POMCP with respect to the original question. We evaluate our approach on Tiger, a standard benchmark for POMDPs, and a real-world problem related to mobile robot navigation. Results show that the approach can exploit human knowledge on the domain, outperforming state-of-the-art anomaly detection methods in identifying unexpected decisions. An improvement of the Area Under Curve up to 47% has been achieved in our tests.

KEYWORDS

POMCP; MAX-SMT; anomaly detection; explainable planning

ACM Reference Format:
Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli . 2021. Identification of Unexpected Decisions in Partially Observable Monte-Carlo Planning: A Rule-Based Approach. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

Planning in a partially observable environment is an important problem in artificial intelligence and robotics. A popular framework to model such problem is *Partially Observable Markov Decision Processes (POMDPs)* [8] which encode dynamic systems where the state is not directly observable but must be inferred from observations. Computing optimal policies, namely functions that map beliefs (i.e., probability distributions over states) to actions, in this context is PSPACE-complete [24]. However, recent approximate and online methods allow handling many real-world problems. A pioneering algorithm for this purpose is *Partially Observable Monte-Carlo Planning (POMCP)* [26] which uses a particle filter to represent the belief and a Monte-Carlo Tree Search based strategy to compute the policy online. Recently, some techniques have been proposed to introduce prior knowledge in this algorithm [9, 10]. The local representation of the policy made by this algorithm however hinders the interpretation and explanation of the policy itself.

Explainability [14] is becoming a key feature of artificial intelligence systems since in several contexts humans need to understand why specific decisions are taken by the agent. Specifically, explainable planning (XAIP) [6, 13] focuses on explainability in planning methods. The presence of erroneous behaviors in these tools (due, for instance, to the wrong setup of internal parameters) may have a strong impact on autonomous cyber-physical and robotic systems that interact with humans, and detecting these errors in automatically generated policies is very hard in practice. For this reason, improving policy explanability is fundamental.

In this work, we propose a methodology for interpreting POMCP policies and detecting their unexpected decisions. Using this approach, experts provide qualitative information on system behaviors (e.g., "the robot should move fast if it is highly confident that the path is not cluttered") and the proposed methodology supplies quantitative details of these statements based on evidence observed in traces (e.g., the approach says that the robot usually moves fast if the probability to be in a cluttered segment is lower than 1%).

Experts are however also interested in identifying states in which the planner does not respect their assumptions. A possible question, in this case, is "Is there a state in which the robot moves at high speed even if it is likely that the environment is cluttered?". To answer this kind of question, our approach allows expressing partially defined assumptions employing logical formulas, called *rule templates*. Parameters of rule templates are then computed from traces by a Satisfiability Modulo Theory (SMT) solver.

The approach we propose formalizes the parameter computation task as a *MAX-SAT* problem which allows to express complex logical formulas and to compute optimal assignments when the template is not fully satisfiable (which happens in the majority of cases in real policy analysis). A second key feature of the approach concerns the identification of decisions that violate trained rules. Decision boundaries over beliefs generated by the proposed method have good interpretability and can be iteratively improved by focusing

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

on specific questions that arise in the analysis of the policy. This iterative process allows to locally explore policy behaviors.

Finally, since the proposed method quantifies the divergence between rule decision boundaries and decisions that do not satisfy the rules, it also identifies decisions that violate expert assumptions. To empirically evaluate this feature, we inject some errors into POMCP by wrongly setting one of its parameters, and we show that our methodology can outperform standard anomaly detection methods identifying policy decisions that violate expert assumptions. This performance improvement is achieved by exploiting the capability of the proposed method to include prior knowledge of the system under investigation.

The contribution of this paper to the state-of-the-art is threefold:

- We propose a novel approach based on SMT for analyzing properties of POMCP traces by means of logical rules specified by human experts, variables are then instantiated by a MAX-SMT solver.
- We propose a method for identifying unexpected decisions using logical rules learned by the MAX-SMT solver.
- We empirically evaluate the performance of the proposed method on two case studies, namely, the Tiger benchmark domain and a problem of velocity regulation for mobile robots.

2 RELATED WORKS

We have identified two main research topics with relationships with our method and goals, namely, policy verification and explainable planning. Formal logic is strongly employed in verification of machine learning and reinforcement learning algorithms.

Policy verification. In recent years, SMT-based approaches have been developed to verify the safety of neural networks [5, 16, 18]. These methodologies encode the neural network into SMT formulas and check if safety properties hold on these formulas, or they provide counterexamples for properties that are not satisfied. To the best of our knowledge, there is no equivalent approach to verify that a specific property holds on policies generated by POMDPs, and in particular by POMCP. A possibility to use SMT-based approaches to verify POMCP policies is to encode the POMDP problem in one of the logic-based frameworks presented in [3, 7, 23, 28], where property guarantees can be formally proved. However, these frameworks use SMT-solvers to build a policy that satisfies predefined properties while we use a MAX-SMT representation of the problem with a different goal, namely to evaluate if the policy satisfies expert assumptions. Namely, we aim at enhancing policy explainability without altering the policy itself.

A work in which verification is achieved by exploiting a simplified representation of the problem provided by an expert is [29]. It describes a method for verifying properties related to the safety of fully observable systems modeled by Markov Decision Processes (MDPs). The approach works on a pre-trained neural network representing a black-box policy. It uses a linear formula summarizing the policy behavior to allow using off-the-shelf verification tools. This differs from our work for two reasons: first, we work on partially observable environments, and our logical formulas work on beliefs instead of states; second, in [29] the formula is used as an input to the verification tool while we use it to interact with humans and improve policy explainability.

Explainable planning. Explainable Artificial Intelligence (XAI) [14] is a rapidly growing research field focusing on human interpretability and understanding of artificial intelligence (AI) systems. In particular Explainable planning (XAIP) [1, 6, 13, 20] aims at investigating planning tools that come with justifications for the decisions they make. In our work, we use the high-level insight provided by the user to build an explanation of the policy in use. A particularly interesting kind of questions analyzed in XAIP are known as *contrastive question* [13]. They are used to structure the interaction between humans and the AI systems to be explained. In these questions, the expert asks the agent question as "Why have you made this decision instead of this other one, that I believe could be a better option?"and the system answers motivating its choice instead of the alternative one. These questions are, however, very difficult to answer in online frameworks as POMCP [11] because the information required to build the answer may not be available to the agent at run time. We, therefore, do not use contrastive questions but ground the interaction between human and planner on logical formulas that are framed by the expert, using her/his insight, and then instantiated by the SMT solver according to the observed behavior of the system. The identification of decisions that violate user's expectation allows then to generate an iterative process in which the expert, that can refine the rule interactively, acquires new understanding about the policy. In [22] a preliminary study on the use of MAX-SMT for explaining POMCP policies was proposed.

3 METHODS

We provide full description of the proposed method using a running example to show direct application of the main concepts.

3.1 Method overview

The methodology proposed in this work, called *XPOMCP* in the following, is summarized in Figure 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 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 to be 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). For instance, a template saying "Do this when the probability of avoiding collisions is at least $\overline{\mathbf{x}}$", with $\overline{\mathbf{x}}$ free variable, is transformed into "Do this when the probability of avoiding collisions is at least 0.85". By defining a rule template the expert provides useful prior knowledge about the structure of the investigated property. Hence, the rule template defines the *question* asked by the expert. The *answer* to this question is provided by the SMT solver (see box 3), which computes optimal values for the free variables in order to make the formula 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, and it allows to split trace steps into two classes, namely, 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 assumption). The quantification of the violation, i.e., the distance between the rule boundary and the violation, also supports the analysis because it provides an explicit way to explain the violations themselves, which could even be completely unexpected due to expert imprecise knowledge or policy errors.

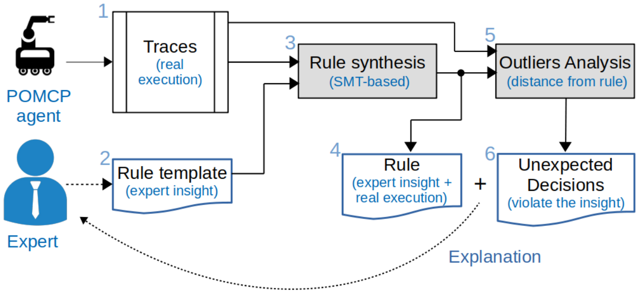
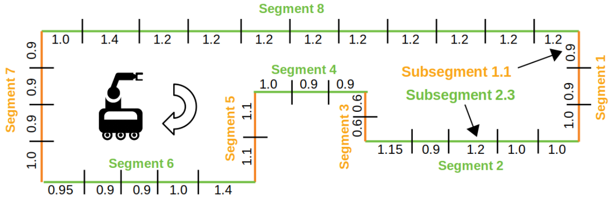


Figure 1: Methodology overview.

3.2 Running example: velocity regulation in mobile navigation

We present a problem of velocity regulation in robotic platforms as a case study to show how XPOMCP works. The same problem is used also in Section 4 to evaluate the performance of our method. A robot travels on a pre-specified path divided into eight *segments* which are in turn divided into *subsegments* of different sizes, as shown in Figure 2. 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 states. The goal of the robot 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 speed of the agent, 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 2.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 2.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).

We are interested in a rule describing when the robot travels at maximum speed (i.e., $a = 2$). We expect that the robot should move at that speed only if it is confident enough to be in an easy-to-navigate segment, but 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 in comparison to the others). To obtain a rule that is compact but informative, we want the rule to be a local approximation of the behavior of the robot, thus we only focus on the current segment without considering the path as a whole when we write this rule. The task of the proposed method is to find the actual bounds on the probability distribution (i.e., belief) that the POMCP algorithm uses to make its decisions and to highlight the (unexpected) decisions that do not comply with this representation.



(a)

f	$p(o = 1 \mid f)$
0	0.0
1	0.5
2	1.0

(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 2: 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 Partially Observable Monte-Carlo Planning

A Partially Observable Markov Decision Process (POMDP) [17] is 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*, $T\colon S \times A \to \Pi(S)$ is the *state-transition model*, with $\Pi(S)$ probability distribution over states, $O\colon S \times A \to \Pi(Z)$ is the *observation model*, $R\colon S \times A \to \mathbb{R}$ is the *reward function* and $\gamma \in [0, 1]$ is a *discount factor*. An agent must maximize the *discounted return* $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. A probability distribution over states, called *belief*, is used to represent the partial observability of the true state. To solve a POMDP it is required to find a *policy*, namely a function $\pi\colon B \to A$ that maps beliefs B into actions.

We use *Partially Observable Monte-Carlo Planning (POMCP)* [26] to solve POMDPs. POMCP is an *online* algorithm that solves POMDPs by using Monte-Carlo techniques. The strength 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 to simulate the environment. POMCP uses a *Monte-Carlo Tree Search* (MCTS) at each time-step to explore the belief space and select the best action. *Upper Confidence Bound for Trees (UCT)* [19] is used as a search strategy to select the subtrees to explore and balance exploration and exploitation. The belief is implemented as a *particle filter*, which is a sampling over the possible states that is updated at every step. At each time-step a particle is selected from the filter, each particle represents a state. This state is used as an initial point to perform a simulation in the Monte-Carlo tree. Each simulation is a sequence of action-observation pairs and it collects a discounted return, and for each action, we can compute the expected reward that can be achieved. The particle filter is updated after receiving an observation. If required, new particles can be generated from the current state through a process of *particle reinvigoration*.

In the following, we call *trace* a set of runs performed by POMCP on a specific problem. Each run is a set of *steps*, and each step corresponds to an action performed by the agent having a belief and receiving an observation from the environment. In the velocity regulation problem, we use 100 runs per trace.

3.4 SMT and MAX-SMT

The problem of reasoning on the satisfiability of formulas involving propositional logic and first-order theories is called *Satisfiability Modulo Theory* (SMT). In XPOMCP, we use propositional logic and the theory of linear real arithmetic to encode the rules that describe the behavior of policies, and we use Z3 [12] to solve the SMT problem. We encode our formulas as a MAX-SMT problem, which has two kinds of clauses, namely, *hard*, that must be satisfied, and *soft* 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 when hard clauses are unsatisfiable. Our rules are intended to describe as many decisions as possible among those taken by the policy hence MAX-SMT provides a perfect formalism to encode this requirement. The Z3 solver is used to solve the MAX-SMT problem [4]. Subsection 3.5 presents the details of this encoding.

The key ingredient for the MAX-SMT formulation are *rules* and *rule templates*. A rule template represents the *question* the expert wants to investigate. It is a set of first-order logic formulas without quantifiers explaining some properties of the policy, and has the following form:

$$\begin{aligned} &r_1 : \texttt{select } a_1 \texttt{ when } (\textstyle\bigvee_{i^1} \texttt{subformula}_{i^1});\\ &\texttt{...}\\ &r_n : \texttt{select } a_n \texttt{ when } (\textstyle\bigvee_{i^n} \texttt{subformula}_{i^n});\\ &[\texttt{ where } \textstyle\bigwedge_j \ (\texttt{requirements}_j);\ ] \end{aligned} \tag{1}$$

where $r_1, \dots, r_n$ are *action rule templates*. A *subformula* is defined as $\bigwedge_k p_s \approx \overline{\mathbf{x}}_k$, where p_s is the probability of state s, symbol $\approx \in \{<, >, \geq, \leq\}$, and $\overline{\mathbf{x}}_k$ is a free variable that is automatically instantiated by the SMT solver analyzing the traces (when the problem is satisfiable). In general, bold letters with an overline (e.g., $\overline{\mathbf{x}}, \overline{\mathbf{y}}$) are used to identify free variable while italic letters (e.g., p, a_i) are used for fixed values read from the trace. The `where` statement can be used to specify an optional set of hard requirements that can take different forms, such a the definition of a minimum value (e.g., $\overline{\mathbf{x}}_0 \geq 0.9$) or a relation (e.g., $\overline{\mathbf{x}}_2 = \overline{\mathbf{x}}_3$). These are used to define prior knowledge on the domain which is used by the rule synthesis algorithm 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 instance, in our running example the actions a_0, a_1 and a_2 represent speeds (i.e., low, medium, high). Each step t contains the partially observable state ($segment^t$, $subsegment^t$, $difficulty^t$) and the selected action a^t. Since *difficulty* is a probability distribution on $3^8 = 6561$ states we do not use this value directly. For the sake of brevity, we introduce the `diff` function which takes a distribution on the possible difficulties `distr`, a segment `seg`, and a required difficulty value `d` as input and returns the probability that segment `s` has difficulty `d` in the distribution `distr`. We can now write the rule template:

$$\begin{aligned} &r_2 : \texttt{select } a_2 \texttt{ when } p_0 \geq \overline{\mathbf{x}}_1 \lor p_2 \leq \overline{\mathbf{x}}_2;\\ &\texttt{where } \overline{\mathbf{x}}_1 \geq 0.9 \ \land \ p_0 = \texttt{diff(distr, seg, 0)} \land\\ &\qquad p_2 = \texttt{diff(distr, seg, 2)} \end{aligned} \tag{2}$$

The first literal of r_2 specifies that we select action a_2 if the probability to be in a segment with low difficulty is greater than a certain threshold, where with $\overline{\mathbf{x}}_1 \geq 0.9$ in the requirement we declare that this threshold must be at least 0.9 (an information that we expect to be true), while the second is an upper bound on the belief that the current segment is hard (i.e., $p_2 \leq \overline{\mathbf{x}}_2$). From now on, we always assume $p_d = \texttt{diff}(\textit{difficulty}, \textit{segment}, d)$ for $d \in \{0, 1, 2\}$ in the context of the velocity regulation problem. To encode Equation (2), for each step t in the trace we add the clauses:

- $p_0^t = \texttt{diff}(\textit{difficulty}^t, \textit{segment}^t, 0)$,
- $p_2^t = \texttt{diff}(\textit{difficulty}^t, \textit{segment}^t, 2)$,
- if the robot performs action a_2 (moving fast), then the formula $(p_0^t \leq \overline{\mathbf{x}}_1 \lor p_2^t \geq \overline{\mathbf{x}}_2)$ is added to the problem,
- if the robot performs a different action (i.e., a_0 or a_1) then the formula $\neg(p_0^t \leq \overline{\mathbf{x}}_1 \lor p_2^t \geq \overline{\mathbf{x}}_2)$ is added.

Finally, we add the constraints:

- $(\overline{\mathbf{x}}_1 \geq 0.0 \land \overline{\mathbf{x}}_1 \leq 1.0) \land (\overline{\mathbf{x}}_2 \geq 0.0 \land \overline{\mathbf{x}}_2 \leq 1.0)$ to ensure that $\overline{\mathbf{x}}_1$ and $\overline{\mathbf{x}}_2$ are probabilities,
- $\overline{\mathbf{x}}_1 \geq 0.9$ to force the hard constraint.

A *learned rule* is a rule template with all free variables instantiated (e.g., $\overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2$). For a rule to properly describe a trace generated by a policy, all steps in the trace should satisfy the rule (i.e., the action defined in the rule should be taken in a step iff the belief satisfies the rule conditions). This is however almost impossible in real traces because the policy is usually a complex formula. For this reason, we implemented a soft mechanism to check clause satisfiability, as described in subsection 3.5. In our example, from rule template (2) and a trace we obtain the learned rule:

$$r_2 : \texttt{select } a_2 \texttt{ when } p_0 \geq 0.945 \lor p_2 \leq 0.07;$$

while an example of output provided by XPOMCP when a rule is violated is:

```
Violation in run 2, step 4:
- Selected action: a_2
- Belief: p_0 = 0.38, p_1 = 0.31, p_2 = 0.31.
```

3.5 Rule Synthesis

Rule synthesis is performed by Algorithm 1 that takes as input a trace *ex* generated by POMCP and a rule template *r*. The output is the rule *r* with all free variables instantiated to satisfy as many steps of *ex* as possible. The *solver* is a Z3 instance used to find a model for the formulas.

Algorithm 1: RuleSynthesis

```
   Input: a trace generated by POMCP ex
          a rule template r
   Output: an instantiation of r
1  solver ← probability constraints for thresholds in r;
2  foreach action rule $r_a$ with $a \in A$ do
3      foreach step t in ex do
4          build new dummy literal $l_{a,t}$;
5          $cost \leftarrow cost \cup l_{a,t}$;
6          compute $p_0^t, \ldots, p_n^t$ from t.particles;
7          $r_{a,t}$ ← instantiate rule $r_a$ using $p_0^t, \ldots, p_n^t$;
8          if $t.action \neq a$ then
9              $r_{a,t} \leftarrow \neg(r_{a,t})$;
10         $solver.add(l_{a,t} \vee r_{a,t})$;
11 solver.minimize(cost);
12 $goodness \leftarrow 1 - distance\_to\_observed\_boundary$;
13 model ← solver.maximize(goodness);
14 return model
```

The *solver* is first initialized and hard constraints are added in line 1 to force all parameters in the template to satisfy the probability constraint (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 satisfying 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 essentially counts the number of fake assignments that correspond to unsatisfied clauses. Afterwards, the belief state probabilities are collected from the particle filter (line 6) and used to instatiate 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 way 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 behavior) or by assigning a true value to the literal $l_{a,t}$ (unexpected behavior). The second kind of assignment however has a cost since the dummy variables have been introduced only to allow partial satisfiability of the rules. In line 11, in fact, the solver is asked to find an assignment of free variable which minimizes the cost function, which considers 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 (lines 12–13) concerns the identification of the assignment which is closer to the behavior observed in the trace. This problem is solved by maximizing a goodness function which 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. Notice that this problem concerns the optimization of real variables and it is solved by the linear arithmetic module.

Theoretically, MAX-SMT is an NP-hard but in practice, Z3 can solve our instances in a reasonable time (as shown in Section 4). The variable in the SMT problem are the free variables specified in the template (a constant number) 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.

3.6 Identification of unexpected decisions

A key element of XPOMCP concerns the characterization of steps that fail to satisfy the rule. They can be seen as *unexpected decisions*, namely, exceptions to the general rule that the expert expects to be true. They can provide useful information for policy interpretation. We define two important classes of exceptions, namely, those related to the approximation made by the logical formula and those actually due to an unpredicted behavior (e.g., an error in the POMCP algorithm, or a decision that cannot be described with only local information). We expect exceptions in the first class to fall quite close to the rule boundary, while exceptions in the second class to be more distant from the boundary. In the following, we call the second kind of exceptions *unexpected decisions* since their behavior is unexpected compared to the expert knowledge on the policy.

In this section, we provide a procedure for differentiating the first class of exceptions from the second one, to find as many unexpected decisions as possible. The input of the procedure is a learned rule r, a set of steps (called *steps*) that violate the rule, and a threshold $\tau \in [0, 1]$. The output of the algorithm is a set of steps related to unexpected decisions. The procedure first randomly generates w samples (i.e., beliefs) $\bar{b}_j, j = 1, \ldots, w$, that satisfy the rule. Then, for each belief b_i in $steps$ a distance measure is computed between b_i and all $\bar{b}_j, j = 1, \ldots, w$. The minimum distance h_i is finally computed for each b_i and compared to a threshold τ. If $h_i \geq \tau$ then b_i is considered an outlier because its distance from the rule boundary is high.

Since beliefs are discrete probability distributions, we use a specific distance measure dealing with such kinds of elements, namely, the *discrete Hellinger distance (H^2)* [15]. This distance is defined as follow:

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

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 is very useful to define a meaningful threshold τ. In Section 4 we discuss how we set this threshold for our experiments.

4 RESULTS

This section provides experimental results on two case studies. The capability of XPOMCP to identify unexpected decisions is compared to that of *isolation forest* [21], a state-of-the-art anomaly detection algorithm, showing that our method outperforms isolation forest in terms of F1-score and accuracy.

4.1 Experimental Setting

We implemented two problems, namely *Tiger* and *velocity regulation*, as black-box simulators in the original C++ version of POMCP [26]. To generate *traces*, we collect both particle distributions and actions selected at each step. The *RuleSynthesis* algorithm (i.e., Algorithm 1) and the procedure for identifying *unexpected decisions* (see Section 3.6) have been developed in Python. The Python binding of Z3 [12] has been used to solve the SMT formulas. Experiments have been performed on a notebook with Intel Core i7-6700HQ and 16GB RAM. The code is available at `https://github.com/GiuMaz/AAMAS2021`.

Error injection. To quantify the capability of the proposed method to identify policy errors we modify the *RewardRange* parameter (called W in the following) in POMCP. This parameter defines the maximum difference between the lowest and the highest possible reward, and it is used by UCT 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 is a creeping error that randomly affects the exploration-exploitation trade-off making POMCP incorrect in some situations. We use this kind of error since parameter W must be set by hand in POMCP and it requires specific values that are not always easy to collect.

Exact solution. We use the *incremental pruning algorithm* [8] implemented in [2] to compute an exact policy for *Tiger.* This is used as a ground-truth for evaluating the performance of our method in detecting wrong actions. Unfortunately, we cannot compute the exact policy for the *velocity regulation* problem since its dimension makes the computation intractable, however, we use this case study to evaluate the applicability of our method to larger problems.

Baseline method. Isolation forest (IF) [21] 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 to be 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* [25] 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.

4.2 Results on Tiger

Tiger is a well-known problem [17] in which an agent has to chose which door to open among two doors, one hiding a treasure and the other hiding a tiger. Finding the treasure yields a reward of +10 while finding the tiger a reward of −100. The agent can also listen (by paying a small penalty of −1) to gain new information. Listening is however not accurate since there is a 0.15 probability of hearing the tiger from the wrong door. A successful policy 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. From the analysis of the observation model and reward function, it is however 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 runs (i.e., a trace) performed using POMCP. Finally, by analyzing the trained rule we understand which is the minimum confidence required by the policy to open a door. The correct value of W is 110 (the reward interval is $[-100, 10]$). For each value of W 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. Lower values of W produce a higher number of errors, as show in Table 1 (see column *% errors*).

Rule synthesis. 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:

$$\begin{aligned} &r_L : \texttt{select}\ \mathit{Listen}\ \texttt{when}\ (p_{right} \leq x_1 \wedge p_{left} \leq x_2);\\ &r_{OR} : \texttt{select}\ \mathit{Open}_R\ \texttt{when}\ p_{right} \geq x_3;\\ &r_{OL} : \texttt{select}\ \mathit{Open}_L\ \texttt{when}\ p_{left} \geq x_4;\\ &\texttt{where}\ (x_1 = x_2) \wedge (x_3 = x_4) \wedge (x_3 > 0.9); \end{aligned} \tag{3}$$

Action rule template r_L describes when the agent should listen, while templates r_{OR} and r_{OL} describe when the agent should open the right and left door, respectively. 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)* a minimum confidence is expected 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 regardless of threshold. 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. We use the *Area Under Curve* (AUC) as a performance measure. 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 $W \in \{85, 65, 40\}$. We do not evaluate the methods in the case with $W = 110$, it is error-free thus it is not possible to have any true positive (AUC and AP are 0).

In Table 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

Table 1: Quantitative performance comparison (AUC and AP) at different values of W. Best results are bold

(a) XPOMCP

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

(b) Isolation Forest

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

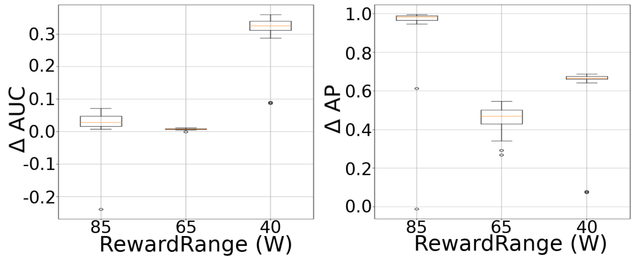


Figure 3: Box-plots of AUC and AP (considering 100 different parameters) with different values of W

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 in the case of *W=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 in the case of *W=85* because it exhibits a large number of false-positive that leads to very low precision and value of AP. Finally, in the case of *W=65*, the difference between the two methods is smaller. In this data-set, both methods achieve their best performance. IF is more effective in identifying true error compared to the case of $W = 85$, but it still generates more false-positive than XPOMCP. Figure 3 displays two boxplots that show how values $\Delta AUC = AUC_{XPOMCP} - AUC_{IF}$ and $\Delta AP = AP_{XPOMCP} - AP_{IF}$ vary in each trace. 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 in the case $W = 85$.

Performance evaluation with optimal parameters. To provide further details on the performance of XPOMCP, here we show its performance with optimal threshold τ (see Section 3.6). To compute the value of τ we performed cross-validation by training XPOMCP on 5 traces and testing it on 45 traces. F1-score about the identification of unexpected decisions was computed on the test set using 100 threshold values uniformly distributed in $[0, 0.5]$, and the test with the best F1-score was selected. The results of this test are presented in Table 2.a. Column *threshold* contains values of threshold used and columns *accuracy* and *F1* show the related performance values on the test set, and *time* shows the average elapsed time (in second). We used the same procedure to tune the *contamination* parameter of IF (Table 2.b). Figure 4 shows a comparison of 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 several true negatives are computed by both methods. However, the F1-score is very different. IF achieves a low score in this metric because it cannot identify some true positive and it generates much more false positives than XPOMCP. In general, IF is faster than XPOMCP of an order of magnitude, but the performance of our methodology is acceptable since it takes POMCP an average of 158.2s to generate a *Tiger* trace with 1000 runs and XPOMCP analyze it in less than 15s.

Table 2: Quantitative performance comparison (F1 and accuracy) using optimal thresholds. Best results are bold

(a) XPOMCP

W	Threshold	F1	Accuracy	time (s)
85	0.061	**0.979**(±0.081)	**0.999**(±0.0001)	14.30(±0.50)
65	0.064	**0.999**(±0.002)	**0.999**(±0.0001)	14.75(±0.80)
40	0.045	**0.980**(±0.072)	**0.987**(±0.049)	12.78(±0.83)

(b) Isolation Forest

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

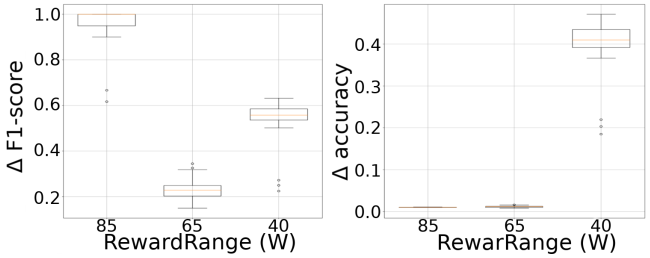


Figure 4: Box-plots of Δ F1-score and Δ accuracy using the optimal thresholds with different values of W

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 W (i.e., $W = 40$). The rule generated by the MAX-SMT solver from this trace is:

$$\begin{aligned} & r_L : \texttt{select}\ Listen\ \texttt{when}\ (p_{right} \le 0.847 \land p_{left} \le 0.847); \\ & r_{OR} : \texttt{select}\ Open_R\ \texttt{when}\ p_{right} \ge 0.966; \\ & r_{OL} : \texttt{select}\ Open_L\ \texttt{when}\ p_{left} \ge 0.966; \end{aligned} \tag{4}$$

It is a compact summary of the policy that highlights the important details in a structured way. There is a gap between the value of rule r_L (i.e., 0.8638) and that of rules r_{OR}, r_{OL} (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. An in depth analysis of this case study is performed in the Supplementary material.

4.3 Results on the velocity regulation problem

To evaluate the performance of XPOMCP on the *velocity regulation* problem, we inspect by-hand each decision that is marked as unexpected, making a detailed analysis of traces (we recall that the exact policy cannot be computed for this problem because the state space is too long). The template used to describe when the robot must move at high speed is as follow:

$$r_2 : \texttt{select action } S_2 \texttt{ when } p_0 \geq \overline{\mathbf{x}}_1 \vee p_2 \leq \overline{\mathbf{x}}_2 \vee (p_0 \geq \overline{\mathbf{x}}_3 \wedge p_1 \geq \overline{\mathbf{x}}_4)$$

$$\texttt{where } \overline{\mathbf{x}}_1 \geq 0.9$$

where $\overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, \overline{\mathbf{x}}_3, \overline{\mathbf{x}}_4$ are free variables and p_0, p_1, p_2 are defined as in Section 3.2. The first two constraints of the rule are identical to the running example, but we add a third constraint $(p_0 \geq \overline{\mathbf{x}}_3 \wedge p_1 \geq \overline{\mathbf{x}}_4)$ that combines p_0 and p_1 to describe when the robot must move at high speed. The correct value of W 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 we obtain the rule:

$$r_2 : \texttt{select action } S_2 \texttt{ when } p_0 \geq 0.910 \vee p_2 \leq 0.013 \vee (p_0 \geq 0.838 \wedge p_1 \geq 0.132)$$

It takes XPOMCP 69.53s to analyze the trace. This rule fails on 33 out of 3500 decisions, but only 4 of them are marked by XPOMCP as unexpected using threhsold $\tau = 0.1$, that we select empirically by analyzing the H^2 of unexpected decisions on *velocity regulation* traces. Table 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 # shows an identification number for the step, columns $p_0, p_1, p2$ show the belief of the step, column H^2 shows the Hellinger distance of the failed steps, and column *unexp.* shows the outcome of the classification based on 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 5–33 cannot be satisfied due to the approximate nature of the rule but do not violate the expert indications.

To visualize the result of our approach we show a *T-distributed Stochastic Neighbor Embedding (t-SNE) projection* [27] in which the belief at each step is used to compute point coordinates and the action taken by POMCP is represented by different colors (see Figure 5). In particular, green, blue, and orange points represent steps in the traces in which POMCP selected, respectively, a low speed, a medium speed ad a high speed. While our rule generated by XPOMCP presents a clear and compact representation of the boundary on the belief that must be satisfied to select speed 2, there are no obvious separations between the points of the three speed values in the graph. Most orange points are grouped in small clusters spread around the graph, but some isolated orange points are also present. The 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 point 2 and 4 are close to one of the clusters. Note that not all isolated points are marked as unexpected, XPOMCP identify the unexpected points not only by using their belief but also the insight provided by the expert.

Table 3: Notable steps in velocity regulation ($W = 90$)

#	p_0	p_1	p_2	H^2	*unexp.*
1	0.335	0.331	0.334	0.3526	yes
2	0.261	0.461	0.278	0.3090	yes
3	0.671	0.198	0.131	0.1717	yes
4	0.678	0.228	0.094	0.1389	yes
5	0.775	0.196	0.029	0.0411	no
6	0.832	0.127	0.041	0.0347	no
32	0.853	0.126	0.021	0.0109	no
33	0.826	0.160	0.014	0.0105	no

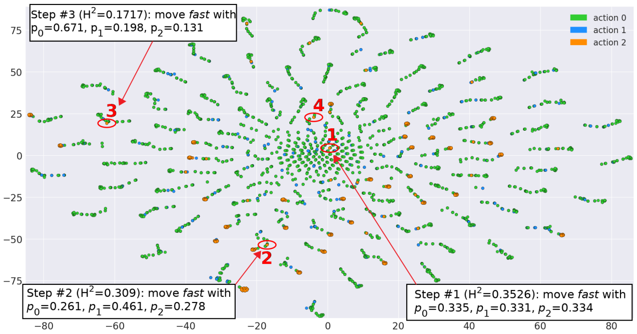


Figure 5: t-SNE of n-uples (belief, action) for the *velocity regulation* with $W = 90$. Our algorithm identify 4 anomalies, they are circled in red

5 CONCLUSIONS AND FUTURE WORK

In this work, we present a methodology that combines high-level indications provided by a human expert with an automatic procedure that analyzes execution traces to synthesize key properties of a policy in the form of rules. We exploit such rules to identify anomalous behavior, and we show that our methodology outperforms a state-of-the-art anomaly detection algorithm by exploiting the high-level indications. This work paves the way towards several interesting research directions. Specifically, we aim at improving the expressiveness of the logical formulas used to formalize the indications of the expert (e.g., by employing temporal logic), and to develop an online interaction between POMCP and XPOMCP, where the rules are generated while POMCP is operating and not only on execution traces.

REFERENCES

[1] Sule Anjomshoae, Amro Najjar, Davide Calvaresi, and Kary Främling. 2019. Explainable Agents and Robots: Results from a Systematic Literature Review. In *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems* (Montreal QC, Canada) *(AAMAS '19)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1078–1088.

[2] Eugenio Bargiacchi, Diederik M. Roijers, and Ann Nowé. 2020. AI-Toolbox: A C++ library for Reinforcement Learning and Planning (with Python Bindings). *Journal of Machine Learning Research* 21, 102 (2020), 1–12. http://jmlr.org/papers/v21/18-402.html

[3] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. 2018. Verifiable Reinforcement Learning via Policy Extraction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 2499–2509.

[4] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. νZ - An Optimizing SMT Solver. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*. Springer-Verlag, Berlin, Heidelberg, 194–199.

[5] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. 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, 3-8 December 2018, Montréal, Canada*. 4795–4804. http://papers.nips.cc/paper/7728-a-unified-view-of-piecewise-linear-neural-network-verification

[6] Michael Cashmore, Anna Collins, Benjamin Krarup, Senka Krivic, Daniele Magazzeni, and David Smith. 2019. Towards Explainable AI Planning as a Service. 2nd ICAPS Workshop on Explainable Planning, XAIP 2019.

[7] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. 2016. A Compilation of the Full PDDL+ Language into SMT. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling* (London, UK) *(ICAPS'16)*. AAAI Press, 79–87. http://dl.acm.org/citation.cfm?id=3038594.3038605

[8] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. 1997. Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *In Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, 54–61.

[9] Alberto Castellini, 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-19*. International Joint Conferences on Artificial Intelligence Organization, 5540–5546.

[10] Alberto Castellini, Enrico Marchesini, and Alessandro Farinelli. 2019. Online Monte Carlo Planning for Autonomous Robots: Exploiting Prior Knowledge on Task Similarities. In *Proceedings of the 6th Italian Workshop on Artificial Intelligence and Robotics (AIRO 2019@AI*IA2019) (CEUR Workshop Proceedings, Vol. 2594)*. CEUR-WS.org, 25–32.

[11] 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 (Lecture Notes in Computer Science, Vol. 12520)*. Springer, 261–276.

[12] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340.

[13] Maria Fox, Derek Long, and Daniele Magazzeni. 2017. Explainable Planning. *CoRR* abs/1709.10256 (2017). http://arxiv.org/abs/1709.10256

[14] David Gunning. 2019. DARPA's Explainable Artificial Intelligence (XAI) Program. *IUI '19: Proceedings of the 24th International Conference on Intelligent User Interfaces*, ii–ii.

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

[16] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 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 (Lecture Notes in Computer Science, Vol. 10426)*. Springer, 3–29.

[17] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artif. Intell.* 101, 1–2 (May 1998), 99–134.

[18] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 97–117.

[19] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Proc. ECML'06*. Springer-Verlag, Berlin, Heidelberg, 282–293.

[20] Pat Langley, Ben Meadows, Mohan Sridharan, and Dongkyu Choi. 2017. Explainable Agency for Intelligent Autonomous Systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (San Francisco, California, USA) *(AAAI'17)*. AAAI Press, 4762–4763.

[21] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*. 413–422.

[22] 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) (CEUR Workshop Proceedings, Vol. 2806)*. CEUR-WS.org, 44–48.

[23] Gethin Norman, David Parker, and Xueyi Zou. 2017. Verification and control of partially observable probabilistic systems. *Real-Time Systems* 53, 3 (2017), 354–402.

[24] Christos H. Papadimitriou and John N. Tsitsiklis. 1987. The Complexity of Markov Decision Processes. *Math. Oper. Res.* 12, 3 (Aug. 1987), 441–450.

[25] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Rron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[26] David Silver and Joel Veness. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe Taylor, R. S. Zemel, and A. Culotta (Eds.). Curran Associates, Inc., 2164–2172. http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps.pdf

[27] Laurens van der Maaten and Geoffrey Hinton. 2008. Viualizing data using t-SNE. *Journal of Machine Learning Research* 9 (11 2008), 2579–2605.

[28] Yue Wang, Swarat Chaudhuri, and Lydia E. Kavraki. 2018. Bounded Policy Synthesis for POMDPs with Safe-Reachability Objectives. *ArXiv* abs/1801.09780 (2018).

[29] He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. 2019. An Inductive Synthesis Framework for Verifiable Reinforcement Learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 686–701.