# Strengthening Deterministic Policies for POMDPs

Leonore Winterer*[1], Ralf Wimmer[2,1], Nils Jansen[3], and Bernd Becker[1]

[1] Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
{winterel, wimmer, becker}@informatik.uni-freiburg.de
[2] Concept Engineering GmbH, Freiburg im Breisgau, Germany
[3] Radboud University, Nijmegen, The Netherlands
n.jansen@science.ru.nl

**Abstract.** The synthesis problem for partially observable Markov decision processes (POMDPs) is to compute a policy that satisfies a given specification. Such policies have to take the full execution history of a POMDP into account, rendering the problem undecidable in general. A common approach is to use a limited amount of memory and randomize over potential choices. Yet, this problem is still NP-hard and often computationally intractable in practice. A restricted problem is to use neither history nor randomization, yielding policies that are called stationary and deterministic. Previous approaches to compute such policies employ mixed-integer linear programming (MILP). We provide a novel MILP encoding that supports sophisticated specifications in the form of temporal logic constraints. It is able to handle an arbitrary number of such specifications. Yet, randomization and memory are often mandatory to achieve satisfactory policies. First, we extend our encoding to deliver a restricted class of randomized policies. Second, based on the results of the original MILP, we employ a preprocessing of the POMDP to encompass memory-based decisions. The advantages of our approach over state-of-the-art POMDP solvers lie (1) in the flexibility to strengthen simple deterministic policies without losing computational tractability and (2) in the ability to enforce the provable satisfaction of arbitrarily many specifications. The latter point allows to take trade-offs between performance and safety aspects of typical POMDP examples into account. We show the effectiveness of our method on a broad range of benchmarks.

## 1 Introduction

Partially observable Markov decision processes (POMDPs) are a formal model for planning under uncertainty in partially observable environments [23,37]. POMDPs adequately model a number of real-world applications, see for instance [43,33]. While an agent operates in a scenario modeled by a POMDP, it receives *observations* and tries to infer the likelihood of the system being in a certain state, the belief state. Together with a belief update function, the space of all belief states forms a (uncountably infinite) *belief MDP* [35,26,5].

* Corresponding author

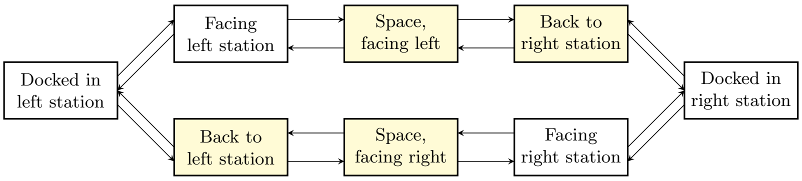


Fig. 1: The space shuttle benchmark – yellow states share an observation. Transitions have been simplified for clarity.

Consider the following simple example [11] as sketched in Fig. 1. A space shuttle has to transport goods between two stations, while docking at these stations is subject to failure with certain probabilities. The perception of the shuttle is limited in the sense that it will only see the stations if it is directly facing them. If not, it can only see empty space and has to infer from the history which of the stations is the next one to deliver goods to.

Traditional POMDP problems typically comprise the computation of a policy that maximizes a cumulative reward over a finite horizon. However, the application may require that the agent's behavior obeys more complicated specifications. For example, temporal logics (e. g., LTL [31]) describe task properties like reachability or liveness that cannot be expressed using reward functions [25]. For the aforementioned space shuttle, maximizing the reward corresponds to maximizing the number of succesful deliveries. Additional specifications may for instance require the shuttle to only navigate in empty space for a limited number of steps.

Policy synthesis for POMDPs is hard. For infinite- or indefinite-horizon problems, computing an optimal policy is undecidable [26]. Optimal action choices depend on the whole history of observations and actions, and thus require an infinite amount of memory. When restricting the specifications to maximizing accumulated rewards over a finite horizon and also limiting the available memory, computing an optimal policy is PSPACE-complete [29]. This problem is, practically, intractable even for small instances [27]. When policies are restricted to be *memoryless*, finding an optimal policy within this set is still NP-hard [39]. For the more general LTL specifications, synthesis of policies with limited memory is even EXPTIME-complete [8].

*State-of-the-art.* The aforementioned hardness and intractability of the computation of exact solutions for the POMDP problems discussed earlier triggered several feasible approaches. Notably, there are approximate [20], point-based [30], or Monte-Carlo-based [36] methods. Yet, none of these approaches provides guarantees for temporal logic specifications. The tool PRISM-pomdp [28] actually provides guarantees by approximating the belief space into a fully observable belief MDP, but is restricted to small examples. Other techniques, such as those employing an incremental satisfiability modulo theory (SMT) solver over a bounded belief space [40] or a simulation over sets of belief models [18], are also restricted to small examples. [42] employs a game-based abstraction approach to efficiently solve problems with specific properties. In [22], finite-state controllers for POMDPs are computed using parameter synthesis for Markov chains [19,21] by applying convex optimization techniques [12,13]. Another work employs machine learning techniques together with formal verification to achieve sound but not optimal solutions [7].

*Our Approach.* The problem we consider in this paper is to compute a policy for a POMDP that provably satisfies one or more specifications such as temporal logic constraints and expected (discounted) reward properties. First, we restrict ourselves to a simple class of policies which are both memoryless and do not randomize over action choices, that is, they are *deterministic*. A natural approach encodes this problem as a mixed-integer linear program (MILP) [34]. We extend previous approaches [2,24] to account for multiple specifications and provide a particular encoding for temporal logic constraints. The advantage is that these MILPs yield simple, small, and easy-to-analyze policies which can be computed by efficient state-of-the-art tools like Gurobi [17].

However, policies that incorporate randomization over choices often trade off the necessity of memory-based decisions [10,1], and randomization may be needed for multiple objectives [15,3]. To preserve the advantages of MILP solving, we propose *static randomization*. We augment the MILP encoding in the following way. In addition to deterministic choices, the policy may to select an arbitrary but fixed distribution over all possible actions. As we will demonstrate in this paper, often any distribution is sufficient as long as randomization is possible.

Yet, for certain problems a notion of (at least finite) memory is required. As a third step to strengthen deterministic policies, we perform a preprocessing of the POMDP regarding previous computations for purely deterministic policies. At states where the choices are bad according to the specifications, we perform a technique we call *observation and state splitting* which essentially encodes finite memory into the state space of the POMDP. Intuitively, we enable a policy to distinguish states that previously shared an observation.

Summarized, we provide three contributions. First, we enable the computation of deterministic polices that provably adhere to multiple specifications. Second, we augment the underlying MILP to account for randomization using fixed distributions over actions. Third, we introduce a novel POMDP preprocessing which encodes finite memory into critical states. We showcase the feasibility and competitiveness of our approach by a thorough experimental evaluation on well-known case studies.

## 2 Preliminaries

For a finite or countably infinite set $X$, $\mu\colon X \to [0,1]$ with $\sum_{x\in X}\mu(x) = 1$ denotes a *probability distribution* over $X$; the set of all probability distributions over $X$ is $\mathit{Dist}(X)$. A partial function $f\colon X \nrightarrow Y$ is a function $f\colon X' \to Y$ for some subset $X' = \mathrm{dom}(f) \subset X$.

**Definition 1 (Markov Decision Process).** *A* Markov Decision Process (MDP) *is a tuple* $\mathsf{M} = (S, s_{\mathit{init}}, \mathit{Act}, P, R)$ *where* $S$ *is a finite set of* states*,* $s_{\mathit{init}} \in S$

*the* initial state*, $Act$ a finite set of* actions*, $P\colon S \times Act \nrightarrow Dist(S)$ a (partial)* probabilistic transition function*, and $R\colon S \times Act \to \mathbb{R}$ a reward function that assigns to every tuple $(s, \alpha) \in \mathrm{dom}(P)$ a real-valued reward.*

The set of actions that are enabled in $s$ is $Act(s) = \{\alpha \in Act \,|\, (s, \alpha) \in \mathrm{dom}(P)\}$ .

A partially observable Markov decision process (*POMDP*) [23] models restricted knowledge of the current state of an MDP.

**Definition 2 (POMDP).** *A* partially observable Markov decision process (POMDP) *is a tuple $\mathsf{D} = (\mathsf{M}, \mathcal{O}, \lambda)$ such that $\mathsf{M} = (S, s_{init}, Act, P, R)$ is the* underlying MDP of $\mathsf{D}$*, $\mathcal{O}$ a finite set of* observations*, and $\lambda\colon S \to \mathcal{O}$ the* observation function.

Note that in our definition of a POMDP each state has exactly one observation. Sometimes a more general definition of POMDPs is used, in which the observation function depends not only on the current state, but also on the previous action, and returns not a fixed observation, but a probability distribution over the possible observations. However, there is a polynomial reduction from this general case to the one we use in this work [9].

**Definition 3 (Path).** *A sequence $\pi = s_0\alpha_0 s_1\alpha_1 \dots$ with $s_i \in S$, $\alpha_i \in Act$ and $P(s_i, \alpha_i)(s_{i+1}) > 0$ for all $i \geq 0$ is called a* path*. Paths can be finite (ending in a state) or infinite. The set of finite paths is $\mathsf{Paths}_{\mathit{fin}}$ and the set of infinite paths $\mathsf{Paths}_{inf}$. For a finite path $\pi$, we denote by $last(\pi)$ the final state of $\pi$.*

**Definition 4 (Observation Sequence).** *If $\pi = s_0\alpha_0 s_1\alpha_1 \dots s_{n-1}\alpha_{n-1}s_n$ is a finite path, then $\theta \coloneqq \lambda(\pi) \coloneqq \lambda(s_0)\alpha_0\lambda(s_1)\alpha_1 \dots \lambda(s_{n-1})\alpha_{n-1}\lambda(s_n)$ is called the* observation sequence *of $\pi$.*

Before a probability space over paths of (PO)MDPs can be defined, the nondeterminism needs to be resolved. This resolution is done by an entity called a policy that determines the next action to execute:

**Definition 5 (Policy).** *A* policy *for a POMDP $\mathsf{D}$ is a function $\sigma\colon \mathsf{Paths}_{\mathit{fin}} \to Dist(Act)$ such that $\sigma(s_0 \dots s_n)(\alpha) > 0$ implies $\alpha \in Act(s_n)$. We denote the set of all possible policies for a POMDP $\mathsf{D}$ with $\Sigma_{\mathsf{D}}$.*

*A policy is* observation-based *if $\sigma(\pi) = \sigma(\pi')$ holds for all $\pi, \pi'$ with $\lambda(\pi) = \lambda(\pi')$. A policy is $\sigma$* stationary *if $\sigma(\pi) = \sigma(\pi')$ holds whenever $last(\pi) = last(\pi')$. The set of all stationary policies for a POMDP $\mathsf{D}$ is $\Sigma_{\mathsf{D}}^{stat}$. Policies that are not stationary, are called* history-dependent*. A policy is* deterministic *if $\sigma(\pi)(\alpha) \in \{0, 1\}$ for all $\pi$ and $\alpha$. Policies that are not deterministic are called* randomized.

Stationary observation-based policies are typically regarded as functions $\sigma\colon \mathcal{O} \to Dist(Act)$ (randomized policy) or $\sigma\colon \mathcal{O} \to Act$ (deterministic policy). As a policy resolves all nondeterminism and partial observability, it turns a (PO)MDP into a discrete-time Markov chain (DTMC), which is a purely stochastic process.

**Definition 6 (Induced DTMC).** *Let $\mathsf{D}$ be a POMDP as defined above with reward function $R$ and $\sigma\colon \mathsf{Paths}_{\mathit{fin}} \to Dist(Act)$ a policy. The induced DTMC is a tuple $\mathsf{D}_\sigma = (\mathsf{Paths}_{\mathit{fin}}, s_{init}, P')$ such that $P'(\pi, \pi') = \sigma(\pi)(\alpha) \cdot P(last(\pi), \alpha, s)$ if $\pi' = \pi\alpha s$, and $P'(\pi, \pi') = 0$ otherwise. The induced reward function $R'\colon \mathsf{Paths}_{\mathit{fin}} \to \mathbb{R}$ is defined as $R'(\pi) = \sum_{\alpha \in Act(last(\pi))} \sigma(\pi)(\alpha) \cdot R(last(\pi), \alpha)$.*

In the following we consider the computation of observation-based policies for POMDPs. The goal is to find a policy such that the induced DTMC satisfies a given specification. For the scope of this paper, we focus on *reachability* and *expected discounted reward* specifications and combinations thereof. Note that general LTL properties for probabilistic systems can be reduced to reachability [4].

**Definition 7 (Reachability).** *Let* $\mathsf{C} = (S, s_{init}, P)$ *be a DTMC and* $T \subseteq S$ *a set of target states. The probability to reach a state in* $T$ *from* $s$ *is the unique solution of the following linear equation system:*

$$x_s = \begin{cases} 1 & \textit{if } s \in T, \\ 0 & \textit{if there is no path from } s \textit{ to } T, \\ \sum_{s' \in \mathrm{succ}(s)} P(s, s') \cdot x_{s'} & \textit{otherwise.} \end{cases}$$

**Definition 8 (Expected discounted rewards).** *For a discount factor* $\beta \in (0, 1)$ *and a DTMC* $\mathsf{C} = (S, s_{init}, P)$*, the* expected discounted reward *of a state state* $s$ *is the unique solution of the following linear equation system:*

$$r_s = R(s) + \beta \cdot \sum_{s' \in \mathrm{succ}(s)} P(s, s') \cdot r_{s'} \qquad \textit{for each } s \in S.$$

Recall that the problem to determine a policy that optimizes expected rewards or probabilities is undecidable [26] in general.

## 3 Solving POMDPs as MILPs

While several sophisticated algorithms exist to compute policies for POMDPs, a simple, small, and easy-to-analyze policy can be obtained by encoding the POMDP into a *Mixed Integer Linear Program* (MILP), which can be solved with linear optimization tools like *Gurobi* [17]. As a central advantage of the MILP formulations, it is straightforward to support multiple specifications simultaniously. For instance, one can maximize the discounted reward under the condition that the probability of reaching a target state is above a given bound and the discounted cost below another threshold.

### 3.1 Maximum Reachability Probabilities

Let $\mathsf{D} = (\mathsf{M}, \mathcal{O}, \lambda)$ be a POMDP and $T \subseteq S$ a set of target states. We assume that the states in $T$ have been made absorbing and that $\mathsf{M}$ contains only states from which $T$ is reachable under at least one possible policy. All other states can be removed from the POMDP. We define the following MILP:

$$\text{maximize:} \quad p_{s_{\mathrm{init}}} \tag{1a}$$

$$\text{subject to:}$$

$$\forall s \in S \setminus T: \quad \sum_{\alpha \in Act(s)} \sigma_{\lambda(s),\alpha} = 1 \tag{1b}$$

$$\forall s \in T\colon \quad p_s = 1 \tag{1c}$$

$$\forall s \in S \setminus T \ \forall \alpha \in Act(s)\colon \quad p_s \leq (1 - \sigma_{\lambda(s),\alpha}) + \sum_{s' \in \mathrm{succ}(s,\alpha)} P(s,\alpha,s') \cdot p_{s'} \tag{1d}$$

$$\forall (s,\alpha) \in Act^{\mathrm{pr}} \ \forall s' \in \mathrm{succ}(s,\alpha)\colon \quad r_s < r_{s'} + 1 - t_{s,s'} \tag{2a}$$

$$\forall (s,\alpha) \in Act^{\mathrm{pr}}\colon \quad p_s \leq 1 - \sigma_{\lambda(s),\alpha} + \sum_{s' \in \mathrm{succ}(s,\alpha)} t_{s,s'} \tag{2b}$$

The variables $p_s \in [0,1]$ store the probability to reach a target state from $s$ under the chosen policy. We maximize this probability for the initial state $s_{\mathrm{init}}$ (1a). The variables $\sigma_{z,\alpha}$ for $z \in \mathcal{O}$ and $\alpha \in Act$ encode the policy. $\sigma_{\lambda(s),\alpha} = 1$ implies that the policy chooses action $\alpha$ in all states with observation $\lambda(s)$ – as we are computing stationary deterministic policies, $\sigma_{\lambda(s),\alpha} \in \{0,1\}$ for all $s \in S$ and $\alpha \in Act(s)$. Thus, (1b) ensures that for each observation exactly one action is selected. (1c) ensures that target states are assigned a probability of 1. For non-target states $s \in S \setminus T$, (1d) recursively defines the probability $p_s$: for actions that are not chosen, i. e., $\sigma_{\lambda(s),\alpha} = 0$, the inequality is always satisfied, as it can be simplified to $p_s \leq 1 + \epsilon$ with $\epsilon \geq 0$. If $\sigma_{\lambda(s),\alpha} = 1$, the probability is defined as the sum of the probability in each of the successors of the current state, multiplied with the probability to proceed to this successor when taking the current action. Maximizing the value of $p_s$ ensures that this constraint is satisfied by equality. If the target states are reachable from all states under all possible policies, (1a)–(1d) are sufficient. We add (2a) and (2b) to avoid computing invalid values under policies that make the targets unreachable from some states: we define the problematic states $S^{\mathrm{pr}}$ as the set of states that can only reach the target states under some policies, and compute them using standard graph algorithms. The problematic actions are then given by $Act^{\mathrm{pr}} = \big\{(s,\alpha) \in S \times Act \,\big|\, \alpha \in Act(s) \land \mathrm{succ}(s,\alpha) \subseteq S^{\mathrm{pr}}\big\}$. We then introduce a ranking over the problematic states: each $s \in S^{\mathrm{pr}}$ is assigned a value $r_s \in [0,1]$. Next, we try to assign a transition to a successor state of $s$ by setting $t_{s,s'} = 1$ such that the value of the rank increases along the transition, i. e., $r_{s'} > r_s$. If this is not the case, (2a) enforces $t_{s,s'} = 0$. If the target state cannot be reached under the current policy, i. e., $t_{s,s'} = 0$ for all successors of $s$, (2b) ensures that $p_s = 0$. This technique is inspired by the reachability constraints from [41] that are used to compute counterexamples for MDPs [14]. An alternative formulation of reachability constraints using flow constraints can be found in [38].

### 3.2 Maximum Expected Discounted Rewards

Let $\mathsf{D} = (\mathsf{M}, \mathcal{O}, \lambda)$ be a POMDP. For a discount factor $\beta \in (0,1)$ and an upper bound on the maximum discounted expected reward $v^*_{\max}$, we can built the MILP as follows:

$$
\begin{aligned}
\text{maximize:}\quad & v_{s_{\text{init}}} && \text{(3a)}\\
\text{subject to:}\quad & && \\
\forall s \in S\colon\quad & \sum_{\alpha \in Act(s)} \sigma_{\lambda(s),\alpha} = 1 && \text{(3b)}\\
\forall s \in S\ \forall \alpha \in Act(s)\colon\quad & v_s \le v^*_{\max} \cdot (1 - \sigma_{\lambda(s),\alpha}) + r(s,\alpha) && \\
& \qquad + \beta \cdot \sum_{s' \in \operatorname{succ}(s,\alpha)} P(s,\alpha,s') \cdot v_{s'} && \text{(3c)}
\end{aligned}
$$

The MILP for maximum discounted reward is analogous to the formulation for maximum reachability, with the following differences: The real-valued variables $v_s \in \mathbb{R}$ for each $s \in S$ store the maximum discounted expected reward corresponding to the selected policy.

As $v_s$ can have values $> 1$, in (3c), we need an upper bound $v^*_{\max}$ on the maximum expected reward. One possibility is setting $v^*_{\max}$ to the maximum expected discounted reward of the underlying MDP $\mathsf{M}$, which serves as an upper bound on the reward that can be achieved in $\mathsf{D}$. An alternative is using $v^*_{\max} := \frac{1}{1-\beta} \cdot \max_{s,\alpha} R(s,\alpha)$. Since we no longer have any target states, the expected reward is computed for an infinite run of $\mathsf{D}$ under the selected policy. $0 < \beta < 1$ guarantees that the expected reward converges to a finite number. Thus, we don't need the reachability constraints we introduced in Sect. 3.1. This simplification makes the MILP considerably smaller and more efficient to solve.

### 3.3 Randomization

Stationary, deterministic policies can be restrictive in many use cases. However, while randomization might often be necessary, sometimes the actual probability distribution does not matter. In Fig. 2, any stationary deterministic policy can reach the blue state with probability of at most 0.5. However, assigning any distribution with $\sigma_{\text{yellow},\alpha} > 0$ and $\sigma_{\text{yellow},\beta} > 0$ leads to a probability of 1.

In order to achieve this effect, we allow (besides deterministic choices) a randomized choice with an arbitrary, but fixed distribution over the enabled actions. This can be done by introducing an additional action $u \notin Act$ that is enabled in the set $S' \subseteq S$ of states with (1) a non-unique observation and (2)

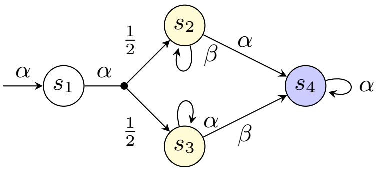
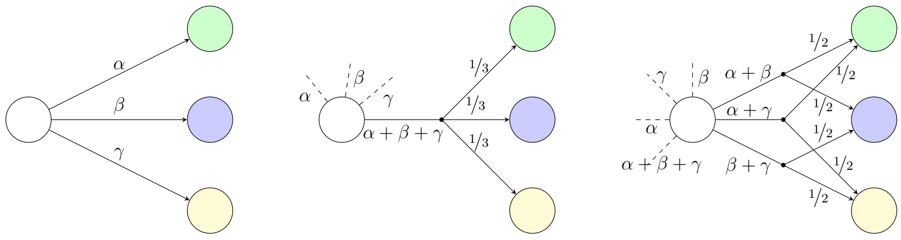


Fig. 2: A simple example that needs arbitrary randomization for maximum reachability of $s_4$.



(a) Pure (b) Light (c) Heavy

Fig. 3: A POMDP without, with light, and with heavy static randomization

more than one enabled action. We replace the underlying MDP $\mathsf{M}$ by $\mathsf{M}' = (S, s_{\text{init}}, Act_u, P_u)$ such that $Act_u := Act \cup \{u\}$ and $P_u$ as follows: $P_u$ coincides with $P$ in states $S \setminus S'$ and whenever $\alpha \neq u$. For instance, consider a state $s \in S'$ where we want to achieve a uniform distribution over all actions. We set $P_u(s, u, s') = \frac{1}{|Act(s)|} \cdot \sum_{\alpha \in Act(s)} P(s, \alpha, s')$ for $s' \in S$.

Any finite set of distributions can be supported that way. We suggest three modes of randomization, as illustrated in Fig. 3: *Pure* (no randomization), *Light* (adding one uniform distribution the enabled actions for each state) and *Heavy* (adding a uniform distribution for each non-empty subset of enabled actions.

## 4 Splitting Observations and States

Finding an optimal stationary policy is a much easier problem than optimizing over all (history-dependent) policies, but the quality of stationary policies can be arbitrarily worse than the quality of a general optimal policy. We attempt to preprocess POMDPs in a way that implicitly adds history locally by encoding previous observations into the states – thus, making stationary policies computed on the augmented POMDP more powerful. In order to do so, we introduce *observation splitting* and *state splitting*.

### 4.1 Observation Splitting

Let $\mathsf{D} = (\mathsf{M}, \mathcal{O}, \lambda)$ be a POMDP with underlying MDP $\mathsf{M} = (S, s_{\text{init}}, Act, P, R)$, $z \in \mathcal{O}$ an observation, and $\lambda^{-1}(z) = \{s \in S \mid z = \lambda(s)\}$ the set of states with observation $z$. W. l. o. g., let $|\lambda^{-1}(\lambda(s_{\text{init}}))| = 1$ and $P(s, s_{\text{init}}) = 0$ for all $s \in S$. An existing POMDP can easily be modified to conform with these requirements.

**Definition 9 (Pre-Observations).** *For $s \in S$, the pre-observations of $s$ are defined as* $pred_{\mathsf{D}}(s) = \big\{(z, \alpha) \in \mathcal{O} \times Act \,\big|\, \exists s' \in S \colon z = \lambda(s') \wedge P(s', \alpha, s) > 0\big\}$.

Assume that $s, s'$ are the only states with observation $z = \lambda(s) = \lambda(s')$ and that the pre-observations of $s$ are disjoint from the pre-observations of $s'$. A history-dependent policy can easily distinguish the two states by remembering the previous observation and action, but a stationary policy can not. Observation splitting assigns distinct observations to the two states. While a memoryless policy on the original POMDP has to make the same decision in $s$ and $s'$, it can make different decisions on the modified system. Therefore, a memoryless policy on the modified system typically corresponds to a history-dependent policy on the original POMDP. Note that this operation does not increase the number of states or transitions.

An observation $z$ can be split if we can partition $\lambda^{-1}(z)$ into two disjoint subsets $A$ and $B$ such that $\big(\bigcup_{s\in A} \mathrm{pred}_{\mathsf{D}}(s)\big) \cap \big(\bigcup_{s\in B} \mathrm{pred}_{\mathsf{D}}(s)\big) = \emptyset$, i. e., when $z$ is observed, we can distinguish states in $A$ from states in $B$ if the observation in the predecessor state as well as the last chosen action are known. This information can be encoded into the POMDP by assigning distinct observations to the states in $A$ and the states in $B$. Formally, we get the POMDP $\mathsf{D}' = (\mathsf{M}, \mathcal{O}', \lambda')$ with

$$\mathcal{O}' = (\mathcal{O} \setminus \{z\}) \,\dot\cup\, \{z_A, z_B\} \text{ and } \lambda'(s) = \begin{cases} \lambda(s) & \text{if } s \notin A \,\dot\cup\, B, \\ z_A & \text{if } s \in A, \\ z_B & \text{if } s \in B. \end{cases}$$

**Theorem 1.** *Let $\mathsf{D}'$ be the POMDP we obtain by splitting some observation $z$ of POMDP $\mathsf{D}$ into new observations $z_A$ and $z_B$. Then:*

1. $\{\mathsf{D}_\sigma \mid \sigma \in \Sigma_{\mathsf{D}}\} = \{\mathsf{D}'_\sigma \mid \sigma \in \Sigma_{\mathsf{D}'}\}$*, and*
2. $\{\mathsf{D}_\sigma \mid \sigma \in \Sigma^{stat}_{\mathsf{D}}\} \subseteq \{\mathsf{D}'_\sigma \mid \sigma \in \Sigma^{stat}_{\mathsf{D}'}\}$.

If we consider the set of all policies, observation splitting does not make a difference as we can obtain the same induced DTMCs before and after observation splitting. However, if we only consider stationary policies, we get more freedom and can choose among a larger set of induced DTMCs.

*Proof.* Let $\mathsf{D}$ be a POMDP and $\mathsf{D}'$ result from $\mathsf{D}$ by splitting observation $z$. Let, for $i = 1, 2$, $\pi_i = s^i_0 \alpha^i_0 s^i_1 \alpha^i_1 \dots s^i_n \in \mathsf{Paths}_{\mathit{fin}}$ be two finite paths in $\mathsf{D}$, and $\pi'_i$ be the corresponding paths in $\mathsf{D}'$. It is easy to see that $\lambda'(\pi'_1) = \lambda'(\pi'_2)$ iff $\lambda(\pi_1) = \lambda(\pi_2)$. That means, for each policy in $\mathsf{D}$ there is a corresponding policy in $\mathsf{D}'$ that makes the same decisions and vice versa.

Additionally, for all states $s_1, s_2$ of $\mathsf{D}$ and $\mathsf{D}'$, we have $\lambda(s_1) \neq \lambda(s_2) \Rightarrow \lambda'(s_1) \neq \lambda'(s_2)$. Therefore a stationary policy that can make different choices in $s_1$ and $s_2$ in $\mathsf{D}$ can make different choices in $\mathsf{D}'$ as well. □

### 4.2 State Splitting

Often, observation splitting is not applicable to a given POMDP. We define state splitting for refining the POMDP to enable observation splitting: In Fig. 4, all states that have the same color share an observation (i. e., $\lambda(s_1) = \lambda(s_2) = \lambda(s_3)$). We have $\mathrm{pred}_{\mathsf{D}}(s_2) = \mathrm{pred}_{\mathsf{D}}(s_1) \cup \mathrm{pred}_{\mathsf{D}}(s_3)$, so the three states cannot be split into disjoint sets by means of their pre-observations and thus, observation splitting cannot be applied. However, by creating two copies $s^1_2$ and $s^2_2$, the pre-observations

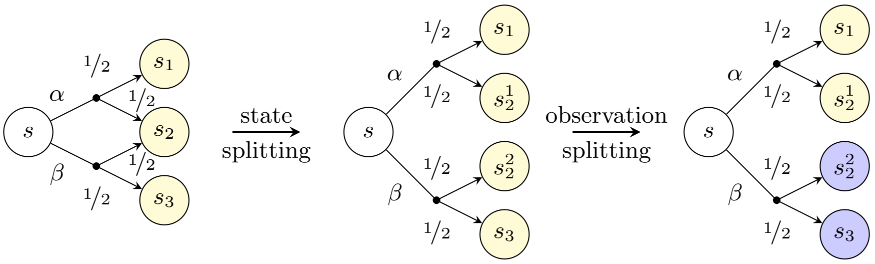


Fig. 4: Applying state splitting and observation splitting to a POMDP. The observations are given by the color of the states.

of $s_2^1$ and $s_1$ on the one hand and $s_2^2$ and $s_3$ on the other hand become disjoint, thus enabling observation splitting on the yellow observation.

Formally, we can split a state $s \in S$ in a POMDP $\mathsf{D} = (\mathsf{M}, \mathcal{O}, \lambda)$ whenever $|\mathrm{pred}_{\mathsf{D}}(s)| > 1$. Again, we assume that $\mathrm{pred}_{\mathsf{D}}(s_{\mathrm{init}}) = \emptyset$. We obtain a modified POMDP $\mathsf{D}' = (\mathsf{M}', \mathcal{O}', \lambda')$ with

- $S' \coloneqq \big(S \setminus \{s\}\big) \mathbin{\dot\cup} \big\{(s, z, \alpha) \,\big|\, (z, \alpha) \in \mathrm{pred}_{\mathsf{D}}(s)\big\}$;
- for all $t \in S'$ we set: $\lambda'(t) \coloneqq \lambda(s)$ if $t = (s, z, \alpha)$ for some $z \in \mathcal{O}$ and $\alpha \in Act$, and $\lambda'(t) \coloneqq \lambda(t)$ otherwise;
- for all $t, t' \in S'$, $\beta \in Act$:

$$P'(t,\beta,t') \coloneqq \begin{cases} P(t,\beta,t') & \text{if } t,t' \in S \setminus \{s\}, \\ P(s,\beta,s) & \text{if } t = (s,z,\alpha) \text{ and } t' = (s,z,\beta) \\ & \qquad \text{for some } z \in \mathcal{O} \text{ and } \alpha \in Act, \\ P(t,\beta,s) & \text{if } t \in S \setminus \{s\} \text{ and } t' = (s,\lambda(t),\beta), \\ P(s,\beta,t') & \text{if } t = (s,z,\alpha) \text{ for some } z \in \mathcal{O} \\ & \qquad \text{and } \alpha \in Act \text{ and } t' \in S \setminus \{s\}, \\ 0 & \text{otherwise;} \end{cases}$$

- for all $t \in S'$ and $\beta \in Act(t)$ we set: $R'(t, \beta) \coloneqq R(s, \beta)$ if $t = (s, z, \alpha)$ for some $z \in \mathcal{O}$ and $\alpha \in Act$, and $R'(t, \beta) \coloneqq R(t, \beta)$ otherwise.

**Theorem 2.** *Let $\mathsf{D}$ be a POMDP and let $\mathsf{D}'$ result from $\mathsf{D}$ by splitting a state $s$ of $\mathsf{D}$. Then $\mathsf{D}$ and $\mathsf{D}'$ are bisimilar.*

After extending the definition of bisimulation to POMDPs, this can be proven by defining an equivalence relation between $s$ and the states $(s, z, \alpha)$ produced by splitting it.

It is well known [16] that bisimilar systems satisfy (among others) the same LTL and PCTL properties, including reachability and discounted expected rewards.

**Algorithm 1:** Splitting Heuristic

```
Input: POMDP D
oldResult ← 0
splitObservations(D)
newResult ← computeMILP(D)
while newResult > oldResult do
    oldResult ← newResult
    splitGroup ← computeSplitGroup(D, oldResult)
    splitStates(D, splitGroup)
    splitObservations(D)
    newResult ← computeMILP(D)
```

## 5 Implementation

We implemented both MILP formulations described in Sect. 3 and use the commercial solver Gurobi [17] to solve them. From our experience, Gurobi often finds a feasible solution, which satisfies all constraints, quickly, but then spends a lot of time trying to improve this initial solution or prove its optimality. However, even this initial solution is often already close to the optimum. Thus, we have implemented a *time limit* mode, in which the solver tries to optimize the result for a predefined number of seconds after the first solution is found.

We implemented the MILPs with three different levels of randomization as in Sect. 3.3, and observation and state splitting as in Sect. 4.

State splitting by itself only increases the size of the state space and yields a bisimilar system. Therefore it only makes sense to apply state splitting when it enables observation splitting, which in turn increases the power of stationary policies. However, it is not clear beforehand which states to split. So as a rule of thumb, we want to determine a small subset of states whose splitting enables a large number of observation splits.

*Splitting Heuristic.* We suggest a splitting heuristic that uses previous results of the MILP to iteratively refine the POMDP by selecting states for splitting, see Algorithm 1 for an outline. First, we apply observation splitting on the original POMDP and compute the optimal stationary policy in that POMDP to get a baseline for the following optimization. Then, we use this solution to determine a set of states for splitting. Similar to what policy iteration for MDPs [32] does, we check if locally changing a selected action would result in an improvement. $\sigma^*$ is the current policy and $v^*_s$ the corresponding value of state $s$. We choose

$$\sigma'(s) :\in \operatorname*{argmax}_{\alpha \in Act(s)} \sum_{s' \in S} P(s, \alpha, s') \cdot v^*_{s'},$$

preferring $\sigma'(s) = \sigma^*(s)$ where possible.

Whenever $\sigma'(s) \neq \sigma^*(s)$ holds, then being able to distinguish $s$ from the other states with the same observation would lead to an improvement. Therefore $s$ is added to the set *splitGroup*. State splitting is applied to all states in *splitGroup*. Afterwards we apply observation splitting as long as it modifies the POMDP, and solve the MILP for the modified POMDP. We repeat this procedure, until no further improvements can be made.

In case of multiple specifications, it can happen that the initial MILP is infeasible on the original POMDP. In this case we apply Algorithm 1 to optimize the first constraint until it is satisfied. Then we optimize the second one under the condition that the first constraint is satisfied, etc. In the end, we either obtain a policy that satisfies all constraints, or at some point we cannot satisfy one of the specifications. This can have two reasons: either the POMDP does not satisfy the specification or state plus observation splitting are not powerful enough to yield a POMDP on which a stationary policy satisfies the constraints. Note that a complete method does not exist due to the undecidability of the problem.

## 6 Experiments

*Experimental Setup.* All experiments were run on a machine with a 3.3 GHz Intel® Xeon® E5-2643 CPU and 64 GB RAM, running Ubuntu 16.04.

We consider seven benchmarks from two different sources. The *4×4grid_avoid* was taken from the PRISM-pomdp model checker[4] and is a maximum reachability probability grid world (with one absorbing "bad" state that needs to be avoided). The other benchmarks were adopted from the POMDP page[5] and slightly modified to fit our definitions. *1d*, *4×4.95*, *cheese.95*, *mini-hall2*, and *parr95.95* are grid worlds in which a reward is issued for reaching certain states. *shuttle.95* describes a space shuttle delivering cargo between two space stations, and a reward is issued for every successful delivery (see Fig. 1).

For two of the benchmarks, we added secondary constraints to demonstrate the effectiveness of our approach to multi-objective model checking. On 4×4grid_avoid, we added a cost of 1 for each step in the grid (except for the self loops in the goal and bad state). We require the reachability probability to be at least 0.25, and minimize the (un-discounted) expected reward. Note that computing un-discounted reward is sound in this case, as we asure the computation of a valid policy by the reachability contraints as seen in 3.1 and a sink state is eventually reached with probability 1. On cheese.95, we added a new state – each time the goal state is reached, there is a choice to continue back into the maze, or to transit to a rewardless sink state. We then declared one state of the grid "bad" and required that the probability to reach this state is at most 0.5, while still maximizing the total expected discounted reward.

We run our MILP implementation using Gurobi 8.1 to solve all benchmarks. To improve runtimes, we used time limits of 5, 10, 30, and 60 s for the optimization part of each solver call (see Sect. 5). We also let the optimization run to termination (with a total time limit of 7200 s) to get an assessment of the quality of the solutions that were found.

[4] `http://www.prismmodelchecker.org/files/rts-poptas/`

[5] `http://www.pomdp.org/examples/`

Table 1: Results for different benchmarks, timeouts, and implementations

| Benchmark | TO | Pure | Light | Heavy | Pure + H | Light + H | Heavy + H | SARSOP | solvePOMDP | PRISM-pomdp |
|---|---|---|---|---|---|---|---|---|---|---|
| 1d | 5 s | **0.61/0.1s** | **0.65/0.1s** | **0.65/0.1s** | **0.83/0.1s** | **0.83/0.7s** | **0.83/0.1s** | 0.95/0.003s | 0.95/1.3s | — |
|  | 10 s |  |  |  |  |  |  |  |  |  |
|  | 30 s |  |  |  |  |  |  |  |  |  |
|  | 60 s |  |  |  |  |  |  |  |  |  |
| 4×4.95 | 5 s | **0.22/0.1s** | **0.41/0.4s** | **3.0/0.7s** | **0.22/0.5s** | 3.55/34.3s | **3.0/4.0s** | 3.55/0.05s | 3.55/20.5s | — |
|  | 10 s |  |  |  |  | 3.55/71.6s |  |  |  |  |
|  | 30 s |  |  |  |  | **3.55/209.2s** |  |  |  |  |
|  | 60 s |  |  |  |  |  |  |  |  |  |
| 4×4grid_avoid | 5 s | **0.21/0.1s** | **0.3/0.2s** | **0.85/0.1s** | **0.21/0.2s** | **0.88/3.3s** | **0.93/9.3s** | — | — | 0.96/346.9s |
|  | 10 s |  |  |  |  |  |  |  |  |  |
|  | 30 s |  |  |  |  |  |  |  |  |  |
|  | 60 s |  |  |  |  |  |  |  |  |  |
| 4×4grid_avoid | 5 s | **UNSAT/0.1s** | **13.63/0.1s** | **4.4/0.2s** | **UNSAT/0.1s** | **3.43/2.8s** | 4.40/17.2s* | — | — | — |
| (p≥0.25, MinR) | 10 s |  |  |  |  |  | 4.40/34.8s* |  |  |  |
|  | 30 s |  |  |  |  |  | 4.21/112.3s* |  |  |  |
|  | 60 s |  |  |  |  |  | 3.95/456.5s* |  |  |  |
| cheese.95 | 5 s | **0.62/0.6s** | **1.2/1.6s** | 1.84/19s* | 3.31/35.3s | 1.2/16.7s | 1.84/34.7s* | 3.40/0.03s | 3.40/13.7s | — |
|  | 10 s |  |  | 1.84/37.7s* | 3.31/72.7s | 2.06/71.7s | 1.84/70.3s* |  |  |  |
|  | 30 s |  |  | 1.84/113.7s* | 3.34/226.2s* | 2.1/222.8s* | 1.84/217s* |  |  |  |
|  | 60 s |  |  | **1.84/162.5s** | 3.34/454.3s* | 2.1/452s* | 1.84/382.4s* |  |  |  |
| cheese.95 | 5 s | **0.40/0.8s** | **0.45/1.9s** | 0.47/19.1s* | **0.40/0.8s** | 0.50/39.8s | 0.47/36.4s* | — | — | — |
| (p≤0.5, MaxR) | 10 s |  |  | 0.47/37.5s* |  | 0.50/77.7s | 0.47/73.3s* |  |  |  |
|  | 30 s |  |  | 0.47/116.2s* |  | 0.50/232.2s | 0.47/229.3s* |  |  |  |
|  | 60 s |  |  | **0.47/148.7s** |  | 0.51/464.6s* | 0.47/370.8s* |  |  |  |
| mini-hall2 | 5 s | **2.43/0.4s** | **2.43/12s** | 2.43/18.1s | 2.5/20.1s | 2.43/29.5s* | 2.43/33.6s | 2.71/0.04s | 2.71/33.8s | — |
|  | 10 s |  |  | 2.43/37.2s | 2.58/38.2s* | 2.43/46.3s* | 2.43/71.2s |  |  |  |
|  | 30 s |  |  | 2.46/114.2s | 2.43/114.2s | 2.43/121.5s* | 2.46/213s |  |  |  |
|  | 60 s |  |  | **2.51/228s** | 2.58/227.9s* | 2.43/235s* | 2.51/434.9s* |  |  |  |
| parr95.95 | 5 s | **6.0/0.2s** | **6.0/0.2s** | **6.0/0.2s** | **6.84/0.5s** | **6.84/0.5s** | **6.84/0.7s** | 6.84/0.02s | 6.84/8.1s | — |
|  | 10 s |  |  |  |  |  |  |  |  |  |
|  | 30 s |  |  |  |  |  |  |  |  |  |
|  | 60 s |  |  |  |  |  |  |  |  |  |
| shuttle.95 | 5 s | **18.0/0.2s** | **18.0/0.4s** | **18.0/1.0s** | 31.25/36.8s* | 31.25/34.1s* | 18.63/18.3s | 31.25/0.05s | 31.25/804s | — |
|  | 10 s |  |  |  | 31.25/74.6s | 31.25/71.1s* | 22.8/67.3s |  |  |  |
|  | 30 s |  |  |  | 31.25/226s* | 31.25/223.2s* | 31.25/217.3s* |  |  |  |
|  | 60 s |  |  |  | 31.25/452s* | 31.25/451.5s* | 31.25/443.6s* |  |  |  |

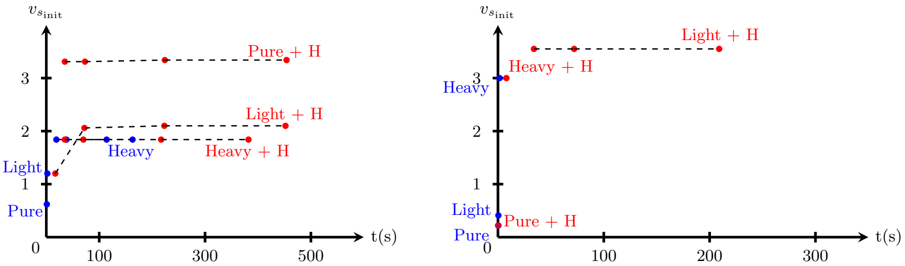
For comparison, we also ran the maximum expected reward benchmarks with the explicit point based POMDP solvers *SARSOP* [6] and *solvePOMDP*[6]. The results for the maximum reachability probability benchmark (4×4grid_avoid) were compared against PRISM-pomdp. All solvers were run using standard parameters. We did not find any solver that could handle the type of multi-objective model checking we implemented for 4×4.95 and cheese.95.

*Results.* Table 1 summarizes our experimental results. The first column has the name of the benchmark – for each benchmark, there are four lines representing the four different timeouts used, as specified in the second column ("TO"). Each entry shows both the result (maximum expected discounted reward or maximum reachability probability) for the initial state of the POMDP, and the run time. For each randomization mode (Pure, Light, Heavy) we show both the result of a single MILP call as well as the result of the state splitting heuristic introduced in Sect. 5, indicated by "+ H" in the column name. For the "+ H" column, the solve time comprises all calls to the solver as well as the time used for splitting states and observations.

Entries printed in **bold** had the same result and runtime as the optimal solution without timeouts, i. e., they were not influenced by the timeouts. In these cases, we omit the entries for higher timeouts, as they had the same values.

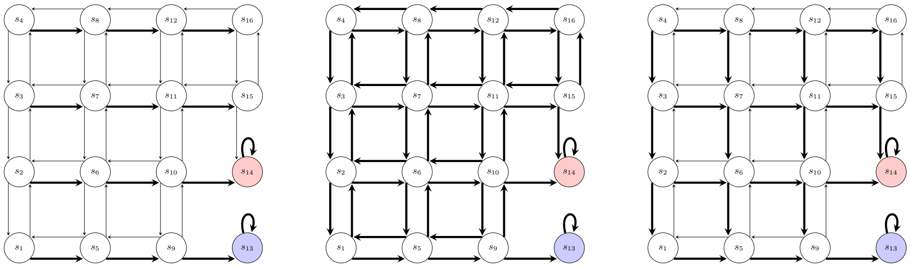
For entries marked with an star (∗), the result is either the same as the optimal

[6] `https://www.erwinwalraven.nl/solvepomdp/`



(a) cheese.95 (b) 4×4.95

Fig. 5: Probability and runtime of the different MILP approaches for different grid world benchmarks



(a) Pure (b) Light (c) Heavy

Fig. 6: Policies computed for the 4×4grid_avoid benchmark with different randomization modes

solution or (for the columns using the splitting heuristic) the same/better than the last iteration that could be solved optimally within two hours. Additionally, the results for cheese.95 and 4×4.95 are also visualized in Fig. 5 – the blue dots represent the results for Pure, Light and Heavy randomization modes without state splitting, while the red dots represent the results when applying the splitting heuristic. Data points that have been produced on the same mode, but using different timeouts, are connected by a dashed line.
Fig. 6 shows the polices computed by the MILP using different randomization modes for the 4×4grid_avoid benchmark.

*Evaluation.* Solving the MILP just once and without any randomization is fast, but doesn't yield a very good result in most cases. However, already the "Light" randomization can improve the result significantly, in same cases up to a factor of 2, without significantly increasing the computation time. Adding the full "Heavy" randomization yields a further improvement in the result – most noticeably for the 4×4.95 benchmark, where the result is improved by factor 7 – but it can also significantly increase the run time of the solver.

While some benchmarks, like 4×4.95 and 4×4grid_avoid, profit immensely from adding randomization, others, like 1d, have an immediate benefit when using preprocessing. parr95.95 and shuttle.95 even achieve the same results as the reference solvers when applying the state splitting heuristic, while randomization had no effect on the results at all. In general, deterministig, history dependend policies are more powerful than stationary, randomized ones and with arbitrary history, randomization can be simulated – e.g., taking an action every second time a state is visited.

All benchmarks can achieve results that are very close to those of the reference solvers. In terms of run time, our approach is slower than SARSOP, but highly outperforms solvePOMDP and PRISM-pomdp.

As can be seen in Fig. 5a, when using different timeouts, the intermediary solution Gurobi returns before fully optimizing the result is already very close to the optimum in many cases. Interestingly, for the mini-hall2 benchmark and the "Pure + H" combination, a timeout of 10 s even yields a better result than a 30 s timeout: the less-optimized result after 10 s causes the heuristic to trigger additional state splits that the benchmark ultimately benefits from.

The same cause can result in the values getting worse when more randomization is added, as seen with the 4×4.95 benchmark. The heuristic picks different states to split, resulting in a higher value for the maximum expected discounted reward in the Light + H case than the Heavy + H approach. However, these additional split states also lead to a higher run time.

For the 4×4grid_avoid benchmark with multi objective model checking, we get UNSAT for the two entries using no randomization, since the required level of reachability probability can not be achieved.

The effects randomization has on a policy are shown in Fig. 6. All sub-figures depict the 4×4grid_avoid benchmark, a grid world with one absorbing goal state (blue) and an absorbing bad state (red). All of the white states share an observation and have four possible actions, although we omit the self-loops that occur when trying to move outside of the grid. The arrows corresponsing to actions chosen under the current policy are drawn **bold**. The system randomly starts in one of the white states. The policy without randomization always chooses to move right – only $s_1$, $s_5$ and $s_9$ can reach the goal state. The policy computed with "Light" randomization enables all actions for all states – now each state has the possibility to reach the goal state, but the probability to get to the bad state is still higher. Only with "Heavy" randomization can the bad state be avoided with a higher probability – each state has equal probability to move down and right, getting the best chance to reach the goal in the lower right corner.

## 7 Conclusion

We introduced a MILP formulation to optimize both reachability probabilities and expected discounted rewards in POMDPs. We used these MILPs to compute optimal stationary deterministic policies and employed the concept of static randomization. Furthermore, we introduced state and observation splitting as preprocessing for a POMDP to locally add history to the otherwise stationary policies. Since blindly splitting states leads to a significant growth of the state space, we proposed a heuristic that iteratively improves solutions by splitting carefully selected states. We show the approaches are competitive to state-of-the-art POMDP solvers, and that MILP formulations for rewards and reachability can easily be combined to find policies that satisfy an arbitrary number of specifications at the same time.

## References

1. Amato, C., Bernstein, D.S., Zilberstein, S.: Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. Autonomous Agents and Multi-Agent Systems **21**(3), 293–320 (2010). https://doi.org/10.1007/s10458-009-9103-z
2. Aras, R., Dutech, A., Charpillet, F.: Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In: ICAPS. pp. 18–25. AAAI (2007), `http://www.aaai.org/Library/ICAPS/2007/icaps07-003.php`
3. Baier, C., Dubslaff, C., Klüppelholz, S.: Trade-off analysis meets probabilistic model checking. In: CSL-LICS. pp. 1:1–1:10. ACM (2014). https://doi.org/10.1145/2603088.2603089
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
5. Braziunas, D.: POMDP solution methods. University of Toronto (2003)
6. Brock, O., Trinkle, J., Ramos, F.: SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Robotics: Science and Systems IV. MIT Press (2009). https://doi.org/10.15607/RSS.2008.IV.009
7. Carr, S., Jansen, N., Wimmer, R., Serban, A.C., Becker, B., Topcu, U.: Counterexample-guided strategy improvement for pomdps using recurrent neural networks. In: IJCAI. pp. 5532–5539. ijcai.org (2019)
8. Chatterjee, K., Chmelík, M., Gupta, R., Kanodia, A.: Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. In: ICRA. pp. 325–330 (2015). https://doi.org/10.1109/ICRA.2015.7139019
9. Chatterjee, K., Chmelík, M., Gupta, R., Kanodia, A.: Optimal cost almost-sure reachability in POMDPs. Artificial Intelligence **234**, 26–48 (2016). https://doi.org/10.1016/j.artint.2016.01.007
10. Chatterjee, K., De Alfaro, L., Henzinger, T.A.: Trading memory for randomness. In: QEST. IEEE (2004). https://doi.org/10.1109/QEST.2004.1348035
11. Chrisman, L.: Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In: AAAI. pp. 183–188. AAAI Press / The MIT Press (1992)
12. Cubuktepe, M., Jansen, N., Junges, S., Katoen, J., Papusha, I., Poonawala, H.A., Topcu, U.: Sequential convex programming for the efficient verification of parametric MDPs. In: TACAS (2). LNCS, vol. 10206, pp. 133–150 (2017)
13. Cubuktepe, M., Jansen, N., Junges, S., Katoen, J., Topcu, U.: Synthesis in pMDPs: A tale of 1001 parameters. In: ATVA. LNCS, vol. 11138, pp. 160–176. Springer (2018)
14. Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.: Fast debugging of PRISM models. In: ATVA. Lecture Notes in Computer Science, vol. 8837, pp. 146–162. Springer (2014)
15. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. Logical Methods in Computer Science **4**(4) (2008). https://doi.org/10.2168/LMCS-4(4:8)2008

16. Givan, R., Dean, T.L., Greig, M.: Equivalence notions and model minimization in markov decision processes. Artificial Intelligence **147**(1-2), 163–223 (2003)
17. Gurobi Optimization, L.: Gurobi optimizer reference manual (2019), `http://www.gurobi.com`
18. Haesaert, S., Nilsson, P., Vasile, C.I., Thakker, R., Agha-mohammadi, A., Ames, A.D., Murray, R.M.: Temporal logic control of POMDPs via label-based stochastic simulation relations. In: ADHS. IFAC-PapersOnLine, vol. 51(16), pp. 271–276. Elsevier (2018)
19. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. Software Tools for Technology Transfer **13**(1), 3–19 (2010)
20. Hauskrecht, M.: Value-function approximations for partially observable Markov decision processes. Journal of Artificial Intelligence Research **13**, 33–94 (2000)
21. Junges, S., Ábrahám, E., Hensel, C., Jansen, N., Katoen, J., Quatmann, T., Volk, M.: Parameter synthesis for markov models. CoRR **abs/1903.07993** (2019)
22. Junges, S., Jansen, N., Wimmer, R., Quatmann, T., Winterer, L., Katoen, J., Becker, B.: Finite-state controllers of POMDPs using parameter synthesis. In: UAI. pp. 519–529. AUAI Press (2018)
23. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence **101**(1), 99–134 (1998)
24. Kumar, A., Mostafa, H., Zilberstein, S.: Dual formulations for optimizing Dec-POMDP controllers. In: ICAPS. pp. 202–210. AAAI Press (2016)
25. Littman, M.L., Topcu, U., Fu, J., Isbell, C., Wen, M., MacGlashan, J.: Environment-independent task specifications via GLTL. arXiv preprint 1704.04341 (2017)
26. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In: AAAI. pp. 541–548. AAAI Press (1999)
27. Meuleau, N., Peshkin, L., Kim, K.E., Kaelbling, L.P.: Learning finite-state controllers for partially observable environments. In: UAI. pp. 427–436. Morgan Kaufmann (1999)
28. Norman, G., Parker, D., Zou, X.: Verification and control of partially observable probabilistic systems. Real-Time Systems **53**(3), 354–402 (2017)
29. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of Markov decision processes. Mathematics of Operations Research **12**(3), 441–450 (1987)
30. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for POMDPs. In: IJCAI. pp. 1025–1032. Morgan Kaufmann (2003)
31. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
32. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley-Interscience (2005)
33. Russell, S.J., Norvig, P.: Artificial Intelligence – A Modern Approach (3. internat. ed.). Pearson Education (2010)
34. Schrijver, A.: Theory of linear and integer programming. Wiley (1999)
35. Shani, G., Pineau, J., Kaplow, R.: A survey of point-based POMDP solvers. Autonomous Agents and Multi-Agent Systems **27**(1), 1–51 (2013)
36. Silver, D., Veness, J.: Monte-carlo planning in large pomdps. In: Lafferty, J.D., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A. (eds.) NIPS. pp. 2164–2172. Curran Associates, Inc. (2010)
37. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. The MIT Press (2005)
38. Velasquez, A.: Steady-state policy synthesis for verifiable control. In: Kraus, S. (ed.) IJCAI. pp. 5653–5661. ijcai.org (2019). https://doi.org/10.24963/ijcai.2019/784

39. Vlassis, N., Littman, M.L., Barber, D.: On the computational complexity of stochastic controller optimization in POMDPs. ACM Trans. on Computation Theory **4**(4), 12:1–12:8 (2012). https://doi.org/10.1145/2382559.2382563
40. Wang, Y., Chaudhuri, S., Kavraki, L.E.: Bounded policy synthesis for pomdps with safe-reachability objectives. In: AAMAS. pp. 238–246. Int'l Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM (2018)
41. Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J.P., Becker, B.: Minimal counterexamples for linear-time probabilistic verification. Theoretical Computer Science **549**, 61–100 (Sep 2014). https://doi.org/10.1016/j.tcs.2014.06.020
42. Winterer, L., Junges, S., Wimmer, R., Jansen, N., Topcu, U., Katoen, J., Becker, B.: Motion planning under partial observability using game-based abstraction. In: CDC. pp. 2201–2208. IEEE (2017)
43. Wongpiromsarn, T., Frazzoli, E.: Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications. In: CDC. pp. 7644–7651. IEEE (2012)