



Fast Computation of Conditional Probabilities in MDPs and Markov Chain Families

Milan Češka¹ , Sebastian Junges² , Luko van der Maas² ,
Filip Macák¹ , and Tim Quatmann³

¹ Brno University of Technology, Brno, Czech Republic

² Radboud University, Nijmegen, The Netherlands

luko.vandermaas@ru.nl

³ RWTH Aachen University, Aachen, Germany



Abstract. Computing optimal conditional reachability probabilities in Markov decision processes (MDPs) is tractable by a reduction to reachability probabilities. Yet, this reduction yields cyclic, challenging MDPs that are often notoriously hard to solve. We present an alternative, practically efficient method to compute optimal conditional reachabilities. This new method is numerically stable, can decide the threshold problem in linear time on acyclic MDPs, and yields performance comparable to standard reachability queries. We also integrate the method in an abstraction-refinement framework to analyse millions of Markov chains at once. We demonstrate the efficacy of the new methods on benchmarks from Bayesian network analysis, probabilistic programs, and runtime monitoring and show speed-ups up to multiple orders of magnitude.

1 Introduction

Markov chains (MCs) are a standard model for stochastic processes. The standard reachability query on Markov chains is, e.g., ‘What is the probability that one reaches an airport before the plane departs?’ As a variation, we study ‘What is the probability that one reaches the airport, if you already know that the buses will have at least a ten-minute delay?’ Such a query is a typical example for *conditional reachability probabilities* [2]. Intuitively, conditional reachability asks for reachability probabilities not over all paths from the initial state, but over paths that match the given evidence. It can be used to implicitly define an alternative initial state, that is obtained from observing a running system. Conditional reachability probabilities have prominent applications: In runtime monitoring, one aims to compute the probability of an imminent threat conditioned on the previous observations by the monitor [34]. In Bayesian networks and MCs derived from them [31], one computes the probability of an event like a break-in conditioned on evidence like a burglary alarm. In probabilistic programs and their MC semantics, one aims to compute the probability of a return value

conditioned on known inputs, e.g., [23]. These examples of conditional reachability probabilities can be efficiently computed [7] on reasonably-sized MCs using modern probabilistic model checking tools [15,22]. In this paper, we bring this performance to Markov decision processes and to Markov chain families by a fresh perspective.

Towards MDPs. Many systems cannot be modelled concisely as a Markov chain: Not all behavior can be captured probabilistically; probabilities may not be known, or the stochastic process must be abstracted by a smaller state space. These observations are standard motivations to add nondeterminism to the model formalism and to thus consider *Markov Decision Processes* (MDPs). The ability to compute optimal (i.e., maximal/minimal) conditional reachability probabilities in MDPs then allows us to bound the conditional reachability probability in models for runtime monitors with nondeterministic agents [24] or with learned probabilities [33], Bayesian networks with interval-valued probabilities [32], or probabilistic programs under abstractions [13] or with explicit nondeterminism [18].

A Bayesian Perspective. Computing conditional reachability probabilities in MCs is straightforward by applying Bayes' theorem: The probability to reach the goal states G conditioned on also reaching evidence states E can be expressed as the fraction of two reachability probabilities: The probability to reach G and E divided by the probability to reach E , i.e.,

$$\Pr(\Diamond G \mid \Diamond E) = \frac{\Pr(\Diamond G \cap \Diamond E)}{\Pr(\Diamond E)}.$$

The upper probability can be computed on the product of a small DFA and the original Markov chain. In MDPs, however, we optimize over the policies that resolve the nondeterminism, which prevents calculating the numerator and denominator independently:

$$\max_{\sigma} \frac{\Pr_{\sigma}(\Diamond G \cap \Diamond E)}{\Pr_{\sigma}(\Diamond E)} \neq \frac{\max_{\sigma} \Pr_{\sigma}(\Diamond G \cap \Diamond E)}{\max_{\sigma} \Pr_{\sigma}(\Diamond E)}.$$

The Restart Method. Baier *et al.* [7] provide a mathematically elegant solution to this challenge, basically mimicking the key idea of rejection sampling. Any probability mass that never satisfies the condition E cannot be reached must be redistributed (i.e., restarted). In terms of the MDP, this means that for all states from which E cannot be reached, the transitions are rewired to the initial state. In these new (restart) MDPs, one simply computes the probability to eventually reach the goal! The downside is that this method introduces a lot of loops, in particular even when the original MDP was acyclic [26]. MDPs with cycles back to the initial states are among the most challenging for methods based on value iteration [19]. Indeed, some(!) restart MDPs are known as very challenging benchmarks of standard reachability queries [20,26].

A Multiobjective Perspective. Intuitively, we consider conditional reachability queries as a variation of a bi-objective model checking problem [14]: Maximize the probability to reach G and E , while minimizing the probability to reach E . Consider the decision problem whether $\max_{\sigma} \Pr_{\sigma}(\Diamond G \mid \Diamond E) \geq \lambda$. Using Bayes' law, some reordering, and some side-conditions detailed below, this can be seen as

$$\max_{\sigma} \Pr_{\sigma}(\Diamond G \cap \Diamond E) - \lambda \Pr_{\sigma}(\Diamond E) \geq 0.$$

This query can be efficiently computed (in practice and theory) using one so-called *total reward query* [15, 30] and is a simple instance of a *relational reachability property* [16]. If the original MDP is acyclic this is possible in linear time with a numerically stable method.

Optimal Conditional Probabilities. The decision method above gives rise to a method to approximate the optimal conditional reachability probability, using a standard bisection method. We accelerate this method using a smarter heuristic. Further, note that we also support computing the optimal conditional reachability probability. The main argument for the bisection to terminate is that the solution can be represented as a reachability probability [7] and that these must be a rational number representable by a bounded number of bits [12]. We therefore adapt a routine that bisects this set of bounded rationals [27] and extend it with termination criteria based on the computed optimal policies.

Towards Markov Chain Families. Beyond MDPs, we are interested in analysing Markov chain families, i.e., large sets of Markov chains. Our interest is twofold: (1) Markov chain families can capture uncertainty about the topology of a system and can capture systems that can be configured in various ways. Such families occur, e.g., when analysing (purely) probabilistic programs with non-deterministic inputs [10, 13]. (2) Given sets of states $E_1, \dots, E_n$, we are interested whether there is some evidence E_i such that $\Pr(\Diamond G \mid \Diamond E_i) \geq \lambda$. This question is natural in runtime monitoring [5, 25], where evidence itself may be uncertain and programs that encode a secret input (e.g., ciphers, see Sect. 6), where we want to ensure a secret cannot be reasonably guessed from any possible evidence.

Embedding into Abstraction-Refinement. We use a standard representation of Markov chain families using *colored MDPs*, where every color-consistent policy of the MDP represents a Markov chain [3]. To solve colored MDPs, we use abstraction-refinement [3], the state-of-the-art method for solving standard reachability queries in colored MDPs. However, the shift to conditional probabilities brings various challenges, that prevent simply applying previous methods: The optimal policies in (standard) MDPs are not memoryless and we only get a value for the initial state. Our experiments show that, nevertheless, abstraction-refinement can bound conditional probabilities on millions of MCs at once.

Related Work. The previous state-of-the-art is the restart method, outlined above, which was experimented with in [26]. The approach in this paper is completely different and rooted in multi-objective MDP model checking. To the best

of our knowledge, the first MDP model checking algorithm for conditional reachability probabilities is presented in [2]: For acyclic MDPs, it suggests a method that computes numerator and denominator under each policy, along with some pruning that is broadly related to multiobjective model checking. However, the approach remains exponential and highly inefficient.

Our approach of tackling the threshold problem and using that as a subroutine also appears in work on conditional expected rewards [8]. Deciding the threshold problem for conditional expected rewards is already PSPACE-complete for acyclic MDPs, in stark contrast to our linear time algorithm. Whereas [8] focusses on establishing this and other complexity bounds, we present a concise algorithm with a focus on (practical) performance: We use value iteration over LPs, we efficiently extract policies, and use efficient bisection methods.

The recent work [13] develops novel probabilistic programming languages (PPLs) to solve various optimization problems over probabilistic inference. It explicitly remarks that computing conditional probabilities on MDPs directly is ill-supported. With this paper, we make reasoning about the operational semantics of PPLs with conditioning more feasible.

Contributions. In summary, this paper yields a significant step forward in computing conditional probabilities on MDPs, building upon progress in (multiobjective) model checking from the last decade. In particular, we provide an empirically superior method for computing optimal conditional reachability probabilities in MDPs rooted in a reduction to a total reward query paired with an optimised bisection method. The method is further equipped with a policy extraction technique that enables an effective integration into the abstraction refinement scheme to solve conditional reachability queries on colored MDPs. Our experiments on benchmarks from several application domains demonstrate speed-ups up to multiple orders of magnitude compared to the restart method [7].

2 Preliminaries

The set of probability distributions over a finite set S is $Dist(S) = \{\mu: S \rightarrow [0, 1] \mid \sum_{s \in S} \mu(s) = 1\}$, $Supp(\mu) = \{s \in S \mid \mu(s) > 0\}$ is the support of $\mu \in Dist(S)$.

Definition 1. A Markov decision process (MDP) is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, s^I)$, where S and Act are finite sets of states and actions, s^I is an initial state, and $\mathbf{P}: S \times Act \rightarrow Dist(S)$ is a (partial) transition probability function such that $Act(s) = \{\alpha \in Act \mid \mathbf{P}(s, \alpha) \text{ is defined}\} \neq \emptyset$ for all $s \in S$.

We fix an MDP $\mathcal{M} = (S, Act, \mathbf{P}, s^I)$. $\|\mathcal{M}\| = |\{(s, \alpha, s') \mid \mathbf{P}(s, \alpha)(s') > 0\}|$ denotes the size of $\mathcal{M}$. An MDP is *acyclic* if all cycles in its transition graph are self-loops. A *path* of MDP $\mathcal{M}$ is an infinite sequence $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots$ with $\alpha_i \in Act(s_i)$ and $s_{i+1} \in Supp(\mathbf{P}(s_i, \alpha_i))$ for all $i \geq 0$. $Paths_{\mathcal{M}}$ denotes the set of paths of $\mathcal{M}$. A finite path $\hat{\pi} = s_0 \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{n-1}} s_n$ is a finite prefix of a path

ending in state $last(\hat{\pi}) = s_n$. For a set of states $G \subseteq S$ we define the set of paths that eventually reach a state in G as

$$\Diamond G = \{s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots \in Paths_{\mathcal{M}} \mid s_i \in G \text{ for some } i \geq 0\}.$$

A *policy* σ for $\mathcal{M}$ maps finite paths $\hat{\pi}$ of $\mathcal{M}$ to enabled actions $\sigma(\hat{\pi}) \in Act(last(\hat{\pi}))$. The set of all policies for $\mathcal{M}$ is denoted as $\Sigma_{\mathcal{M}}$, when $\mathcal{M}$ is clear from context, we also write Σ . A memoryless policy satisfies $\sigma(\hat{\pi}) = \sigma(\hat{\pi}')$ for all finite paths $\hat{\pi}$ and $\hat{\pi}'$ with $last(\hat{\pi}) = last(\hat{\pi}')$. As usual, we also write $\sigma: S \rightarrow Act$ for memoryless policies, mapping last states to actions. MDP $\mathcal{M}$ and policy σ for $\mathcal{M}$ induce a probability measure $\Pr_{\mathcal{M}}^{\sigma}$ on measurable sets of paths defined via a standard cylinder set construction [6]. For two measurable sets of paths $\Pi_1, \Pi_2 \subseteq Paths_{\mathcal{M}}$, the conditional probability of Π_1 given Π_2 is defined as

$$\Pr_{\mathcal{M}}^{\sigma}(\Pi_1 \mid \Pi_2) = \Pr_{\mathcal{M}}^{\sigma}(\Pi_1 \cap \Pi_2) / \Pr_{\mathcal{M}}^{\sigma}(\Pi_2).$$

The probability is undefined iff $\Pr_{\mathcal{M}}^{\sigma}(\Pi_2) = 0$. We define

$$\Pr_{\mathcal{M}}^{\max}(\Pi_1 \mid \Pi_2) = \sup \{ \Pr_{\mathcal{M}}^{\sigma}(\Pi_1 \mid \Pi_2) \mid \sigma \in \Sigma_{\mathcal{M}} \text{ with } \Pr_{\mathcal{M}}^{\sigma}(\Pi_2) \neq 0 \}.$$

$\Pr_{\mathcal{M}}^{\min}(\Pi_1 \mid \Pi_2)$ is defined similarly. When $\mathcal{M}$ is clear from context and $s \in S$ is a state of $\mathcal{M}$, we also write $\Pr_s^{\sigma}$, $\Pr_s^{\max}$, and $\Pr_s^{\min}$ for the corresponding probability measures that arise from replacing the initial state of $\mathcal{M}$ by s .

A *reward function* $\mathcal{R}: S \times Act \times S \rightarrow \mathbb{R}$ for MDP $\mathcal{M}$ assigns a reward to transitions. The reward accumulated on a path $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots$ until reaching a set $T \subseteq S$ is given by $\mathcal{R}_{\Diamond T}(\pi) = \sum_{0 \leq i < \min\{n \in \mathbb{N} \mid s_n \in T\}} \mathcal{R}(s_i, \alpha_i, s_{i+1})$.¹ The *expected total reward* of $\mathcal{M}$ for $\mathcal{R}_{\Diamond T}$ under a policy $\sigma \in \Sigma_{\mathcal{M}}$ is the expected value of the accumulated reward: $\text{Ex}_{\mathcal{M}}^{\sigma}(\mathcal{R}_{\Diamond T}) = \int_{\pi} \mathcal{R}_{\Diamond T}(\pi) d\Pr_{\mathcal{M}}^{\sigma}(\pi)$. As for $\Pr_{\mathcal{M}}^{\sigma}$, we write $\text{Ex}_{\mathcal{M}}^{\max}$ and $\text{Ex}_{\mathcal{M}}^{\min}$ for the *maximal* expected values (assuming initial state $s \in S$). The values $\text{Ex}_s^{\max}(\mathcal{R}_{\Diamond T})$ for each $s \in S$ coincide with the least fixed point $\mathbf{x}: S \rightarrow \mathbb{R}$ of the *Bellman operator* $\Phi: (S \rightarrow \mathbb{R}) \rightarrow (S \rightarrow \mathbb{R})$ given by $\Phi(\mathbf{x})(s) = \max_{\alpha \in Act(s)} \sum_{s' \in S} \mathbf{P}(s, \alpha, s') \cdot (\mathbf{x}(s') + \mathcal{R}(s, \alpha, s'))$ for $s \notin T$ and $\Phi(\mathbf{x})(s) = 0$ for $s \in T$. Common algorithms for computing $\text{Ex}_s^{\max}(\mathcal{R})$ for each $s \in S$ are based on linear programming, policy iteration, and value iteration. When mixtures of positive and negative rewards are present (as in our case), these algorithms commonly require collapsing end components as a preprocessing to ensure a *unique* fixed point of the Bellman operator [1].

3 Deciding Conditional Reachability Properties

This section presents our new approach for answering the following problem.

¹ In our setting, this sum always converges to a finite value. Generally, it might be infinite or undefined—due to possible mixtures of positive and negative rewards.

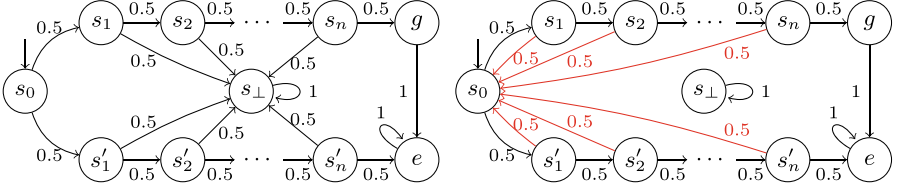


Fig. 1. Example MDPs $\mathcal{M}_1$ (left) and $\mathcal{M}'_1$ (right).

Problem 1: Conditional Reachability Threshold Problem

Given an MDP $\mathcal{M}$ with sets of states $G, E \subseteq S$ and a threshold $\lambda \in [0, 1]$, decide whether $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \leq \lambda$.

For the rest of the section, we fix an MDP $\mathcal{M}$ with state sets $G, E \subseteq S$. We assume that the conditional probability $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E)$ is defined, i.e., $\Pr_{\mathcal{M}}^{\sigma}(\Diamond E) > 0$ for some $\sigma \in \Sigma_{\mathcal{M}}$. This assumption can be checked using standard qualitative MDP model checking algorithms [6]. The restart method outlined in [7] provides a polynomial time algorithm for computing $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E)$ and thus for solving Problem 1. Roughly, the approach constructs a new MDP $\mathcal{M}'$ from $\mathcal{M}$, where transitions leading to states that do not reach the condition E are rewired to the initial state. The conditional reachability probability $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \leq \lambda$ in $\mathcal{M}$ coincides with the (unconditional) reachability probability $\Pr_{\mathcal{M}'}^{\max}(\Diamond G)$ in $\mathcal{M}'$. The latter can be computed using standard MDP model checking methods. Since $\mathcal{M}'$ has approximately the same size as $\mathcal{M}$, the linear programming approach yields a polynomial time algorithm for Problem 1.

Example 1. Fig. 1 shows an *acyclic* MDP $\mathcal{M}_1$ and the corresponding MDP $\mathcal{M}'_1$ that arises from applying the restart method for the conditional probability $\Pr_{\mathcal{M}}^{\max}(\Diamond\{g\} \mid \Diamond\{e\})$. In $\mathcal{M}'_1$, transitions that are rewired to the initial state are depicted in red. In $\mathcal{M}_1$, these transitions lead to $s_\perp$ which cannot reach e . We have $\Pr_{\mathcal{M}}^{\max}(\Diamond\{g\} \mid \Diamond\{e\}) = \Pr_{\mathcal{M}'_1}^{\max}(\Diamond\{g\}) = 0.5$. However, computing the reachability probability in $\mathcal{M}'_1$ is challenging due to its cyclic structure. A similar MDP is used in [19, Fig. 3] to illustrate convergence issues of value iteration.

We present an alternative approach for answering Problem 1 that does not introduce loops into the MDP structure. Let $\Sigma_{\mathcal{M}, E} = \{\sigma \in \Sigma \mid \Pr_{\mathcal{M}}^{\sigma}(\Diamond E) > 0\}$ denote the set of policies that reach E with positive probability. Our procedure is based on the following key observation (proof in [35]).

Theorem 1. For $\lambda \in [0, 1]$ and $\sim \in \{<, \leq, =, \geq, >\}$:

$$\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \sim \lambda \quad \text{iff} \quad \max_{\sigma \in \Sigma_{\mathcal{M}, E}} \left(\Pr_{\mathcal{M}}^{\sigma}(\Diamond G \cap \Diamond E) - \lambda \cdot \Pr_{\mathcal{M}}^{\sigma}(\Diamond E) \right) \sim 0.$$

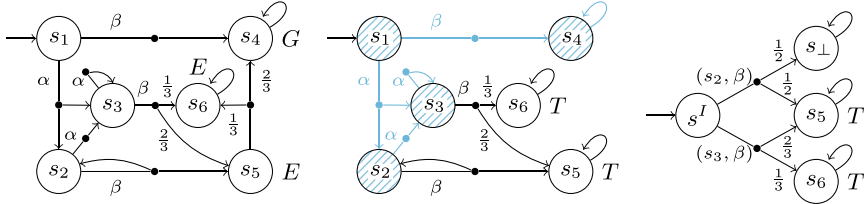


Fig. 2. Example MDPs $\mathcal{M}_2$ (left), $\mathcal{M}_{2,\circ}$ (middle), $\tilde{\mathcal{M}}_2$ (right). The initial component is shown in blue, together with all their actions that are not exits. All unmarked transition probabilities are uniformly distributed over all transitions of an action.

Using Theorem 1, our approach is to answer Problem 1 by computing the value

$$V(\lambda) := \max_{\sigma \in \Sigma_{\mathcal{M},E}} V^\sigma(\lambda) \quad \text{with} \quad V^\sigma(\lambda) := \left(\Pr_{\mathcal{M}}^\sigma(\Diamond G \cap \Diamond E) - \lambda \cdot \Pr_{\mathcal{M}}^\sigma(\Diamond E) \right)$$

and comparing it against 0. The computation of $V(\lambda)$ is done in two steps. (1) Restate $V^\sigma(\lambda)$ as an expected total reward. (2) Construct an equivalent MDP $\mathcal{M}'$ for which the set E is reached with positive probability under all policies (i.e., $\Sigma_{\mathcal{M}',E} = \Sigma_{\mathcal{M}'}$). After step (2), standard MDP algorithms for expected total rewards can be employed to decide $V(\lambda) \leq 0$.

Step 1: Reward Characterization. The value $V(\lambda)$ above is characterized using an expected total reward for the reward function $\mathcal{R}^\lambda$ for $\lambda \in [0, 1]$, given as:

$$\mathcal{R}^\lambda(s, \alpha, s') = \begin{cases} \Pr_{s'}^{\max}(\Diamond G) - \lambda & \text{if } s' \in E \\ \Pr_{s'}^{\max}(\Diamond E) \cdot (1 - \lambda) & \text{if } s' \in G \setminus E \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, once a state in E is reached, a policy σ that maximizes $V^\sigma(\lambda)$ as above must maximize the probability to also reach G . Similarly, once a state in G is reached, such a policy maximizes the probability to also reach E .

Lemma 1. $V(\lambda) = \max_{\sigma \in \Sigma_{\mathcal{M},E}} \text{Ex}_{\mathcal{M}}^\sigma(\mathcal{R}_{\Diamond G \cup E}^\lambda)$

Proof (sketch). We can write $V^\sigma(\lambda)$ as an expected value of a function f^λ with

$$f^\lambda(\pi) = [\pi \in \Diamond G \cap \Diamond E] - \lambda \cdot [\pi \in \Diamond E].$$

The (maximal) expected values of f^λ and $\mathcal{R}_{\Diamond G \cup E}^\lambda$ coincide.

We now exemplify how we apply the reward characterization to solve Problem 1.

Example 2. Consider the example in Fig. 2, for which we solve an instance of Problem 1 with the property $\Pr_{\mathcal{M}_2}^{\max}(\Diamond\{s_4\} \mid \Diamond\{s_5, s_6\}) \leq 1/2$. By Theorem 1 we need to compute $V(1/2)$. Thus, we consider the reward function $\mathcal{R}^{1/2}$ on $\mathcal{M}_2$. By definition we must compute the rewards for transitions into $E = \{s_5, s_6\}$, these gain a reward of $\Pr_{s_5}^{\max}(\Diamond\{s_4\}) - 1/2 = 2/3 - 1/2 = 1/6$, $\Pr_{s_6}^{\max}(\Diamond\{s_4\}) = 0 - 1/2 = -1/2$ respectively, and transitions into $G = \{s_4\}$: $\Pr_{s_4}^{\max}(\Diamond\{s_5, s_6\}) \cdot (1 - 1/2) = 0 \cdot 1/2 = 0$. The memoryless policy $\sigma(s_1) = \alpha, \sigma(s_2) = \beta, \sigma(s_3) = \alpha$ has maximal reward $\text{Ex}_{\mathcal{M}_2}^{\sigma}(\mathcal{R}_{\Diamond\{s_4, s_5, s_6\}}^{\lambda=1/2}) = 1/12$. The policy σ reaches E with positive probability, thus $\sigma \in \Sigma_{\mathcal{M}_2, E}$, and since $\text{Ex}_{\mathcal{M}_2}^{\sigma}(\mathcal{R}_{\Diamond\{s_4, s_5, s_6\}}^{\lambda=1/2}) > 0$, the property does not hold (Theorem 1).

Step 2a: Preprocessing $\mathcal{M}$. The example above does not yet discuss a method to find a policy $\sigma \in \Sigma_{\mathcal{M}, E}$. We first simplify the expected total reward query by considering the MDP $\mathcal{M}_{\odot}$ obtained from $\mathcal{M}$ by making the states in $G \cup E$ absorbing, i.e., all outgoing transitions of each state $s \in G \cup E$ are replaced by a self-loop $s \xrightarrow{\perp} s$. For ease of notation, reward functions $\mathcal{R}$ for $\mathcal{M}$ also apply to $\mathcal{M}_{\odot}$, where $\mathcal{R}(s, \perp, s) = 0$ for all $s \in G \cup E$. On $\mathcal{M}_{\odot}$, we consider a new set of terminal states

$$T := E \cup \{s \in G \mid \Pr_s^{\max}(\Diamond E) > 0\}.$$

Intuitively, paths of $\mathcal{M}$ that reach a state in $G \setminus T$ (without visiting a state in E before) have reward 0 and do not reach E .

Example 3. Consider $\mathcal{M}_{2, \odot}$ in Fig. 2. We have $T = \{s_5, s_6\}$. The goal state $s_4 \in G$ is not part of T , since it has no path to the evidence set $E = \{s_5, s_6\}$.

The following lemma establishes that $V(\lambda)$ can be obtained by computing (i) the non-zero rewards of $\mathcal{R}^{\lambda}$ and (ii) an optimal expected total reward on $\mathcal{M}_{\odot}$. The former can be derived using two standard MDP model checking calls to obtain the probabilities $\Pr_s^{\max}(\Diamond G)$ for each $s \in E$ and $\Pr_s^{\max}(\Diamond E)$ for each $s \in G \setminus E$.

Lemma 2. $V(\lambda) = \max_{\sigma \in \Sigma_{\mathcal{M}_{\odot}, T}} \text{Ex}_{\mathcal{M}_{\odot}}^{\sigma}(\mathcal{R}_{\Diamond T}^{\lambda})$

Proof (Sketch). Given a policy $\sigma \in \Sigma_{\mathcal{M}, E}$, we obtain a policy $\sigma' \in \Sigma_{\mathcal{M}_{\odot}, T}$ that mimics σ until an absorbing state $s \in G \cup E$ is reached. Similarly, for a policy $\sigma' \in \Sigma_{\mathcal{M}_{\odot}, T}$ a policy $\sigma \in \Sigma_{\mathcal{M}, E}$ is constructed that mimics σ' until a state in $G \cup E$ is reached from which point on the policy σ mimics a policy that maximizes the probability to reach E . In both cases, we have $\text{Ex}_{\mathcal{M}}^{\sigma}(\mathcal{R}_{\Diamond G \cup E}^{\lambda}) = \text{Ex}_{\mathcal{M}_{\odot}}^{\sigma'}(\mathcal{R}_{\Diamond T}^{\lambda})$.

Remark 1 (Minimal Conditional Probabilities). Our approach can be straightforwardly generalized to minimal conditional probabilities $\Pr_{\mathcal{M}}^{\min}(\Diamond G \mid \Diamond E)$ except one edge case: For the equivalent of Lemma 2, we might have $\Sigma_{\mathcal{M}_{\odot}, T} = \emptyset$ even if $\Sigma_{\mathcal{M}, E} \neq \emptyset$. This case occurs if all paths that reach E visit a state in $G \setminus T = \{s \in G \mid \Pr_s^{\min}(\Diamond E) = 0\}$ before. If this holds, we get $\Pr_{\mathcal{M}}^{\min}(\Diamond G \mid \Diamond E) = 1$.

Step 2b: Discard Policies with Undefined Value. Using our processing from Step 2a, we may assume an MDP $\mathcal{M}$ with terminal states $T \subseteq S$, all absorbing, and a reward function $\mathcal{R}^\lambda: S \times \text{Act} \times S \rightarrow \mathbb{R}$ where $\mathcal{R}^\lambda(s, \alpha, s') \neq 0$ implies $s' \in T$. We now provide an approach to decide whether $\max_{\sigma \in \Sigma_{\mathcal{M}, T}} \text{Ex}_{\mathcal{M}}^\sigma(\mathcal{R}_{\diamond T}^\lambda) \sim 0$ for $\sim \in \{<, \leq, =, \geq, >\}$. Our procedure then allows to decide Problem 1 using Lemma 2 and Theorem 1. The challenge is to limit the expected total reward analysis to policies $\sigma \in \Sigma_{\mathcal{M}, T}$ that reach T with non-zero probability.

For a policy $\sigma \in \Sigma_{\mathcal{M}}$, the set $\text{Reach}_{\mathcal{M}}(\sigma) = \{s \in S \mid \Pr_{\mathcal{M}}^\sigma(\diamond\{s\}) > 0\}$ contains states that are reachable from the initial state s^I of $\mathcal{M}$ under σ with non-zero probability. For $\sigma \in \Sigma_{\mathcal{M}} \setminus \Sigma_{\mathcal{M}, T}$, the set T is reached with probability 0, i.e., $\text{Reach}_{\mathcal{M}}(\sigma) \cap T = \emptyset$. Furthermore, for each state $s \in \text{Reach}_{\mathcal{M}}(\sigma)$, there is a policy under which T is reached from s with probability 0, yielding

$$C^I := \bigcup_{\sigma \in \Sigma_{\mathcal{M}} \setminus \Sigma_{\mathcal{M}, T}} \text{Reach}_{\mathcal{M}}(\sigma) \subseteq \{s \in S \mid \Pr_s^{\min}(\diamond T) = 0\}.$$

We call the set $C^I \subseteq S$ the *initial component* of $\mathcal{M}$ and observe that strategies reaching T must exit C^I eventually. We emphasise that C^I does not need to be a strongly connected component and is thus not a (traditional) end component.

Example 4. Consider $\mathcal{M}_{2, \odot}$ from Fig. 2. The initial component of $\mathcal{M}_{2, \odot}$ with states $C^I = \{s_1, s_2, s_3, s_4\}$ is highlighted. It is not strongly connected. For each of its states there exists a policy which stays inside C^I . For example for state s_2 , the policy $\sigma(s_2) = \alpha, \sigma(s_3) = \alpha$ never reaches a state in $T = \{s_5, s_6\}$.

Lemma 3. *For all policies $\sigma \in \Sigma_{\mathcal{M}}$:*

$$\Pr_{\mathcal{M}}^\sigma(\diamond T) > 0 \quad \text{iff} \quad \sigma \in \Sigma_{T, \mathcal{M}} \quad \text{iff} \quad \Pr_{\mathcal{M}}^\sigma(\diamond(S \setminus C^I)) > 0.$$

Proof. The first equivalence is by definition of $\Sigma_{T, \mathcal{M}}$. Moreover, $\Pr_{\mathcal{M}}^\sigma(\diamond T) > 0$ implies $\Pr_{\mathcal{M}}^\sigma(\diamond(S \setminus C^I)) > 0$ because C^I does not contain any state in T , i.e., $T \subseteq (S \setminus C^I)$. Finally, $\Pr_{\mathcal{M}}^\sigma(\diamond T) = 0$ implies $\Pr_{\mathcal{M}}^\sigma(\diamond(S \setminus C^I)) = 0$ because for $\sigma \notin \Sigma_{T, \mathcal{M}}$ we have $\text{Reach}_{\mathcal{M}}(\sigma) \subseteq C^I$ by definition of C^I and $\Pr_{\mathcal{M}}^\sigma(\diamond\{s\}) = 0$ for all $s \in (S \setminus \text{Reach}_{\mathcal{M}}(\sigma)) \supseteq (S \setminus C^I)$.

If $C^I = \emptyset$, there is no policy $\sigma \in \Sigma_{\mathcal{M}} \setminus \Sigma_{\mathcal{M}, T}$, i.e. $\Sigma_{\mathcal{M}, T} = \Sigma_{\mathcal{M}}$. We assume $C^I \neq \emptyset$. Lemma 3 yields that $\sigma \in \Sigma_{T, \mathcal{M}}$ iff there is some state $s \in C^I \cap \text{Reach}_{\mathcal{M}}(\sigma)$ via which the initial component is exited, i.e. $(s, \sigma(s)) \in \text{Exits}(C^I)$ for

$$\text{Exits}(C^I) := \{(s, \alpha) \in S \times \text{Act} \mid s \in C^I, \alpha \in \text{Act}(s), \text{ and } \text{Supp}(\mathbf{P}(s, \alpha)) \not\subseteq C^I\}.$$

We construct a new MDP $\tilde{\mathcal{M}}$ from $\mathcal{M}$ where the choice of an exiting state-action pair is explicit, while ensuring that the maximal expected reward is positive in $\mathcal{M}$ iff it is positive in $\tilde{\mathcal{M}}$.

Definition 2. The elimination of the initial component $C^I \neq \emptyset$ of $\mathcal{M}$ yields the MDP $\tilde{\mathcal{M}} = (\tilde{S}=(S \setminus C^I) \cup \{s^I, s_\perp\}, \text{Act} \cup \text{Exits}(C^I) \cup \{\perp\}, \tilde{\mathbf{P}}, s^I)$, where $\tilde{\mathbf{P}}$ is defined as follows—using $\mu_{s,\alpha} \in \text{Dist}(\tilde{S})$ with $\mu_{s,\alpha}(s') = \mathbf{P}(s, \alpha)(s')$ for $s' \in S \setminus C^I$ and $\mu_{s,\alpha}(s_\perp) = \sum_{s' \in C^I} \mathbf{P}(s, \alpha, s')$:

$$\tilde{\mathbf{P}}(s, \alpha) = \begin{cases} \mu_{s,\alpha} & \text{if } s \in S \setminus C^I, \alpha \in \text{Act}, \\ \mu_{s_e, \alpha_e} & \text{if } s = s^I, \alpha = (s_e, \alpha_e) \in \text{Exits}(C^I). \\ \{s_\perp \mapsto 1\} & \text{if } s = s_\perp, \alpha = \perp. \end{cases}$$

The reward function $\mathcal{R}^\lambda$ for $\mathcal{M}$ is straightforwardly applied to $\tilde{\mathcal{M}}$. In particular, $\mathcal{R}(s, \alpha, s')$ only depends on s' and only assigns non-zero values for $s' \in S \setminus C^I$.

Example 5. The initial component C^I of $\mathcal{M}_{2,\odot}$ from Fig. 2 (middle) has two exits: (s_2, β) and (s_3, β) . Eliminating C^I results in $\tilde{\mathcal{M}}_2$ in Fig. 2 (right). The transition probabilities of (s_2, β) are modified such that the probability of going back into the initial component is instead redirected to the state $s_\perp$.

Lemma 4. For $\sim \in \{<, \leq, =, \geq, >\}$:

$$\max_{\sigma \in \Sigma_{\mathcal{M},T}} \text{Ex}_{\mathcal{M}}^\sigma(\mathcal{R}_{\diamond T}^\lambda) \sim 0 \quad \text{iff} \quad \max_{\tilde{\sigma} \in \Sigma_{\tilde{\mathcal{M}}}} \text{Ex}_{\tilde{\mathcal{M}}}^{\tilde{\sigma}}(\mathcal{R}_{\diamond T}^\lambda) \sim 0.$$

Proof (Sketch). For the implication, let $\sigma \in \Sigma_{\mathcal{M},T}$ be a policy of $\mathcal{M}$ that maximises the expected reward $\text{Ex}_{\mathcal{M}}^\sigma(\mathcal{R}_{\diamond T}^\lambda)$ among all policies in $\Sigma_{\mathcal{M},T}$. We construct a policy $\tilde{\sigma}$ of $\tilde{\mathcal{M}}$ such that the comparisons of the induced expected rewards to zero coincide. To see this, consider a case distinction on $\text{val}(s, \alpha)$ defined for $(s, \alpha) \in \text{Exits}(C^I)$ as $\text{val}(s, \alpha) = \sum_{s' \in S \setminus C^I} \mathbf{P}(s, \alpha, s') \cdot (\text{Ex}_{s'}^{\max}(\mathcal{R}_{\diamond T}^\lambda) + \mathcal{R}^\lambda(s, \alpha, s'))$:

- If $\text{val}(s, \alpha) > 0$ for some $(s, \alpha) \in \text{Exits}(C^I)$, then the maximal expected reward in $\mathcal{M}$ is positive (as σ is optimal, it will prefer staying inside the initial component C^I and collect zero reward over picking an exit with negative value). Policy $\tilde{\sigma}$ for $\tilde{\mathcal{M}}$ mimics this by picking that exit in s^I .
- If $\text{val}(s', \alpha') \leq 0$ for all $(s', \alpha') \in \text{Exits}(C^I)$ and $\text{val}(s, \alpha) = 0$ for some $(s, \alpha) \in \text{Exits}(C^I)$, the maximal expected reward in $\mathcal{M}$ is 0 due to a similar argument and $\tilde{\sigma}$ for $\tilde{\mathcal{M}}$ can select an action at s^I with zero value.
- Otherwise, if $\text{val}(s, \alpha) < 0$ for all $(s, \alpha) \in \text{Exits}(C^I)$, all policies induce a negative value and the choice of $\tilde{\sigma}$ does not matter.

For the converse, any policy $\tilde{\sigma} \in \Sigma_{\tilde{\mathcal{M}}}$ can be converted to a policy $\sigma \in \Sigma_{\mathcal{M},T}$ such that the comparisons of the induced expected rewards to zero coincide. In particular, if $\tilde{\sigma}(s^I) = (s, \alpha)$, then σ mimics this by only exiting C^I via the exit (s, α) . Paths of $\mathcal{M}$ that do not take that exit gain zero reward.

Example 6. Recall the reward function $\mathcal{R}^\lambda$ for $\lambda = 1/2$ of $\mathcal{M}_{2,\odot}$ from Example 2. Applying $\mathcal{R}^\lambda$ on $\tilde{\mathcal{M}}_2$ gives a reward of $1/6$ to the transitions into s_5 , and a reward of $-1/2$ to the transitions into s_6 . The maximum expected reward is attained by the policy $\sigma(s^I) = (s_2, \beta)$, which induces $\text{Ex}_{\mathcal{M}_2}^\sigma(\mathcal{R}_{\diamond \{s_5, s_6\}}^\lambda) = 1/3$. While this value is not equal to the value found in Example 2, it is on the same side of zero.

Efficiently Handling Acyclic MDPs. For acyclic MDPs, expected total rewards and reachability probabilities can be computed in linear time with dynamic programming [29]. Our constructions preserve acyclicity.

Proposition 1. *For acyclic MDPs, Problem 1 can be solved in $\mathcal{O}(\|\mathcal{M}\|)$ time.*

Policy Extraction. Applications—for example planning problems, colored MDPs (Sect. 5), and our policy tracking optimization from Sect. 4—often require extracting an optimal policy on $\mathcal{M}$ which witnesses $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \sim \lambda$, i.e., $V(\lambda) \sim 0$. In contrast to standard reachability probabilities, memoryless policies are not sufficient for conditional reachability [2]. Inspired by [2, 7], we split the behavior of the policy into three memory states: i) neither G nor E states were reached yet, ii) some state $s \in G$ was reached, and iii) some state $s \in E$ was reached. Intuitively, in ii) and iii) we seek the optimal policy for query $\Pr_{\mathcal{M}}^{\max}(\Diamond E)$ and $\Pr_{\mathcal{M}}^{\max}(\Diamond G)$, respectively. In case i), we directly use $\tilde{\sigma}(s)$ obtained by solving $\tilde{\mathcal{M}}$ for states $s \notin C^I$. Finally, we construct the policy for states in C^I . The action $\tilde{\sigma}(s^I)$ defines the desired initial component exit, i.e., if $\tilde{\sigma}(s^I) = (s_e, \alpha_e)$, then the constructed policy chooses α_e in state s_e . For the other states in C^I we compute a policy that reaches s_e with non-zero probability while only taking actions that stay in C^I , which can be implemented by standard graph algorithms.

Example 7. Continuing from Example 6, the policy $\tilde{\sigma}(s^I) = (s_2, \beta)$ induced the maximum value of $V^{(1/2)}$ in $\tilde{\mathcal{M}}_2$. The policy for $\mathcal{M}_2$ will have three memory states: $\sigma' : \{m^\emptyset, m^E, m^G\} \rightarrow S \rightarrow \text{Act}$. $\sigma'(m^E)$ and $\sigma'(m^G)$ (case ii and iii) take the only available action in the goal and evidence states. For $\sigma'(m^\emptyset)$ (case i), we assign $\sigma'(m^\emptyset)(s_2) = \beta$ as it is the chosen exit. The other initial component states get assigned the actions $\sigma'(m^\emptyset)(s_1) = \alpha$ and $\sigma'(m^\emptyset)(s_3) = \alpha$ since σ' needs to reach s_2 and otherwise stay in C^I to avoid reaching an evidence state.

4 Computing Optimal Conditional Reachability

This section concerns optimal conditional reachability probabilities:

Problem 2: Conditional Reachability Optimization Problem

Given an MDP $\mathcal{M}$ with sets of states $G, E \subseteq S$ and a precision $\varepsilon \in [0, 1]$, compute a value λ with $|\lambda - \Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E)| \leq \varepsilon$.

Our approach for Problem 2 embeds the idea of Problem 1 into a bisection algorithm outlined in Algorithm 1. The algorithm maintains a lower bound ℓ and an upper bound u on the optimal conditional probability. In each iteration, we check $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \sim \lambda$ for $\sim \in \{\leq, \geq\}$ and the midpoint $\lambda = (\ell + u)/2$ using our techniques from Sect. 3. The bounds ℓ and u are updated accordingly, halving the width of the interval $[\ell, u]$ in each step.

Proposition 2. *Algorithm 1 answers Problem 2 and for $\varepsilon > 0$ terminates within $\lceil \log_2(1/(2 \cdot \varepsilon)) \rceil$ iterations.*

Input : MDP $\mathcal{M}$ with $G, E \subseteq S$, $\varepsilon \in [0, 1]$
Output : $\lambda \in [0, 1]$ with $|\lambda - \Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E)| \leq \varepsilon$
1 $\ell \leftarrow 0$; $u \leftarrow 1$ // *Invariant: $\ell \leq \Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \leq u$*
2 **while** $(u - \ell)/2 > \varepsilon$ **do**
3 $\lambda \leftarrow (\ell + u)/2$ // *current candidate*
 // *using $V(\lambda) = \max_{\sigma \in \Sigma_{\mathcal{M}, E}} (\Pr_{\mathcal{M}}^{\sigma}(\Diamond G \cap \Diamond E) - \lambda \cdot \Pr_{\mathcal{M}}^{\sigma}(\Diamond E))$*
4 **if** $V(\lambda) \geq 0$ **then** $\ell \leftarrow \lambda$ // *λ is a lower bound*
5 **if** $V(\lambda) \leq 0$ **then** $u \leftarrow \lambda$ // *λ is an upper bound*
6 **return** $(\ell + u)/2$

Algorithm 1: Bisection for Computing Optimal Conditional Probabilities

We discuss three modifications of the bisection algorithm. The first handles exact computations, i.e., $\varepsilon = 0$. The latter two aim to accelerate convergence. All three modifications can be combined. Section 6 empirically compares them.

Exact Computation. We exploit that $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E)$ is a rational number to enable exact computations. Specifically, we adapt the candidate selection in Line 3 and pick the rational in the interval $[\ell, u]$ with minimal denominator instead. The resulting procedure yields a traversal of the *Stern-Brocot tree* [17] which allows us to find every rational in finitely many steps. To ensure progress, our implementation decides heuristically in each iteration whether (i) a candidate close to the middle of the search interval $[\ell, u]$ is taken or (ii) whether a candidate in $[\ell, u]$ with minimal denominator is taken.

Terminating Early Through Policy Tracking For $\sigma \in \Sigma_{\mathcal{M}, E}$, we write $V^{\sigma}(\lambda) = \Pr_{\mathcal{M}}^{\sigma}(\Diamond G \cap \Diamond E) - \lambda \cdot \Pr_{\mathcal{M}}^{\sigma}(\Diamond E)$ as in Sect. 3. We say that σ induces $V(\lambda)$ if $V^{\sigma}(\lambda) = V(\lambda)$. The next modification is based on the following observation:

Lemma 5. *Let $0 \leq \ell \leq u \leq 1$. A policy $\sigma \in \Sigma_{\mathcal{M}, E}$ that induces $V(\ell)$ and $V(u)$ also induces $V(\lambda)$ for all $\lambda \in [\ell, u]$.*

Proof. For any $\sigma' \in \Sigma_{\mathcal{M}, E}$ we have $V^{\sigma'}(\ell) \leq V^{\sigma}(\ell)$ and $V^{\sigma'}(u) \leq V^{\sigma}(u)$. Since $V^{\sigma'}(\lambda)$ and $V^{\sigma}(\lambda)$ are linear in λ , this yields $V^{\sigma'}(\lambda) \leq V^{\sigma}(\lambda)$ for all $\lambda \in [\ell, u]$.

When updating the bounds ℓ and u in Lines 4 and 5, our modification of Algorithm 1 keeps track of a policy σ_{ℓ} that induces $V(\ell)$ and a policy σ_u that induces $V(u)$. If both policies coincide, they induce the optimal conditional probability. In particular, Lemma 5 yields that $\sigma^* = \sigma_{\ell} = \sigma_u$ induces the optimal value $V^{\sigma}(\lambda^*) = V(\lambda^*)$ for $\lambda^* = \Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \in [\ell, u]$. With Theorem 1, we get $V^{\sigma^*}(\lambda^*) = 0$, yielding

$$\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) = \lambda^* = \frac{\Pr_{\mathcal{M}}^{\sigma^*}(\Diamond G \cap \Diamond E)}{\Pr_{\mathcal{M}}^{\sigma^*}(\Diamond E)} = \Pr_{\mathcal{M}}^{\sigma^*}(\Diamond G \mid \Diamond E).$$

We finally evaluate the found policy σ^* —by computing the above fraction—to obtain the resulting conditional probability.

Example 8. We compute $\Pr_{\mathcal{M}_2}^{\max}(\Diamond\{s_4\} \mid \Diamond\{s_5, s_6\})$ for MDP $\mathcal{M}$ from Fig. 2 using policy tracking bisection. The bisection algorithm starts with the bounds $[0, 1]$ and no upper or lower bound policies. The current candidate λ becomes $1/2$. As seen in Example 3, $V(1/2) \geq 0$, thus the lower bound becomes $1/2$, and the lower bound policy is $\sigma_\ell(s_1) = \alpha, \sigma_\ell(s_2) = \beta, \sigma_\ell(s_3) = \alpha$. Next, the candidate becomes $\lambda = 3/4$ and $V(3/4) < 0$. The upper bound is updated to $3/4$ and the upper bound policy turns out to be equal to the lower bound policy $\sigma_u = \sigma_\ell$. Thus, we can apply Lemma 5 with $\sigma^* = \sigma_\ell = \sigma_u$ and calculate $\Pr_{\mathcal{M}_2}^{\sigma^*}(\Diamond\{s_4\} \mid \Diamond\{s_5, s_6\}) = \frac{1/3}{1/2} = 2/3$.

Advanced Bisection Bounds. The following results allow us to establish tighter bounds on the conditional probability based on the values $V(\lambda)$.

Lemma 6. *Let $\lambda \in [0, 1]$. If $V(\lambda) \geq 0$ then*

$$\lambda + V(\lambda)/\Pr_{\mathcal{M}}^{\max}(\Diamond E) \leq \Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \leq \lambda + V(\lambda)/\Pr_{\mathcal{M}}^{\min}(\Diamond E).$$

If $V(\lambda) \leq 0$ then

$$\lambda + V(\lambda)/\Pr_{\mathcal{M}}^{\min}(\Diamond E) \leq \Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) \leq \lambda + V(\lambda)/\Pr_{\mathcal{M}}^{\max}(\Diamond E).$$

The proof is in [35]. When updating the bounds ℓ and u in Lines 4 and 5 of Algorithm 1, sharper bounds based on the inequalities in Lemma 6 may be chosen. This modification requires pre-computing $\Pr_{\mathcal{M}}^{\min}(\Diamond E)$ and $\Pr_{\mathcal{M}}^{\max}(\Diamond E)$.

5 Conditional Reachability in Colored MDPs

Colored MDPs [3, 21] extend MDPs by grouping states. Intuitively, policies must take the same action across a group. This model captures various synthesis problems where additional constraints on policies are imposed. Each policy σ that satisfies these constraints induces an MC $\mathcal{M}^\sigma$. Colored MDPs thus compactly represent families of MCs, each induced by such a policy.

This section shows how the methods above can be effectively combined with an abstraction-refinement (AR) method [3] to solve conditional reachability problem for families of MCs, represented by colored MDPs. In the experiments, we use this for probabilistic program verification and monitoring benchmarks.

For conciseness, we use a simplified definition of colored MDPs from [25]:

Definition 3 (Colored MDPs). *A colored MDP is a tuple $\mathcal{M}^C = (\mathcal{M}, C, c)$, where $\mathcal{M}$ is an MDP with states S , C is a set of colors, and $c: S \rightarrow C$.*

Definition 4 (Color-Consistent Policy). *A memoryless policy σ for a colored MDP is color consistent, if for all states $s, s': c(s) = c(s')$ implies $\sigma(s) = \sigma(s')$.*

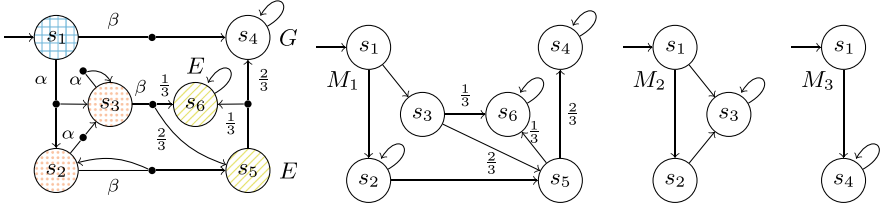


Fig. 3. (left) Colored MDP $\mathcal{M}^C$ with underlying MDP from Fig. 2. The coloring restricts the policy space to policies where $\sigma(s_2) = \sigma(s_3)$ (s_5 and s_6 are irrelevant as only one action is available here). (right) three Markov chains in the family of MCs given by color consistent policies in $\mathcal{M}^C$.

Policy synthesis for colored MDPs asks for a color-consistent policy such that the reachability probability exceeds a given threshold. This problem is NP-hard, but efficient heuristics based on abstraction-refinement, counterexample generalization, and SMT exist [3, 21]. We lift the problem to conditional reachability.

Problem 3: Threshold Conditional Reachability for colored MDPs

Given a colored MDP $\mathcal{M}^C = (\mathcal{M}, C, c)$ with $G, E \subseteq S$ and $\lambda \in [0, 1]$, decide whether there is a color-consistent policy σ such that $\Pr_{\mathcal{M}}^{\sigma}(\Diamond G \mid \Diamond E) \geq \lambda$.

In the remainder of this section, we clarify the connection of colored MDPs to family of MCs and adapt the AR algorithm from [3] to our Problem 3.

Family of MCs. Many synthesis problems consider MDPs under a restricted set of policies. The set of induced Markov chains is a *family*:

Definition 5 (Family of MCs). A family of MCs $\mathcal{F}$ given by colored MDP $\mathcal{M}^C$ is defined as $\mathcal{F} = \{\mathcal{M}^{\sigma} \mid \sigma \text{ is a color consistent policy for } \mathcal{M}^C\}$.

Example 9. Consider the colored MDP $\mathcal{M}^C$ in Fig. 3 with underlying MDP from Fig. 2 and a coloring with $c(s_2) = c(s_3)$, so color-consistent policies must satisfy $\sigma(s_2) = \sigma(s_3)$. While the underlying MDP has eight policies, only four are color consistent, corresponding to the three MCs in Fig. 3: (M_1) from $\sigma(s_1) = \alpha$, $\sigma(s_2) = \sigma(s_3) = \beta$, (M_2) from $\sigma(s_1) = \sigma(s_2) = \sigma(s_3) = \alpha$, and (M_3) $\sigma(s_1) = \beta$, $\sigma(s_2) = \sigma(s_3) = \alpha/\beta$ where s_2 and s_3 unreachable.

While an arbitrary family of MCs can be encoded as a colored MDP, the efficiency of this encoding depends on the similarity of the MCs in the family [3]. Synthesis problems are not given by families of MCs, but immediately by colored MDPs, which are concise if the transition relation for most states coincides across family members. Previous work shows use cases where families of over 10^{100} MCs are efficiently represented via colored MDPs [3].

Abstraction Refinement for Colored MDPs. Consider a colored MDP $\mathcal{M}^C = (\mathcal{M}, C, c)$ and a constraint $\geq \lambda$. Since the consistent policies in $\mathcal{M}^C$ are a subset of all policies, $\mathcal{M}$ provides a sound abstraction of $\mathcal{M}^C$. Therefore the policy $\sigma_{\max}$ maximising the conditional probability $Pr_{\mathcal{M}}(\Diamond G \mid \Diamond E)$ in the MDP $\mathcal{M}$ provides the upper bound ub on maximal conditional probability achieved by any consistent policy in $\mathcal{M}^C$. If $ub < \lambda$ then all policies, including the consistent policies violate $\geq \lambda$ and thus there is no consistent policy satisfying the constraint. If $ub \geq \lambda$ and $\sigma_{\max}$ is consistent, we have the solution of Problem 3. In all other cases, we cannot draw any conclusion and the abstraction needs to be refined.

The goal of the refinement is to remove the inconsistent policy $\sigma_{\max}$ from the policies in the abstraction. This is achieved by splitting the set of policies in $\mathcal{M}$ into two (disjoint) subsets such that $\sigma_{\max}$ is not included in either subset. Let $\sigma_{\max}$ be inconsistent in the states s and s' , i.e. $c(s) = c(s')$ while $\sigma(s) = \alpha$ and $\sigma(s') = \beta$ (where α and β are two different actions). We can efficiently represent the subsets by two (colored) sub-MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$. In $\mathcal{M}_1$, we disable α in states s, s' while in $\mathcal{M}_2$, we disable β . Any consistent policy from $\mathcal{M}$ exists in either $\mathcal{M}_1$ and $\mathcal{M}_2$. Hence, the AR leads to an iterative algorithm that analyses a sequence of colored MDPs $\mathcal{M}_i^C = (\mathcal{M}_i, C, c)$. In every iteration, either a satisfying consistent policy is found or $\mathcal{M}_i^C$ is either discarded (if $ub < \lambda$), or refined. The finite number of policies implies termination of the algorithm.

Example 10. Take the colored MDP $\mathcal{M}^C$ from Fig. 3 and a threshold $\geq 0.6 = \lambda$. We obtain $Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E) = \frac{2}{3}$ and policy $\sigma_{\max}$ with $\sigma_{\max}(s_1) = \sigma_{\max}(s_3) = \alpha$ and $\sigma_{\max}(s_2) = \beta$. Since the value exceeds λ , we check policy consistency. The policy is inconsistent: it selects different actions in s_2 and s_3 . We refine the abstraction into two disjoint colored sub-MDPs: $\mathcal{M}_1^C$, α is enforced in s_2 and s_3 , and $\mathcal{M}_2^C$, where α is not allowed in s_2, s_3 . In $\mathcal{M}_1^C$ no policy reaches evidence states, so no policy can satisfy the threshold. For $\mathcal{M}_2^C$, we obtain $Pr_{\mathcal{M}_2}^{\max}(\Diamond G \mid \Diamond E) = \frac{5}{9} < \lambda$. Hence $\mathcal{M}_2^C$ is also discarded. Thus, no consistent policy satisfies $\lambda \geq 0.6$ in $\mathcal{M}^C$.

Refinement for Conditional Reachability. We integrate the approaches in Sects. 3 and 4 with abstraction refinement [3] and outline differences to the standard reachability case. First, for Problem 3, the use of Sect. 3 suffices. Second, the policy $\sigma_{\max}$ is typically inconsistent in several states, which raises a design choice for the refinement strategy. In [3], the heuristic is based on prioritizing states based on the expected number of visits multiplied by the probability of reaching the goal from that state, both under $\sigma_{\max}$. Conditional reachability queries do not provide these probabilities for every state [2]. Based on our preliminary experiments, we use a simple backward refinement that always refines the inconsistent state that has the maximal distance from s^I .

6 Experiments

We compare the **restart** method [7] with our novel **total reward reduction** (**treat**) based methods in Sects. 3 and 4. We answer the following research questions:

- (RQ1). What is the fastest way to solve Problem 1 and Problem 2 (on MDPs)?
- (RQ2). Can the novel algorithms for the computation of conditional properties speed up solving of Problem 3 (on colored MDPs)?
- (RQ3). How does explicit support for conditional reachability in STORM [22] affect runtime monitoring experiments as in [24]?

Summary of the Results. For (RQ1), the use of our new algorithms speeds up the computation on most benchmarks, for both floating-point and exact computations. The floating point results are accurate, whereas they are not always for the old restart method, and the performance has improved by orders of magnitude on some (acyclic) benchmarks. For (RQ2), the results demonstrate not only the feasibility of adapting abstraction-refinement towards conditional probabilities, but also the importance of across-the-board performance improvements on small/easy MDPs in this setting for both floating-point and exact computations. Beyond what is reported in (RQ1), the answers to (RQ3) highlight the impact of preprocessing, where already the old restart method with the novel preprocessing outperforms the previous ‘outside’ construction of the restart MDP and its analysis by up to orders of magnitude.

Setup. We run single-threaded experiments on an AMD Threadripper 5965wx, with a 10 min timeout for RQ1 and RQ3 and a 15 min timeout for RQ2. Memory consumption remains below 32 GB RAM. In the experiments, we report the model checking times, which exclude loading the model (from an explicit-state input) but include the necessary preprocessing. The supplementary material [11] contains all data points and the source code used to perform the experiments.

RQ1 Computing Conditional Properties on MDPs

Implementation. All methods (including **restart**) are implemented within the model checker STORM [22] and share the same process of model building and preprocessing, including the elimination of the initial component (Definition 2) All methods support exact and floating-point arithmetic and can be instantiated with any MDP solver. Here, we focus on the standard configurations in STORM².

Benchmarks. We use 318 benchmarks from four sources. (1) We use (all) MDP benchmarks from [7, 26], which are adapted from standard benchmarks (*consensus*, *wlan*). (2) From runtime monitoring for MDPs [24], we collect challenging benchmarks³ using the same approach that was used to create benchmarks in [20]

² with exact guarantees using exact arithmetic and value iteration without hard guarantees for floating point arithmetic: like [20], we observed that benchmarks where VI is wrong yield timeouts for optimistic VI.

³ Which are not solved by all approaches within a 0.1 seconds.

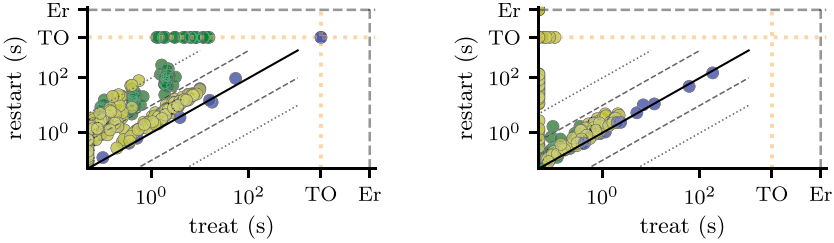


Fig. 4. Runtime for solving Problem 1: *restart* vs *treat*. Exact arithmetic left, numerical computations right. Points refer to benchmark instances (from: BNs, MDPs, IMCs, RunMon). Points above the diagonal correspond to a speedup of *treat*, points above dashed lines to a 10x or 100x speedup.

(*refuel*, *patrol*, *evade*). (3) We use MDPs from (uncertain/parametric) Bayesian networks [32], which correspond to the MDP semantics for the (interval) MCs [4] underlying these Bayesian networks (BNs) (*alarm*, *child*, and others). (4) Similarly, we add (interval-based) uncertainty to (all) MC benchmarks from [7, 26], which are adapted from standard benchmarks (*brp*, *egl*, *crowds*) and translate them to MDPs as above. Where applicable, we evaluate models on multiple properties. The runtime monitoring, BN and the *brp* benchmark are acyclic.

Solving the Threshold Problem. We compare *restart* with *treat*. In Fig. 4 we provide a log-log scale scatter plot, where points correspond to benchmarks and provide a runtime comparison between two methods.

The Exact Case. We observe a considerable speed-up ($>2\times$) on many models and a multi-order magnitude speed-up on some models. For a few benchmarks, the impact of the method choice is marginal as the initial component elimination alone yields tiny MDPs. The highest speed-up occurs on acyclic models with large initial components, where *treat* uses dynamic programming and *restart* requires repeated exact LU decompositions for the policy iteration.

The Numerical Case. Numerical approaches relying on *restart* do not (always) converge, as was observed before [20, 26]. Even when thresholding, this does lead to some errors: These errors are not due to unfortunate selections of thresholds, but due to values being completely off. For benchmarks where the numerical computations converge, we see significantly lower runtimes and a similar picture as in the exact case, although the effects are less pronounced.

Solving the Optimization Problem. We compare *restart* with bisection-variants on top of *treat*: We use standard (bis-std) and advanced bounds (bis-adv) and run it without and with policy tracking (pt-bis-std, pt-bis-adv). In Fig. 5, we compare pt-bis-std – the empirically strongest configuration – with *restart*. With exact arithmetic, results coincide and are used as a ground truth, whereas the

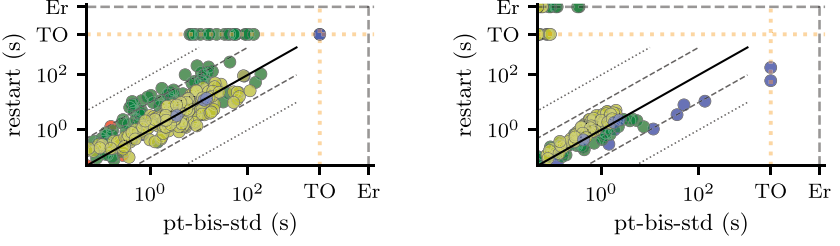


Fig. 5. Runtime for solving Problem 2: restart vs pt-bis-std, analogous to Fig. 4.

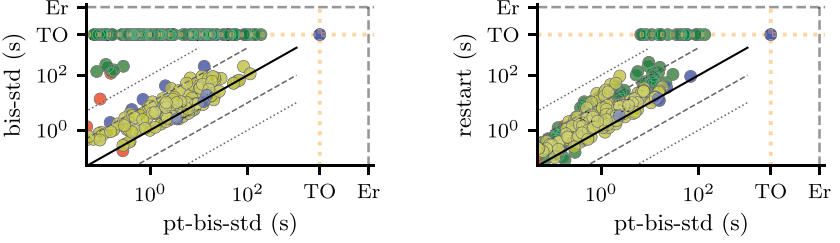


Fig. 6. Runtime for solving Problem 2. Left: Exact arithmetic bis-std vs pt-bis-std. Right: ϵ -optimal pt-bis-std vs. exact restart. Legend as in Fig. 4.

numerical methods sometimes yield significant errors⁴. The supplementary material contains additional scatter plots.

The Exact Case. The configuration pt-bis-std overall outperforms restart, but not on every model: The restart suffers significantly on acyclic models, but computing an exact and potentially exponentially large rational number with bisection can also be challenging. In fact, while restart times out $60\times$, bis-std and bis-adv $106\times$ and $97\times$, respectively. Policy tracking is effective, see also Fig. 6(left) and reduces time-outs to $9\times$ for pt-bis-std and $57\times$ for pt-bis-adv.

The Numerical Case. The restart method does not reliably provide accurate results (as before) and provides wrong results on 171 benchmarks, whereas pt-bis-std is always correct and solves many of them instantly. The overhead of any bisection method is limited as the number of iterations required to converge to a target precision of 10^{-6} is upper bounded to twenty for the most naive bisection strategy possible. Indeed, our bisection variations all yield lower number of iterations. Standard bis-std times out $13\times$ compared to $6-7\times$ for any improved variant.

Exact ε -Guarantees. Due to the slow convergence of VI, solving reachability with exact arithmetic up to an ε -error is not effective. However, when using bisection-based methods, we can use treat for computing exact results, but terminate

⁴ We mark results that have a relative error $> 10^{-3}$ as incorrect, unless ground truth and reported result are $< 10^{-8}$. We use these liberal interpretation numbers to highlight the significance of the errors.

Table 1. Performance comparison for color MDP problems. We report runtime in seconds and number of iterations per second. The left side reports results for numerical computation and the right side reports results for exact computation. The boldface values denote the fastest algorithm for each computation type.

Model	$ \mathcal{M}^C $	$ S $	numerical						exact					
			treat		pt-bis-std		restart		treat		pt-bis-std		restart	
			time	it/s	time	it/s	time	it/s	time	it/s	time	it/s	time	it/s
ceas-4-10	1e6	265	<1 s	1978	<1 s	1978	120 s	1	<1 s	848	<1 s	712	<1 s	574
ceas-10-8	1e8	451	29 s	1500	30 s	1459	TO	<1	66 s	654	71 s	610	86 s	507
ladder	8.6e11	251	TO	1204	TO	1157	TO	917	TO	411	TO	356	TO	196
ladder-in	8.6e12	503	305 s	517	315 s	532	TO	<1	742 s	225	TO	163	TO	130
dpm	1620.0	1976	4 s	75	46 s	6	TO	<1	9 s	29	TO	<1	14 s	19
dpm-q	1.6e4	1733	5 s	58	49 s	6	TO	<1	10 s	28	TO	<1	15 s	19
virus	1.2e4	118	2 s	1312	TO	<1	5 s	659	12 s	344	205 s	19	10 s	414
airport	1.3e12	51	58 s	4726	94 s	4754	TO	3329	204 s	1339	194 s	2299	631 s	2142
airport-b	1.8e14	115	21 s	1583	29 s	1610	TO	565	39 s	852	60 s	786	361 s	672

once the exact result is approximated up to 10^{-6} . As in the numerical case, this efficiently bounds the number of iterations, but provides strong guarantees. The resulting Fig. 6(right) clearly shows the advantages of the novel approach.

RQ2 Computing Conditional Properties on Colored MDPs

Implementation and Setup. We extended PAYNT [3], a tool for solving colored MDPs, to support conditional reachability. PAYNT uses STORM internally for model checking DTMCs and MDPs. Therefore, we can evaluate the advantage of *treat* versus *restart* in the context of Problem 3. We also evaluate the effect of only thresholding against computing the optimal value: In particular, we compare with *pt-bis-std* that typically provides the best performance (see RQ1; results for the other bisection methods are presented in [35]). We report the overall runtime for Problem 3 and the abstraction-refinement iterations per second, for numerical and exact computations.

Benchmarks. We use benchmarks from three different domains. (1) probabilistic program verification: the *Caesar cipher* (*ceas*) with different input length and alphabet size and two variants of *ladder network* (*ladder*) [23] with varying possible inputs. (2) Families of MCs from PAYNT, but with conditional probability specifications (*dpm* [28], *virus*). (3) Colored MDPs exported from monitor verification [25] where the problem is to verify whether a proposed monitor has no missed alarms. Table 1 reports the number of color-consistent policies $|\mathcal{M}^C|$ in the colored MDP $\mathcal{M}^C$ and the number of underlying states $||\mathcal{M}||$.

Role of the Refinement Strategy. Different policies computed by the individual algorithms can significantly affect the abstraction-refinement strategy (in terms

Table 2. Runtime monitoring examples: 3 methods, each evaluated on ten traces of 250 steps. We report the average and maximal time to evaluate a trace.

Model	treat		restart		previous	
	t_{avg}	t_{max}	t_{avg}	t_{max}	t_{avg}	t_{max}
airportB-3-300-30	0.42	0.54	6.71	12.2	21.7	23.9
evadeI-10	0.10	0.12	0.35	0.44	1.37	2.67
evadeV-9-3	0.86	1.71	6.62	24.8	17.6	54.1
refuelA-50-80	0.29	0.38	1.00	1.39	3.38	6.13

of the number of iterations and the structure of the hardness of the consecutive queries) and thus the overall performance.

Using Treat vs Restart. For the numerical computation, the **restart** method is unable to solve a majority of the benchmarks and **treat** based methods are clearly superior. Except for the *airport* benchmarks, this is due to the timing for individual queries. For *airport*, we observe the impact of the refinement strategy: despite a comparable number of MDP queries, **restart** times out. For the exact computation, the advantage of **treat** based methods is less prominent, but except for the smallest benchmarks **restart** still lags considerably behind.

Using Treat vs. pt-bi-std. In most cases, **treat** outperforms the bisection-based methods, since it reduces the number of value iterations needed to solve the given MDP query. However, the advantage of **treat** can sometimes be mitigated by the impact of different policies on refinement.

RQ3 Runtime Monitoring Revisited

In this experiment, we evaluate the impact of support for conditional probabilities in MDPs into STORM in general and the use of **treat** in particular, when considering an application in runtime monitoring. In particular, we replicate experiments from [24]. Somewhat simplified, that paper presents a method that takes a trace (of colors) τ to compare a series of conditional probabilities:

$$\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \tau_1) \leq \lambda, \dots, \Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \tau_n) \leq \lambda,$$

where $\Pr(\Diamond G \mid \tau_i)$ denotes the probability to reach states in G , given that the states visited in the first i steps match the coloring given in τ . By unrolling, these queries can be reformulated into $\Pr_{\mathcal{M}}^{\max}(\Diamond G \mid \Diamond E)$ [7].

Setup. The **previous** approach put forward in [24] unrolls the MDP immediately into the **restart** MDP and computes the conditional reachability probabilities using off-the-shelf MDP reachability in STORM. We adapted the code to only unroll and use the conditional reachability model checking. In contrast to RQ1, we did not filter for challenging individual instances, but run ten traces of 250 steps on models from the [24]-repository.

Result. In Table 2, we observe not only the advantage of `treat` vs `restart`, but also how STORM provides efficient preprocessing that significantly improves the performance. The improvements in average time are particularly relevant for learning runtime monitors from data [25], whereas a low maximal response time is particularly relevant when deploying a monitor [9].

7 Conclusion

Our novel approach to compute conditional reachability probabilities in MDPs clearly outperforms the previous state of the art. We show the benefits for various benchmarks from the literature, for Markov chain families, and in runtime monitoring applications. We would like to lift these approaches to symbolic state spaces and programs, as well as to interval MDPs or stochastic games. The extension towards Markov chain families raises additional questions how to generally handle specifications that require history-dependent policies. We furthermore hope to integrate support for conditional reachability into [21], which allows reasoning about families while directly supporting assumptions, e.g., on the reachability of E .

Acknowledgments.  This work has been executed under the project VASSAL: “Verification and Analysis for Safety and Security of Applications in Life” funded by the European Union under Horizon Europe WIDERA Coordination and Support Action/Grant Agreement No. 10116002. This work was partly supported by the NWO grant FuRoRe (OCENW.M.22.282) and by the KI-Starter Project Verifying AI Systems under Partial Observability of the Ministry of Culture and Science of the German State of North Rhine-Westphalia. Additionally, this work has been funded by the Czech Science Foundation grant QUAK (25-18318S).

Data Availability Statement. An artifact with source code and experimental data is available at <https://doi.org/10.5281/zenodo.19739061> [11]. It contains the modifications done to STORM [22], PAYNT [3], and PREMISE [24]. All algorithms implemented are part of STORM version 1.13.0 available at <https://stormchecker.org>.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. de Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University, USA (1997)
2. Andrés, M.E., van Rossum, P.: Conditional probabilities over probabilistic and nondeterministic systems. In: TACAS, pp. 157–172. Lecture Notes in Computer Science, Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_12
3. Andriushchenko, R., Ceska, M., Macák, F., Junges, S., Katoen, J.: An oracle-guided approach to constrained policy synthesis under uncertainty. J. Artif. Intell. Res. **82**, 433–469 (2025). <https://doi.org/10.1613/JAIR.1.16593>

4. Badings, T., Sim ao, T.D., Suilen, M., Jansen, N.: Decision-making under uncertainty: beyond probabilities. *Int. J. Softw. Tools Technol. Transf.* **25**(3), 375–391 (2023). <https://doi.org/10.1007/S10009-023-00704-3>
5. Badings, T., Volk, M., Junges, S., Stoelinga, M., Jansen, N.: CTMCs with imprecisely timed observations. In: TACAS (2), pp. 258–278. *Lecture Notes in Computer Science*, Springer (2024). https://doi.org/10.1007/978-3-031-57249-4_13
6. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008). https://doi.org/10.1007/978-3-031-97439-7_7
7. Baier, C., Klein, J., Klüppelholz, S., Märcker, S.: Computing conditional probabilities in Markovian models efficiently. In: TACAS, pp. 515–530. *Lecture Notes in Computer Science*, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_43
8. Baier, C., Klein, J., Klüppelholz, S., Wunderlich, S.: Maximizing the conditional expected reward for reaching the goal. In: TACAS (2), pp. 269–285. *Lecture Notes in Computer Science* (2017). https://doi.org/10.1007/978-3-662-54580-5_16
9. Bartocci, E., et al.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In: *Lectures on Runtime Verification*, pp. 135–175. *Lecture Notes in Computer Science*, Springer (2018). https://doi.org/10.1007/978-3-319-75632-5_5
10. Batz, K., Chen, M., Junges, S., Kaminski, B.L., Katoen, J., Matheja, C.: Probabilistic program verification via inductive synthesis of inductive invariants. In: TACAS (2), pp. 410–429. *Lecture Notes in Computer Science*, Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_25
11. Ceska, M., Junges, S., van der Maas, L., Macák, F., Quatmann, T.: Artifact of “fast computation of conditional probabilities in MDPs and Markov chain families” (2026). <https://doi.org/10.5281/zenodo.19739061>
12. Chatterjee, K., Henzinger, T.A.: Value iteration. In: 25 Years of Model Checking, pp. 107–138. *Lecture Notes in Computer Science*, Springer (2008). https://doi.org/10.1007/978-3-540-69850-0_7
13. Cho, M., Gouwar, J., Holtzen, S.: Scaling optimization over uncertainty via compilation. *Proc. ACM Program. Lang.* **9**(OOPSLA1), 1546–1574 (2025). <https://doi.org/10.1145/3720500>
14. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. *Log. Methods Comput. Sci.* **4**(4) (2008). [https://doi.org/10.2168/LMCS-4\(4:8\)2008](https://doi.org/10.2168/LMCS-4(4:8)2008)
15. Forejt, V., Kwiatkowska, M.Z., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: TACAS, pp. 112–127. *Lecture Notes in Computer Science*, Springer (2011). https://doi.org/10.1007/978-3-642-19835-9_11
16. Gerlach, L., Winkler, T., Ábrahám, E., Bonakdarpour, B., Junges, S.: Efficient probabilistic model checking for relational reachability. In: CAV (1), pp. 127–147. *Lecture Notes in Computer Science*, Springer (2025). https://doi.org/10.1007/978-3-031-98668-0_6
17. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics: A Foundation for Computer Science, 2nd Ed. Addison-Wesley (1994)
18. Gürtler, T., Kaminski, B.L.: noDice: Inference for discrete probabilistic programs with nondeterminism and conditioning. *Proc. ACM Program. Lang.* **10**(OOPSLA1) (2026). <https://doi.org/10.1145/3798215>
19. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.* **735**, 111–131 (2018). <https://doi.org/10.1016/J.TCS.2016.12.003>

20. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A practitioner's guide to MDP model checking algorithms. In: TACAS (1), pp. 469–488. Lecture Notes in Computer Science, Springer (2023). https://doi.org/10.1007/978-3-031-30823-9_24
21. Heck, L., Macák, F., Ceska, M., Junges, S.: Constrained and robust policy synthesis with satisfiability-modulo-probabilistic-model-checking. In: AAAI, pp. 36253–36261. AAAI Press (2026). <https://doi.org/10.1609/AAAI.V40I43.40944>
22. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker storm. *Int. J. Softw. Tools Technol. Transf.* **24**(4), 589–610 (2022). <https://doi.org/10.1007/S10009-021-00633-Z>
23. Holtzen, S., den Broeck, G.V., Millstein, T.D.: Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.* **4**(OOPSLA), 140:1–140:31 (2020). <https://doi.org/10.1145/3428208>
24. Junges, S., Torfah, H., Seshia, S.A.: Runtime monitors for Markov decision processes. In: CAV (2), pp. 553–576. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_26
25. van der Maas, L., Junges, S.: Learning verified monitors for hidden Markov models. In: ATVA, pp. 180–203. Lecture Notes in Computer Science, Springer (2025). https://doi.org/10.1007/978-3-032-08707-2_9
26. Märcker, S., Baier, C., Klein, J., Klüppelholz, S.: Computing conditional probabilities: implementation and evaluation. In: SEFM, pp. 349–366. LNCS, Springer (2017). https://doi.org/10.1007/978-3-319-66197-1_22
27. Mathur, U., Bauer, M.S., Chadha, R., Sistla, A.P., Viswanathan, M.: Exact quantitative probabilistic model checking through rational search. *Formal Methods Syst. Des.* **56**(1), 90–126 (2020). <https://doi.org/10.1007/S10703-020-00348-Y>
28. Norman, G., Parker, D., Kwiatkowska, M.Z., Shukla, S.K., Gupta, R.K.: Formal analysis and validation of continuous-time Markov chain based system level power management strategies. In: HLDVT, pp. 45–50. IEEE Computer Society (2002). <https://doi.org/10.1109/HLDVT.2002.1224427>
29. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). <https://doi.org/10.1002/9780470316887>
30. Quatmann, T., Katoen, J.: Multi-objective optimization of long-run average and total rewards. In: TACAS (1), pp. 230–249. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-72016-2_13
31. Salmani, B., Katoen, J.: Bayesian inference by symbolic model checking. In: QEST, pp. 115–133. LNCS, Springer (2020). https://doi.org/10.1007/978-3-030-59854-9_9
32. Salmani, B., Katoen, J.: Automatically finding the right probabilities in Bayesian networks. *J. Artif. Intell. Res.* **77**, 1637–1696 (2023). <https://doi.org/10.1613/JAIR.1.14044>
33. Skurka, A., van der Maas, L., Junges, S., Torfah, H.: Learning robust Markov models for safe runtime monitoring. In: AAMAS. International Foundation for Autonomous Agents and Multiagent Systems / ACM (2026)
34. Stoller, S.D., et al.: Runtime verification with state estimation. In: RV, pp. 193–207. Lecture Notes in Computer Science, Springer (2011). https://doi.org/10.1007/978-3-642-29860-8_15
35. Češka, M., Junges, S., van der Maas, L., Macák, F., Quatmann, T.: Fast computation of conditional probabilities in MDPS and Markov chain families (2026). <https://doi.org/10.48550/arXiv.2605.11897>, <https://arxiv.org/abs/2605.11897>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

