Shepherding Hordes of Markov Chains

Milan Češka[1], Nils Jansen[2], Sebastian Junges[3], and Joost-Pieter Katoen[3]

[1] Brno University of Technology, Brno, Czech Republic
[2] Radboud University, Nijmegen, The Netherlands
[3] RWTH Aachen University, Aachen, Germany

Abstract. This paper considers large families of Markov chains (MCs) that are defined over a set of parameters with finite discrete domains. Such families occur in software product lines, planning under partial observability, and sketching of probabilistic programs. Simple questions, like 'does at least one family member satisfy a property?', are NP-hard. We tackle two problems: distinguish family members that satisfy a given quantitative property from those that do not, and determine a family member that satisfies the property optimally, i.e., with the highest probability or reward. We show that combining two well-known techniques, MDP model checking and abstraction refinement, mitigates the computational complexity. Experiments on a broad set of benchmarks show that in many situations, our approach is able to handle families of millions of MCs, providing superior scalability compared to existing solutions.

1 Introduction

Randomisation is key to research fields such as dependability (uncertain system components), distributed computing (symmetry breaking), planning (unpredictable environments), and probabilistic programming. Families of alternative designs differing in the structure and system parameters are ubiquitous. Software dependability has to cope with configuration options, in distributed computing the available memory per process is highly relevant, in planning the observability of the environment is pivotal, and program synthesis is all about selecting correct program variants. The automated analysis of such families has to face a formidable challenge — in addition to the state-space explosion affecting each family member, the family size typically grows exponentially in the number of features, options, or observations. This affects the analysis of (quantitative) software product lines [17, 27, 42, 44, 45], strategy synthesis in planning under partial observability [11, 13, 28, 35, 40], and probabilistic program synthesis [9, 12, 26, 39].

This paper considers families of Markov chains (MCs) to describe configurable probabilistic systems. We consider finite MC families with finite-state family members. Family members may have different transition probabilities and distinct topologies — thus different reachable state spaces. The latter aspect goes beyond the class of parametric MCs as considered in parameter synthesis [10, 21, 23, 30] and model repair [6, 15, 41].

For an MC family $\mathfrak{D}$ and quantitative specification φ, with φ a reachability probability or expected reward objective, we consider the following synthesis

problems: (a) does some member in $\mathfrak{D}$ satisfy a threshold on φ? (aka: *feasibility* synthesis), (b) which members of $\mathfrak{D}$ satisfy this threshold on φ and which ones do not? (aka: *threshold synthesis*), and (c) which family member(s) satisfy φ optimally, e.g., with highest probability? (aka: *optimal synthesis*).

The simplest synthesis problem, feasibility, is NP-complete and can naively be solved by analysing all individual family members — the so-called *one-by-one* approach. This approach has been used in [17] (and for qualitative systems in e.g. [18]), but is infeasible for large systems. An alternative is to model the family $\mathfrak{D}$ by a single Markov decision process (MDP) — the so-called *all-in-one* MDP [17]. The initial MDP state non-deterministically chooses a family member of $\mathfrak{D}$, and then evolves in the MC of that member. This approach has been implemented in tools such as ProFeat [17], and for purely qualitative systems in [19]. The MDP representation avoids the individual analysis of all family members, but its size is proportional to the family size. This approach therefore does not scale to large families. A symbolic BDD-based approach is only a partial solution as family members may induce different reachable state-sets.

This paper introduces an *abstraction-refinement* scheme over the MDP representation[4]. The abstraction *forgets* in which family member the MDP operates. The resulting *quotient* MDP has a single representative for every reachable state in a family member. It typically provides a very compact representation of the family $\mathfrak{D}$ and its analysis using off-the-shelf MDP model-checking algorithms yields a speed-up compared to the all-in-one approach. Verifying the quotient MDP yields under- and over-approximations of the min and max probability (or reward), respectively. These bounds are safe as all *consistent* schedulers, i.e., those that pick actions according to a single family member, are contained in all schedulers considered on the quotient MDP. (CEGAR-based MDP model checking for partial information schedulers, a slightly different notion than restricting schedulers to consistent ones, has been considered in [29]. In contrast to our setting, [29] considers history-dependent schedulers and in this general setting no guarantee can be given that bounds on suprema converge [28]).

Model-checking results of the quotient MDP do provide useful insights. This is evident if the resulting scheduler is consistent. If the verification reveals that the min probability exceeds r for a specification φ with a $\leq r$ threshold, then — even for inconsistent schedulers — it holds that all family members violate φ. If the model checking is inconclusive, i.e., the abstraction is too coarse, we iteratively refine the quotient MDP by splitting the family into sub-families. We do so in an efficient manner that avoids rebuilding the sub-families. Refinement employs a light-weight analysis of the model-checking results.

We implemented our abstraction-refinement approach using the Storm model checker [24]. Experiments with case studies from software product lines, planning, and distributed computing yield possible speed-ups of up to 3 orders of magnitude over the one-by-one and all-in-one approaches (both symbolic and explicit). Some

[4] Classical CEGAR for model checking of software product lines has been proposed in [20]. This uses feature transition systems, is purely qualitative, and exploits existential state abstraction.

benchmarks include families of millions of MCs where family members are thousands of states. The experiments reveal that — as opposed to parameter synthesis [10,23,30] — the threshold has a major influence on the synthesis times.

To summarise, this work presents: a) MDP-based abstraction-refinement for various synthesis problems over large families of MCs, b) a refinement strategy that mitigates the overhead of analysing sub-families, and c) experiments showing substantial speed-ups for many benchmarks. Extra material can be found in [1].

2 Preliminaries

We present the basic foundations for this paper, for details, we refer to [4,5].

Probabilistic models. A *probability distribution* over a finite or countably infinite set X is a function $\mu\colon X \to [0,1]$ with $\sum_{x\in X} \mu(x) = \mu(X) = 1$. The set of all distributions on X is denoted $\mathit{Distr}(X)$. The support of a distribution μ is $\mathrm{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$. A distribution is *Dirac* if $|\mathrm{supp}(\mu)| = 1$.

Definition 1 (MC) *A* discrete-time Markov chain *(MC) D is a triple $(S, s_0, \mathbf{P})$, where S is a finite set of states, $s_0 \in S$ is an initial state, and $\mathbf{P}\colon S \to \mathit{Distr}(S)$ is a transition probability matrix.*

MCs have unique distributions over successor states at each state. Adding nondeterministic choices over distributions leads to Markov decision processes.

Definition 2 (MDP) *A* Markov decision process *(MDP) is a tuple $M = (S, s_0, \mathit{Act}, \mathcal{P})$ where S, s_0 as in Def. 1, Act is a finite set of actions, and $\mathcal{P}\colon S \times \mathit{Act} \nrightarrow \mathit{Distr}(S)$ is a partial transition probability function.*

The *available actions* in $s \in S$ are $\mathit{Act}(s) = \{a \in \mathit{Act} \mid \mathcal{P}(s,a) \neq \bot\}$. An MDP with $|\mathit{Act}(s)| = 1$ for all $s \in S$ is an MC. For MCs (and MDPs), a state-reward function is $\mathit{rew}\colon S \to \mathbb{R}_{\geq 0}$. The reward $\mathit{rew}(s)$ is earned upon leaving s.

A *path* of an MDP M is an (in)finite sequence $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$, where $s_i \in S$, $a_i \in \mathit{Act}(s_i)$, and $\mathcal{P}(s_i, a_i)(s_{i+1}) \neq 0$ for all $i \in \mathbb{N}$. For finite π, $\mathrm{last}(\pi)$ denotes the last state of π. The set of (in)finite paths of M is $\mathsf{Paths}^M_{\mathit{fin}}$ (Paths^M). The notions of paths carry over to MCs (actions are omitted). Schedulers resolve all choices of actions in an MDP and yield MCs.

Definition 3 (Scheduler) *A* scheduler *for an MDP $M = (S, s_0, \mathit{Act}, \mathcal{P})$ is a function $\sigma\colon \mathsf{Paths}^M_{\mathit{fin}} \to \mathit{Act}$ such that $\sigma(\pi) \in \mathit{Act}(\mathrm{last}(\pi))$ for all $\pi \in \mathsf{Paths}^M_{\mathit{fin}}$. Scheduler σ is* memoryless *if $\mathrm{last}(\pi) = \mathrm{last}(\pi') \implies \sigma(\pi) = \sigma(\pi')$ for all $\pi, \pi' \in \mathsf{Paths}^M_{\mathit{fin}}$. The set of all schedulers of M is Σ^M.*

Definition 4 (Induced Markov Chain) *The MC* induced by MDP M and $\sigma \in \Sigma^M$ *is given by $M_\sigma = (\mathsf{Paths}^M_{\mathit{fin}}, s_0, \mathbf{P}^\sigma)$ where:*

$$\mathbf{P}^\sigma(\pi, \pi') = \begin{cases} \mathcal{P}(\mathrm{last}(\pi), \sigma(\pi))(s') & \text{if } \pi' = \pi \xrightarrow{\sigma(\pi)} s' \\ 0 & \text{otherwise.} \end{cases}$$

Specifications. For a MC D, we consider unbounded reachability specifications of the form $\varphi = \mathbb{P}_{\sim\lambda}(\Diamond G)$ with $G \subseteq S$ a set of goal states, $\lambda \in [0,1] \subseteq \mathbb{R}$, and $\sim \in \{<, \leq, \geq, >\}$. The probability to satisfy the path formula $\phi = \Diamond G$ in D is denoted by $\texttt{Prob}(D, \phi)$. If φ holds for D, that is, $\texttt{Prob}(D, \phi) \sim \lambda$, we write $D \models \varphi$. Analogously, we define expected reward specifications of the form $\varphi = \mathbb{E}_{\sim\kappa}(\Diamond G)$ with $\kappa \in \mathbb{R}_{\geq 0}$. We refer to λ/κ as *thresholds*. While we only introduce reachability specifications, our approaches may be extended to richer logics like arbitrary PCTL [31], PCTL* [3], or ω-regular properties.

For an MDP M, a specification φ holds ($M \models \varphi$) if and only if it holds for the induced MCs of all schedulers. The maximum probability $\texttt{Prob}^{\max}(M, \phi)$ to satisfy a path formula ϕ for an MDP M is given by a maximising scheduler $\sigma^{\max} \in \Sigma^M$, that is, there is no scheduler $\sigma' \in \Sigma^M$ such that $\texttt{Prob}(M_{\sigma^{\max}}, \phi) < \texttt{Prob}(M_{\sigma'}, \phi)$. Analogously, we define the minimising probability $\texttt{Prob}^{\min}(M, \phi)$, and the maximising (minimising) expected reward $\texttt{ExpRew}^{\max}(M, \phi)$ ($\texttt{ExpRew}^{\min}(M, \phi)$).

The probability (expected reward) to satisfy path formula ϕ from state $s \in S$ in MC D is $\texttt{Prob}(D, \phi)(s)$ ($\texttt{ExpRew}(D, \phi)(s)$). The notation is analogous for maximising and minimising probability and expected reward measures in MDPs. Note that the expected reward $\texttt{ExpRew}(D, \phi)$ to satisfy path formula ϕ is only defined if $\texttt{Prob}(D, \phi) = 1$. Accordingly, the expected reward for MDP M under scheduler $\sigma \in \Sigma^M$ requires $\texttt{Prob}(M_\sigma, \phi) = 1$.

3 Families of MCs

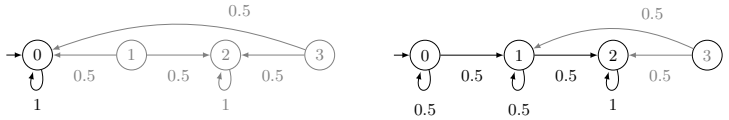
We present our approaches on the basis of an explicit representation of a *family of MCs* using a parametric transition probability function. While arbitrary probabilistic programs allow for more modelling freedom and complex parameter structures, the explicit representation alleviates the presentation and allows to reason about practically interesting synthesis problems. In our implementation, we use a more flexible high-level modelling language, cf. Sect 5.

Definition 5 (Family of MCs) *A* family of MCs *is defined as a tuple* $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ *where* S *is a finite set of states,* $s_0 \in S$ *is an initial state,* K *is a finite set of discrete parameters such that the domain of each parameter* $k \in K$ *is* $T_k \subseteq S$*, and* $\mathfrak{P} \colon S \to Distr(K)$ *is a family of transition probability matrices.*

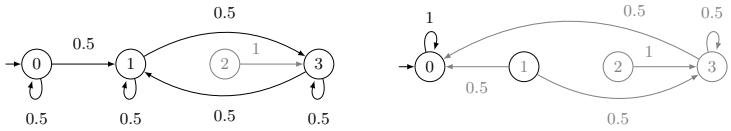
The transition probability function of MCs maps states to distributions over successor states. For families of MCs, this function maps states to distributions over parameters. Instantiating each of these parameters with a value from its domain yields a "concrete" MC, called a *realisation*.

Definition 6 (Realisation) *A* realisation *of a family* $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ *is a function* $r \colon K \to S$ *where* $\forall k \in K \colon r(k) \in T_k$*. A realisation* r *yields a MC* $D_r = (S, s_0, \mathfrak{P}(r))$*, where* $\mathfrak{P}(r)$ *is the transition probability matrix in which each* $k \in K$ *in* $\mathfrak{P}$ *is replaced by* $r(k)$*. Let* $\mathcal{R}^{\mathfrak{D}}$ *denote the* set of all realisations *for* $\mathfrak{D}$*.*

As a family $\mathfrak{D}$ of MCs is defined over finite parameter domains, the number of family members (i.e. realisations from $\mathcal{R}^{\mathfrak{D}}$) of $\mathfrak{D}$ is finite, viz. $|\mathfrak{D}| := |\mathcal{R}^{\mathfrak{D}}| =$



(a) D_{r_1} with $r_1(k_1) = 0, r_1(k_2) = 2$ (b) D_{r_2} with $r_2(k_1) = 1, r_2(k_2) = 2$



(c) D_{r_3} with $r_3(k_1) = 1, r_3(k_2) = 3$ (d) D_{r_4} with $r_4(k_1) = 0, r_4(k_2) = 3$

Fig. 1. The four different realisations of $\mathfrak{D}$.

$\prod_{k \in K} |T_k|$, but exponential in $|K|$. Subsets of $\mathcal{R}^{\mathfrak{D}}$ induce so-called *subfamilies* of $\mathfrak{D}$. While all these MCs share the same state space, their *reachable* states may differ, as demonstrated by the following example.

Example 1 (Family of MCs). Consider a family of MCs $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ where $S = \{0, 1, 2, 3\}$, $s_0 = 0$, and $K = \{k_0, k_1, k_2\}$ with domains $T_{k_0} = \{0\}, T_{k_1} = \{0, 1\}$, and $T_{k_2} = \{2, 3\}$. The parametric transition function $\mathfrak{P}$ is defined by:

$$\mathfrak{P}(0) = 0.5 \colon k_0 + 0.5 \colon k_1 \qquad \mathfrak{P}(1) = 0.5 \colon k_1 + 0.5 \colon k_2$$
$$\mathfrak{P}(2) = 1 \colon k_2 \qquad \mathfrak{P}(3) = 0.5 \colon k_1 + 0.5 \colon k_2$$

Fig. 1 shows the four MCs that result from the realisations $\{r_1, r_2, r_3, r_4\} = \mathcal{R}^{\mathfrak{D}}$ of $\mathfrak{D}$. States that are unreachable from the initial state are greyed out.

We state two synthesis problems for families of MCs. The first is to identify the set of MCs satisfying and violating a given specification, respectively. The second is to find a MC that maximises/minimises a given objective. We call these two problems *threshold synthesis* and *max/min synthesis*.

Problem 1 (Threshold synthesis) *Let $\mathfrak{D}$ be a family of MCs and φ a probabilistic reachability or expected reward specification. The* threshold synthesis problem *is to partition $\mathcal{R}^{\mathfrak{D}}$ into T and F such that $\forall r \in T \colon D_r \vDash \varphi$ and $\forall r \in F \colon D_r \nvDash \varphi$.*

As a special case of the threshold synthesis problem, the *feasibility synthesis problem* is to find just one realisation $r \in \mathcal{R}^{\mathfrak{D}}$ such that $D_r \vDash \varphi$.

Problem 2 (Max synthesis) *Let $\mathfrak{D}$ a family of MCs and $\phi = \Diamond G$ for $G \subseteq S$ The* max synthesis problem *is to find a realisation $r^* \in \mathcal{R}^{\mathfrak{D}}$ such that $\texttt{Prob}(D_{r^*}, \phi) = \max_{r \in \mathcal{R}_{\mathfrak{D}}} \{\texttt{Prob}(D_r, \phi)\}$. The problem is defined analogously for an expected reward measure or minimising realisations.*

Example 2 (Synthesis problems). Recall the family of MCs $\mathfrak{D}$ from Example 1. For the specification $\varphi = \mathbb{P}_{\geq 0.1}(\Diamond\{1\})$, the solution to the threshold synthesis problem is $T = \{r_2, r_3\}$ and $F = \{r_1, r_4\}$, as the goal state 1 is not reachable for D_{r_1} and D_{r_4}. For $\phi = \Diamond\{1\}$, the solution to the max synthesis problem on $\mathfrak{D}$ is r_2 or r_3, as D_{r_2} and D_{r_3} have probability one to reach state 1.

Approach 1 (One-by-one [17]) *A straightforward solution to both synthesis problems is to enumerate all realisations $r \in \mathcal{R}^{\mathfrak{D}}$, model check the MCs D_r, and either compare all results with the given threshold or determine the maximum.*

We already saw that the number of realisations is exponential in $|K|$.

Theorem 1 *The feasibility synthesis problem is NP-complete.*

The theorem even holds for almost-sure reachability properties. The proof is a straightforward adaption of results for augmented interval Markov chains [16, Theorem 3], partial information games [14], or partially observable MDPs [13].

4 Guided Abstraction-Refinement Scheme

In the previous section, we introduced the notion of a family of MCs, two synthesis problems and the one-by-one approach. Yet, for a sufficiently high number of realisations such a straightforward analysis is not feasible. We propose a novel approach allowing us to more efficiently analyse families of MCs.

4.1 All-in-one MDP

We first consider a single MDP that subsumes all individual MCs of a family $\mathfrak{D}$, and is equipped with an appropriate action and state labelling to identify the underlying realisations from $\mathcal{R}^{\mathfrak{D}}$.

Definition 7 (All-in-one MDP [17, 27, 42]) *The* all-in-one MDP *of a family $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ of MCs is given as $M^{\mathfrak{D}} = (S^{\mathfrak{D}}, s_0^{\mathfrak{D}}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}})$ where $S^{\mathfrak{D}} = S \times \mathcal{R}^{\mathfrak{D}} \cup \{s_0^{\mathfrak{D}}\}$, $Act^{\mathfrak{D}} = \{a^r \mid r \in \mathcal{R}^{\mathfrak{D}}\}$, and $\mathcal{P}^{\mathfrak{D}}$ is defined as follows:*

$$\mathcal{P}^{\mathfrak{D}}(s_0^{\mathfrak{D}}, a^r)((s_0, r)) = 1 \quad \textit{and} \quad \mathcal{P}^{\mathfrak{D}}((s, r), a^r)((s', r)) = \mathfrak{P}(r)(s)(s').$$

Example 3 (All-in-one MDP). Fig. 2 shows the all-in-one MDP $M^{\mathfrak{D}}$ for the family $\mathfrak{D}$ of MCs from Example 1. Again, states that are not reachable from the initial state $s_0^{\mathfrak{D}}$ are marked grey. For the sake of readability, we only include the transitions and states that correspond to realisations r_1 and r_2.

From the (fresh) initial state $s_0^{\mathfrak{D}}$ of the MDP, the choice of an action a_r corresponds to choosing the realisation r and entering the concrete MC D_r. This property of the all-in-one MDP is formalised as follows.

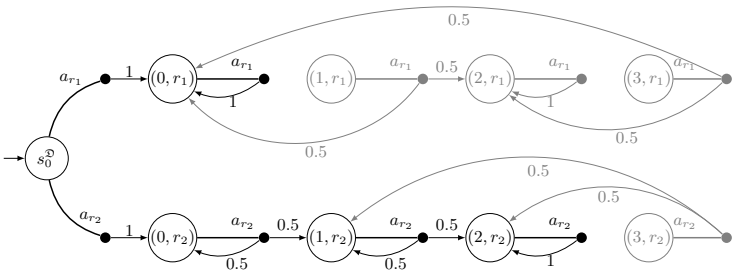


Fig. 2. Reachable fragment of the all-in-one MDP $M^{\mathfrak{D}}$ for realisations r_1 and r_2.

Corollary 1 *For the all-in-one MDP $M^{\mathfrak{D}}$ of family $\mathfrak{D}$ of MCs:*

$$\{M^{\mathfrak{D}}_{\sigma^r} \mid \sigma^r \text{ memoryless deterministic scheduler } \} = \{D_r \mid r \in \mathcal{R}^{\mathfrak{D}}\}^5.$$

Consequently, the feasibility synthesis problem for φ has the solution $r \in \mathcal{R}^{\mathfrak{D}}$ iff there exists a memoryless deterministic scheduler σ^r such that $M^{\mathfrak{D}}_{\sigma^r} \vDash \varphi$.

Approach 2 (All-in-one [17]) *Model checking the all-in-one MDP determines max or min probability (or expected reward) for all states, and thereby for all realisations, and thus provides a solution to both synthesis problems.*

As also the all-in-one MDP may be too large for realistic problems, we merely use it as formal starting point for our abstraction-refinement loop.

4.2 Abstraction

First, we define a predicate abstraction that at each state of the MDP *forgets* in which realisation we are, i.e., abstracts the second component of a state (s, r).

Definition 8 (Forgetting) *Let $M^{\mathfrak{D}} = (S^{\mathfrak{D}}, s_0^{\mathfrak{D}}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}})$ be an all-in-one MDP.* Forgetting *is an equivalence relation $\sim_f \subseteq S^{\mathfrak{D}} \times S^{\mathfrak{D}}$ satisfying*

$$(s, r) \sim_f (s', r') \iff s = s' \text{ and } s_0^{\mathfrak{D}} \sim_f (s_0^{\mathfrak{D}}, r) \; \forall r \in \mathcal{R}^{\mathfrak{D}}.$$

Let $[s]_\sim$ denote the equivalence class wrt. $\sim_f$ containing state $s \in S^{\mathfrak{D}}$.

Forgetting induces the quotient MDP $M^{\mathfrak{D}}_{\sim} = (S^{\mathfrak{D}}_{\sim}, [s_0^{\mathfrak{D}}]_\sim, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}}_{\sim})$, *where* $\mathcal{P}^{\mathfrak{D}}_{\sim}([s]_\sim, a_r)([s']_\sim) = \mathfrak{P}(r)(s)(s')$.

At each state of the quotient MDP, the actions correspond to any realisation. It includes states that are unreachable in every realisation.

Remark 1 (Action space). According to Def. 8, for every state $[s]_\sim$ there are $|\mathfrak{D}|$ actions. Many of these actions lead to the same distributions over successor states. In particular, two different realisations r and r' lead to the same distribution in s if $r(k) = r'(k)$ for all $k \in K$ where $\mathfrak{P}(s)(k) \neq 0$. To avoid this spurious blow-up of actions, we *a-priori* merge all actions yielding the same distribution.

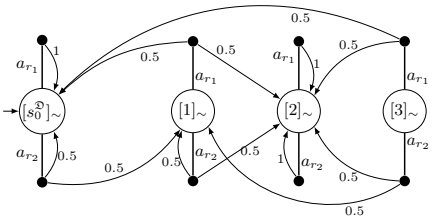


Fig. 3. The quotient MDP $M^{\mathfrak{D}}_{\sim}$ for realisations r_1 and r_2.

The quotient MDP under forgetting involves that the available actions allow to switch realisations and thereby create induced MCs different from any MC in $\mathfrak{D}$. We formalise the notion of a consistent realisation with respect to parameters.

Definition 9 (Consistent realisation) *For a family $\mathfrak{D}$ of MCs and $k \in K$, k-*realisation-consistency *is an equivalence relation $\approx_k \subseteq \mathcal{R}^{\mathfrak{D}} \times \mathcal{R}^{\mathfrak{D}}$ satisfying:*

$$r \approx_k r' \iff r(k) = r'(k).$$

Let $[r]_{\approx_k}$ denote the equivalence class w.r.t. $\approx_k$ containing $r \in \mathcal{R}^{\mathfrak{D}}$.

Definition 10 (Consistent scheduler) *For quotient MDP $M^{\mathfrak{D}}_{\sim}$ after forgetting and $k \in K$, a scheduler $\sigma \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ is k-*consistent *if for all $\pi, \pi' \in \mathsf{Paths}^{M^{\mathfrak{D}}_{\sim}}_{fin}$:*

$$\sigma(\pi) = a_r \wedge \sigma(\pi') = a_{r'} \implies r \approx_k r' \ .$$

*A scheduler is K-*consistent *(short:* consistent*) if it is k-consistent for all $k \in K$.*

Lemma 1. *For the quotient MDP $M^{\mathfrak{D}}_{\sim}$ of family $\mathfrak{D}$ of MCs:*

$$\{ \left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}} \mid \sigma^{r^*} \text{ consistent scheduler } \} = \{ D_r \mid r \in \mathcal{R}^{\mathfrak{D}} \}.$$

Proof (Idea). For $\sigma^r \in \Sigma^{M^{\mathfrak{D}}}$, we construct $\sigma^{r^*} \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ such that $\sigma^{r^*}([s]_\sim) = a_r$ for all s. Clearly σ^{r^*} is consistent and $M^{\mathfrak{D}}_{\sigma^r} = \left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}}$ is obtained via a map between (s, r) and $[s]_\sim$. For $\sigma^{r^*} \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$, we construct $\sigma^r \in \Sigma^{M^{\mathfrak{D}}}$ such that if $\sigma^{r^*}([s]_\sim) = a_r$ then $\sigma^r(s_0^{\mathfrak{D}}) = a_r$. For all other states, we define $\sigma^r((s, r')) = a^{r'}$ independently of σ^{r^*}. Then $M^{\mathfrak{D}}_{\sigma^r} = \left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}}$ is obtained as above.

The following theorem is a direct corollary: we need to consider exactly the consistent schedulers.

Theorem 2 *For all-in-one MDP $M^{\mathfrak{D}}$ and specification φ, there exists a memoryless deterministic scheduler $\sigma^r \in \Sigma^{M^{\mathfrak{D}}}$ such that $M^{\mathfrak{D}}_{\sigma^r} \vDash \varphi$ iff there exists a consistent deterministic scheduler $\sigma^{r^*} \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ such that $\left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}} \vDash \varphi$.*

[5] The original initial state s_0 of the family of MCs needs to be the initial state of $M^{\mathfrak{D}}_{\sigma^r}$.

Example 4. Recall the all-in-one MDP $M^{\mathfrak{D}}$ from Ex. 3. The quotient MDP $M^{\mathfrak{D}}_{\sim}$ is depicted in Fig. 3. Only the transitions according to realisations r_1 and r_2 are included. Transitions from previously unreachable states, marked grey in Ex. 3, are now available due to the abstraction. The scheduler $\sigma \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ with $\sigma([s_0^{\mathfrak{D}}]_{\sim}) = a_{r_2}$ and $\sigma([1]_{\sim}) = a_{r_1}$ is *not* k_1*-consistent* as different values are chosen for k_1 by r_1 and r_2. In the MC $M^{\mathfrak{D}}_{\sim\sigma}$ induced by σ and $M^{\mathfrak{D}}_{\sim}$, the probability to reach state $[2]_{\sim}$ is one, while under realisation r_1, state 2 is not reachable.

Approach 3 (Scheduler iteration) *Enumerating all consistent schedulers for $M^{\mathfrak{D}}_{\sim}$ and analysing the induced MC provides a solution to both synthesis problems.*

However, optimising over exponentially many consistent schedulers solves the NP-complete feasibility synthesis problem, rendering such an iterative approach unlikely to be efficient. Another natural approach is to employ solving techniques for NP-complete problems, like satisfiability modulo linear real arithmetic.

Approach 4 (SMT) *A dedicated SMT-encoding (in [1]) of the induced MCs of consistent schedulers from $M^{\mathfrak{D}}_{\sim}$ that solves the feasibility problem.*

4.3 Refinement Loop

Although iterating over consistent schedulers (Approach 3) is not feasible, model checking of $M^{\mathfrak{D}}_{\sim}$ still provides useful information for the analysis of the family $\mathfrak{D}$. Recall the feasibility synthesis problem for $\varphi = \mathbb{P}_{\leq\lambda}(\phi)$. If $\texttt{Prob}^{\max}(M^{\mathfrak{D}}_{\sim}, \phi) \leq \lambda$, then all realisations of $\mathfrak{D}$ satisfy φ. On the other hand, $\texttt{Prob}^{\min}(M^{\mathfrak{D}}_{\sim}, \phi) > \lambda$ implies that there is no realisation satisfying φ. If λ lies between the min and max probability, and the scheduler inducing the min probability is not consistent, we cannot conclude anything yet, i.e., the abstraction is too coarse. A natural countermeasure is to refine the abstraction represented by $M^{\mathfrak{D}}_{\sim}$, in particular, split the set of realisations leading to two synthesis sub-problems.

Definition 11 (Splitting) *Let $\mathfrak{D}$ be a family of MCs, and $\mathcal{R} \subseteq \mathcal{R}^{\mathfrak{D}}$ a set of realisations. For $k \in K$ and predicate A_k over S,* splitting *partitions $\mathcal{R}$ into*

$$\mathcal{R}_{\top} = \{r \in \mathcal{R} \mid A_k(r(k))\} \quad \textit{and} \quad \mathcal{R}_{\bot} = \{r \in \mathcal{R} \mid \neg A_k(r(k))\}.$$

Splitting the set of realisations, and considering the subfamilies separately, rather than splitting states in the quotient MDP, is crucial for the performance of the synthesis process as we avoid rebuilding the quotient MDP in each iteration. Instead, we only restrict the actions of the MDP to the particular subfamily.

Definition 12 (Restricting) *Let $M^{\mathfrak{D}}_{\sim} = (S^{\mathfrak{D}}_{\sim}, [s_0^{\mathfrak{D}}]_{\sim}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}}_{\sim})$ be a quotient MDP and $\mathcal{R} \subseteq \mathcal{R}^{\mathfrak{D}}$ a set of realisations. The* restriction *of $M^{\mathfrak{D}}_{\sim}$ wrt. $\mathcal{R}$ is the MDP $M^{\mathfrak{D}}_{\sim}[\mathcal{R}] = (S^{\mathfrak{D}}_{\sim}, [s_0^{\mathfrak{D}}]_{\sim}, Act^{\mathfrak{D}}[\mathcal{R}], \mathcal{P}^{\mathfrak{D}}_{\sim})$ where $Act^{\mathfrak{D}}[\mathcal{R}] = \{a_r \mid r \in \mathcal{R}\}$.*[6]

[6] Naturally, $\mathcal{P}^{\mathfrak{D}}_{\sim}$ in $M^{\mathfrak{D}}_{\sim}[\mathcal{R}]$ is restricted to $Act^{\mathfrak{D}}[\mathcal{R}]$.

Algorithm 1 Threshold synthesis

Input: A family $\mathfrak{D}$ of MCs with the set $\mathcal{R}^{\mathfrak{D}}$ of realisations, and specification $\mathbb{P}_{\leq\lambda}(\phi)$
Output: A partition of $\mathcal{R}^{\mathfrak{D}}$ into subsets T and F according to Problem 1.

$F \leftarrow \emptyset,\ T \leftarrow \emptyset,\ U \leftarrow \{\mathcal{R}^{\mathfrak{D}}\}$
$M^{\mathfrak{D}}_{\sim} \leftarrow \texttt{buildQuotientMDP}(\mathfrak{D}, \mathcal{R}^{\mathfrak{D}}, \sim_f)$ ▷ Applying Def. 7 and 8
while $U \neq \emptyset$ **do**
  select $\mathcal{R} \in U$ **and** $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathcal{R}\}$
  $M^{\mathfrak{D}}_{\sim}[\mathcal{R}] \leftarrow \texttt{restrict}(M^{\mathfrak{D}}_{\sim}, \mathcal{R})$ ▷ Applying Def. 12
  $(\max, \sigma_{\max}) \leftarrow \texttt{solveMaxMDP}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)$
  $(\min, \sigma_{\min}) \leftarrow \texttt{solveMinMDP}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)$
  if $\max < \lambda$ **then** $T \leftarrow T \cup \mathcal{R}$
  if $\min > \lambda$ **then** $F \leftarrow F \cup \mathcal{R}$
  if $\min \leq \lambda \leq \max$ **then**
    $U \leftarrow U \cup \texttt{split}(\mathcal{R}, \texttt{selPredicate}(\max, \sigma_{\max}, \min, \sigma_{\min}))$ ▷ See Sect. 4.4
return T, F

The splitting operation is the core of the proposed abstraction-refinement. Due to space constraints, we do not consider feasibility separately.

Algorithm 1 illustrates the *threshold synthesis* process. Recall that the goal is to decompose the set $\mathcal{R}^{\mathfrak{D}}$ into realisations satisfying and violating a given specification, respectively. The algorithm uses a set U to store subfamilies of $\mathcal{R}^{\mathfrak{D}}$ that have not been yet classified as satisfying or violating. It starts building the quotient MDP with merged actions. That is, we never construct the all-in-one MDP, and we merge actions as discussed in Rem. 1. For every $\mathcal{R} \in U$, the algorithm restricts the set of realisations to obtain the corresponding subfamily. For the restricted quotient MDP, the algorithm runs standard MDP model checking to compute the max and min probability and corresponding schedulers, respectively. Then, the algorithm either classifies $\mathcal{R}$ as satisfying/violating, or splits it based on a suitable predicate, and updates U accordingly. We describe the splitting strategy in the next subsection. The algorithm terminates if U is empty, i.e., all subfamilies have been classified. As only a finite number of subfamilies of realisations has to be evaluated, termination is guaranteed.

The refinement loop for max synthesis is very similar, cf. Alg. 2. Recall that now the goal is to find the realisation r^* that maximises the satisfaction probability $\max^*$ of a path formula. The difference between the algorithms lies in the interpretation of the results of the underlying MDP model checking. If the max probability for $\mathcal{R}$ is below $\max^*$, $\mathcal{R}$ can be discarded. Otherwise, we check whether the corresponding scheduler $\sigma_{\max}$ is consistent. If consistent, the algorithm updates r^* and $\max^*$, and discards $\mathcal{R}$. If the scheduler is not consistent but $\min > \max^*$ holds, we can still update $\max^*$ and improve the pruning process, as it means that some realisation (we do not know which) in $\mathcal{R}$ induces a higher probability than $\max^*$. Regardless whether $\max^*$ has been updated, the algorithm has to split $\mathcal{R}$ based on some predicate, and analyse its subfamilies as they may include the maximising realisation.

Algorithm 2 Max synthesis

Input: A family $\mathfrak{D}$ of MCs with the set $\mathcal{R}^{\mathfrak{D}}$ of realisations, and a path formula ϕ
Output: A realisation $r^* \in \mathcal{R}^{\mathfrak{D}}$ according to Problem 2.

$\max^* \leftarrow -\infty,\ U \leftarrow \{\mathcal{R}^{\mathfrak{D}}\}$
$M_{\sim}^{\mathfrak{D}} \leftarrow \texttt{buildQuotientMDP}(\mathfrak{D}, \mathcal{R}^{\mathfrak{D}}, \sim_f)$ ▷ Applying Def. 7 and 8
while $U \neq \emptyset$ **do**
  select $\mathcal{R} \in U$ **and** $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathcal{R}\}$
  $M_{\sim}^{\mathfrak{D}}[\mathcal{R}] \leftarrow \texttt{restrict}(M_{\sim}^{\mathfrak{D}}, \mathcal{R})$ ▷ Applying Def. 12
  $(\max, \sigma_{\max}) \leftarrow \texttt{solveMaxMDP}(M_{\sim}^{\mathfrak{D}}[\mathcal{R}], \phi)$
  $(\min, \sigma_{\min}) \leftarrow \texttt{solveMinMDP}(M_{\sim}^{\mathfrak{D}}[\mathcal{R}], \phi)$
  if $\max > \max^*$ **then**
    if $\texttt{isConsistent}(\sigma_{\max})$ **then** $r^* \leftarrow q_{\max}, \max^* \leftarrow \max$
    else
      if $\min > \max^*$ **then** $\max^* \leftarrow \min$
      $U \leftarrow U \cup \texttt{split}(\mathcal{R}, \texttt{selPredicate}(\max, \sigma_{\max}, \min, \sigma_{\min}))$ ▷ See Sect. 4.4
return r^*

4.4 Splitting strategies

If verifying the quotient MDP $M_{\sim}^{\mathfrak{D}}[\mathcal{R}]$ cannot classify the (sub-)realisation $\mathcal{R}$ as satisfying or violating, we split $\mathcal{R}$, while we guide the splitting strategy by using the obtained verification results. The splitting operation chooses a suitable parameter $k \in K$ and predicate A_k that partition the realisations $\mathcal{R}$ into $\mathcal{R}_\top$ and $\mathcal{R}_\bot$ (see Def. 11). A good splitting strategy globally reduces the number of model-checking calls required to classify all $r \in \mathcal{R}$.

The two key aspects to locally determine a good k are: 1) the *variance*, that is, how the splitting may narrow the difference between $\max = \texttt{Prob}^{\max}(M_{\sim}^{\mathfrak{D}}[\mathcal{X}], \phi)$ and $\min = \texttt{Prob}^{\min}(M_{\sim}^{\mathfrak{D}}[\mathcal{X}], \phi)$ for both $\mathcal{X} = \mathcal{R}_\top$ or $\mathcal{X} = \mathcal{R}_\bot$, and 2) the *consistency*, that is, how the splitting may reduce the inconsistency of the schedulers $\sigma_{\max}$ and $\sigma_{\min}$. These aspects cannot be evaluated precisely without applying all the split operations and solving the new MDPs $M_{\sim}^{\mathfrak{D}}[\mathcal{R}_\bot]$ and $M_{\sim}^{\mathfrak{D}}[\mathcal{R}_\top]$. Therefore, we propose an efficient strategy that selects k and A_k based on a light-weighted analysis of the model-checking results for $M_{\sim}^{\mathfrak{D}}[\mathcal{R}]$. The strategy applies two *scores* $\texttt{variance}(k)$ and $\texttt{consistency}(k)$ that estimate the influence of k on the two key aspects. For any k, the scores are accumulated over all *important states* s (reachable via $\sigma_{\max}$ or $\sigma_{\min}$, respectively) where $\mathfrak{P}(s)(k) \neq 0$. A state s is important for $\mathcal{R}$ and some $\delta \in \mathbb{R}_{\geq 0}$ if

$$\frac{\texttt{Prob}^{\max}(M_{\sim}^{\mathfrak{D}}[\mathcal{R}], \phi)(s) - \texttt{Prob}^{\min}(M_{\sim}^{\mathfrak{D}}[\mathcal{R}], \phi)(s)}{\texttt{Prob}^{\max}(M_{\sim}^{\mathfrak{D}}[\mathcal{R}], \phi) - \texttt{Prob}^{\min}(M_{\sim}^{\mathfrak{D}}[\mathcal{R}], \phi)} \geq \delta$$

where $\texttt{Prob}^{\min}(.)(s)$ and $\texttt{Prob}^{\max}(.)(s)$ is the min and max probability in the MDP with initial state s. To reduce the overhead of computing the scores, we simplify the scheduler representation. In particular, for $\sigma_{\max}$ and every $k \in K$, we extract a map $C_{\max}^k \colon T_k \to \mathbb{N}$, where $C_{\max}^k(t)$ is the number of important states for which $\sigma_{\max}(s) = a_r$ with $r(k) = t$. The mapping $C_{\min}^k$ represents $\sigma_{\min}$.

We define $\texttt{variance}(k) = \sum_{t \in T_k} |C^k_{\max}(t) - C^k_{\min}(t)|$, leading to high scores if the two schedulers vary a lot. Further, we define $\texttt{consistency}(k) = \texttt{size}\left(C^k_{\max}\right) \cdot \texttt{max}\left(C^k_{\max}\right) + \texttt{size}\left(C^k_{\min}\right) \cdot \texttt{max}\left(C^k_{\min}\right)$, where $\texttt{size}\,(C) = |\{t \in T_k \mid C(t) > 0\}| - 1$ and $\texttt{max}\,(C) = \max_{t \in T_k}\{C(t)\}$, leading to high scores if the parameter has clear favourites for $\sigma_{\max}$ and $\sigma_{\min}$, but values from its full range are chosen.

As indicated, we consider different strategies for the two synthesis problems. For threshold synthesis, we favour the impact on the variance as we principally do not need consistent schedulers. For the max synthesis, we favour the impact on the consistency, as we need a consistent scheduler inducing the max probability.

Predicate A_k is based on reducing the variance: The strategy selects $T' \subset T_k$ with $|T'| = \frac{1}{2}\lceil |T_k| \rceil$, containing those t for which $C^k_{\max}(t) - C^k_{\min}(t)$ is the largest. The goal is to get a set of realisations that induce a large probability (the ones including T' for parameter k) and the complement inducing a small probability.

Approach 5 (MDP-based abstraction refinement) *The methods underlying Algorithms 1 and 2, together with the splitting strategies, provide solutions to the synthesis problems and are referred to as* MDP abstraction *methods.*

5 Experiments

We implemented the proposed synthesis methods as a Python prototype using Storm [24]. In particular, we use the Storm Python API for model-adaption, -building, and -checking as well as for scheduler extraction. For SMT solving, we use Z3 [38] via pySMT [25]. The tool-chain takes a PRISM [37] or JANI [8] model with open integer constants, together with a set of expressions with possible values for these constants. The model may include the parallel composition of several modules/automata. The open constants may occur in guards[7], probability definitions, and updates of the commands/edges. Via adequate annotations, we identify the parameter values that yield a particular action. The annotations are key to interpret the schedulers, and to restrict the quotient without rebuilding.

All experiments were executed on a Macbook MF839LL/A with 8GB RAM memory limit and a 12h time out. All algorithms can significantly benefit from coarse-grained parallelisation, which we therefore do not consider here.

5.1 Research questions and benchmarks

The goal of the experimental evaluation is to answer the research question: *How does the proposed MDP-based abstraction methods (Approaches 3–5) cope with the inherent complexity (i.e. the NP-hardness) of the synthesis problems (cf. Problems 1 and 2)?* To answer this question, we compare their performance with Approaches 1 and 2 [17], representing state-of-the-art solutions and the base-line algorithms. The experiments show that the performance of the MDP abstraction significantly varies for different case studies. Thus, we consider benchmarks from various application domains to *identify the key characteristics of the synthesis problems affecting the performance of our approach.*

[7] slight care by the user is necessary to avoid deadlocks.

Table 1. Benchmarks and timings for Approach 1–3

				Member size		Quotient size			Run time		
Bench.	Range	$\|K\|$	$\|\mathcal{D}\|$	Avg. $\|S\|$	Avg. $\|T\|$	$\|S\|$	$\|A\|$	$\|T\|$	1-by-1	All-in-1	Sched. Enum.
Pole	[3.35,3.82]	17	1327104	5689	16896	6793	7897	22416	130k*	MO	26k
Maze	[9.8,9800]	20	1048576	134	211	203	277	409	28k	TO	2.7k
Herman	[1.86,2.44]	9	576	5287	6948	21313	102657	184096	55	72	246
DPM	[68,210]	9	32768	5572	18147	35154	66096	160146	2.9k	MO	7.2k
BSN	[0,0.988]	10	1024	116	196	382	457	762	31	2	2

Benchmarks description. We consider the following case studies: *Maze* is a planning problem typically considered as POMDP, e.g. in [40]. The family describes all MCs induced by small-memory [13, 34] observation-based deterministic strategies (with a fixed upper bound on the memory). We are interested in the expected time to the goal. In [34], parameter synthesis was used to find randomised strategies, using [21]. *Pole* considers balancing a pole in a noisy and unknown environment (motivated by [2, 11]). At deploy time, the controller has a prior over a finite set of environment behaviours, and should optimise the expected behavior without depending on the actual (hidden) environment. The family describes schedulers that do not depend on the hidden information. We are interested in the expected time until failure. *Herman* is an asynchronous encoding of the distributed Herman protocol for self-stabilising rings [32, 36]. The protocol is extended with a bit of memory for each station in the ring, and the choice to flip various unfair coins. Nodes in the ring are anonymous, they all behave equivalently (but may change their local memory based on local events). The family describes variations of memory-updates and coin-selection, but preserves anonymity. We are interested in the expected time until stabilisation. *DPM* considers a partial information scheduler for a disk power manager motivated by [7, 26]. We are interested in the expected energy consumption. *BSN* (Body sensor network, [42]) describes a network of connected sensors that identify health-critical situations. We are interested in the reliability. The family contains various configurations of the used sensors. *BSN* is the largest software product line benchmark used in [17]. We drop some implications between features (parameters for us) as this is not yet supported by our modelling language. We thereby extended the family.

Table 1 shows the relevant statistics for each benchmark: the benchmark name, the (approximate) range of the min and max probability/reward for the given family, the number of non-singleton parameters $|K|$, and the number of family members $|\mathfrak{D}|$. Then, for the family members the average number of states and transitions of the MCs, and the states, actions ($= \sum_{s \in S} |Act(s)|$), and transitions of the quotient MDP. Finally, it lists in seconds the run time of the base-line algorithms and the consistent scheduler enumeration[8]. The base-line algorithms employ the one-by-one and the all-in-one technique, using either a BDD or a sparse matrix representation. We report the best results. MOs indicate breaking the memory limit. Only the all-in-one approach required significant memory. As expected, the SMT-based implementation provides an inferior performance and thus we do not report its results.

[8] Values with a * are estimated by sampling a large fraction of the family.

Table 2. Results for threshold synthesis via abstraction-refinement

Inst	λ	#Below	# Subf below	#Above	# Subf above	Singles	#Iter	Time	Build	Check	Anal.	Speedup
Pole	3.37	697	176	1326407	2186	920	4723	308	117	60	118	**421**
	3.73	1307077	7854	20027	3279	1294	22265	1.7k	576	317	396	**77**
	3.76	1322181	3140	4923	1025	1022	8329	584	187	114	197	**222**
	3.79	1326502	572	602	123	74	1389	58	23	10	23	**2.2k**
Maze	10	4	3	1048572	92	4	189	5	<1	3	< 1	**26k**
	20	4247	2297	1044329	4637	3400	13867	114	21	43	29	**246**
	30	18188	9934	1030388	18004	14010	55875	608	80	127	270	**46**
	8000	1046285	846	2291	1125	969	3941	136	9	106	13	**1.0k**
Herman	1.9	6	6	570	368	320	747	333	303	11	18	**0.2**
	1.71	0	0	576	258	184	515	232	206	8	17	**0.3**
DPM	80	160	141	32608	1292	356	2865	1.0k	602	322	64	**3**
	70	6	6	32762	443	40	897	380	190	156	32	**8**
	60	0	0	32768	104	6	207	99	42	48	8	**29**
BSN	.965	544	81	480	81	25	321	2	<1	<1	<1	**1**
	.985	994	41	30	8	5	97	<1	<1	<1	<1	**3**

5.2 Results and discussion

To simplify the presentation, we focus primarily on the threshold synthesis problem as it allows a compact presentation of the key aspects. Below, we provide some remarks about the performance for the max and feasibility synthesis.

Results. Table 2 shows results for threshold synthesis. The first two columns indicate the benchmark and the various thresholds. For each threshold λ, the table lists the number of family members below (above) λ, each with the number of subfamilies that together contain these instances, and the number of singleton subfamilies that were considered. The last table part gives the number of iterations of the loop in Alg. 1, and timing information (total, build/restrict times, model checking times, scheduler analysis times). The last column gives the speed-up over the best base-line (based on the estimates).

Key observations. The speed-ups drastically vary, which shows that the MDP abstraction often achieves a superior performance but may also lead to a performance degradation in some cases. We identify four key factors.

Iterations. As typical for CEGAR approaches, the key characteristic of the benchmark that affects the performance is the number N of iterations in the refinement loop. The abstract action introduces an overhead per iteration caused by performing two MDP verification calls and by the scheduler analysis. The run time for *BSN*, with a small $|\mathfrak{D}|$ is actually significantly affected by the initialisation of various data structures; thus only a small speedup is achieved.

Abstraction size. The size of the quotient, compared to the average size of the family members, is relevant too. The quotient includes at least all reachable states of all family members, and may be significantly larger if an inconsistent scheduler reaches states which are unreachable under any consistent scheduler. The existence of such states is a common artefact from encoding families in high-level languages. Table 1, however, indicates that we obtain a very compact representation for *Maze* and *Pole*.

Thresholds. The most important aspect is the threshold λ. If λ is closer to the optima, the abstraction requires a smaller number of iterations, which directly improves the performance. We emphasise that in various domains, thresholds that ask for close-to-optimal solutions are indeed of highest relevance as they typically represent the system designs developers are most interested in [43]. *Why do thresholds affect the number of iterations?* Consider a family with $T_k = \{0, 1\}$ for each k. Geometrically, the set $\mathcal{R}^{\mathfrak{D}}$ can be visualised as $|K|$-dimensional cube. The cube-vertices reflect family members. Assume for simplicity that one of these vertices is optimal with respect to the specification. Especially in benchmarks where parameters are equally important, the induced probability of a vertex roughly corresponds to the Manhattan distance to the optimal vertex. Thus, vertices above the threshold induce a diagonal hyperplane, which our splitting method approximates with orthogonal splits. Splitting diagonally is not possible, as it would induce optimising over observation-based schedulers. Consequently, we need more and more splits the more the diagonal goes through the middle of the cube. *Even when splitting optimally, there is a combinatorial blow-up in the required splits when the threshold is further from the optimal values.* Another effect is that thresholds far from optima are more affected by the over-approximation of the MDP model-checking results and thus yield more inconclusive answers.

Refinement strategy. So far, we reasoned about optimal splits. Due to the computational overhead, our strategy cannot ensure optimal splits. Instead, the strategy depends mostly on information encoded in the computed MDP strategies. *In models where the optimal parameter value heavily depends on the state, the obtained schedulers are highly inconsistent and carry only limited information for splitting.* Consequently, in such benchmarks we split sub-optimally. The sub-optimality has a major impact on the performance for *Herman* as all obtained strategies are highly inconsistent – they take a different coin for each node, which is good to speed up the stabilisation of the ring.

Summary. MDP abstraction is not a silver bullet. It has a lot of potential in threshold synthesis when the threshold is close to the optima. Consequently, *feasibility synthesis with unsatisfiable specifications is handled perfectly well by MDP abstraction*, while this is the worst-case for enumeration-based approaches. Likewise, *max synthesis* can be understood as threshold synthesis with a shifting threshold $\max^*$: If the $\max^*$ is quickly set close to max, MDP abstraction yields superior performance. Roughly, we can quickly approximate $\max^*$ when some of the parameter values are clearly beneficial for the specification.

6 Conclusion and Future Work

We contributed to the efficient analysis of families of Markov chains. In particular, we discussed and implemented existing approaches to solve practically interesting synthesis problems, and devised a novel abstraction refinement scheme that mitigates the computational complexity of the synthesis problems, as shown by the empirical evaluation. In the future, we will include refinement strategies based on counterexamples as in [22, 33].

References

1. Additional material, https://github.com/moves-rwth/shepherd
2. Arming, S., Bartocci, E., Chatterjee, K., Katoen, J.P., Sokolova, A.: Parameter-independent strategies for pMDPs via POMDPs. In: QEST. LNCS, vol. 11024, pp. 53–70. Springer (2018)
3. Aziz, A., Singhal, V., Balarin, F., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: It usually works: The temporal logic of stochastic systems. In: CAV. LNCS, vol. 939, pp. 155–165. Springer (1995)
4. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking, pp. 963–999. Springer (2018)
5. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
6. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C., Smolka, S.: Model repair for probabilistic systems. In: TACAS, LNCS, vol. 6605, pp. 326–340. Springer (2011)
7. Benini, L., Bogliolo, A., Paleologo, G., Micheli, G.D.: Policy optimization for dynamic power management. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **8**(3), 299–316 (2000)
8. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: TACAS. LNCS, vol. 10206, pp. 151–168 (2017)
9. Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Efficient synthesis of robust models for stochastic systems. Journal of Systems and Software **143**, 140 – 158 (2018)
10. Češka, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M., Brim, L.: Precise parameter synthesis for stochastic biochemical systems. Acta Informatica **54**(6), 589–623 (2017)
11. Chades, I., Carwardine, J., Martin, T.G., Nicol, S., Sabbadin, R., Buffet, O.: MOMDPs: A solution for modelling adaptive management problems. In: AAAI. AAAI Press (2012)
12. Chasins, S., Phothilimthana, P.M.: Data-driven synthesis of full probabilistic programs. In: CAV. LNCS, vol. 10426, pp. 279–304. Springer (2017)
13. Chatterjee, K., Chmelik, M., Davies, J.: A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In: AAAI. pp. 3225–3232. AAAI Press (2016)
14. Chatterjee, K., Kößler, A., Schmid, U.: Automated analysis of real-time scheduling using graph games. In: HSCC. pp. 163–172. ACM (2013)
15. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M.Z., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: TASE. pp. 85–92. IEEE (2013)
16. Chonev, V.: Reachability in augmented interval Markov chains. CoRR **abs/1701.02996** (2017)
17. Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C.: ProFeat: feature-oriented engineering for family-based probabilistic model checking. Formal Asp. Comput. **30**(1), 45–75 (2018)
18. Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.: Model checking software product lines with SNIP. STTT **14**(5), 589–612 (2012)
19. Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.: Formal semantics, modular specification, and symbolic verification of product-line behaviour. Sci. Comput. Program. **80**, 416–439 (2014)
20. Cordy, M., Heymans, P., Legay, A., Schobbens, P.Y., Dawagne, B., Leucker, M.: Counterexample guided abstraction refinement of product-line behavioural models. In: SIGSOFT FSE. pp. 190–201. ACM (2014)

21. Cubuktepe, M., Jansen, N., Junges, S., Katoen, J.P., Topcu, U.: Synthesis in pMDPs: A tale of 1001 parameters. In: ATVA. LNCS, vol. 11138, pp. 160–176. Springer (2018)
22. Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.P.: Fast debugging of PRISM models. In: ATVA. LNCS, vol. 8837, pp. 146–162. Springer (2014)
23. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.P., Ábrahám, E.: PROPhESY: A PRObabilistic ParamEter SYNnthesis Tool. In: CAV. LNCS, vol. 9206, pp. 214–231. Springer (2015)
24. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: A modern probabilistic model checker. In: CAV. LNCS, vol. 10427, pp. 592–600. Springer (2017)
25. Gario, M., Micheli, A.: Pysmt: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT Workshop 2015 (2015)
26. Gerasimou, S., Calinescu, R., Tamburrelli, G.: Synthesis of probabilistic models for quality-of-service software engineering. Autom. Softw. Eng. **25**(4), 785–831 (2018)
27. Ghezzi, C., Sharifloo, A.M.: Model-based verification of quantitative non-functional properties for software product lines. Information & Software Technology **55**(3), 508–524 (2013)
28. Giro, S., D'Argenio, P.R., Fioriti, L.M.F.: Distributed probabilistic input/output automata: Expressiveness, (un)decidability and algorithms. Theor. Comput. Sci. **538**, 84–102 (2014)
29. Giro, S., Rabe, M.N.: Verification of partial-information probabilistic systems using counterexample-guided refinements. In: ATVA. LNCS, vol. 7561, pp. 333–348. Springer (2012)
30. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. Software Tools for Technology Transfer **13**(1), 3–19 (2011)
31. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6**(5), 512–535 (1994)
32. Herman, T.: Probabilistic self-stabilization. Inf. Process. Lett. **35**(2), 63–67 (1990)
33. Jansen, N., Wimmer, R., Ábrahám, E., Zajzon, B., Katoen, J.P., Becker, B., Schuster, J.: Symbolic counterexample generation for large discrete-time Markov chains. Sci. Comput. Program. **91**, 90–114 (2014)
34. Junges, S., Jansen, N., Wimmer, R., Quatmann, T., Winterer, L., Katoen, J.P., Becker, B.: Finite-state controllers of POMDPs using parameter synthesis. In: UAI. pp. 519–529. AUAI Press (2018)
35. Kochenderfer, M.J.: Decision Making Under Uncertainty: Theory and Application. The MIT Press, 1st edn. (2015)
36. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic verification of hermans self-stabilisation algorithm. Formal Aspects of Computing **24**(4), 661–670 (2012)
37. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011)
38. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
39. Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D.: Efficient synthesis of probabilistic programs. In: PLDI. pp. 208–217. ACM (2015)
40. Norman, G., Parker, D., Zou, X.: Verification and control of partially observable probabilistic systems. Real-Time Systems **53**(3), 354–402 (2017)
41. Pathak, S., Ábrahám, E., Jansen, N., Tacchella, A., Katoen, J.P.: A greedy approach for the efficient repair of stochastic models. In: NFM. LNCS, vol. 9058, pp. 295–309. Springer (2015)

42. Rodrigues, G.N., Alves, V., Nunes, V., Lanna, A., Cordy, M., Schobbens, P., Sharifloo, A.M., Legay, A.: Modeling and verification for probabilistic properties in software product lines. In: HASE. pp. 173–180. IEEE (2015)
43. Skaf, J., Boyd, S.: Techniques for exploring the suboptimal set. Optimization and Engineering **11**(2), 319–337 (2010)
44. Vandin, A., ter Beek, M.H., Legay, A., Lluch-Lafuente, A.: QFLan: A tool for the quantitative analysis of highly reconfigurable systems. In: FM. LNCS, vol. 10951, pp. 329–337. Springer (2018)
45. Varshosaz, M., Khosravi, R.: Discrete time Markov chain families: modeling and verification of probabilistic software product lines. In: SPLC Workshops. pp. 34–41. ACM (2013)