A Comparison of Time- and Reward-Bounded Probabilistic Model Checking Techniques

Ernst Moritz Hahn[1(✉)] and Arnd Hartmanns[2(✉)]

[1] Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Twente, Enschede, The Netherlands
hahn@ios.ac.cn, a.hartmanns@utwente.nl

Abstract. In the design of probabilistic timed systems, requirements concerning behaviour that occurs within a given time or energy budget are of central importance. We observe that model-checking such requirements for probabilistic timed automata can be reduced to checking reward-bounded properties on Markov decision processes. This is traditionally implemented by unfolding the model according to the bound, or by solving a sequence of linear programs. Neither scales well to large models. Using value iteration in place of linear programming achieves scalability but accumulates approximation error. In this paper, we correct the value iteration-based scheme, present two new approaches based on scheduler enumeration and state elimination, and compare the practical performance and scalability of all techniques on a number of case studies from the literature. We show that state elimination can significantly reduce runtime for large models or high bounds.

1 Introduction

Probabilistic timed automata (PTA, [17]) are a popular formal model for probabilistic real-time systems. They combine nondeterministic choices as in Kripke structures, discrete probabilistic decisions as in Markov chains, and hard real-time behaviour as in timed automata. We are interested in properties of the form "what is the best/worst-case probability to eventually reach a certain system state while accumulating at most b reward", i.e. in calculating reward-bounded reachability probabilities. Rewards can model a wide range of aspects, e.g. the number of retransmissions in a network protocol (accumulating reward 1 for each), energy consumption (accumulating reward at a state-dependent wattage over time), or time itself (accumulating reward at rate 1 everywhere). Reachability probabilities for PTA with rewards can be computed by first turning a PTA into an equivalent Markov decision process (MDP) using the digital clocks semantics [17] and then performing standard probabilistic model checking [3].

The naïve approach to compute specifically *reward-bounded* reachability probabilities is to *unfold* [1] the state space of the model. For the example of

This work was supported by the 3TU.BSR project, by CDZ project 1023 (CAP), by the Chinese Academy of Sciences Fellowship for International Young Scientists, and by the National Natural Science Foundation of China (grant no. 61550110506).

M. Fränzle et al. (Eds.): SETTA 2016, LNCS 9984, pp. 85–100, 2016.
DOI: 10.1007/978-3-319-47677-3_6

time-bounded properties, this means adding a new clock variable that is never reset [17]. In the general case on the level of MDP [19], in addition to the current state of the model, one keeps track of the reward accumulated so far, up to b. This turns the reward-bounded problem into standard unbounded reachability. Unfolding blows up the model size (the number of states, or the number of variables and constraints in the corresponding linear program) and causes the model checking process to run out of memory even if the original (unbounded) model was of moderate size (cf. Table 1). For PTA, unfolding is the only approach that has been considered so far. A more efficient technique has been developed for MDP, and via the digital clocks semantics it is applicable to PTA just as well:

The probability for bound i depends only on the values for previous bounds $\{ i-r, \ldots, i-1 \}$ where r is the max. reward in the automaton. We can thus avoid the monolithic unfolding by sequentially computing the values for its "layers" where the accumulated reward is $i = 0$, 1, etc. up to b, storing the current layer and the last r result vectors only. This process can be implemented by solving a sequence of b linear programming (LP) problems no larger than the original unbounded model [2]. While it solves the memory problem in principle, LP is known not to scale to large MDP in practice. Consequently, LP has been replaced by value iteration to achieve scalability in the most recent implementation [14]. Value iteration is an approximative numeric technique to compute reachability probabilities up to a predefined error bound ϵ. When used in sequence, this error accumulates, and the final result for bound b may differ from the actual probability by more than ϵ. This has not been taken into account in [14].

In this paper, we first make a small change to the value iteration-based scheme to counteract the error accumulation. We then present two new ways to compute reward-bounded reachability probabilities for MDP (with a particular interest in the application to PTA via digital clocks) without unfolding (Sect. 3). Using either scheduler enumeration or MDP state elimination, they both reduce the model such that a reward of 1 is accumulated on all remaining transitions. A *reward-bounded* property in the original model corresponds to a *step-bounded* property in the reduced model. We use standard step-bounded value iteration [3] to check these properties efficiently and exactly. Observe that we improve the practical efficiency of computing reward-bounded probabilities, but the problem is Exp-complete in general [6]. It can be solved in time polynomial in the size of the MDP and the *value* of b, i.e. it is only pseudo-polynomial in b. Like all related work, we only present solutions for the case of nonnegative integer rewards.

The unfolding-free techniques also provide the probability for all lower bounds $i < b$. This has been exploited to obtain quantiles [2], and we use it more generally to compute the entire cumulative (sub)distribution function (*cdf* for short) over the bound up to b at no extra cost. We have implemented all techniques in the mcsta tool (Sect. 4) of the Modest Toolset [10]. It is currently the only publicly available implementation of reward-bounded model checking for PTA and MDP without unfolding. We use it to study the relative performance and scalability of the previous and new techniques on six example models from the literature (Sect. 5). State elimination in particular shows promising performance.

Other Related Work. Randour et al. [18] have studied the complexity of computing reward-bounded probabilities (framed as percentile queries) for MDP with *multiple* rewards and reward bounds. They propose an algorithm based on unfolding. For the soft real-time model of Markov automata, which subsumes MDP, reward bounds can be turned into time bounds [13]. Yet this only works for rewards associated to Markovian states, whereas immediate states (i.e. the MDP subset of Markov automata) always implicitly get zero reward.

2 Preliminaries

$\mathbb{N}$ is $\{0, 1, \dots\}$, the set of natural numbers. 2^S is the powerset of S. $\mathrm{Dom}(f)$ is the domain of the function f.

Definition 1. *A (discrete)* probability distribution *over a set* Ω *is a function* $\mu \in \Omega \to [0,1]$ *such that* $\mathrm{support}(\mu) \overset{\text{def}}{=} \{\omega \in \Omega \mid \mu(\omega) > 0\}$ *is countable and* $\sum_{\omega \in \mathrm{support}(\mu)} \mu(\omega) = 1$. $\mathrm{Dist}(\Omega)$ *is the set of all probability distributions over* Ω. $\mathcal{D}(s)$ *is the* Dirac distribution *for* s, *defined by* $\mathcal{D}(s)(s) = 1$.

Markov Decision Processes. To move from one state to another in a Markov decision process, first a transition is chosen nondeterministically; each transition then leads into a probability distribution over rewards and successor states.

Definition 2. *A* Markov decision process *(MDP) is a triple* $M = \langle S, T, s_{init} \rangle$ *where* S *is a finite set of states,* $T \in S \to 2^{\mathrm{Dist}(\mathbb{N} \times S)}$ *is the transition function, and* $s_{init} \in S$ *is the initial state. For all* $s \in S$, *we require that* $T(s)$ *is finite and non-empty.* M *is a* discrete-time Markov chain *(DTMC) if* $\forall\, s \in S\colon |T(s)| = 1$.

We write $s \to_T \mu$ for $\exists\, \mu \in T(s)$ and call it a *transition.* We write $s \xrightarrow{r}_T s'$ if additionally $\langle r, s' \rangle \in \mathrm{support}(\mu)$. $\langle r, s' \rangle$ is a *branch* with reward r. If T is clear from the context, we write just $\to$. Graphically, transitions are lines to an intermediate node from which branches labelled with reward (if not zero) and probability lead to successor states. We may omit the intermediate node and probability 1 for transitions into Dirac distributions, and we may label transitions to refer to them in the text. Figure 1 shows an example MDP M^e with 5 states, 7 (labelled) transitions and 10 branches. Using branch rewards instead of the more standard transition rewards leads to more compact models; in the example, we assign reward 1 to the branches back to s and t to count the number of "failures" before reaching v. In practice, high-level formalisms like Prism's [15] guarded command language are used to specify MDP. They extend MDP with variables over finite domains that can be used in expressions to e.g. enable/disable transitions. This allows to compactly describe very large MDP.

Definition 3. *A* finite path *in* $M = \langle S, T, s_{init} \rangle$ *is defined as a finite sequence* $\pi_{\mathrm{fin}} = s_0\, \mu_0\, r_0\, s_1\, \mu_1\, r_1\, s_2 \dots \mu_{n-1}\, r_{n-1}\, s_n$ *where* $s_i \in S$ *for all* $i \in \{0, \dots, n\}$ *and* $s_i \to \mu_i \wedge \langle r_i, s_{i+1} \rangle \in \mathrm{support}(\mu_i)$ *for all* $i \in \{0, \dots, n-1\}$. *Let* $|\pi_{\mathrm{fin}}| \overset{\text{def}}{=} n$, $\mathrm{last}(\pi_{\mathrm{fin}}) \overset{\text{def}}{=} s_n$, *and* $\mathrm{reward}(\pi_{\mathrm{fin}}) = \sum_{i=0}^{n-1} r_i$. $\mathrm{Paths}_{\mathrm{fin}}(M)$ *is the set of all finite*

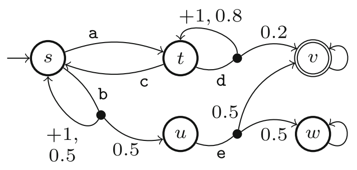

Fig. 1. Example MDP M^e

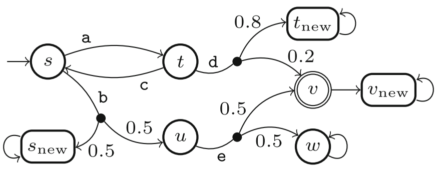

Fig. 2. Transformed MDP $M^e{\downarrow_{F^e}}{\downarrow_{\mathrm{R}}}$

paths starting with s_{init}. A path *is an infinite sequence $\pi = s_0\,\mu_0\,r_0\,s_1\,\mu_1\,r_1\ldots$ where $s_i \in S$ and $s_i \to \mu_i \wedge \langle r_i, s_{i+1}\rangle \in \mathrm{support}(\mu_i)$ for all $i \in \mathbb{N}$.* Paths(M) *is the set of all paths starting with s_{init}. We define $s \in \pi \overset{\text{def}}{\Leftrightarrow} \exists\, i\colon s = s_i$.*

Definition 4. *Given $M = \langle S, T, s_{init}\rangle$, $\mathfrak{S} \in \mathrm{Paths_{fin}}(M) \to \mathrm{Dist}(\mathrm{Dist}(\mathbb{N} \times S))$ is a* scheduler *for M if $\forall\,\pi_{\text{fin}}\colon \mu \in \mathrm{support}(\mathfrak{S}(\pi_{\text{fin}})) \Rightarrow \mathrm{last}(\pi_{\text{fin}}) \to \mu$. The set of all schedulers of M is* Sched(M)*. $\mathfrak{S}$ is* reward-positional *if* $\mathrm{last}(\pi_1) = \mathrm{last}(\pi_2) \wedge \mathrm{reward}(\pi_1) = \mathrm{reward}(\pi_2)$ *implies $\mathfrak{S}(\pi_1) = \mathfrak{S}(\pi_2)$,* positional *if* $\mathrm{last}(\pi_1) = \mathrm{last}(\pi_2)$ *alone implies $\mathfrak{S}(\pi_1) = \mathfrak{S}(\pi_2)$, and* deterministic *if* $|\mathrm{support}(\mathfrak{S}(\pi))| = 1$, *for all finite paths π, π_1 and π_2, respectively. A* simple *scheduler is positional and deterministic. The set of all simple schedulers of M is* SSched(M)*.*

Let $M{\downarrow_{\mathfrak{S}_s}} \overset{\text{def}}{=} \langle S, T', s_{init}\rangle$ with $T'(s) \overset{\text{def}}{=} \{\,\mu \mid \mathfrak{S}_s(s) = \mathcal{D}(\mu)\}$ for $\mathfrak{S}_s \in \mathrm{SSched}(M)$. $M{\downarrow_{\mathfrak{S}_s}}$ is a DTMC. Using the standard cylinder set construction [3], a scheduler $\mathfrak{S}$ induces a probability measure $\mathcal{P}_M^{\mathfrak{S}}$ on measurable sets of paths starting from s_{init}. We define the *extremal* values $\mathcal{P}_M^{\max}(\Pi) = \sup_{\mathfrak{S}\in\mathrm{Sched}(M)} \mathcal{P}_M^{\mathfrak{S}}(\Pi)$ and $\mathcal{P}_M^{\min}(\Pi) = \inf_{\mathfrak{S}\in\mathrm{Sched}(M)} \mathcal{P}_M^{\mathfrak{S}}(\Pi)$ for measurable $\Pi \subseteq \mathrm{Paths}(M)$.

For an MDP M and goal states $F \subseteq S$, we define the *unbounded*, *step-bounded* and *reward-bounded* reachability probabilities for $opt \in \{\max, \min\}$:

- $\mathrm{P}_{opt}(F) \overset{\text{def}}{=} \mathcal{P}_M^{opt}(\{\,\pi \in \mathrm{Paths}(M) \mid \exists\, s \in F\colon s \in \pi\})$ is the extremal probability of eventually reaching a state in F.
- $\mathrm{P}_{opt}^{\mathrm{S}\le b}(F)$ is the extremal probability of reaching a state in F via at most $b \in \mathbb{N}$ transitions, defined as $\mathcal{P}_M^{opt}(\Pi_b^{\mathrm{T}})$ where Π_b^{T} is the set of paths that have a prefix of length at most b that contains a state in F.
- $\mathrm{P}_{opt}^{\mathrm{R}\le b}(F)$ is the extremal probability of reaching a state in F with accumulated reward at most $b \in \mathbb{N}$, defined as $\mathcal{P}_M^{opt}(\Pi_b^{\mathrm{R}})$ where Π_b^{R} is the set of paths that have a prefix π_{fin} containing a state in F with $\mathrm{reward}(\pi_{\text{fin}}) \le b$.

Theorem 1. *For an unbounded property, there exists an optimal simple scheduler, i.e. one that attains the extremal value* [3]*. For a reward-bounded property, there exists an optimal deterministic reward-positional scheduler* [12]*.*

Continuing our example, let $F^e = \{\,v\}$. We maximise the probability to eventually reach F^e in M^e by always scheduling transition `a` in s and `d` in t, so $\mathrm{P}_{\max}(F^e) = 1$ with a simple scheduler. We get $\mathrm{P}_{\max}^{\mathrm{R}\le 0}(F^e) = 0.25$ by scheduling `b` in s. For higher bound values, simple schedulers are no longer sufficient: we get $\mathrm{P}_{\max}^{\mathrm{R}\le 1}(F^e) = 0.4$ by first trying `a` then `d`, but falling back to `c` then `b` if we return to t. We maximise the probability for higher bound values n by trying `d` until the accumulated reward is $n - 1$ and then falling back to `b`.

Probabilistic Timed Automata. Probabilistic timed automata (PTA [17]) extend MDP with *clocks* and *clock constraints* as in timed automata to model real-time behaviour and requirements. PTA have two kinds of rewards: branch rewards as in MDP and *rate rewards* that accumulate at a certain rate over time. Time itself is a rate reward that is always 1. The digital clocks approach [17] is the only PTA model checking technique that works well with rewards. It works by replacing the clock variables by bounded integers and adding self-loop edges to increment them synchronously as long as time can pass. The reward of a self-loop edge is the current rate reward. The result is (a high-level model of) a finite *digital clocks MDP*. All the algorithms that we develop for MDP in this paper can thus be applied to PTA. While time- and branch reward-bounded properties on PTA are decidable [17], general rate reward-bounded properties are not [4].

Probabilistic Model Checking. Probabilistic model checking for MDP (and thus for PTA via the digital clocks semantics) works in two phases: (1) state space *exploration* turns a given high-level model into an in-memory representation of the underlying MDP, then (2) a numerical *analysis* computes the value of the property of interest. In phase 1, the goal states are made absorbing:

Definition 5. *Given $M = \langle S, T, s_{init} \rangle$ and $F \subseteq S$, we define the F-*absorbing *MDP as $M{\downarrow_F} = \langle S, T', s_{init} \rangle$ with $T'(s) = \{\, \mathcal{D}(\langle 1, s \rangle) \}$ for all $s \in F$ and $T'(s) = T(s)$ otherwise. For $s \in S$, we define $M[s] = \langle S, T, s \rangle$.*

An efficient algorithm for phase 2 and unbounded properties is (unbounded) value iteration [3]. We denote a call to a value iteration implementation by $\mathtt{VI}(V, M{\downarrow_F}, opt, \epsilon)$ with initial value vector $V \in S \to [0, 1]$ and $opt \in \{\max, \min\}$. Internally, it iteratively approximates over all states s a (least) solution for

$$V(s) = opt_{\mu \in T(s)} \textstyle\sum_{\langle r, s' \rangle \in \mathrm{support}(\mu)} \mu(\langle r, s' \rangle) \cdot V(s')$$

up to (relative) error ϵ. Let initially $V = \{\, s \mapsto 1 \mid s \in F \} \cup \{\, s \mapsto 0 \mid s \in S \setminus F \}$. Then on termination of $\mathtt{VI}(V, M{\downarrow_F}, opt, \epsilon)$, we have $V(s) \approx_\epsilon \mathrm{P}_{opt}(F)$ in $M[s]$ for all $s \in S$. All current implementations in model checking tools like Prism [15] use a simple convergence criterion based on ϵ that in theory only guarantees $V(s) \leq \mathrm{P}_{opt}(F)$, yet in practice delivers ϵ-close results on most, but not all, case studies. Guaranteed ϵ-close results could be achieved at the cost of precomputing and reducing a maximal end component decomposition of the MDP [7]. In this paper, we thus write VI to refer to an ideal ϵ-correct algorithm, but for the sake of comparison use the standard implementation in our experiments in Sect. 5.

For a step-bounded property, the call $\mathtt{StepBoundedVI}(V = V_0, M{\downarrow_F}, opt, b)$ with bound b can be implemented [3] by computing for all states

$$V_i(s) := opt_{\mu \in T(s)} \textstyle\sum_{\langle r, s' \rangle \in \mathrm{support}(\mu)} \mu(\langle r, s' \rangle) \cdot V_{i-1}(s')$$

iteratively for $i = 1, \ldots, b$. After iteration i, we have $V_i(s) = \mathrm{P}_{opt}^{\mathrm{S} \leq i}(F)$ in $M[s]$ for all $s \in S$ when starting with V as in the unbounded case above. Note that this algorithm computes exact results (modulo floating-point precision and errors) without any costly preprocessing and is very easy to implement and parallelise.

Reward-bounded properties can naïvely be checked by *unfolding* the model according to the accumulated reward: we add a variable v to the model prior to phase 1, with branch reward r corresponding to an assignment $v := v + r$. To check $\mathrm{P}_{opt}^{\mathrm{R}\leq b}(F)$, phase 1 thus creates an MDP that is up to b times as large as without unfolding. In phase 2, $\mathrm{P}_{opt}(F')$ is checked using `VI` as described above where F' corresponds to the states in F where additionally $v \leq b$ holds.

3 Reward-Bounded Analysis Techniques

We describe three techniques that allow the computation of reward-bounded reachability probabilities on MDP (and thus PTA) without unfolding. The first one is a reformulation of the value iteration-based variant [14] of the algorithm introduced in [2]. We incorporate a simple fix for the problem that the error accumulation over the sequence of value iterations had not been accounted for and refer to the result as algorithm modvi. We then present two new techniques senum and elim that avoid the issues of unbounded value iteration by transforming the MDP such that step-bounded value iteration can be used instead.

From now on, we assume that all rewards are either zero or one. This simplifies the presentation and is in line with our motivation of improving time-bounded reachability for PTA: in the corresponding digital clocks MDP, all transitions representing the passage of time have reward 1 while the branches of all other transitions have reward 0. Yet it is without loss of generality: for modvi, it is merely a matter of a simplified presentation, and for the two new algorithms, we can preprocess the MDP to replace each branch with reward $r > 1$ by a chain of r Dirac transitions with reward 1. While this may blow up the state space, we found that most models in practice only use rewards 0 and 1 in the first place: among the 15 MDP and DTMC models currently distributed with PRISM [15], only 2 out of the 12 examples that include a reward structure do not satisfy this assumption. It also holds for all case studies that we present in Sect. 5.

For all techniques, we need a transformation $\downarrow_{\mathrm{R}}$ that redirects each reward-one branch to a copy s'_{new} of the branch's original target state s'. In effect, this replaces branch rewards by branches to a distinguished category of "new" states:

Definition 6. *Given $M = \langle S, T, s_{init}\rangle$, we define $M{\downarrow_{\mathrm{R}}}$ as $\langle S \uplus S_{\mathrm{new}}, T^{\downarrow}, s_{init}\rangle$ with $S_{\mathrm{new}} = \{\, s_{\mathrm{new}} \mid s \in S \}$,*

$$T^{\downarrow}(s) = \begin{cases} \{\, \mathit{Conv}(s,\mu) \mid \mu \in T(s) \} & \text{if } s \in S \\ \{\, \mathcal{D}(\langle 0, s\rangle) \} & \text{if } s \in S_{\mathrm{new}} \end{cases}$$

and $\mathit{Conv}(s,\mu) \in \mathrm{Dist}(\mathbb{N} \times S \uplus S_{\mathrm{new}})$ is defined by $\mathit{Conv}(s,\mu)(\langle 0, s'\rangle) = \mu(\langle 0, s'\rangle)$ and $\mathit{Conv}(s,\mu)(\langle 1, s'_{\mathrm{new}}\rangle) = \mu(\langle 1, s'\rangle)$ over all $s' \in S$.

For our example MDP M^e and $F^e = \{\, v \}$, we show $M^e{\downarrow_{F^e}}{\downarrow_{\mathrm{R}}}$ in Fig. 2. Observe that $M^e{\downarrow_{F^e}}$ is the same as M^e, except that the self-loop of goal state v

```
function ModVI($V, M = \langle S, T, s_{init} \rangle, F, b, opt, \epsilon$)
    for $i = 1$ to $b$ do
        foreach $s_{\text{new}} \in S_{\text{new}}$ do $V(s_{\text{new}}) := V(s)$
        VI($V, M{\downarrow_F}{\downarrow_{\text{R}}}, opt, \frac{\epsilon}{b+1}$)
```

Algorithm 1. Sequential value iterations for reward-bounded reachability

gets reward 1. $M^e{\downarrow_{F^e}}{\downarrow_{\text{R}}}$ is then obtained by redirecting the three reward-one branches (originally going to s, t and v) to new states s_{new}, t_{new} and v_{new}.

All of the algorithm descriptions we present take a value vector V as input, which they update. V must initially contain the probabilities to reach a goal state in F with zero reward, which can be computed for example via a call to $\texttt{VI}(V = V_F^0, M{\downarrow_F}{\downarrow_{\text{R}}}, opt, \epsilon)$ with sufficiently small ϵ and

$$V_F^0 \overset{\text{def}}{=} \{\, s \mapsto 1, s_{\text{new}} \mapsto 0 \mid s \in F \} \cup \{\, s \mapsto 0, s_{\text{new}} \mapsto 0 \mid s \in S \setminus F \}.$$

3.1 Sequential Value Iterations

We recall the technique for model-checking reward-bounded properties of [2] that avoids unfolding. It was originally formulated as a *sequence* of linear programming (LP) problems LP_i, each corresponding to bound $i \leq b$. Each LP_i is of the same size as the original (non-unfolded) MDP, representing its state space, but uses the values computed for $LP_{i-r}, \ldots, LP_{i-1}$ with r being the maximal reward that occurs in the MDP. Since LP does not scale to large MDP [7], the technique has been reconsidered using value iteration instead [14]. Using the transformations and assumptions introduced above, we can formulate it as in Algorithm 1. Initially, V contains the probabilities to reach a goal state with zero reward. In contrast to [14], when given an overall error bound ϵ, we use bound $\frac{\epsilon}{b+1}$ for the individual value iteration calls. At the cost of higher runtime, this counteracts the accumulation of error over multiple calls to yield an ϵ-close final result:

Consider $M{\downarrow_F}{\downarrow_{\text{R}}} = \langle S, T, s_{init} \rangle$ and $f \in (S \to [0,1]) \to (S \to [0,1])$ with $f = \lim_i f_i$ where for $V \in S \to [0,1]$ it is $f_0(V)(s) = V(s)$ and $f_{i+1}(V)(s) = opt_\mu \sum_{s'} \mu(s') \cdot f_i(V)(s')$, i.e. f corresponds to performing an ideal value iteration with error $\epsilon = 0$. Thus, performing Algorithm 1 using f would result in an error of 0. If we limit the error in each value iteration to $\frac{\epsilon}{b+1}$, then the function we use can be stated as $f' = f_n$ for n large enough such that $||f'(V) - f(V)||_{\max} \leq \frac{\epsilon}{b+1}$ for all V used in the computations. Let V_0, $V_0' = V_0 + \delta_0$ be the initial value vectors, $\delta_0 < \frac{\epsilon}{b+1}$. Further, let $V_i, V_i' \in S \to [0,1]$, $i \in \{1, \ldots, b\}$, be the value vectors after the i-th call to VI for the case without (V_i) and with error (V_i'). We can then show by induction that $||V_i - V_i'||_{\max} \leq (i+1)\frac{\epsilon}{b+1}$. Initially, we have $V_0' = V_0 + \delta_0$. Therefore, we have $V_1' = f'(V_0') = f(V_0') + \delta_1 = f(V_0) + \sum_{j=0}^{0} \delta_j + \delta_1$ for some $\delta_1 \in S \to [0,1]$ with $||\delta_1||_{\max} \leq \frac{\epsilon}{b+1}$. Then, we have for some $\delta_j \in S \to [0,1]$, $j \in \{0, \ldots, i\}$, with $||\delta_j||_{\max} \leq \frac{\epsilon}{b+1}$:

$$V_{i+1}' = f'(V_i') = f(V_i') + \delta_{i+1} \overset{*}{=} f(V_i) + \sum\nolimits_{j=0}^{i} \delta_j + \delta_{i+1} = f(V_i) + \sum\nolimits_{j=0}^{i+1} \delta_j$$

```
function SEnum($V, M = \langle S, T, s_{init} \rangle, F, b, opt$)
    $T'' := \varnothing$, $M' := M{\downarrow_F}{\downarrow_{\mathrm{R}}} = \langle S \uplus S_{\mathrm{new}}, T', s_{init} \rangle$
    foreach $s \in \{ s_{init} \} \cup \{ s'' \mid \exists\, s' \colon s' \xrightarrow{1}_T s'' \}$ do
        foreach $\mathfrak{S} \in \mathrm{SSched}(M'[s])$ do        // enumeration of simple schedulers
            $T''(s) := T''(s) \cup \{ \texttt{ComputeProbs}(M'[s]{\downarrow_{\mathfrak{S}}}) \}$
    $T'' := T'' \cup \{ \bot \mapsto \{ \mathcal{D}(\langle 0, \bot \rangle) \} \}$, $V(\bot) := 0$
    StepBoundedVI($V, M'' = \langle \mathrm{Dom}(T''), T'', s_{init} \rangle, b, opt$)     // step-bounded iter.
function ComputeProbs($M = \langle S \uplus S_{\mathrm{new}}, \ldots \rangle$)                  // M is a DTMC
    $\mu := \{ \langle 0, s \rangle \mapsto \mathrm{P}_{\max=\min}(\{ s_{\mathrm{new}} \}) \mid s_{\mathrm{new}} \in S_{\mathrm{new}} \}$
    return $\mu \cup \{ \langle 0, \bot \rangle \mapsto 1 - \sum_{s_{\mathrm{new}} \in S_{\mathrm{new}}} \mu(\langle 0, s \rangle) \}$
```

Algorithm 2. Reward-bounded reachability via scheduler enumeration

where $*$ holds by the induction assumption. Finally, $|| \sum_{j=0}^{i} \delta_j ||_{\max} \leq (i+1) \frac{\epsilon}{b+1}$, so $||V_i - V_i'||_{\max} \leq (i+1) \frac{\epsilon}{b+1}$, which is what had to be proved.

3.2 Scheduler Enumeration

Our first new technique, senum, is summarised as Algorithm 2. The idea is to replace the entire sub-MDP between a "relevant" state and the new states (that follow immediately after what was a reward-one branch before the $\downarrow_{\mathrm{R}}$ transformation) by *one direct* transition to a distribution over the new states *for each* simple scheduler. The actual reward-bounded probabilities can be computed on the result MDP M'' using the standard step-bounded algorithm (line 7), since one *step* now corresponds to a *reward* of 1.

The relevant states, which remain in the result MDP M'', are the initial state plus those states that had an incoming reward-one branch. We iterate over them in line 3. In an inner loop (line 4), we iterate over the simple schedulers for each relevant state. For each scheduler, ComputeProbs determines the distribution μ s.t. for each new state s_{new}, $\mu(s_{\mathrm{new}})$ is the probability of reaching it (accumulating 1 reward on the way) and $\mu(\bot)$ is the probability of getting stuck in an end component without being able to accumulate any more reward ever. A transition to preserve μ in M'' is created in line 5. The total number of simple schedulers for n states with max. fan-out m is in $\mathcal{O}(m^n)$, but we expect the number of schedulers that lead to different distributions from one relevant state up to the next reward-one steps to remain manageable (cf. column "avg" in Table 2).

ComputeProbs is implemented either using value iterations, one for each new state, or—since $M'[s]{\downarrow_{\mathfrak{S}}}$ is a DTMC—using DTMC *state elimination* [8]. The latter successively eliminates the non-new states as shown schematically in Fig. 3 while preserving the reachability probabilities, all in one go.

3.3 State Elimination

Instead of performing a probability-preserving DTMC state elimination for each scheduler as in senum, technique elim applies a new scheduler- and

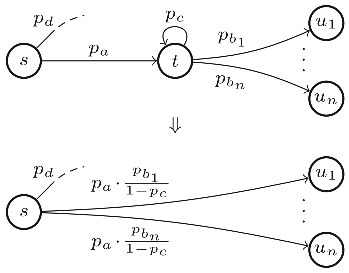
p_d
p_a
p_c
p_{b_1}
p_{b_n}
s
t
u_1
u_n
$\Downarrow$
$p_a \cdot \frac{p_{b_1}}{1-p_c}$
$p_a \cdot \frac{p_{b_n}}{1-p_c}$

Fig. 3. DTMC state elimination [8]

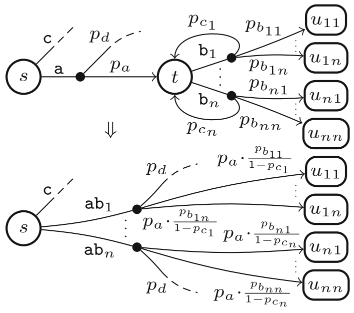
c
a
p_d
p_a
p_{c_1}
p_{c_n}
b$_1$
b$_n$
$p_{b_{11}}$
$p_{b_{1n}}$
$p_{b_{n1}}$
$p_{b_{nn}}$
u_{11}
u_{1n}
u_{n1}
u_{nn}
$\Downarrow$
ab$_1$
ab$_n$
$p_a \cdot \frac{p_{b_{11}}}{1-p_{c_1}}$
$p_a \cdot \frac{p_{b_{1n}}}{1-p_{c_1}}$
$p_a \cdot \frac{p_{b_{n1}}}{1-p_{c_n}}$
$p_a \cdot \frac{p_{b_{nn}}}{1-p_{c_n}}$

Fig. 4. MDP state elimination

```
function Elim($V, M = \langle S, T, s_{init} \rangle, F, b, opt$)
    $M' := M{\downarrow_F}{\downarrow_R} = \langle S \uplus S_{\text{new}}, \ldots \rangle$
    $\langle S \uplus S_{\text{new}}, T', s_{init} \rangle :=$ Eliminate($M', S$)          // MDP state elimination
    $T'' := \{ \bot \mapsto \{ \mathcal{D}(\langle 0, \bot \rangle) \} \}$, $V(\bot) := 0$, $\mu' := \varnothing$
    foreach $s_{\text{new}} \in S_{\text{new}}$ and $\mu \in T'(s)$ do          // state merging
        $\mu' := \mu' \cup \{ \bot \mapsto \sum_{\langle 0, s' \rangle \in \text{support}(\mu) \wedge s' \in S} \mu(\langle 0, s' \rangle) \}$
        $\mu' := \mu' \cup \{ s' \mapsto \mu(\langle 0, s'_{\text{new}} \rangle) \mid \langle 0, s'_{\text{new}} \rangle \in \text{support}(\mu) \wedge s'_{\text{new}} \in S_{\text{new}} \}$
        $T''(s_{\text{new}}) := T''(s_{\text{new}}) \cup \{ \mu' \}$, $\mu' := \varnothing$
    StepBoundedVI($V, \langle \text{Dom}(T''), T'', s_{init} \rangle, b, opt$)          // step-bounded iteration
```

Algorithm 3. Reward-bounded reachability via MDP state elimination

probability-preserving state elimination algorithm to the entire MDP. The state elimination algorithm is described by the schema shown in Fig. 4; states with Dirac self-loops will remain. Observe how this elimination process preserves the options that simple schedulers have, and in particular relies on their positional character to be able to redistribute the loop probabilities p_{c_i} onto the same transition only.

elim is shown as Algorithm 3. In line 3, the MDP state elimination procedure is called to eliminate all the regular states in S. We can ignore rewards here since they were transformed by $\downarrow_R$ into branches to the distinguished new states. As an extension to the schema of Fig. 4, we also preserve the original outgoing transitions when we eliminate a relevant state (defined as in Sect. 3.2) because we need them in the next step: In the loop starting in line 5, we redirect (1) all branches that go to non-new states to the added bottom state $\bot$ instead because they indicate that we can get stuck in an end component without reward, and (2) all branches that go to new states to the corresponding original states instead. This way, we merge the (absorbing, but not eliminated) new states with the corresponding regular (eliminated from incoming but not outgoing transitions) states. Finally, in line 8, the standard step-bounded value iteration is performed on the eliminated-merged MDP as in senum. Figure 5 shows our example MDP after state elimination, and Fig. 6 shows the subsequent merged MDP. For clarity, transitions to the same successor distributions are shown in a combined way.

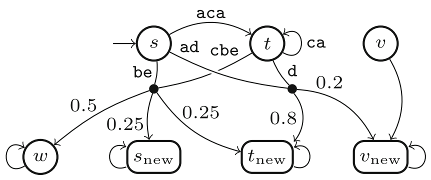

Fig. 5. $M_{\downarrow}^{e}$ after state elimination

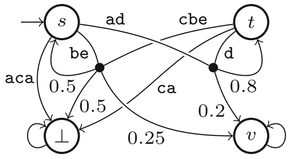

Fig. 6. $M_{\downarrow}^{e}$ eliminated and merged

3.4 Correctness and Complexity

Correctness. Let $\mathfrak{S}$be a deterministic reward-positional scheduler for $M{\downarrow_F}$. It corresponds to a sequence of simple schedulers $\mathfrak{S}_i$ for $M{\downarrow_F}$ where $i \in \{ b, \ldots, 0\}$ is the remaining reward that can be accumulated before the bound is reached. For each state s of $M{\downarrow_F}$ and $i > 0$, each such $\mathfrak{S}_i$ induces a (potentially substochastic) measure μ_s^i such that $\mu_s^i(s')$ is the probability to reach s' from s in $M{\downarrow_F}{\downarrow_{\mathfrak{S}_i}}$ over paths whose last step has reward 1. Let μ_s^0 be the induced measure such that $\mu_s^0(s')$ is the probability under $\mathfrak{S}_0$ to reach s' without reward if it is a goal state and 0 otherwise. Using the recursion $\overline{\mu}_s^i(s') \overset{\text{def}}{=} \sum_{s'' \in S} \mu_s^i(s'') \cdot \overline{\mu}_{s''}^{i-1}(s')$ with $\overline{\mu}_s^0 \overset{\text{def}}{=} \mu_s^0$, the value $\overline{\mu}_s^b(s')$ is the probability to reach goal state s' from s in $M{\downarrow_F}$ under $\mathfrak{S}$. Thus we have $\max_{\mathfrak{S}} \overline{\mu}_s^b(s') = \mathrm{P}_{\max}^{\mathrm{R} \leq b}(F)$ and $\min_{\mathfrak{S}} \overline{\mu}_s^b(s') = \mathrm{P}_{\min}^{\mathrm{R} \leq b}(F)$ by Theorem 1. If we distribute the maximum operation into the recursion, we get

$$\max_{\mathfrak{S}} \overline{\mu}_s^i(s') = \sum\nolimits_{s'' \in S} \max_{\mathfrak{S}_i} \mu_s^i(s'') \cdot \max_{\mathfrak{S}} \overline{\mu}_{s''}^{i-1}(s') \qquad (1)$$

and an analogous formula for the minimum. By computing extremal values w.r.t. simple schedulers for each reward step, we thus compute the value w.r.t. an optimal deterministic reward-positional scheduler for the bounded property overall. The correctness of `senum` and `elim` now follows from the fact that they implement precisely the right-hand side of (1): $\overline{\mu}_s^0$ is always given as the initial value of V as described at the very beginning of this section. In `senum`, we enumerate the relevant measures $\mu_s^{\cdot}$ induced by all the simple schedulers as one transition each, then choose the optimal transition for each i in the i-th iteration inside `StepBoundedVI`. The argument for `elim` is the same, the difference being that state elimination is what transforms all the measures into single transitions.

Complexity. The problem that we solve is Exp-complete [6]. We make the following observations about `senum` and `elim`: Let $n_{\text{new}} \leq n$ be the number of new states, $n_{\text{s}} \leq n$ the max. size of any relevant reward-free sub-MDP (i.e. the max. number of states reachable from the initial or a new state when dropping all reward-one branches), and $s_{\text{s}} \leq m^n$ the max. number of simple schedulers in these sub-MDP. The reduced MDP created by `senum` and `elim` have n_{new} states and up to $n_{\text{new}} \cdot s_{\text{s}} \cdot n_{\text{s}}$ branches. The bounded value iterations thus involve $\mathcal{O}(b \cdot n_{\text{new}} \cdot s_{\text{s}} \cdot n_{\text{s}})$ arithmetic operations overall. Note that in the worst case, $s_{\text{s}} = m^n$, i.e. it is exponential in the size of the original MDP. To obtain the reduced MDP, `senum` enumerates $\mathcal{O}(n_{\text{new}} \cdot s_{\text{s}})$ schedulers; for each, value iteration or DTMC state elimination is done on a sub-MDP of $\mathcal{O}(n_\mathrm{s})$ states. elim needs to eliminate $n - n_\mathrm{new}$ states, with each elimination requiring $\mathcal{O}(s_\mathrm{s} \cdot n_\mathrm{s})$ operations.

4 Implementation

We have implemented the three unfolding-free techniques within MCSTA, the MODEST TOOLSET's model checker for PTA and MDP. When asked to compute $\mathrm{P}^{\mathrm{R}\leq b}_{opt}(\cdot)$, it delivers *all* values $\mathrm{P}^{\mathrm{R}\leq i}_{opt}(\cdot)$ for $i \in \{0, \ldots, b\}$ since the algorithms allow doing so at no overhead. Instead of a single value, we thus get the entire (sub-)cdf. Every single value is defined via an individual optimisation over schedulers. However, we have seen in Sect. 3.4 that an optimal scheduler for bound i can be extended to an optimal scheduler for $i + 1$, so there exists an optimal scheduler for all bounds. The max./min. cdf represents the probability distribution induced by that scheduler. We show these functions for the randomised consensus case study [16] in Fig. 7. The top (bottom) curve is the max. (min.) probability for the protocol to terminate within the number of coin tosses given on the x-axis. For comparison, the left and right dashed lines show the means of these distributions. Note that the min. expected value corresponds to the max. bounded probabilities and vice-versa. As mentioned, using the unfolding-free techniques, we compute the curves in the same amount of memory otherwise sufficient for the means only. We also implemented a convergence criterion to detect when the result will no longer increase for higher bounds, i.e. when the unbounded probability has been reached up to ϵ. For the functions in Fig. 7, this happens at 4016 coin tosses for the max. and 5607 for the min. probability.

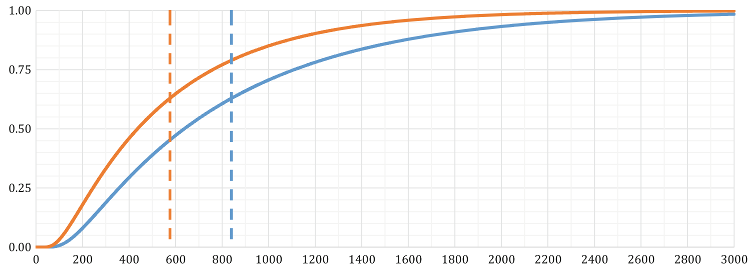

Fig. 7. Cdfs and means for the randomised consensus model ($H = 6, K = 4$)

5 Experiments

We use six case studies from the literature to evaluate the applicability and performance of the three unfolding-free techniques and their implementation:

– **BEB** [5]: MDP of a bounded exponential backoff procedure with max. backoff value $K = 4$ and $H \in \{5, \ldots, 10\}$ parallel hosts. We compute the max. probability of any host seizing the line while all hosts enter backoff $\leq b$ times.

- **BRP** [9]: The PTA model of the bounded retransmission protocol with $N \in \{32, 64\}$ frames to transmit, retransmission bound $MAX \in \{6, 12\}$ and transmission delay $TD \in \{2, 4\}$ time units. We compute the max. and min. probability that the sender reports success in $\leq b$ time units.
- **RCONS** [16]: The randomised consensus shared coin protocol MDP as described in Sect. 4 for $N \in \{4, 6\}$ parallel processes and constant $K \in \{2, 4, 8\}$.
- **CSMA** [9]: PTA model of a communication protocol using CSMA/CD, with max. backoff counter $BCMAX \in \{1, \dots, 4\}$. We compute the min. and max. probability that both stations deliver their packets by deadline b time units.
- **FW** [16]: PTA model ("Impl" variant) of the IEEE 1394 FireWire root contention protocol with either a short or a long cable. We ask for the min. probability that a leader (root) is selected before time bound b.
- **IJSS** [14]: MDP model of Israeli and Jalfon's randomised self-stabilising algorithm with $N \in \{18, 19, 20\}$ processes. We compute the min. probability to reach a stable state in $\leq b$ steps of the algorithm (query Q2 in [14]). This is a step-bounded property; we consider IJSS here only to compare with [14].

Experiments were performed on an Intel Core i5-6600T system (2.7 GHz, 4 cores) with 16 GB of memory running 64-bit Windows 10 and a timeout of 30 min.

Looking back at Sect. 3, we see that the only extra states introduced by modvi compared to checking an unbounded probabilistic reachability or expected-reward property are the new states s_{new}. However, this was for the presentation only, and is avoided in the implementation by checking for reward-one branches on-the-fly. The transformations performed in senum and elim, on the other hand, will reduce the number of states, but may add transitions and branches. elim may also create large intermediate models. In contrast to modvi, these two techniques may thus run out of memory even if unbounded properties can be checked. In Table 1, we show the state-space sizes (1) for the traditional unfolding approach ("unfolded") for the bound b where the values have converged, (2) when unbounded properties are checked or modvi is used ("non-unfolded"), and (3) after state elimination and merging in elim. We report thousands (K) or millions (M) of states, transitions ("trans") and branches ("branch"). Column "avg" lists the average size of all relevant reward-free sub-MDP. The values for senum are the same as for elim. Times are for the state-space exploration phase only, so the time for "non-unfolded" will be incurred by all three unfolding-free algorithms. We see that avoiding unfolding is a drastic reduction. In fact, 16 GB of memory are not sufficient for the larger unfolded models, so we used MCSTA's disk-based technique [11]. State elimination leads to an increase in transitions and especially branches, drastically so for BRP, the exceptions being BEB and IJSS. This appears related to the size of the reward-free subgraphs, so state elimination may work best if there are few steps between reward increments.

In Table 2, we report the performance results for all three techniques when run until the values have converged at bound value b (except for IJSS, where we follow [14] and set b to the 99th percentile). For senum, we used the variant based on value iteration since it consistently performed better than the one

Table 1. State spaces

			unfolded		non-unfolded					eliminated		
	model	b	states	time	states	trans	branch	time	avg	states	trans	branch
BEB	5	90	1 M	6 s	10 k	12 k	22 k	0 s	3.7	3 k	5 k	19 k
	6	143	6 M	35 s	45 k	60 k	118 k	0 s	3.3	16 k	28 k	104 k
	7	229	45 M	273 s	206 k	304 k	638 k	1 s	3.0	80 k	149 k	568 k
	8	371			1.0 M	1.6 M	3.4 M	3 s	2.8	0.4 M	0.8 M	3.1 M
	9	600	> 30 min		4.6 M	8.3 M	18.9 M	26 s	2.6	2.0 M	4.2 M	16.6 M
	10	n/a			22.2 M	44.0 M	102.8 M	138 s	2.5	> 16 GB		
BRP	32, 6, 2	179	9 M	40 s	0.1 M	0.1 M	0.1 M	0 s	24.2	0.1 M	0.4 M	7.1 M
	32, 6, 4	347	50 M	206 s	0.2 M	0.2 M	0.2 M	1 s	22.8	0.2 M	1.0 M	20.3 M
	32, 12, 2	179	22 M	90 s	0.2 M	0.2 M	0.3 M	1 s	22.8	0.2 M	1.1 M	22.1 M
	32, 12, 4	347	122 M	499 s	0.6 M	0.7 M	0.7 M	2 s	21.3	0.6 M	3.2 M	62.0 M
	64, 6, 2	322	38 M	157 s	0.1 M	0.2 M	0.2 M	0 s	47.1	0.1 M	1.3 M	53.8 M
	64, 6, 4	630	207 M	826 s	0.4 M	0.4 M	0.5 M	1 s	44.4	0.4 M	3.8 M	153.7 M
	64, 12, 2	322	107 M	427 s	0.5 M	0.5 M	0.5 M	1 s	44.4	0.4 M	4.1 M	166.0 M
	64, 12, 4	630	> 30 min		1.3 M	1.4 M	1.5 M	4 s	41.3	> 16 GB		
RCONS	4, 4	2653	54 M	365 s	41 k	113 k	164 k	0 s	4.5	35 k	254 k	506 k
	4, 8	7793			80 k	220 k	323 k	0 s	4.1	68 k	499 k	997 k
	6, 2	2175	> 30 min		1.2 M	5.0 M	7.2 M	5 s	11.7	1.1 M	23.6 M	47.1 M
	6, 4	5607			2.3 M	9.4 M	13.9 M	9 s	9.1	2.2 M	42.2 M	84.3 M
CSMA	1	2941	31 M	276 s	13 k	13 k	13 k	0 s	1.4	13 k	13 k	15 k
	2	3695	191 M	1097 s	96 k	96 k	97 k	0 s	1.3	95 k	95 k	110 k
	3	5229	> 30 min		548 k	548 k	551 k	2 s	1.2	545 k	545 k	637 k
	4	8219			2.7 M	2.7 M	2.7 M	9 s	1.2	2.7 M	2.7 M	3.2 M
FW	short	2487	9 M	150 s	4 k	6 k	6 k	0 s	4.0	4 k	111 k	413 k
	long	3081	> 30 min		0.2 M	0.5 M	0.5 M	1 s	3.4	0.2 M	2.4 M	7.7 M
IJSS	18	445			0.3 M	2.6 M	5.0 M	5 s	1.0	0.3 M	2.5 M	4.3 M
	19	498	> 30 min		0.5 M	5.5 M	10.5 M	10 s	1.0	0.5 M	5.2 M	9.0 M
	20	553			1.0 M	11.5 M	22.0 M	22 s	1.0	1.0 M	11.0 M	18.9 M

using DTMC state elimination. "iter" denotes the time needed for (unbounded or step-bounded) value iteration, while "enum" and "elim" are the times needed for scheduler enumeration resp. state elimination and merging. "#" is the total number of iterations performed over all states inside the calls to `VI`. "avg" is the average number of schedulers enumerated per relevant state; to get the approx. total number of schedulers enumerated for a model instance, multiply by the number of states for elim in Table 1. "rate" is the number of bound values computed per second, i.e. b divided by the time for value iteration. Memory usage in columns "mem" is MCSTA's peak working set, including state space exploration, reported in mega- (M) or gigabytes (G). MCSTA is garbage-collected, so these values are higher than necessary since full collections only occur when the system runs low on memory. The values related to value iteration for senum are the same as for elim. In general, we see that senum uses less memory than elim,

Table 2. Runtime and memory usage

			modvi				senum			elim			
	model	b	iter	#	mem	rate	enum	mem	avg	elim	iter	mem	rate
BEB	5	90	0 s	422	43 M	425 $\frac{1}{s}$	0 s	45 M	4.2	0 s	0 s	48 M	∞ $\frac{1}{s}$
	6	143	1 s	666	54 M	168 $\frac{1}{s}$	1 s	65 M	6.4	0 s	0 s	107 M	1430 $\frac{1}{s}$
	7	229	6 s	1070	132 M	34 $\frac{1}{s}$	11 s	210 M	10.7	2 s	0 s	409 M	1145 $\frac{1}{s}$
	8	371	56 s	1734	374 M	6 $\frac{1}{s}$	88 s	588 M	19.9	12 s	2 s	1.5 G	247 $\frac{1}{s}$
	9	600	487 s	2798	2.0 G	1 $\frac{1}{s}$	960 s	2.6 G	40.8	67 s	14 s	6.6 G	43 $\frac{1}{s}$
	10	n/a	> 30 min				> 30 min			> 16 GB			
BRP	32, 6, 2	179	1 s	837	56 M	209 $\frac{1}{s}$	160 s	474 M	6.1	4 s	1 s	775 M	164 $\frac{1}{s}$
	32, 6, 4	347	5 s	1520	87 M	81 $\frac{1}{s}$	498 s	1.2 G	5.9	12 s	7 s	2.6 G	61 $\frac{1}{s}$
	32, 12, 2	179	3 s	837	92 M	70 $\frac{1}{s}$	569 s	1.3 G	5.9	13 s	4 s	2.8 G	56 $\frac{1}{s}$
	32, 12, 4	347	17 s	1520	196 M	26 $\frac{1}{s}$	1467 s	3.5 G	5.5	40 s	22 s	6.1 G	20 $\frac{1}{s}$
	64, 6, 2	322	4 s	1451	77 M	105 $\frac{1}{s}$	> 30 min			31 s	16 s	4.7 G	24 $\frac{1}{s}$
	64, 6, 4	630	20 s	2735	140 M	38 $\frac{1}{s}$				114 s	91 s	14.1 G	8 $\frac{1}{s}$
	64, 12, 2	322	11 s	1451	149 M	34 $\frac{1}{s}$				132 s	51 s	13.9 G	8 $\frac{1}{s}$
	64, 12, 4	630	62 s	2735	328 M	12 $\frac{1}{s}$				> 16 GB			
RCONS	4, 4	2653	43 s	21763	61 M	108 $\frac{1}{s}$	2 s	126 M	17.5	1 s	3 s	224 M	1842 $\frac{1}{s}$
	4, 8	7793	260 s	66739	87 M	56 $\frac{1}{s}$	4 s	187 M	15.8	2 s	16 s	384 M	933 $\frac{1}{s}$
	6, 2	2175	1608 s	19291	680 M	2 $\frac{1}{s}$	> 30 min			136 s	169 s	11.9 G	20 $\frac{1}{s}$
	6, 4	5607	> 30 min							275 s	879 s	13.4 G	11 $\frac{1}{s}$
CSMA	1	2941	4 s	12220	45 M	1363 $\frac{1}{s}$	5 s	46 M	3.7	0 s	0 s	60 M	∞ $\frac{1}{s}$
	2	3695	30 s	15363	64 M	244 $\frac{1}{s}$	> 30 min			1 s	3 s	190 M	2437 $\frac{1}{s}$
	3	5229	226 s	21780	187 M	46 $\frac{1}{s}$				3 s	24 s	839 M	429 $\frac{1}{s}$
	4	8219	1689 s	34009	617 M	10 $\frac{1}{s}$				19 s	192 s	3.8 G	86 $\frac{1}{s}$
FW	short	2487	1 s	6608	40 M	2763 $\frac{1}{s}$	29 s	79 M	476.5	0 s	1 s	113 M	3109 $\frac{1}{s}$
	long	3081	60 s	9610	108 M	51 $\frac{1}{s}$	205 s	880 M	82.7	13 s	23 s	1.4 G	132 $\frac{1}{s}$
IJSS	18	445	55 s	891	527 M	8 $\frac{1}{s}$	5 s	876 M	10.0	16 s	3 s	1.7 G	23 $\frac{1}{s}$
	19	498	126 s	997	947 M	4 $\frac{1}{s}$	10 s	1.7 G	10.5	35 s	7 s	3.7 G	12 $\frac{1}{s}$
	20	553	304 s	1107	1.8 G	2 $\frac{1}{s}$	22 s	3.5 G	11.0	83 s	16 s	7.6 G	6 $\frac{1}{s}$

but is much slower in all cases except IJSS. If elim works and does not blow up the model too much, it is significantly faster than modvi, making up for the time spent on state elimination with much faster value iteration rates.

6 Conclusion

We presented three approaches to model-check reward-bounded properties on MDP without unfolding: a small correction of recent work based on unbounded value iteration [14], and two new techniques that reduce the model such that step-bounded value iteration can be used, which is efficient and exact. We also consider the application to time-bounded properties on PTA and provide the first implementation that is publicly available, within the Modest Toolset at modestchecker.net. By avoiding unfolding and returning the entire probability distribution up to the bound at no extra cost, this could finally make reward- and time-bounded probabilistic timed model checking feasible in practical applications. As we presented the algorithms in this paper, they compute reachability probabilities. However all of them can easily be adapted to compute reward-bounded expected accumulated rewards and instantaneous rewards, too.

Outlook. The digital clocks approach for PTA was considered limited in scalability. The presented techniques lift some of its most significant practical limitations. Moreover, time-bounded analysis without unfolding and with computation of the entire distribution in this manner is not feasible for the traditionally more scalable zone-based approaches because zones abstract from concrete timing. We see the possibility to improve the state elimination approach by removing transitions that are linear combinations of others and thus unnecessary. This may reduce the transition and branch blowup on models like the BRP case. Going beyond speeding up simple reward-bounded reachability queries, state elimination also opens up ways towards a more efficient analysis of long-run average and long-run reward-average properties.

References

1. Andova, S., Hermanns, H., Katoen, J.-P.: Discrete-time rewards model-checked. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 88–104. Springer, Heidelberg (2004). doi:10.1007/978-3-540-40903-8_8
2. Baier, C., Daum, M., Dubslaff, C., Klein, J., Klüppelholz, S.: Energy-utility quantiles. In: Badger, J.M., Rozier, K.Y. (eds.) NFM 2014. LNCS, vol. 8430, pp. 285–299. Springer, Heidelberg (2014). doi:10.1007/978-3-319-06200-6_24
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Massachusetts (2008)
4. Berendsen, J., Chen, T., Jansen, D.N.: Undecidability of cost-bounded reachability in priced probabilistic timed automata. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 128–137. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02017-9_16
5. Giro, S., D'Argenio, P.R., Ferrer Fioriti, L.M.: Partial order reduction for probabilistic systems: a revision for distributed schedulers. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 338–353. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04081-8_23
6. Haase, C., Kiefer, S.: The odds of staying on budget. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 234–246. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_19
7. Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 125–137. Springer, Heidelberg (2014). doi:10.1007/978-3-319-11439-2_10
8. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. STTT **13**(1), 3–19 (2011)
9. Hartmanns, A., Hermanns, H.: A Modest approach to checking probabilistic timed automata. In: QEST, pp. 187–196. IEEE Computer Society (2009)

10. Hartmanns, A., Hermanns, H.: The Modest Toolset: an integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_51
11. Hartmanns, A., Hermanns, H.: Explicit model checking of very large MDP using partitioning and secondary storage. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 131–147. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24953-7_10
12. Hashemi, V., Hermanns, H., Song, L.: Reward-bounded reachability probability for uncertain weighted MDPs. In: Jobstmann, B., Leino, K.R.M. (eds.) VMCAI 2016. LNCS, vol. 9583, pp. 351–371. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49122-5_17
13. Hatefi, H., Braitling, B., Wimmer, R., Fioriti, L.M.F., Hermanns, H., Becker, B.: Cost vs. time in stochastic games and Markov automata. In: Li, X., Liu, Z., Yi, W. (eds.) SETTA 2015. LNCS, vol. 9409, pp. 19–34. Springer, Heidelberg (2015). doi:10.1007/978-3-319-25942-0_2
14. Klein, J., Baier, C., Chrszon, P., Daum, M., Dubslaff, C., Klüppelholz, S., Märcker, S., Müller, D.: Advances in symbolic probabilistic model checking with PRISM. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 349–366. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_20
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
16. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: QEST, pp. 203–204. IEEE Computer Society (2012)
17. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. FMSD **29**(1), 33–78 (2006)
18. Randour, M., Raskin, J.-F., Sankur, O.: Percentile queries in multi-dimensional Markov decision processes. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 123–139. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21690-4_8
19. Ummels, M., Baier, C.: Computing quantiles in Markov reward models.break In: Pfenning, F. (ed.) FoSSaCS 2013. LNCS, vol. 7794, pp. 353–368. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37075-5_23