Statistical Approximation of Optimal Schedulers for Probabilistic Timed Automata

Pedro R. d'Argenio, Arnd Hartmanns, Axel Legay, Sean Sedwards

Statistical Approximation of Optimal Schedulers for Probabilistic Timed Automata

Pedro R. D'Argenio[1], Arnd Hartmanns[2], Axel Legay[3], and Sean Sedwards[3]

[1] Universidad Nacional de Córdoba, Córdoba, Argentina
[2] University of Twente, Enschede, The Netherlands
[3] INRIA Rennes – Bretagne Atlantique, Rennes, France

Abstract. The verification of probabilistic timed automata involves finding schedulers that optimise their nondeterministic choices with respect to the probability of a property. In practice, approaches based on model checking fail due to state-space explosion, while simulation-based techniques like statistical model checking are not applicable due to the nondeterminism. We present a new lightweight on-the-fly algorithm to find near-optimal schedulers for probabilistic timed automata. We make use of the classical region and zone abstractions from timed automata model checking, coupled with a recently developed smart sampling technique for statistical verification of Markov decision processes. Our algorithm provides estimates for both maximum and minimum probabilities. We compare our new approach with alternative techniques, first using tractable examples from the literature, then motivate its scalability using case studies that are intractable to numerical model checking and challenging for existing statistical techniques.

1 Introduction

Probabilistic timed automata (PTA) [17] are a popular modelling formalism for the analysis of real-time systems. As a generalisation of timed automata (TA) [1], they support (discrete) nondeterministic choices as well as (continuous) nondeterministic timing with hard bounds. As a generalisation of Markov decision processes (MDP), they additionally allow (discrete) probabilistic choices. A PTA model can thus combine hard real-time aspects (using fixed or nondeterministic time bounds) with soft real-time features (using probabilistically chosen delays). PTA also permit abstraction, introducing nondeterminism to reduce the model's size, and allow choices between enabled events to be specified as probabilistic if information on the frequency of their occurrence is available, or as nondeterministic otherwise. Examples of verification questions that can be answered with PTA include "what is the worst-case probability of the modelled process meeting its deadline?", "can it terminate with probability greater than p?", and "is the probability to spend more than 2 s in an unsafe state greater than zero?"

All PTA verification questions include a quantification over schedulers, i.e. over resolutions of the nondeterministic decisions: the "worst-case probability" is the lowest probability achievable for any scheduler; when we ask whether

something "can happen with probability greater than p", we need to find at least one scheduler that makes the probability greater than p. The key challenge in PTA verification thus lies in finding *optimal schedulers*, i.e. those that maximise or minimise the probability for the question of interest. If all time constraints in a PTA rely on integer bounds, optimal schedulers can be found by analysing an MDP abstraction of the PTA's semantics using probabilistic model checking. Whether using regions [17], digital clocks [16] or zones [17, 18] for the abstraction, this approach inevitably fails for large models due to state-space explosion. While the number of extra states needed to capture information about time can be small when using zones, realistic PTA models use compact model descriptions with a parallel composition operator and discrete state variables, both of which already cause the underlying state spaces to be intractably large in practice.

For the analysis of purely stochastic systems, such as Markov chains, an alternative to traditional model checking with exhaustive state-space exploration is *statistical model checking* (SMC, [10, 20]): a number of *simulation* runs is performed on the model, generating traces that can be used to statistically *estimate* the probability of a given path formula with some level of confidence. By definition, however, nondeterministic decisions cannot be simulated, so SMC cannot be applied directly to models like MDP or PTA. For the former, some SMC-like approaches have recently been developed. They either work by iteratively optimising the decisions of an explicitly-stored scheduler [4, 9], or by sampling from the scheduler space and iteratively improving a *set* of candidate near-optimal schedulers [5]. The former are *heavyweight* techniques because the size of the description of the (memoryless) scheduler is significant, and in the worst case is the size of the state space. The latter is a *lightweight* approach that uses $\mathcal{O}(1)$ memory to represent each (history-dependent) scheduler.

Uppaal-smc [7] implements a stochastic model of timed automata that can be used to perform SMC on PTA. By instantiating invariants as either uniform or exponential distributions over time, it can estimate the expected probability of a PTA property under a specific stochastic scheduler. Making use of the same model, Uppaal Stratego [6] handles a more general model than PTA. It combines a scheduler *synthesis* phase with a subsequent SMC analysis of the model under this scheduler, but is limited by an explicit representation of schedulers.

In this paper we develop a lightweight technique to approximate optimal schedulers for PTA, based on the lightweight approach for MDP of [5]. While PTA can be abstracted to MDP, allowing the approach of [5] to be applied directly, the need to explicitly simulate many small delay steps has a catastrophic affect on performance. In addition, the region and digital clocks abstractions are often unnecessarily fine grained, resulting in an explosion of possible schedulers and potentially making near-optimal schedulers very rare. As simulation is inherently a forwards exploration technique, of the zone-based approaches only the one of [17] would be applicable. Unfortunately, it admits unrealistically powerful schedulers, and thus delivers *upper bounds* on maximum and *lower bounds* on minimum probabilities. This is fundamentally incompatible with sampling sched-

ulers as in [5], which delivers *lower* bounds on maximum and *upper* bounds on minimum probabilities. Our technique nevertheless uses zones on-the-fly to perform a forwards exploration, but selects a single target region after each discrete jump. This avoids the problems of [17], while the conceptual blow-up of state space does not affect our technique because it simulates a trace by storing only one state in memory at a time.

We report on a prototype implementation of our new technique. We test it with standard models from the literature that are tractable for probabilistic model checking, in order to compare the near-optimal schedulers that we find with the optimal schedulers computed by PRISM [15]. We then show that our technique works well for some intractably large examples. We also compare our results with those produced by the single-scheduler approach of UPPAAL-SMC.

2 Preliminaries

$\mathbb{N}$ is $\{0, 1, \dots\}$, the set of natural numbers. $\mathbb{R}^+$ is $(0, \infty)$, the set of positive real numbers. $\mathbb{R}_0^+$ is $[0, \infty)$, the set of nonnegative real numbers. $a.b$ denotes the concatenation of two sequences a and b or of two objects interpreted as bitstrings.

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

We write $\mathcal{D}(s)$ for the *Dirac distribution* for s, defined by $\mathcal{D}(s)(s) = 1$.

Definition 2. *A uniform* pseudo-random number generator *(PRNG) is a an object $\mathcal{U}$ that, once initialised with a* seed $i \in \mathbb{N}$ *(denoted $\mathcal{U} := \mathrm{PRNG}(i)$), can be* iterated *(denoted $\mathcal{U}()$) to produce a new value that is pseudo-uniformly distributed in $[0, 1)$ and pseudo-statistically independent of previous iterates. $\mathcal{U}$ is* deterministic *if, for a given seed, the sequence of iterates is always the same.*

We only consider deterministic PRNG. Determinism is standard in commonly used PRNG [12]. We denote by $\mathcal{U}(\mu)$ the pseudo-random selection of a value from $\mathrm{support}(\mu)$ according to a value sampled from $\mathcal{U}$ and the probabilities in μ. In what follows, when we write "random" w.r.t. a choice made by a PRNG we implicitly mean "pseudo-random" unless qualified otherwise.

2.1 Markov Decision Processes

Markov decision processes combine nondeterminism and probabilistic choices: to move from one state to another, first a transition is chosen nondeterministically; each transition then leads into a probability distribution over successor states.

Definition 3. *A* Markov decision process *(MDP) is a 4-tuple $\langle S, A, T, s_{init} \rangle$ where S is a countable set of states, A is a countable set of transition labels, $T \in S \to 2^{A \times \mathrm{Dist}(S)}$ is the transition function with $T(s)$ countable for all $s \in S$, and $s_{init} \in S$ is the initial state.*

A triple $\langle s, a, \mu \rangle$ such that $\langle a, \mu \rangle \in T(s)$ is called a *transition*. We also write $s \xrightarrow{a} \mu$ for the transition $\langle s, a, \mu \rangle$.

2.2 Probabilistic Timed Automata

Probabilistic timed automata deal with time through *clocks*: variables whose domain is $\mathbb{R}_0^+$ that advance synchronously with rate 1 over time. Given a set of clocks $\mathcal{C}$, the valuation $\mathbf{0} \in \mathit{Val}$ (with $\mathit{Val} = \mathcal{C} \to \mathbb{R}_0^+$) assigns zero to every clock $c \in \mathcal{C}$. For $v \in \mathit{Val}$ and $t \in \mathbb{R}_0^+$, we denote by $v + t$ the valuation where all clock variables have been incremented by t, and by $v[X]$ the one where all clocks in $X \subseteq \mathcal{C}$ have been reset to zero. *Clock constraints* are expressions of the form

$$\mathcal{CC} ::= \mathit{true} \mid \mathit{false} \mid \mathcal{CC} \wedge \mathcal{CC} \mid c \sim n \mid c_1 - c_2 \sim n$$

where $\sim \in \{>, \geq, <, \leq\}$, $c, c_1, c_2 \in \mathcal{C}$ and $n \in \mathbb{N}$. The form $c_1 - c_2 \sim n$ is called a *diagonal*, and a clock constraint without diagonals is *diagonal-free*. If all comparison operators used in a clock constraint are in $\{\geq, \leq\}$, it is *closed*. $[\![e]\!]$ for $e \in \mathcal{CC}$ is the set of valuations $v \in \mathit{Val}$ such that e evaluated in v is *true*.

Definition 4. *A* probabilistic timed automaton *(PTA for short) is a 6-tuple $\langle \mathit{Loc}, \mathcal{C}, A, E, l_{init}, \mathit{Inv} \rangle$ where Loc is a countable set of locations, $\mathcal{C}$ is a finite set of clocks, A is a countable set of edge labels, $E \in \mathit{Loc} \to 2^{\mathcal{CC} \times A \times \mathrm{Dist}(2^{\mathcal{C}} \times \mathit{Loc})}$ is the edge function with $E(l)$ finite for all $l \in \mathit{Loc}$, $l_{init} \in \mathit{Loc}$ is the initial location, and $\mathit{Inv} \in \mathit{Loc} \to \mathcal{CC}$ is the invariant function.*

A 4-tuple $\langle l, g, a, \mu \rangle$ such that $\langle g, a, \mu \rangle \in E(l)$ is called an *edge*. It consists of the guard g, the label a and the probability distribution μ over sets of clocks to reset and target locations. We also write $l \xrightarrow{g,a} \mu$ for the edge $\langle l, g, a, \mu \rangle$. Using PTA to directly build models of complex systems is cumbersome. Instead, higher-level formalisms such as PRISM's [15] guarded command language are used. Aside from a parallel composition operator, they add to PTA variables that take values from finite domains. In a *PTA with variables* (VPTA), guards and invariants can include Boolean expressions over the variables, the set of clock resets is extended by assignments of new values to variables, and the probabilities of target locations can be computed based on the current valuation. The semantics of a VPTA M is a PTA whose locations are pairs $\langle l, v \rangle$ of a location of M and a valuation v for the variables. VPTA can compactly describe very large PTA.

The semantics of a PTA is as follows: When in location l, time can pass as long as the invariant $\mathit{Inv}(l)$ remains satisfied. An edge $e \in E(l)$ as above can be taken if its guard is satisfied at the current point in time. When this happens, a target $\langle X, l' \rangle$ is chosen according to the probability distribution μ, the clocks in X are reset, and we move to the successor location l'.

Example 1. An example PTA is shown in Fig. 1. It has two clocks x and y. The edge labelled `a` from location l_0 has guard $x \leq 1$ and leads into the probability distribution $\{ \langle \varnothing, l_1 \rangle \mapsto \frac{1}{2}, \langle \{ y \}, l_2 \rangle \mapsto \frac{1}{2} \}$. The invariant of l_2 is $y \leq 0$, thus no time can pass while the system is in that location.

Definition 5. *A* timed probabilistic transition system *(TPTS for short) is a 4-tuple $\langle S, A, T, s_{init} \rangle$ where S is a set of states, $A = \mathbb{R}^+ \uplus A'$ is a set of transition labels partitioned into delays in $\mathbb{R}^+$ and jump labels in A', $T \in S \to 2^{A \times \mathrm{Dist}(S)}$ is*

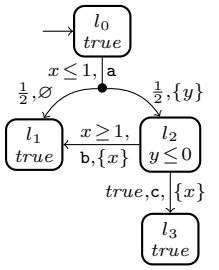

Fig. 1. Example PTA

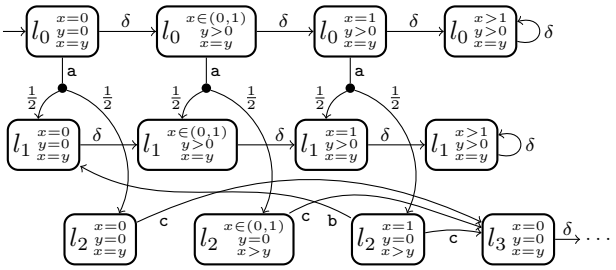

Fig. 2. The region MDP of the example PTA

the transition function, and $s_{init} \in S$ is the initial state. The delay-labelled transitions must lead into Dirac distributions and be time-deterministic and time-additive.

TPTS can be seen as uncountably infinite-state, uncountably-branching Markov decision processes. We use them to formally define the semantics of PTA:

Definition 6. *The semantics of a well-formed PTA $M = \langle Loc, \mathcal{C}, A, E, l_{init}, Inv\rangle$ is the TPTS $[\![M]\!] = \langle Loc \times Val, \mathbb{R}^+ \uplus A, T_M, \langle l_{init}, \mathbf{0}\rangle\rangle$ where T_M is the smallest function such that the following two inference rules are satisfied:*

$$\frac{l \xrightarrow{g,a}_E \mu \quad v \in [\![g]\!]}{\langle l, v\rangle \xrightarrow{a}_{T_M} \mu^v_M} \text{ (jump)} \qquad \frac{t \in \mathbb{R}^+ \quad \forall\, t' \le t\colon (v + t') \in [\![Inv(l)]\!]}{\langle l, v\rangle \xrightarrow{t}_{T_M} \mathcal{D}(\langle l, v + t\rangle)} \text{ (delay)}$$

where $\mu^v_M(\langle l', v'\rangle) = \mu(\langle X, l'\rangle)$ if $v' = v[X]$ and $\mu^v_M(s) = 0$ otherwise.

We refer to the transitions resulting from the respective inference rules as *jumps* and *delays*. It is undesirable to be able to jump into a location l' such that $Inv(l')$ is immediately violated. If this is not possible in a PTA, we say that it is *well-formed*. Non-well-formed PTA need to be rejected as modelling errors.

2.3 Probabilistic Timed Reachability

A behaviour of a TPTS $M = \langle S, A \uplus \mathbb{R}^+, T, s_{init}\rangle$ is a *path* $\pi \in Paths(M)$: an infinite sequence $\langle s_0, a_0\rangle\langle s_1, a_1\rangle \cdots \in (S \times A \uplus \mathbb{R}^+)^\omega$ of states and actions or delays. The system starts in the initial state $s_0 = s_{init}$. Assuming that the current state is s_i, the choice of the next transition $s_i \xrightarrow{a_i} \mu_i$ is nondeterministic. Such a choice is made by a scheduler:

Definition 7. *For a TPTS as above, a (memoryless deterministic)* scheduler *is a function $\mathfrak{S} \in S \to A \times \mathrm{Dist}(S)$ s.t. $\mathfrak{S}(s) \in T(s)$ for all $s \in S$.*

Once the scheduler has chosen $\mathfrak{S}(s_i) = s_i \xrightarrow{a_i} \mu_i$, the next state s_{i+1} is selected randomly according to μ_i. Using the usual cylinder set construction [17], every scheduler $\mathfrak{S}$ defines a probability measure $\mathbb{P}_{\mathfrak{S}}$ over the set of all paths. Let $\delta(\pi)$ be the sum of all transition labels in $\mathbb{R}^+$ on path π. Following the standard approach,

we restrict to time-divergent schedulers, i.e. we only consider schedulers $\mathfrak{S}$ where $\mathbb{P}_{\mathfrak{S}}(\{\, \pi \in \mathit{Paths}(M) \mid \delta(\pi) = \infty \,\}) = 1$.

Given a PTA M, we are interested in the verification of *probabilistic timed reachability* properties, which are queries of the form "starting from the initial state, what is the maximum/minimum probability of eventually/within time t reaching a location $l \in L$ when c holds" (quantitative form) resp. "is this probability less/greater than or equal to p" (qualitative form) for $L \subseteq \mathit{Loc}$, $c \in \mathcal{C}$, and $t \in \mathbb{R}_0^+$. The time-bounded questions can be turned into unbounded ones by adding a new clock c_t to M that is never reset, and using the clock constraint $c_t \leq t \wedge c$ in place of c. It thus suffices to consider the time-unbounded questions, which is why we do *not* need history-dependent schedulers. L and c together characterise a set F of states of $[\![M]\!]$. We are thus interested in the extremal probabilities $\sup_{\mathfrak{S}} \mathbb{P}_{\mathfrak{S}}(\Pi_F)$ (the maximum probability) and $\inf_{\mathfrak{S}} \mathbb{P}_{\mathfrak{S}}(\Pi_F)$ (the minimum probability) where Π_F is the set of paths containing a state in F. If we can compute them, we can also compute e.g. the probabilities of linear-time (safety) path properties by running M in parallel with a finite state machine observer and using its final states for L. If schedulers exist that realise the sup (inf) above (which is always the case for MDP), we call them *optimal* or *maximising* (*minimising*) schedulers.

Example 2. Two properties of interest on the PTA of Ex. 1 are (1) the maximum probability to reach l_1 with $x < 1$, which is $\frac{1}{2}$, and (2) the minimum probability to reach a location in $\{\, l_1, l_2 \,\}$, which is 0 (since we can stay in l_0 forever).

2.4 Digital Clocks, Regions and Zones

To compute reachability probabilities for a (finite) PTA, we cannot construct its semantics since this is an uncountably infinite object. However, three countable (finite) abstractions have been developed to be used for model checking:

Digital clocks. We can replace the clock variables by bounded integers and add self-loop edges to increment them synchronously as long as the location invariant is satisfied. If all clock constraints are closed and diagonal-free, this turns a PTA into a (finite) MDP with variables, while preserving reachability probabilities [16]. The number of states of the underlying *digital clocks MDP* is exponential in the number of clocks and the maximum constants they are compared to. In practice, however, it is often small enough to be amenable to model checking.

Regions. The *region MDP* is a (finite) abstraction that preserves reachability probabilities for any (finite) PTA [17]. Like the region graph for TA, it is the quotient of the TPTS semantics of a PTA under the equivalence relation that groups the states that cannot be distinguished by any clock constraint (up to the largest value any clock is compared with in the PTA). Its states are thus pairs $\langle l, [v]_{\mathcal{C}} \rangle$ of a location l and a *clock equivalence class* $[v]_{\mathcal{C}}$. In this paper we write *region* to refer to a clock equivalence class, i.e. a set of valuations. The region graph construction suffers from the same blow-up as the digital clocks approach, but region graphs are almost always too large to be useful for model checking.

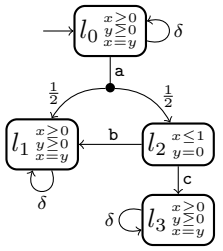

Fig. 3. The zone MDP

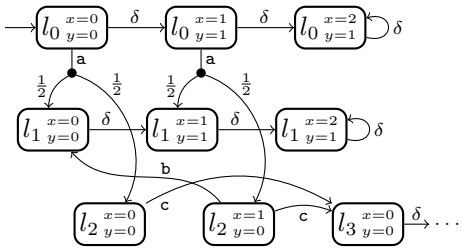

Fig. 4. The digital clocks MDP of the example PTA

Zones. A PTA's behaviour often is the same for many regions. This observation has already led to the development of zone-based approaches for TA. A *zone* is a set of valuations characterised by a clock constraint, or equivalently, it is a convex union of regions. Using zones we can construct significantly smaller MDP abstractions of PTA than with individual regions. However, if the standard TA forwards reachability procedure is used for PTA, the resulting *zone MDP* admits schedulers that are too powerful, and thus the reachability probabilities computed in this abstraction are upper/lower bounds on the PTA's respective maximum/minimum probabilities only [17]. To obtain compact zone graphs that do not exhibit this problem, a backwards analysis is needed [18].

In the zone-based algorithms that we present later in this paper, we will write $z_1 \sqcup z_2$ for the convex union of the two zones z_1 and z_2, i.e. the minimal zone that contains both z_1 and z_2, $z{\uparrow}$ for the delay zone $\{\, v+t \mid v \in z \wedge t \in \mathbb{R}_0^+ \,\}$, and $\mathit{Reg}(z)$ for the set of regions in zone z.

Example 3. For the PTA of Ex. 1, we show the region MDP in Fig. 2, the digital clocks MDP in Fig. 4, and the zone MDP in Fig. 3. Transitions representing delays are labelled δ. If we compute the probabilities of Ex. 2 on these abstractions, we find that we obtain the correct values with regions and digital clocks. For the former, this was not guaranteed since property (1) contains a non-closed clock constraint. On the zone MDP, however, we obtain probability 1 for property (1). This is because in the original PTA the delay chosen in l_0 determines whether l_1 can be reached from l_2 after the probabilistic jump. In the zone MDP, however, this choice is effectively moved to l_2, giving schedulers extra power.

3 Lightweight Verification of MDP

We briefly recall the lightweight verification technique for MDP of [5] that underpins our new approach for PTA. As a statistical model checking (SMC) technique, it is based on generating a number of simulation runs through the MDP and then statistically estimating the (reachability) probability of interest. When simulating a fully stochastic model, e.g. a Markov chain, the individual probabilistic decisions are resolved randomly, and thus *a run is sampled* faithfully from the probability measure over all runs of the system. However, in MDP,

Input: MDP $\langle S, A, T, s_{init} \rangle$, path property ϕ, scheduler id $\sigma \in \mathbb{Z}$
Output: Sampled path π

1 $s := s_{init}$, $\pi := s_{init}$
2 **while** $\phi(\pi) =$ undecided **do**
3 $\quad \mathcal{U}_{\text{nd}} := \text{PRNG}(\mathcal{H}(\sigma.s))$ *// use hash of σ and s as seed for $\mathcal{U}_{\text{nd}}$*
4 $\quad$ **if** $T(s) = \varnothing$ **then break** *// end of run due to deadlock*
5 $\quad \langle a, \mu \rangle := \lceil \mathcal{U}_{\text{nd}}() \cdot |T(s)| \rceil$-th element of $T(s)$ *// use $\mathcal{U}_{\text{nd}}$ to select transition*
6 $\quad s' := \mathcal{U}_{\text{pr}}(\mu)$ *// use $\mathcal{U}_{\text{pr}}$ to select target according to μ*
7 $\quad \pi := \pi.s'$, $s := s'$ *// append the new state s to π*
8 **return** π

Algorithm 1: Path generation for an MDP and a fixed scheduler

the nondeterministic choices need to be resolved, too. The lightweight approach of [5] addresses this problem by also *sampling individual schedulers* from the overall space of all schedulers. An adapted SMC analysis is performed for each scheduler, and a set of candidate near-optimal schedulers is iteratively refined by keeping those that deliver the highest (lowest) probabilities. An iterative "smart sampling" technique [5] maximises the probability of finding an optimal scheduler with a finite simulation budget.

To avoid storing schedulers as explicit mappings, our lightweight approach constructs them on-the-fly using pseudo-random number generators. It uses two independent PRNG $\mathcal{U}_{\text{pr}}$ and $\mathcal{U}_{\text{nd}}$ to resolve the probabilistic and nondeterministic choices, respectively. A single integer $\sigma \in \mathbb{Z}$ of b_σ bits identifies and fully specifies a scheduler. At its core, the adapted SMC analysis uses Algorithm 1 to perform simulation runs. We assume that the MDP is given in some compact representation, e.g. as a network of MDP with variables, where a state of the concrete MDP can be seen as a valuation for the system variables v_i with the value of each v_i being represented by a number of bits b_i. A state can thus be represented by the concatenation of the bits of the system variables. In line 3, the scheduler identifier σ is concatenated to the bits representing the current state s (denoted $\sigma.s$). $\mathcal{U}_{\text{nd}}$ is then initialised using the hash code $h = \mathcal{H}(\sigma.s)$. $\mathcal{H}$ maps $\sigma.s$ to a seed that is deterministically dependent on the state and the scheduler. $\mathcal{U}_{\text{nd}}$ maps the seed to a value that is uniformly distributed but also deterministically dependent on the trace and the scheduler (line 5). For the fixed σ, this use of $\mathcal{U}_{\text{nd}}$ thus results in exactly the behaviour of a memoryless scheduler.

In an outer loop, the lightweight approach performs multiple SMC analyses with different sampled values for σ to estimate the property's probability for different schedulers, keeping track of the highest (lowest) overall estimate, which results from the scheduler closest to optimality that has been sampled so far. In a typical implementation on current hardware, a hash function and PRNG may span $\sim 10^{19}$ schedulers. This is usually orders of magnitude more than the number of schedulers sampled. To avoid a cumulative error when choosing a single probability estimate from a number of alternatives, [5] defines a Chernoff bound for multiple estimates. Note that this ensures that the statistical

confidence w.r.t. individual estimates is well-defined, but does not provide confidence w.r.t. the optimality of the overall estimate: For maximum (minimum) reachability probabilities, the overall estimate is a lower (upper) bound on the actual probability.

4 Lightweight Verification of PTA

To adapt the lightweight approach described in the previous section to work for PTA, we could use it as-is on the digital clocks or region abstraction. While this works, we find that such a naive adaptation is inefficient: Letting time pass within a location corresponds to a sequence of states and δ-transitions within the digital clocks or region MDP. This makes simulation dependent on the absolute value of delays, reducing performance in models with longer delays because many more transitions need to be simulated. It also means that there are exponentially more schedulers to consider and that they are more likely to pick short delays. These phenomena have the potential to make near-optimal schedulers infeasibly rare.

Example 4. Consider using the region MDP of Fig. 2 with Algorithm 1, and let s be the initial state. $\mathcal{U}_{\text{nd}}$ is used to select one of the outgoing transitions in line 5. Given a fixed σ, i.e. scheduler, this is a deterministic selection. σ is fixed *within* one SMC analysis, but uniformly randomly chosen for each analysis. The probability of using a scheduler that chooses a given transition from s is thus $\frac{1}{2}$, so the probability to pick one that delays up to the top-rightmost state is only $\frac{1}{8}$. However, for property (2) of Ex. 2, these schedulers are the only ones that lead to the actual minimum probability of 0, while all others (which have probability mass $\frac{7}{8}$) lead to probability 1, a correct but useless upper bound.

The example shows that we would prefer schedulers for every delay to be equally likely. This can be achieved with zones: every state of the zone MDP has an outgoing jump for each edge that ever becomes enabled over time in the corresponding location (cf. Fig. 3). However, as we have already observed, a zone MDP's optimal schedulers may lead to true upper (lower) bounds on maximum (minimum) probabilities. If we were to use the lightweight approach on the zone MDP, we would get lower bounds on upper bounds on maximum probabilities (and vice versa for minimum), i.e. *some* value whose relation to the actual probability is unknown, which is arguably of little use for formal verification.

To avoid this problem, after deciding to perform a particular jump, we also select the concrete region from which the jump takes place, using $\mathcal{U}_{\text{nd}}$ again since this is resolving nondeterminism. Doing so *does* conceivably blow up the state space compared to the zone MDP, but this is irrelevant because we only ever construct the state space local to a simulation. Note that it *does not* introduce extra transitions that could lead to the problems we encountered with regions or digital clocks. Furthermore, we distinguish between being allowed to delay into a deadlock situation vs. having an enabled edge at the upper bound of a location's invariant. In effect, we thus simulate the region MDP, but exploit zones to do so in a way suitable for the lightweight approach. Algorithm 2 shows the concrete

Input: PTA $M = \langle Loc, \mathcal{C}, A, E, l_{init}, Inv \rangle$, path property ϕ, scheduler id $\sigma \in \mathbb{Z}$
Output: Simulation trace ω

1 $l := l_{init}$; $z := \{ \mathbf{0} \}$; $\omega := \langle l_{init}, z \rangle$
2 **while** $\phi(\omega)$ = undecided **do**
3 **if** $z \cap [\![Inv(l)]\!]$ is empty **then** raise error *// check that M is well-formed*
4 $J := \varnothing$; $z_J := z$; $z' := z{\uparrow} \cap [\![Inv(l)]\!]$ *// let time pass as the invariant allows*
5 **foreach** $\langle g, a, \mu \rangle \in E(l)$ **where** $z' \cap [\![g]\!] \neq \varnothing$ **do**
6 $J := J \cup \{ \langle \mu, z' \cap [\![g]\!] \rangle \}$ *// store edge distr., zone where it is enabled*
7 $z_J := z_J \sqcup (z' \cap [\![g]\!])$ *// z_J always eventually has an enabled edge*
8 **if** $J = \varnothing$ **then** $\omega := \omega.\langle l, z' \rangle$; **break** *// can only delay into deadlock*
9 **if** $z' \neq z_J$ **then** $J := J \cup \{ \mathcal{D}(\langle \varnothing, l \rangle), z' \setminus z_J \}$ *// possible delay into deadlock*
10 $\mathcal{U}_{\text{nd}} := \text{PRNG}(\mathcal{H}(\sigma.l.z'))$ *// use hash of σ, l and z' as seed for $\mathcal{U}_{\text{nd}}$*
11 $\langle \mu, z_\mu \rangle := \lceil \mathcal{U}_{\text{nd}}() \cdot |J| \rceil$-th element of J *// use $\mathcal{U}_{\text{nd}}$ to select one of the edges*
12 $\langle X, l' \rangle := \mathcal{U}_{\text{pr}}(\mu)$ *// use $\mathcal{U}_{\text{pr}}$ to select resets, target according to μ*
13 $r := \lceil \mathcal{U}_{\text{nd}}() \cdot |Reg(z_\mu)| \rceil$-th element of $Reg(z_\mu)$ *// use $\mathcal{U}_{\text{nd}}$ to select region in z_μ*
14 $r' := r[X]$ *// reset the clocks in X*
15 $\omega := \omega.\langle l, z \sqcup r \rangle.\langle l', r' \rangle$, $l := l'$, $z := r'$ *// append delay and jump to trace ω*
16 **return** ω

Algorithm 2: Trace generation for a PTA and a fixed scheduler using zones

zone-based simulation that we use for PTA in place of Algorithm 1 for MDP. Otherwise, the lightweight approach remains as described previously.

Algorithm 2 computes the set of edges that become enabled while time passes in the current location, subject to the invariant, in lines 4 to 7. Additionally, the maximum delay after which there is still an enabled edge is computed as z_J. This is necessary to allow the scheduler to choose delaying into a deadlock, like delaying beyond $x = 1$ in l_1 in Fig. 1, when this is possible (line 9, implemented by a self loop into a situation where line 8 will apply). $\mathcal{U}_{\text{nd}}$ is then initialised for the current scheduler identifier σ together with the current location l and zone z', and used in line 11 to select one of the edges. Subsequently, a concrete region needs to be chosen out of the range of delays allowed by that edge's guard, which is done by $\mathcal{U}_{\text{nd}}$ in line 13. The simulation trace is updated in lines 8 and 15, taking care to include every relevant intermediate state since the property at hand may refer to clocks as described in Sect. 2.3. In our implementation, the trace is never stored explicitly, instead the evaluation of ϕ occurs on-the-fly and incrementally.

In line 13 of Algorithm 2, one region is selected from a zone, using $\mathcal{U}_{\text{nd}}$ to choose a region pseudo-randomly with respect to all possible schedulers (the choice made by an individual scheduler is of course deterministic). Since we do not know which region is optimal, we require the choice to be uniformly random. To maintain efficiency, we also require to select directly from the data structure defining the zone, without enumerating all the regions. To achieve this we have implemented two algorithms, one using rejection sampling, to guarantee uniform coverage, the other using conditional sampling, to maximise efficiency.

Both algorithms assume that the increasing integer and fractional values of clocks, corresponding to the division of regions, are represented by a monotonically increasing set of integers. A region is then uniquely identified by a tuple of indices. Some zones are unbounded above, so the maximum region the algorithms consider is the minimum region that exceeds the maximum clock bound in the model. All choices are thus made from a finite range of indices. Neither algorithm considers clocks that are not constrained by bounds within the model, such as the global clock used by the time bounded properties.

The rejection sampling algorithm works by first selecting a value "uniformly at random" (i.e., using $\mathcal{U}_{\mathrm{nd}}$) from a cube that encloses the range of indices of the zone. This is achieved by selecting a value uniformly at random from the range of indices corresponding to each clock. As each individual ordinate is chosen, the feasibility of the tentative region is checked against the constraints of the zone. If at any point the region is judged to be infeasible, the sampling process is restarted.

Given that the initial zone is non-empty, the rejection sampling algorithm will eventually terminate with a valid selection, however the execution time scales exponentially with the number of clocks. The conditional sampling algorithm avoids this complexity by ensuring that every choice is made from a feasible interval. Using $\mathcal{U}_{\mathrm{nd}}$, the algorithm first chooses a "random" order of clocks. This is necessary because the order in which the clocks are sampled may bias the choices. Then, for each clock in order, an index is chosen uniformly at random with respect to the current range of permissible values. After each choice the data structure representing the zone is updated to reflect the choice, restricting the bounds of the remaining clocks and thus the range of permissible indices.

Note that while our case studies do not motivate the use of the less efficient algorithm (i.e., we found no benefit in terms of optimality), we nevertheless consider efficient uniform region selection a subject of ongoing research.

5 Experiments

We have created a prototype implementation of our new approach for lightweight PTA analysis. It is intended to become part of the Plasma toolset for statistical model checking [11, 3], to take advantage of Plasma's integrated development environment and distribution algorithms [3]. We use the name Plasma Seta (Statistical Estimation of Timed Automata) to identify our results in the figures. The tool is written in Java and uses its own implementation of the standard difference-bound matrix (DBM) data structure to represent invariants, clock constraints, zones and regions in memory. Operations on DBM are used to advance the state of a simulation and check whether the current state satisfies a property. We use a textual modelling language that mirrors the annotated graph-based structure traditionally used to describe timed automata and that facilitates simple conversion between existing graphical and textual formalisms. In contrast to SMC for Markov chain-based models, a state of our simulations represents a set of feasible schedulings (cf. Algorithm 2), whose value with respect

to optimality is dependent on the context of whether a minimum or maximum is sought. It is therefore not sufficient to use the standard Monte Carlo estimator and bounded time linear temporal logic. Accordingly, we have implemented a simple continuous time logic that expresses non-nested time-bounded properties of the syntactic forms $\Box_{\leq t}\,\phi$, $\Diamond_{\leq t}\,\phi$ and $\phi\,\mathsf{U}_{\leq t}\,\phi$.

We have applied our implementation to PTA models from the PRISM benchmark suite [13], in order to evaluate its effectiveness (can it solve examples that are intractable to related tools?), its efficiency (what is the performance compared to related tools?), and its usefulness (what is the quality of the results, i.e. the bounds on extremal probabilities that we obtain, and how do these results compare with those of related tools?). Where possible we compare with (a) PRISM's default game-based engine [14] for traditional exhaustive PTA model checking, with (b) UPPAAL-SMC as a statistical model checker that uses a single stochastic scheduler, and with (c) the original lightweight approach (as described in Sect. 3) on a digital clocks abstraction of the PTA.

All experiments were performed on an Intel Xeon 2.8 GHz system running Ubuntu 15.04 with 8 GB of memory. We used PRISM 4.3 with default settings and UPPAAL 4.1.19 with statistical parameter $\epsilon = 0.01$. UPPAAL's estimates are thus given $\pm \leq 0.01$ with probability ≥ 0.95, w.r.t. the value being estimated. For the original lightweight technique and our new approach, we used smart sampling with a per-iteration simulation budget of 3×10^4 and parameters $\epsilon = \delta = 0.01$. This guarantees that the computed results are $\pm \leq 0.01$ with probability ≥ 0.99, w.r.t. the values being estimated.

Firewire. We first look at the `firewire_abst` PRISM PTA benchmark, which models the IEEE 1394 FireWire root contention protocol in an abstract manner. The property we check is $\Diamond_{\leq t}\mathit{done}$, for deadline $t \in \{0.4, 0.8, 1.2, \ldots, 10\}$ µs and where *done* signifies that both stations have completed their transmissions. The results are shown in Fig. 5. The shaded regions denote a ± 0.01 error interval around the values calculated by PRISM, corresponding to our specified confidence. The circles show the probability estimates for the best maximising/minimising scheduler found by our new approach. The crosses mark the single probability estimated by UPPAAL-SMC.

We see that our approach finds schedulers that are very close to the optimal schedulers found by model checking. For deadlines of 5µs or less, the estimates produced by our approach are within the specified statistical confidence, noting that this confidence is not with respect to optimality. For greater deadlines our minimising estimates are less accurate with respect to the true minima, but are nevertheless clearly discernible from the maximising estimates. By contrast, the estimates produced by UPPAAL-SMC lie arbitrarily between the maxima and minima. Very approximately, UPPAAL-SMC's results denote the performance of the "average" scheduler found by our approach.

CSMA-2 and CSMA-3. We now turn to larger and more challenging models. The first is the `csma` PRISM PTA benchmark, which models the IEEE 802.3 CSMA/CD protocol for shared medium access with two senders (CSMA-2). The second is a new model that adds a third sender (CSMA-3). We also consider a

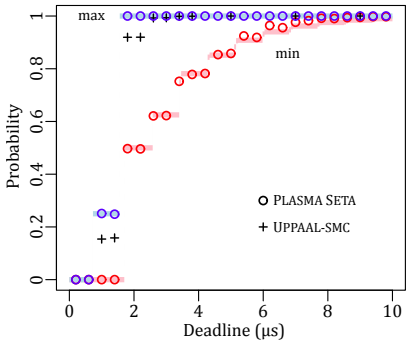

Fig. 5. Firewire results

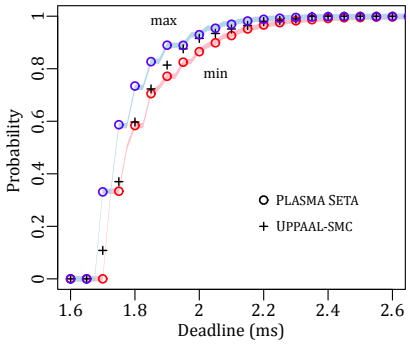

Fig. 6. CSMA-2 results

digital clocks version of CSMA-2. For both, we seek the maximum and minimum probability that all senders successfully finish sending one message within t ms. The experimental results for CSMA-2 are shown in Fig. 6. Once again our approach finds schedulers that very closely approximate the optima calculated by PRISM. In this case all the results are within the specified statistical confidence. As before, the single estimates produced by UPPAAL-SMC lie arbitrarily between minimum and maximum: for deadlines less than ~ 1.9 ms they lie close to the minimum, while above this deadline they lie close to the maximum.

CSMA-2 is tractable for PRISM using values of t from 0 to 3 ms and well beyond. With CSMA-3, however, PRISM runs out of memory with even the lowest interesting deadline. Our approach remains tractable with CSMA-3 and arbitrary deadlines, albeit with increased running time. We report the following results for CSMA-3, although we currently have no means to independently verify their accuracy:

deadline (ms)	2	3	4	6	8
max. prob.	0.0	0.544	0.640	0.650	0.671
min. prob.	0.0	0.192	0.246	0.249	0.244

Performance. Our results have been produced using a single execution thread, but a significant benefit of statistical approaches is that they may be easily parallelised to give near-linear speedup with additional threads. For the specified confidence and interesting deadlines, we need about the same time to generate a result as PRISM (between 400 and 500 s) for CSMA-2. With parallelisation, we expect one or two orders of magnitude improvement with our statistical approach.

To further motivate our new approach, we consider a digital clocks version of the CSMA-2 model, which is possible because its non-closed clock constraints can be worked around for the property we consider. We can thus use the original lightweight technique for MDP. The use of discrete time, however, incurs the penalty of having to explicitly consider every delay step in the time bound of the property. Our results with CSMA-2 suggest that this penalty leads to at least an order of magnitude increase in computational time.

We have also profiled our implementation using the CSMA-2 model. We find that around 56 % of the total runtime is spent on DBM operations excluding region selection (as described at the end of Sect. 4), which account for a further 10 %. Around 32 % is used by the simulation loop to enumerate choices and compute synchronisations in the model. Profiling with the more challenging CSMA-3 model reveals that DBM operations continue to account for around 55 % of the execution time, while the amount of time dedicated to region selection is approximately doubled to 22 %. Synchronising and selecting actions accounts for a further 22 %. It is clear that optimising all DBM operations, including those for region selection, will be a profitable direction of future work to improve performance.

6 Conclusion

We have provided the first algorithm and implementation for statistical approximation of optimal schedulers for PTA. This enables statistical model checking of PTA with proper consideration of nondeterminism. Our algorithm is built on top of the smart sampling technique [5] using zones first, for selecting an enabled edge, and then regions to define the particular (abstract) moment in which the scheduler determines the execution of a transition. This two-step technique minimises possible bias towards selecting fast or slow schedulers. Using zones improves performance compared to a digital clocks or region-based techniques.

Our experiments have validated our technique, reporting near-optimal estimates that are close to the true optima calculated via probabilistic model checking with Prism. As a simulation-based tool, our technique can report results when probabilistic model checking runs out of memory, as shown for the CSMA-3 example. We have also compared it with Uppaal-smc, the only other tool that can simulate PTA. Uppaal-smc assumes that the sojourn time at a location is either uniformly distributed, if the invariant limits it, or exponentially distributed otherwise. It thus uses a single fully stochastic scheduler and reports only a single estimated value. As we have seen in the experiments, this value can occur at any point within the interval bounded by the actual minimum and maximum probabilities. We cannot in general quantify how close our approximations are to the true optima, however they must lie within the interval spanned by the true optimal values or else lie outside with quantified statistical confidence. Thus the reported values can always be used to reject the model when the desired maximum (minimum) probability is smaller (greater) than the estimated one.

Future work. This work opens new directions of possible research. We could consider priced PTA to estimate e.g. energy consumption or financial costs. Ideas in this direction have already been reported in [19] for non-timed models. If extended to stochastic timed automata (STA) [2], our approach could be useful in combination with the STA model checking technique of [8], which delivers upper/lower bounds, while we obtain lower/upper bounds on maximum/minimum

probabilities. We would also like to reduce the sample space of schedulers to increase the likelihood of choosing near-optimal schedulers in our algorithm.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
2. Bohnenkamp, H., D'Argenio, P., Hermanns, H., Katoen, J.P.: MoDeST: A compositional modeling formalism for real-time and stochastic systems. IEEE Trans. Softw. Eng. 32(10), 812–830 (2006)
3. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A flexible, distributable statistical model checking library. In: QEST, LNCS, vol. 8054, pp. 160–164. Springer (2013)
4. Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: ATVA. LNCS, vol. 8837, pp. 98–114. Springer (2014)
5. D'Argenio, P., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. STTT 17(4), 469–484 (2015)
6. David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Uppaal stratego. In: TACAS. LNCS, vol. 9035, pp. 206–211. Springer (2015)
7. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Uppaal SMC tutorial. STTT 17(4), 397–415 (2015)
8. Hahn, E.M., Hartmanns, A., Hermanns, H.: Reachability and reward checking for stochastic timed automata. ECEASST 70 (2014)
9. Henriques, D., Martins, J.G., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: Quantitative Evaluation of Systems, 2012 Ninth International Conference on. pp. 84–93. IEEE (2012)
10. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. LNCS, vol. 2937, pp. 73–84. Springer (2004)
11. Jégourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking – PLASMA. In: TACAS. LNCS, vol. 7214, pp. 498–503. Springer (2012)
12. Knuth, D.E.: The Art of Computer Programming: Sorting and Searching, vol. 3. Addison-Wesley, 2nd edn. (1998)
13. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proc. 9th International Conference on Quantitative Evaluation of SysTems (QEST'12). pp. 203–204. IEEE CS Press (September 2012)
14. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: FORMATS. LNCS, vol. 5813, pp. 212–227. Springer (2009)
15. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011)
16. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. FMSD 29(1), 33–78 (2006)
17. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. 282(1), 101–150 (2002)
18. Kwiatkowska, M.Z., Norman, G., Sproston, J., Wang, F.: Symbolic model checking for probabilistic timed automata. Inf. Comput. 205(7), 1027–1077 (2007)

19. Legay, A., Sedwards, S., Traonouez, L.: Estimating rewards & rare events in non-deterministic systems. ECEASST 72 (2015)
20. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. LNCS, vol. 2404, pp. 223–235. Springer (2002)