Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

EFFICIENT SIMULATION-BASED VERIFICATION OF PROBABILISTIC TIMED AUTOMATA

Arnd Hartmanns
University of Twente
Drienerlolaan 5
7522 NB Enschede
NETHERLANDS

Sean Sedwards
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku
Tokyo 101-8430
JAPAN

Pedro R. D'Argenio
Universidad Nacional de Córdoba
Avenida Medina Allende s/n
X5000HUA Córdoba
ARGENTINA

ABSTRACT

Probabilistic timed automata are a formal model for real-time systems with discrete probabilistic and nondeterministic choices. To overcome the state space explosion problem of exhaustive verification, a symbolic simulation-based approach that soundly treats nondeterminism to approximate maximum and minimum reachability probabilities has recently become available. Its use of difference-bound matrices to handle continuous real time however leads to poor performance: most operations are *cubic* or even *exponential* in the number of clock variables. In this paper, we propose a novel region-based approach and data structure that reduce the complexity of all operations to being *linear*. It relies on a particular mapping between symbolic regions and concrete representative valuations. Using an implementation within the MODEST TOOLSET, we show that the new approach is not only easier to implement, but indeed significantly outperforms all current alternatives on standard benchmark models.

1 INTRODUCTION

Probabilistic timed automata (PTA, Kwiatkowska et al. 2002) are a formal model for real-time systems with nondeterministic and discrete-probabilistic choices and delays. Their most prominent application is in the study of distributed algorithms such as network protocols, where real-time behavior (e.g. transmission delays) and requirements (e.g. on response times) meet uncertain operating environments (e.g. sporadic message loss) and randomized algorithms (e.g. exponential backoff). PTA also serve as the semantic foundation for domain-specific languages (van den Berg et al. 2015, Hartmanns, Hermanns, and Bungert 2016) and for the analysis of more expressive formalisms (e.g. Hahn, Hartmanns, and Hermanns 2014).

Nondeterminism is a crucial feature of PTA. It enables abstraction, concurrency, and the representation of absence of knowledge. For example, in a PTA model of wireless communication, each station may nondeterministically choose the exact time when it starts to send. In this way, the model remains abstract w.r.t. any particular ordering and timing of communication. We would then like to answer questions such as "what is the minimum probability to transmit a complete file within 12 s" or "what is the maximum expected time to success". In both cases, we implicitly ask for an *optimal scheduler* to concretely resolve all nondeterministic choices so as to minimize or maximize the quantity of interest. In our example, this could mean choosing send times that cause many, few, or just the right combination of collisions on the shared channel. PTA *model checking* (Norman, Parker, and Sproston 2013) is a formal verification technique to precisely compute measures such as the above. It is limited by the *state space explosion* problem: the model's space of reachable configurations, exponential in the number of model variables and their domains, must be explored and stored in memory for the quantities of interest to be computed via e.g. value iteration.

As *statistical model checking* (SMC, Younes and Simmons 2002, Hérault et al. 2004), the use of Monte Carlo simulation to analyze formal models has become popular because it avoids state space explosion: in a simulation run, only the current and next states are stored, so memory usage is constant. However, SMC is limited to fully stochastic models like Markov chains or semi-Markov processes since nondeterminism is incompatible with simulation. It *is* possible when a *concrete* scheduler to decide all nondeterministic

choices is assumed, but doing so naively leads to non-optimal results. In the first query above, for *some* scheduler, the resulting estimate would lie *somewhere* between the min. and max. probabilities. This implicit use of a hidden scheduler is implemented for PTA in UPPAAL SMC (David et al. 2011), and its results indeed fall unpredictably close to or far from the optima on standard examples (D'Argenio et al. 2016).

Previous work. Performing *sound* SMC for nondeterministic models is a hard problem. For Markov decision processes, Henriques et al. (2012) first proposed the use of machine learning to incrementally improve a partial scheduler, although their approach later turned out to be unsound. Brázdil et al. (2014) proposed a sound alternative. Both suffer from memory usage linear in the number of explored states as all scheduler decisions must be stored explicitly. The same problem affects UPPAAL STRATEGO (David et al. 2015), which explicitly synthesizes a good scheduler before using it for a standard SMC analysis. Classic memory-efficient sampling approaches such as the one of Kearns, Mansour, and Ng (2002) address discounted models only. We recently presented the first approach to perform SMC for PTA (D'Argenio et al. 2016) that (1) soundly deals with nondeterminism by statistically approximating optimal schedulers while also (2) keeping memory usage constant via a particular use of pseudo-random number generators and hash functions (Legay et al. 2014). It is a symbolic simulation approach using *zones* to compactly represent convex sets of clock valuations and perform large delays in a single step. Yet the complexity of most zone operations is in $\mathcal{O}(n^3)$, where n is the number of clock variables, and the currently best method to select a region uniformly at random is *exponential* in n. Performance thus scales unsatisfactorily.

Our contribution. With this paper, we make sound SMC for PTA practical: we present a new algorithm to simulate PTA and approximate optimal schedulers where all operations are in $\mathcal{O}(n)$. It uses *regions* instead of zones. Its efficiency is due to a novel data structure for regions and a particular mapping between regions and representative clock valuations that we introduce in this paper. In contrast to a naive region-based approach, the latter crucially allows us to still perform large delays in a single step. Regions have so far only been considered as a *theoretical* tool to prove decidability of verification problems on (probabilistic) timed automata via the region graph construction: in practice, the region graph is prohibitively large for any exhaustive approach. However, this does not affect simulation. Consequently, and to the best of our knowledge, this paper is the first to investigate efficient data structures and operations for regions, motivated by the simulation perspective. We describe the implementation of our new algorithm, publicly available as part of the MODEST TOOLSET (Hartmanns and Hermanns 2014), and evaluate its performance on a number of case studies from the literature. We see that the new approach consistently outperforms the zone-based method: it is already faster for single-clock PTA (where theoretical complexity is the same), and provides increasing gains as the number of clocks grows. Not least, it is also far simpler to implement.

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 $[0,\infty)$. $\mathbb{Q}_0^+$ is the set of nonnegative rational numbers. $\mathbb{Z}_{32}$ is the set of 32-bit signed integers. $\mathscr{P}(S)$ is the powerset of S. If S is a continuous subset of $\mathbb{R}$, $\mathscr{I}(S)$ is the set of intervals over S. lb_i and ub_i denote the lower resp. upper bounds of $i \in \mathscr{I}(S)$. The concatenation of two objects interpreted as bitstrings is denoted by $a.b$.

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. $\mathscr{D}(\omega)$ is the *Dirac distribution* for ω, defined by $\mathscr{D}(\omega)(\omega) = 1$.

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

For a probability distribution μ, let $\mathscr{U}(\mu)$ denote the pseudo-random selection of a value from $\mathrm{support}(\mu)$ according to a value sampled from $\mathscr{U}$ 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.

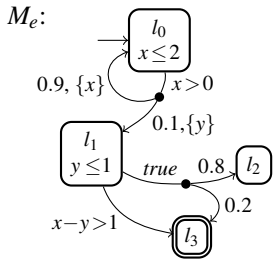

Figure 1: Example PTA M_e.

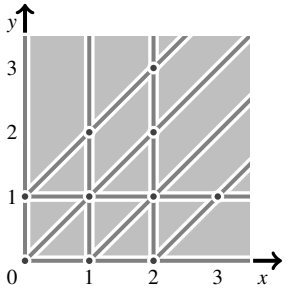

Figure 2: Regions of M_e.

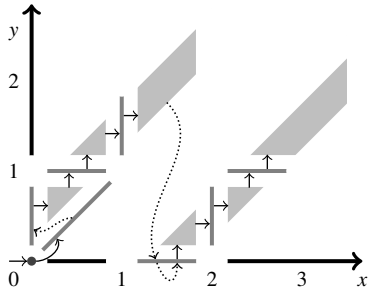

Figure 3: A symbolic trace through M_e.

2.1 Probabilistic Timed Automata

Probabilistic timed automata deal with time through *clocks*: variables with domain $\mathbb{R}_0^+$ that advance synchronously with rate 1 over time. Given a set of clocks $\mathscr{C}$, the valuation $\mathbf{0} \in Val \stackrel{\text{def}}{=} \mathscr{C} \to \mathbb{R}_0^+$ assigns zero to every clock $c \in \mathscr{C}$. For $v \in Val$ and $t \in \mathbb{R}_0^+$, we denote by $v+t$ the valuation where all clocks have been incremented by t, i.e. $\forall c \in \mathscr{C}\colon (v+t)(c) = v(c)+t$, and by $v[X]$ the one where all clocks in $X \subseteq \mathscr{C}$ have been reset to zero. *Clock constraints* are expressions of the form $\mathscr{CC} ::= true \mid false \mid \mathscr{CC} \wedge \mathscr{CC} \mid c \sim n \mid c_1 - c_2 \sim n$ where $\sim \in \{>, \ge, <, \le\}$, $c, c_1, c_2 \in \mathscr{C}$ 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 $\{\ge, \le\}$, it is *closed*. $[\![e]\!]$ for $e \in \mathscr{CC}$ is the set of valuations $v \in Val$ such that e evaluated in v is *true*.

Definition 3 A *probabilistic timed automaton* (PTA) is a 6-tuple $M = \langle Loc, \mathscr{C}, E, l_{init}, tpc \rangle$ where Loc is a countable set of *locations*, $\mathscr{C}$ is a finite set of *clocks*, $E \in Loc \to \mathscr{P}(\mathscr{CC} \times \mathrm{Dist}(T))$ is the *edge function* with $E(l)$ finite for all $l \in Loc$ and targets $T \stackrel{\text{def}}{=} \mathscr{P}(\mathscr{C}) \times Loc$, $l_{init} \in Loc$ is the initial location, and $tpc \in Loc \to \mathscr{CC}$ maps each location to a *time progress condition*.

A triple $\langle l, g, \mu \rangle$ s.t. $\langle g, \mu \rangle \in E(l)$ is an *edge*, also written as $l \xrightarrow{g} \mu$. It consists of the *guard* g and the probability distribution $\mu \in T$ over sets of clocks to reset and target locations. Intuitively, the semantics of a PTA is as follows: When in location l, time can pass while $tpc(l)$ is satisfied. Edge $l \xrightarrow{g} \mu$ can be taken if g is *true* at the current point in time. Then a target $\langle X, l' \rangle$ is chosen according to μ, the clocks in X are reset, and we move to the target location l'. PTA are timed *Markov decision processes* (MDP): an MDP is a PTA with $\mathscr{C} = \varnothing$ where all time progress conditions are *false* and all guards are *true*.

Example 1 We show an example PTA M_e in Figure 1. It has locations $\{l_0, l_1, l_2, l_3\}$ and clocks $\{x, y\}$. The time progress condition is $x \le 2$ in the initial location l_0, and $y \le 1$ in l_1. We omit *true* time progress conditions. There is one edge out of l_0 with guard $x > 0$ into distribution μ_1: back to l_0 with probability 0.9, resetting x, and to l_1 with probability 0.1, resetting y. There are two edges out of l_1. The one with diagonal guard $x - y > 1$ goes to l_3 with probability 1. We omit empty clock reset sets. In l_0 and l_1, the time to delay before taking an edge is nondeterministic; in l_1 additionally the choice of edge is nondeterministic.

Using PTA to directly build models of complex systems is cumbersome. Instead, higher-level formalisms such as MODEST (Bohnenkamp et al. 2006) are used. Aside from a parallel composition operator, they add to PTA variables over finite domains. This allows to compactly describe very large PTA.

2.2 Semantics and Reachability Probabilities

Let us fix a PTA M as in Definition 3. A *state* of M is a pair $\langle l, v \rangle \in S \stackrel{\text{def}}{=} Loc \times Val$ of a location and a valuation for the clocks. To define the valid behaviors of M, we formalize the choices available from a state $\langle l, v \rangle$: either take an edge whose guard is true in v, or delay for $t \in \mathbb{R}^+$ time units if allowed by $tpc(l)$:

Definition 4 Let $\mathrm{Enabled}(\langle l, v \rangle) \stackrel{\text{def}}{=} \{\, \mu \mid l \xrightarrow{g} \mu \wedge v \in [\![g]\!] \,\}$ be the set of enabled target distributions in state $\langle l, v \rangle$. Define the allowed delays as $\mathrm{Delay}(\langle l, v \rangle) \stackrel{\text{def}}{=} \{\, t \in \mathbb{R}^+ \mid \forall t' \in [0, t]\colon v + t' \in [\![tpc(l)]\!] \,\}$. A (memoryless deterministic) *scheduler* is a function $\mathfrak{S}$ s.t. $\mathfrak{S}(\langle l, v \rangle) \in \mathrm{Delay}(\langle l, v \rangle) \uplus \mathrm{Enabled}(\langle l, v \rangle)$ for all states $\langle l, v \rangle$.

A scheduler resolves all nondeterminism over edges and delays. If it chooses an edge, the set of clocks to reset and the target location are chosen probabilistically according to μ. A valid behavior is a path:

Definition 5 A *path* is an infinite alternating sequence $\langle l_0, v_0\rangle\, a_0\, \langle l_1, v_1\rangle\, a_1 \ldots$ of states $\langle l_i, v_i\rangle \in S$ with $\langle l_0, v_0\rangle = \langle l_{init}, \mathbf{0}\rangle$ and delays or target distributions $a_i \in \mathbb{R}^+ \uplus \mathrm{Dist}(T)$ if there exists a scheduler $\mathfrak{S}$ such that $\forall i \in \mathbb{N}\colon\ \mathfrak{S}(\langle l_i, v_i\rangle) = a_i$ and either $a_i = t \in \mathbb{R}^+$ and $\langle l_{i+1}, v_{i+1}\rangle = \langle l_i, v_i + t\rangle$ (a delay step), or $a_i = \mu \in \mathrm{Dist}(T)$, $\langle X, l'\rangle \in \mathrm{support}(\mu)$, and $\langle l_{i+1}, v_{i+1}\rangle = \langle l', v_i[X]\rangle$ (an edge is taken).

Using the usual cylinder set construction (Kwiatkowska et al. 2002), every scheduler $\mathfrak{S}$ defines a probability measure $\mathbb{P}_{\mathfrak{S}}$ on the set of all paths. Let $\delta(\pi)$ be the sum of all $a_i \in \mathbb{R}^+$ on path π. As is standard, we restrict to time-divergent schedulers, i.e. we only consider $\mathfrak{S}$ where $\mathbb{P}_{\mathfrak{S}}(\{\,\pi \in \mathit{Paths}(M) \mid \delta(\pi) = \infty\,\}) = 1$.

Example 2 Let us write valuations v as tuples $\langle v(x), v(y)\rangle$. Then one concrete path in M_e that reaches location l_3 is $\langle l_0, \langle 0,0\rangle\rangle\, 0.8\, \langle l_0, \langle 0.8, 0.8\rangle\rangle\, \mu_1\, \langle l_0, \langle 0, 0.8\rangle\rangle\, 1.1\, \langle l_0, \langle 1.1, 1.9\rangle\rangle\, \mu_1\, \langle l_1, \langle 1.1, 0\rangle\rangle\, \mathscr{D}(\langle \varnothing, l_3\rangle)\langle l_3, \langle 1.1, 0\rangle\rangle \ldots$

We want to answer queries of the form "what is the maximum/minimum probability of eventually/within time t reaching a location $l \in L$ when $c \in \mathscr{CC}$ holds". Time-bounded queries can be turned into unbounded ones by adding a new clock c_t to M that is never reset, and using $c_t \leq t \wedge c$ in place of c. An objective $\langle L, c\rangle$ characterizes the measurable set Π of paths that include a state $\langle l, v\rangle$ such that $l \in L \wedge v \in [\![c]\!]$. We are thus interested in the extremal *reachability probabilities* $\sup_{\mathfrak{S}} \mathbb{P}_{\mathfrak{S}}(\Pi)$ (the maximum probability) and $\inf_{\mathfrak{S}} \mathbb{P}_{\mathfrak{S}}(\Pi)$ (the minimum probability) and refer to them by $\mathrm{P}_{\max}(L \wedge c)$ and $\mathrm{P}_{\min}(L \wedge c)$, respectively, omitting set notation when L is a singleton for readability. Schedulers that realize the sup (inf) exist, and we call them *optimal* or *maximizing* (*minimizing*) schedulers. We can also compute the probabilities of linear-time safety path properties by running M in parallel with a deterministic timed automaton observer and using its final states for L. We abstractly treat such observers as *path properties* ϕ.

Example 3 In M_e, the minimum probability to eventually reach l_3 is $\mathrm{P}_{\min}(l_3) = 0.2$. The maximum is $\mathrm{P}_{\max}(l_3) = 1$; it is only achieved by always waiting in l_0 until x is greater than 1 before taking the edge.

2.3 Symbolic PTA: Regions and Zones

The semantics of a PTA is uncountable: states contain real-valued clock valuations, and paths contain real-valued delay steps. To prove the decidability of PTA verification, i.e. of computing reachability probabilities, Kwiatkowska et al. (2002) have shown that it suffices to consider the finite *region* graph of a PTA, a concept originally introduced by Alur and Dill (1994) for timed automata. The region graph is too large to be useful in practice; effective timed automata verification tools instead use *zones*. Both constructions exploit the restriction that clocks can only be reset to zero and compared to integers in PTA:

Definition 6 Let $k_c \in \mathbb{N}$ be the maximum constant appearing in any comparison involving clock c in a given PTA M as in Definition 3. Then a *zone* is a non-empty set of valuations that can be described by a clock constraint in which all diagonals have the form $c_1 - c_2 \sim n_{c_1 c_2}$ for $n_{c_1 c_2} \in \{\,0, \ldots, \max\{k_{c_1}, k_{c_2}\}\,\}$ and all other comparisons have the form $c \sim n_c$ for $n_c \in \{\,0, \ldots, k_c\,\}$.

Definition 7 A *region* is a minimal zone. The *successor* of a region r is the unique region r^+ such that $\forall v \in r\colon\ v + \min\{\,t \in \mathbb{R}^+ \mid v + t \notin r\,\} \in r^+$, i.e. the first other region encountered when delaying from any valuation in r. For $X \subseteq \mathscr{C}$, $r[X]$ is the unique region obtained from region r by resetting all clocks in X.

Example 4 In M_e, we have $k_x = 2$ and $k_y = 1$. The regions of M_e are visualized in Figure 2: Every gray point, line segment and area is a region. With three clocks, the bounded regions would be points, line segments, triangles and tetrahedrons, and so on for higher dimensions. To find a region's successor, follow a 45-degree line from any point within up to the next region. In Figure 3, we show a symbolic trace through the regions of M_e that includes the behavior of the path shown in Example 2 (omitting the locations). Solid arrows are delay steps to the successor region and dotted arrows correspond to taking an edge.

The standard data structure to represent zones in a verification tool are difference-bound matrices (DBMs, see e.g. Bengtsson and Yi 2003). For a PTA with $\mathscr{C} = \{\,c_1, \ldots, c_l\,\}$, a DBM is a $(|\mathscr{C}|+1) \times (|\mathscr{C}|+1)$ matrix that stores the strict or inclusive upper bound on $c_i - c_j$ at position $\langle i, j\rangle$, where c_0 represents the

Input: MDP $M = \langle Loc, \varnothing, E, l_{init}, \textbf{false} \rangle$, path property ϕ, scheduler identifier $\sigma \in \mathbb{Z}_{32}$
Output: Sampled trace ω

1 $l := l_{init}$, $\omega := \langle l_{init}, \varnothing \rangle$
2 **while** $\phi(\omega) = \text{undecided} \wedge E(\langle l, \varnothing \rangle) \neq \varnothing$ **do** *// run until property decided or deadlock*
3 $\quad \mathscr{U}_{\text{nd}} := \text{PRNG}(\mathscr{H}(\sigma.l))$ *// seed $\mathscr{U}_{\text{nd}}$ with hash of σ and l*
4 $\quad \langle g, \mu \rangle := \lceil \mathscr{U}_{\text{nd}}() \cdot |E(\langle l, \varnothing \rangle)| \rceil$-th element of $E(\langle l, \varnothing \rangle)$ *// use $\mathscr{U}_{\text{nd}}$ to select an edge*
5 $\quad \langle l', \varnothing \rangle := \mathscr{U}_{\text{pr}}(\mu)$ *// use $\mathscr{U}_{\text{pr}}$ to select a target according to μ*
6 $\quad l := l'$, $\omega := \omega.\langle l', \varnothing \rangle$ *// update current location and append to trace*

Algorithm 1: Lightweight simulation for an MDP and a scheduler identifier.

constant 0. Most operations on DBMs, such as computing the intersection of two zones or resetting a set of clocks, require the solution of an all-pairs-shortest-path problem on a graph represented by the matrix and thus take time in $\mathscr{O}(|\mathscr{C}|^3)$, although some can be optimized to $\mathscr{O}(|\mathscr{C}|^2)$ (Bengtsson and Yi 2003). To select a single region uniformly at random from within a zone, the best algorithm we know uses rejection sampling and takes time exponential in n (D'Argenio et al. 2016).

3 LIGHTWEIGHT APPROXIMATION OF OPTIMAL SCHEDULERS

Simple Monte Carlo simulation is not sufficient to compute reachability probabilities for a nondeterministic model: resolving nondeterminism in a randomized way leads to estimates that only lie *somewhere* between the desired extremal values. In addition to computing probabilities, we also need to find (near-)optimal schedulers. One approach is to use simulation-based machine learning algorithms following the ideas of Brázdil et al. (2014) to incrementally improve a candidate optimal scheduler. However, these methods need to store the scheduler's decisions for all (visited) states. This may lead to excessive memory usage.

3.1 The Lightweight Approach for MDP

Legay et al. (2014) introduced a *lightweight* scheduler sampling approach for MDP that identifies a scheduler by a single integer (typically of 32 bits, providing more than sufficient schedulers in practice). This allows to randomly select a large number of schedulers (i.e. integers), perform Monte Carlo simulation/SMC for each, and report the maximum and minimum estimates over all sampled schedulers as approximations of the actual extremal probabilities. Care needs to be taken to account for the accumulation of statistical error introduced by the repeated simulation experiments, and performance can be improved by sampling in a smart way (for details, see D'Argenio et al. 2015). We show the core of the lightweight approach—performing a simulation run for a given scheduler identifier σ—as Algorithm 1. It uses two PRNGs: $\mathscr{U}_{\text{pr}}$ to simulate the probabilistic choices in the MDP in line 5, and $\mathscr{U}_{\text{nd}}$ to resolve the nondeterministic choices in line 4. We want σ to represent a deterministic memoryless scheduler. Therefore, within one simulation run as well as in different runs for the same value of σ, $\mathscr{U}_{\text{nd}}$ must always make the same choice for the same location l. To achieve this, $\mathscr{U}_{\text{nd}}$ is reinitialized with a seed based on σ and l in every step (line 3).

The overall effectiveness of the lightweight approach then depends on the likelihood of selecting a σ that represents a (near-)optimal scheduler. Since we do not know a priori what makes a scheduler optimal, we want to sample "uniformly" from the space of all schedulers. This at least avoids actively biasing against "good" schedulers. More precisely, a uniformly random choice of σ shall result in a uniformly chosen (but fixed) resolution of all nondeterministic choices. Algorithm 1 achieves this naturally for MDP.

3.2 The Naive Extension to PTA Fails

The naive extension of the lightweight approach to PTA is to use Algorithm 1 to generate concrete paths like the one shown in Example 2. The input to $\mathscr{U}_{\text{nd}}$ is then a hash of σ and the current *state* $\langle l, v \rangle$ of

location l plus clock valuation v. $\mathscr{U}_{\text{nd}}$ selects a delay permitted by the time progress condition followed by an enabled edge, if available. A fundamental problem of the naive approach is that it can make (near-)optimal schedulers infeasibly rare. Consider PTA M_e of Example 1 and $\mathrm{P}_{\max}(l_3)$ again. An optimal scheduler must achieve the maximum probability of 1 and thus has to select a delay > 1 whenever we arrive in l_0 (and also select the lower edge from l_1 to l_3 whenever possible). Assuming we pick σ uniformly at random, what is the probability of thus picking an optimal scheduler? Let us write $\mathrm{P}_\sigma(l_3)$ for the probability to reach l_3 with scheduler identifier σ and step through the naive adaptation to PTA of Algorithm 1:

1. Every run starts in $s_0 = \langle l_0, \langle 0,0 \rangle \rangle$. We initialize $\mathscr{U}_{\text{nd}}$ with the hash of σ and s_0 and use it to choose a delay $d_1 \in [0,2]$. The probability that we chose σ such that $\mathscr{U}_{\text{nd}}$ picks a value ≤ 1 is 0.5. We then delay into $\langle l_0, \langle d_1, d_1 \rangle \rangle$ and take the edge. In the case we picked $d \leq 1$, we go to l_1 and then l_2 with probability $0.1 \cdot 0.8 = 0.08$. Thus, with probability 0.5, we picked σ such that $\mathrm{P}_\sigma(l_3) < 0.92$.
2. Now assume we did pick a "good" σ and d_1 is greater than 1, but $\mathscr{U}_{\text{pr}}$ causes us to loop back to l_0 on the current run. We then reinitialize $\mathscr{U}_{\text{nd}}$ with a completely different hash (since the state now includes $v(y) = d_1 \neq 0$) to pick another delay $d_2 \in [0,2]$. The probability that we chose σ such that $\mathscr{U}_{\text{nd}}$ *now* picks a value ≤ 1 is *again* 0.5. Thus, with probability $0.5 + 0.25$, we picked σ such that $\mathrm{P}_\sigma(l_3) \leq 0.928$.

Even though σ is fixed, we get to make a new decision every time we come back to l_0 because the value of y is a different real number in $[0,2]$ every time. The optimal scheduler would have to choose $d_i > 1$ for every $i \in \mathbb{N}$, and the probability of choosing a σ that identifies such a scheduler is zero. If σ is picked from $\mathbb{Z}_{32}$, likely *no* σ corresponds to an optimal scheduler. Even near-optimal schedulers are rare: the probability of having chosen a σ such that $\mathrm{P}_\sigma(l_3) \leq 1 - \varepsilon$ grows quickly while ε decreases slowly. The problem is that we sample from an uncountable space of schedulers where the optimal schedulers have measure zero. We must find an alternative approach where the probability of sampling an optimal scheduler is high.

3.3 Lightweight Zone-Based SMC for PTA

To increase the probability of sampling a (near-)optimal scheduler, we can reduce the space of schedulers. For PTA, some finite abstractions preserve reachability probabilities. On a finite abstraction, there are only finitely many (memoryless deterministic) schedulers. Such abstractions include the region and zone graphs, but also the digital clocks approach (Norman, Parker, and Sproston 2013). In the latter, clocks are replaced by integer variables that are incremented as long as the time progress condition is satisfied, resulting in an MDP. However this only works if all clock constraints in the PTA are closed and diagonal-free. We could naively use Algorithm 1 on the digital clocks MDP (if we accept its restrictions) or the region graph. Both cases are conceptually the same; with the latter, the states are pairs $\langle l, r \rangle$ of the current location l and the clock region r. Since all except the point regions represent uncountably many clock valuations, this is a symbolic simulation technique and we generate runs like the one shown in Figure 3. That figure highlights the main drawback of such an approach: performing a long delay corresponds to a large number of simulation steps to sequentially move through all the successor regions. We found this to cause significant performance problems. The second issue is that we cannot sample schedulers uniformly: As long as the location's time progress condition is satisfied, the only reasonable way to implement the scheduler is to let $\mathscr{U}_{\text{nd}}$ choose uniformly between delaying or taking an edge. The total delay per scheduler thus follows a geometric distribution, biasing the algorithm towards taking edges early.

To solve these two problems, in (D'Argenio et al. 2016) we proposed an approach based on zones, using the standard DBM data structure. We can easily obtain and represent an entire sequence of regions as a single zone, determine the edges enabled throughout that zone, and use $\mathscr{U}_{\text{nd}}$ to uniformly (but deterministically for fixed σ and symbolic state) select one. The resulting algorithm (shown as Algorithm 2 in D'Argenio et al. 2016) is not a simple extension of Algorithm 1 for several reasons that we explore in that paper. In particular, when taking an edge, we need to select a single region from within the target zone. This is to avoid over-/underapproximating the max./min. reachability probabilities since we perform a forwards exploration (Kwiatkowska et al. 2002). As previously mentioned, most operations on zones are in $\mathscr{O}(|\mathscr{C}|^2)$ or $\mathscr{O}(|\mathscr{C}|^3)$ and region selection is currently exponential in $|\mathscr{C}|$ if we aim for uniformly random selection.

Not surprisingly, profiling our previous implementation with standard case studies showed that indeed most of the runtime (~ 66 to $77\,\%$) was spent in various zone operations. In the remainder of this paper, we show how to overcome the two problems that so far prevented the use of regions, in order to significantly improve performance both theoretically and practically.

4 AN EFFICIENT DATA STRUCTURE FOR REGIONS

The concept of regions has been key to proving decidability of various verification problems for (probabilistic) timed automata, including the computation of optimal reachability probabilities for PTA. The total number of regions of a PTA is in $\mathcal{O}(|\mathscr{C}|! \cdot 2^{|\mathscr{C}|} \cdot \prod_{c \in \mathscr{C}}(2 \cdot k_c + 2))$ (Alur and Dill 1994), so any standard model checking approach that exhaustively explores and stores the region graph is infeasible. Consequently, and to the best of our knowledge, the question of finding an efficient data structure for regions has not been considered yet, but it is central to an efficient implementation of the lightweight approach for PTA. In this section we describe our efficient data structure for regions, shown as Algorithm 2. Most importantly, it is set up such that we can perform long delays without having to enumerate successor regions.

4.1 Storing Regions

A region represents a set of clock valuations that cannot be distinguished by any clock constraint. It is fully characterized by the integral part of the value of each clock and whether that value is integer (i.e. whether it is in $\mathbb{N}$ or in $\mathbb{R}_0^+ \setminus \mathbb{N}$) plus the relative order of the fractional parts. We store the integer part of the clock values in vector v_{int} (line 4). Vector o_{frac} (line 5) stores, for each clock, the *index* of the fractional part of its value in the order of fractional values. If *isInt* (line 2) is *true*, the values of the clocks c with $o_{frac}(c) = 0$ are integer; otherwise all clocks have fractional values. This fully characterizes the region; as an optimization for operations defined below, we also store the number of different fractional values as n_d. Note that $n_d - 1 = \max_{c \in \mathscr{C}} o_{frac}(c)$; if all clock values are fractional, then n_d is the *dimension* of the region.

Example 5 With four clocks, the region that contains the valuation $\langle 2, 3.43, 4.6, 4.43 \rangle$ is stored as $v_{int} = \langle 2,3,4,4 \rangle$, $o_{frac} = \langle 0,1,2,1 \rangle$ and $isInt = true$. As two fractional values are the same, we also store $n_d = 3$. Since $isInt = true$, the region is a two-dimensional object (a triangle) in the four-dimensional space spanned by the values of the four clocks. Its successor, with $isInt = false$, is three-dimensional (a tetrahedron).

4.2 Delay Steps and Resets

The standard operations on regions are `delay()` (moving to the successor region) and `reset`(r) (resetting clock r to zero). The implementation of the former is straightforward by observing (cf. Figure 2) that successor regions alternate between having (a) some clocks with integer values (i.e. $isInt = true$) and (b) *only* fractionally-valued clocks (i.e. $isInt = false$). The relative order and dimension do not change. In case (a), all we need to do is set *isInt* to *false* (line 12). In case (b), the clocks with the highest fractional values move up to the next integer value, their index in the fractional order becomes zero, and all other clocks move one index up (lines 10 and 11). The implementation of `reset`(r) is technically more involved: n_d may change depending on whether another clock's value has the same fractional part or is already integer; if it does change, we need to shift the fractional order and/or fill the "hole" left by r. This is implemented in lines 18 to 27. To reset a set of clocks, we call `reset`(r) for each, but this could be optimized.

4.3 Representatives and Durational Delays

The two operations described above were sufficient for the theoretical works using regions that we mentioned before, but result in the problem of having to call `delay()` many times to perform a long delay. To overcome this, we propose two key extensions: The first is the `value`(c) function (line 6) that returns a concrete *representative* value in $\mathbb{Q}_0^+$ for every clock c. For brevity, we will write r not only for the (symbolic) region r, but also for its (concrete) representative clock valuation $\{ c \mapsto r.\texttt{value}(c) \mid c \in \mathscr{C} \}$ whenever a

```
1  type region
2      isInt := true ∈ {true, false}                                  // are one or more clocks integer?
3      n_d := 1 ∈ {1, ..., n}                                         // number of different fractional values
4      v_int := 0 ∈ 𝒞 → ℕ                                             // integer values for all clocks
5      o_frac := 0 ∈ 𝒞 → ℕ                                            // position in the order of fractional values
6      function value(c ∈ 𝒞)
7          return v_int(c) + (2 · o_frac + (if isInt then 0 else 1))/(2 · n_d)
8      function delay()
9          if ¬isInt then foreach c ∈ 𝒞 do
10             o_frac(c) := o_frac(c) + 1 mod n_d                     // cyclic shift of the order of fractional values
11             if o_frac(c) = 0 then v_int(c) := v_int(c) + 1         // highest fractional clocks go to next integer
12         isInt := ¬isInt
13     function delay(d/(2 · n_d))                                    // equivalent to calling delay() d times
14         foreach c ∈ 𝒞 do
15             v_int(c) := v_int(c) + ⌊(2 · o_frac(c) + (if isInt then 0 else 1) + d)/(2 · n_d)⌋
16             o_frac(c) := (o_frac(c) + ⌊(d + (if isInt then 0 else 1))/2⌋) mod n_d
17         if d is odd then isInt := ¬isInt
18     function reset(r ∈ 𝒞)
19         if isInt ∧ o_frac(r) = 0 then v_int(c) := 0; return        // r is already integer
20         same := {c ∈ 𝒞 \ {r} | o_frac(c) = o_frac(r)} ≠ ∅           // other clock at same position in order?
21         if ¬same then n_d := n_d − 1                               // r's old fractional order position becomes free
22         if ¬isInt then n_d := n_d + 1                              // no other clock has fractional value of 0 yet
23         foreach c ≠ r do
24             if ¬same ∧ o_frac(c) > o_frac(r) then
25                 o_frac(c) := o_frac(c) − 1 mod n_d                 // fill gap of r's old position
26             if ¬isInt then o_frac(c) := o_frac(c) + 1 mod n_d      // make room for r's new o_frac
27         o_frac(r) := 0, v_int(r) := 0, isInt := true               // perform the reset for r
```

Algorithm 2: An efficient data structure for regions.

valuation is required by the context. The second extension is the `delay`(d') function (line 13) that, given a certain delay based on a representative valuation, performs that entire delay in one go.

The concrete choice of representative valuations is the key insight of our data structure. For every clock, we select a multiple of $1/(2 \cdot n_d)$: even multiples if the values of one or more clocks are integer, odd multiples otherwise. For illustration, we show as black dots in Figure 4 the representatives of some regions of our example PTA M_e: the one of the region $x = y = 0$ (which has $n_d = 1$), the one of the region $0 < x < y \wedge y = 0$ (with $n_d = 2$), and all successors. The choice is not immediately obvious for all-fractional regions (i.e. triangles in the two-dimensional case) because we select an off-center point. However, it is the only choice where the representatives for a fixed n_d are equally spaced: for any region r, if we delay from its representative for $1/(2 \cdot n_d)$ time units, we arrive exactly at the representative of r^+. This means that the only concrete delays to account for are for multiples of $1/(2 \cdot n_d)$ time units, which makes it possible to implement `delay`(d') with $d' = d/(2 \cdot n_d)$ very efficiently: for each clock, the new integer value is obtained via addition and integer division by $2 \cdot n_d$ (line 15, since a delay of 1 time unit corresponds to $d = 2 \cdot n_d$) while the new fractional order index is obtained by adding $d/2$ modulo n_d (line 16, the division

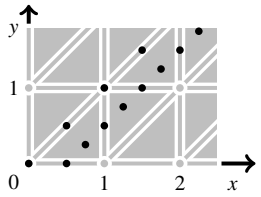

Figure 4: Representatives.

$$\texttt{delays}(v, \mathit{true}) = [0,\infty) \qquad \texttt{delays}(v, \mathit{false}) = \varnothing$$
$$\texttt{delays}(v, c \geq n) = [n - v(c), \infty) \qquad \texttt{delays}(v, c \leq n) = [0, n - v(c)]$$
$$\texttt{delays}(v, c > n) = (n - v(c), \infty) \qquad \texttt{delays}(v, c < n) = [0, n - v(c))$$
$$\texttt{delays}(v, cc_1 \wedge cc_2) = \texttt{delays}(v, cc_1) \cap \texttt{delays}(v, cc_2)$$
$$\texttt{delays}(v, c_1 - c_2 \sim n) = \text{if } v(c_1) - v(c_2) \sim n \text{ then } [0,\infty) \text{ else } \varnothing$$

Figure 5: Mapping clock constraints to delay intervals.

by 2 being due to the alternation between regions where *isInt* is *true* resp. *false* as time passes). Another consequence is that our implementation only needs to work with integer numbers—multiples of the current value of $2 \cdot n_d$—instead of floating-point or rational numbers, giving both efficiency and precision.

4.4 Complexity

The complexity of all operations on our data structure is at most linear in the number of clocks. This drastic reduction compared to zones is because zones are much more expressive than regions—expressiveness that we do not need for our simulation algorithm. For space reasons, Algorithm 2 is simplified: When the values of all clocks are sufficiently large, we must drop upper bounds and eventually stay in the unbounded regions (resulting in the non-triangular areas and half lines in Figure 2). This is possible in linear time, too.

5 REGION-BASED SIMULATION FOR PTA

The new region data structure now allows us to implement the lightweight approach for PTA efficiently: All operations are in $\mathcal{O}(|\mathscr{C}|)$ and we can perform any delay in a single step (i.e. in $\mathcal{O}(|\mathscr{C}|)$ independent of the length of the delay). The overall approach (Algorithm 3 in D'Argenio et al. 2015) remains the same:
1. Randomly select a set M of m integers in $\mathbb{Z}_{32}$ (which represent the schedulers we sample);
2. for each $\sigma \in M$, perform n random simulation runs, resulting in the reachability probability estimate p_σ;
3. return $\max_{\sigma \in M} p_\sigma$ as an approximation for $\mathrm{P}_{\max}(\cdot)$ and $\min_{\sigma \in M} p_\sigma$ as an approximation for $\mathrm{P}_{\min}(\cdot)$.
In step 2, in place of Algorithm 1, which is for MDP, or Algorithm 2 of D'Argenio et al. (2016), which is inefficient, we can now use our new region-based simulation algorithm for PTA that is shown as Algorithm 3.

It crucially relies on the ability to convert between a region and its representative valuation. In line 3, it computes the interval I of possible delays admitted by the current location's time progress condition using the `delays` function. Shown in Figure 5, this recursive function takes any valuation v plus a clock constraint cc—e.g. the time progress condition—and returns the interval $I \in \mathscr{I}(\mathbb{R}_0^+)$ of delays where cc is valid, i.e. $\forall d \in I\colon v + d \in [\![cc]\!]$. I is intersected with each edge's guard (an interval of delays obtained in the same way) in line 6. When the algorithm arrives at line 10, it has collected all enabled edges and the interval of delays during which each is enabled. At that point, it reinitializes $\mathscr{U}_{\text{nd}}$ to act as a memoryless scheduler and lets it (deterministically) select one of the edges (line 11). In lines 12 and 13, we then exploit our particular choice of representatives to let $\mathscr{U}_{\text{nd}}$ select one out of a finite set of delays, each leading directly to one of the successor regions where the selected edge is enabled. The $\texttt{round}_s(G)$ function takes an interval G and returns the largest closed interval $G' \subseteq G$ such that its bounds are multiples of s. For example, $\texttt{round}_{0.5}((0.5, 1.6]) = [1.0, 1.5]$. $\mathscr{U}_{\text{pr}}$ finally selects a target (implementing the Monte Carlo aspect of the simulation) in line 14 and we perform the delay (in a single step) and take the edge in line 15.

The algorithm contains some technicalities to deal with the possibility to let time pass beyond the last enabled edge ("delay into deadlock", but not necessarily timelock). We record all delays in the trace so as not to "jump over" intermediate points where a clock constraint in the property is temporarily satisfied.

6 EXPERIMENTS

We have extended the MODEST TOOLSET (Hartmanns and Hermanns 2014) with implementations of (1) the zone-based approach of D'Argenio et al. (2016), replacing the earlier prototype used for the experiments

Input: PTA $M = \langle Loc, \mathscr{C}, A, E, l_{init}, tpc \rangle$, path property ϕ, scheduler identifier $\sigma \in \mathbb{Z}_{32}$
Output: Sampled trace ω

$l := l_{init}$; $r := \mathbf{0}$; $\omega := \langle l_{init}, r \rangle$
while $\phi(\omega) =$ undecided **do**
  $I := \texttt{delays}(r, tpc(l))$; $J := \varnothing$; $j := 0$ *// get delay interval for the time progress condition*
  if $0 \notin I$ **then** $I := [0,0]$ *// time progress condition does not allow time to pass*
  foreach $l \xrightarrow{g} \mu$ **do** *// for each edge in this location:*
    $J := J \cup \{ \langle \mu, I \cap \texttt{delays}(r,g) \rangle \}$ *// store its distribution and the enabled interval*
    $j := \max\{ j, \mathrm{ub}_{\texttt{delays}(r,g)} \}$ *// keep track of the max. delay that enables an edge*
  if $J = \varnothing$ **then** $\omega := \omega.\mathrm{ub}_I$; **break** *// can only delay into deadlock*
  if $j \neq \mathrm{ub}_I$ **then** $J := J \cup \{ \mathscr{D}(\langle \varnothing, l \rangle), [j,j] \}$ *// possible delay into deadlock*
  $\mathscr{U}_{\mathrm{nd}} := \mathrm{PRNG}(\mathscr{H}(\sigma.l.r))$ *// use hash of σ, l and r as seed for $\mathscr{U}_{\mathrm{nd}}$*
  $\langle \mu, G \rangle := \lceil \mathscr{U}_{\mathrm{nd}}() \cdot |J| \rceil$-th element of J *// use $\mathscr{U}_{\mathrm{nd}}$ to select one of the edges*
  $G := \texttt{round}_{1/(2 \cdot r.n_d)}(G)$ *// make guard interval closed and round its bounds*
  $t := \lfloor \mathscr{U}_{\mathrm{nd}}() \cdot (\mathrm{ub}_G - \mathrm{lb}_G) \cdot 2 \cdot r.n_d \rfloor / (2 \cdot r.n_d) + \mathrm{lb}_G$ *// use $\mathscr{U}_{\mathrm{nd}}$ to select representative delay*
  $\langle X, l' \rangle := \mathscr{U}_{\mathrm{pr}}(\mu)$ *// use $\mathscr{U}_{\mathrm{pr}}$ to select resets, target according to μ*
  $r.\texttt{delay}(t)$, $r.\texttt{reset}(X)$, $l := l'$, $\omega := \omega.t.\langle l, r \rangle$ *// delay, reset, update location and trace*

Algorithm 3: Lightweight symbolic simulation for a PTA and a scheduler identifier using regions.

in that paper, (2) the naive adaptation of Algorithm 1 to the region graph with iterative calls to `delay()` to step through successor regions as described in Section 3.3, and finally (3) our new approach described in Section 5 based on the new region data structure of Section 4 where even long delays are only a single step using the $\texttt{delays}(v, cc)$ and $\texttt{delay}(d)$ functions. The MODEST TOOLSET is publicly available at www.modestchecker.net. We have applied this implementation to four case studies from the literature, the first three adapted from the PRISM benchmark suite (Kwiatkowska, Norman, and Parker 2012):

firewire: A MODEST PTA model of the IEEE 1394 FireWire root contention protocol. We calculate the minimum and maximum probability of termination within 4000 ns.

wlan: A MODEST PTA model of IEEE 802.11 wireless LAN with two stations, using the original timing parameters from the standard (e.g. a maximum transmission time of 15717 µs). We calculate the minimum and maximum probability that either station's backoff counter reaches value 2 with one transmission.

csmacd$_n^D$: A MODEST PTA model of the exponential backoff procedure in the IEEE 802.3 CSMA/CD protocol with n stations on a shared medium. We calculate the minimum and maximum probability of all stations correctly delivering their packets within D µs.

mpeg4: A flexible scenario-aware dataflow (xSADF) model (Hartmanns, Hermanns, and Bungert 2016) of a streaming MPEG-4 SP decoder with 4 kernels and 1 detector operating in 9 scenarios. In contrast to the original SADF model, we have made the execution time of the `VLD` kernel nondeterministic for `I`-frames. We calculate the minimum and maximum probability for kernel `MC` to fire within 1000 model time units.

All our experiments used a 2.7-3.5 GHz Intel Core i5 6600T system running 64-bit Windows 10, using 3 simulation threads. We show the results in Table 1. Column "$|\mathscr{C}|$" lists the number of clocks in each model. The *firewire* model and two of the *csmacd* instances are small enough to be amenable to exhaustive model checking, so we can report the true interval $[\mathrm{P}_{\min}, \mathrm{P}_{\max}]$ of the min. and max. probability under heading "mc". For the three simulation-based approaches, we sampled $m = 500$ schedulers and performed $n = 29\,068$ random simulation runs for each. Thus by the adapted Chernoff bound of D'Argenio et al. (2015), with probability 0.95, estimates (an upper bound for the min. and a lower bound for the max. probability) will be within ± 0.01 of the true values *for the sampled schedulers*. Columns "time" report the simulation time in minutes for all 14 534 000 runs, "$|\omega|$" are the average lengths of each run (in simulation steps, i.e. transitions taken and calls to $\texttt{delay}(d)$ or `delay()`), and "results" list the obtained estimates.

Table 1: Experimental results.

		mc	zones		**regions**			regions (naive)		
model	$\lvert\mathscr{C}\rvert$	results	time	results	$\lvert\omega\rvert$	time	results	$\lvert\omega\rvert$	time	results
firewire	1	$[0.781, 1]$	1.2	$[0.78, 1]$	8	0.8	$[0.79, 1]$	3811	84.4	$[1, 1]$
wlan	2	—	16.9	$[0.04, 0.07]$	71	12.5	$[0.04, 0.07]$	> 10h		
$csmacd_2^{1800}$	4	$[0.729, 0.872]$	24.4	$[0.72, 0.87]$	43	8.0	$[0.72, 0.85]$	2351	266.3	$[0.73, 0.88]$
$csmacd_3^{2700}$	5	$[0.663, 0.892]$	71.0	$[0.76, 0.87]$	106	23.8	$[0.77, 0.85]$	3881	482.1	$[0.88, 0.90]$
$csmacd_4^{3600}$	6	—	152.8	$[0.79, 0.86]$	197	45.7	$[0.80, 0.85]$	> 10h		
mpeg4	5	—	191.2	$[0.61, 0.62]$	347	71.2	$[0.61, 0.62]$	1811	168.2	$[0.61, 0.62]$

We see that, as expected, the zone-based approach (heading "zones") is slow, and scales poorly with the number of clocks. This is despite our implementation in the MODEST TOOLSET using a non-uniform algorithm for region selection that is faster than the original rejection sampling-based method of D'Argenio et al. (2016). If we use regions, but restrict to the `delay()` and `reset(`r`)` functions (heading "regions (naive)"), we have to iterate through a large number of successor regions individually for large delays and bias towards short delays as explained in Section 3.3. We see that, accordingly, the simulation runs have many steps and mostly take an order of magnitude more time than the other approaches. Finally, our new approach using regions with representatives and the `delay(`d`)` function (heading "regions") is consistently the fastest. Its performance depends mostly on the length of the simulation runs and not significantly on $|\mathscr{C}|$. The estimates are not always close to both optimal probabilities where we could model-check. This could likely be improved by simulating a much larger number of schedulers on a compute cluster with higher m, or by using the smart sampling approach of D'Argenio et al. (2015). The estimates of the three approaches are not always the same: In addition to the low number of schedulers, this is likely because region selection is not uniform in the zone-based approach, and we heavily bias towards shorter delays in the naive approach. The latter means that we did not happen to sample any scheduler that leads to a minimum probability < 1 for the *firewire* model.

7 CONCLUSION

The lightweight scheduler sampling technique is a practical method to exploit the scalability of Monte Carlo simulation for the verification of the nondeterministic formalism of probabilistic timed automata. Motivated by this simulation perspective, we presented a novel data structure for regions. To the best of our knowledge, it is the first investigation of this problem. Building on the ability to convert regions to a particular form of representative valuations in order to perform long delays in a single step, we proposed a new symbolic simulation algorithm for use within the lightweight technique. In contrast to previous zone-based approaches, the complexity of a single simulation step is reduced from *cubic* or even *exponential* to only *linear* in the number of clocks. We demonstrated its superior performance on several case studies.

ACKNOWLEDGMENTS

This work is supported by the 3TU.BSR project, the JST ERATO HASUO Metamathematics for Systems Design project (JPMJER1603), ERC grant 695614 (POWVER), and SeCyT-UNC projects 05/BP12, 05/B497.

REFERENCES

Alur, R., and D. L. Dill. 1994. "A Theory of Timed Automata". *Theor. Comput. Sci.* 126 (2): 183–235.

Bengtsson, J., and W. Yi. 2003. "Timed Automata: Semantics, Algorithms and Tools". In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, Volume 3098 of *LNCS*, 87–124: Springer.

Bohnenkamp, H. C., P. R. D'Argenio, H. Hermanns, and J.-P. Katoen. 2006. "MoDeST: A Compositional Modeling Formalism for Hard and Softly Timed Systems". *IEEE Trans. Software Eng.* 32 (10): 812–830.

Brázdil, T., K. Chatterjee, M. Chmelik, V. Forejt, J. Kretínský, M. Z. Kwiatkowska, D. Parker, and M. Ujma. 2014. "Verification of Markov Decision Processes Using Learning Algorithms". In *ATVA*, Volume 8837 of *LNCS*, 98–114: Springer.

D'Argenio, P. R., A. Hartmanns, A. Legay, and S. Sedwards. 2016. "Statistical Approximation of Optimal Schedulers for Probabilistic Timed Automata". In *iFM*, Volume 9681 of *LNCS*, 99–114: Springer.

D'Argenio, P. R., A. Legay, S. Sedwards, and L.-M. Traonouez. 2015. "Smart Sampling for Lightweight Verification of Markov Decision Processes". *Software Tools for Technology Transfer* 17 (4): 469–484.

David, A., P. G. Jensen, K. G. Larsen, M. Mikucionis, and J. H. Taankvist. 2015. "Uppaal Stratego". In *TACAS*, Volume 9035 of *LNCS*, 206–211: Springer.

David, A., K. G. Larsen, A. Legay, M. Mikucionis, and Z. Wang. 2011. "Time for Statistical Model Checking of Real-Time Systems". In *CAV*, Volume 6806 of *LNCS*, 349–355: Springer.

Hahn, E. M., A. Hartmanns, and H. Hermanns. 2014. "Reachability and Reward Checking for Stochastic Timed Automata". *Electronic Communications of the EASST* 70.

Hartmanns, A., and H. Hermanns. 2014. "The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification". In *TACAS*, Volume 8413 of *LNCS*, 593–598: Springer.

Hartmanns, A., H. Hermanns, and M. Bungert. 2016. "Flexible Support for Time and Costs in Scenario-Aware Dataflow". In *EMSOFT*: ACM.

Henriques, D., J. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. 2012. "Statistical Model Checking for Markov Decision Processes". In *QEST*, 84–93: IEEE Computer Society.

Hérault, T., R. Lassaigne, F. Magniette, and S. Peyronnet. 2004. "Approximate Probabilistic Model Checking". In *VMCAI*, Volume 2937 of *LNCS*, 73–84: Springer.

Kearns, M. J., Y. Mansour, and A. Y. Ng. 2002. "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes". *Machine Learning* 49 (2-3): 193–208.

Kwiatkowska, M. Z., G. Norman, and D. Parker. 2012. "The PRISM Benchmark Suite". In *QEST*, 203–204: IEEE Computer Society.

Kwiatkowska, M. Z., G. Norman, R. Segala, and J. Sproston. 2002. "Automatic Verification of Real-Time Systems with Discrete Probability Distributions". *Theor. Comput. Sci.* 282 (1): 101–150.

Legay, A., S. Sedwards, and L.-M. Traonouez. 2014. "Scalable Verification of Markov Decision Processes". In *WS-FMDS*, Volume 8938 of *LNCS*, 350–362: Springer.

Norman, G., D. Parker, and J. Sproston. 2013. "Model Checking for Probabilistic Timed Automata". *Formal Methods in System Design* 43 (2): 164–190.

van den Berg, F., J. Hooman, A. Hartmanns, B. R. Haverkort, and A. Remke. 2015. "Computing Response Time Distributions Using Iterative Probabilistic Model Checking". In *EPEW*, Volume 9272 of *LNCS*, 208–224: Springer.

Younes, H. L. S., and R. G. Simmons. 2002. "Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling". In *CAV*, Volume 2404 of *LNCS*, 223–235: Springer.

AUTHOR BIOGRAPHIES

ARND HARTMANNS is a postdoctoral researcher at the University of Twente in the Netherlands. His research interests include modeling formalisms, scalable verification algorithms and efficient user-friendly tools for the quantitative analysis of critical cyber-physical systems. His email address is a.hartmanns@utwente.nl.

SEAN SEDWARDS works at the National Institute of Informatics in Tokyo, Japan. Focusing on scalable formal verification, he is currently interested in applying machine learning to verification and formally verifying machine-learned systems in the area of autonomous vehicles. His email address is sedwards@nii.ac.jp.

PEDRO R. D'ARGENIO is a full professor at Universidad Nacional de Córdoba and researcher of CONICET, Argentina, and visiting professor at Saarland University, Germany. His research interests include modeling and verification of concurrent systems, performance and dependability evaluation, and the mathematical foundations of nondeterministic and probabilistic systems. His email address is dargenio@famaf.unc.edu.ar.