On Time with Minimal Expected Cost!

Alexandre David[1], Peter G. Jensen[1], Kim Guldstrand Larsen[1], Axel Legay[2], Didier Lime[3], Mathias Grund Sørensen[1], and Jakob H. Taankvist[1]

[1] Aalborg University, Denmark
[2] INRIA Rennes, France
[3] École Centrale de Nantes, IRCCyN, Nantes, France
{adavid,kgl}@cs.aau.dk, {pgje09,mgso09,jtaank09}@student.aau.dk,
axel.legay@inria.fr, didier.lime@irccyn.ec-nantes.fr

Abstract. (Priced) timed games are two-player quantitative games involving an environment assumed to be completely antogonistic. Classical analysis consists in the synthesis of strategies ensuring safety, time-bounded or cost-bounded reachability objectives. Assuming a randomized environment, the (priced) timed game essentially defines an infinite-state Markov (reward) decision proces. In this setting the objective is classically to find a strategy that will minimize the expected reachability cost, but with no guarantees on worst-case behaviour. In this paper, we provide efficient methods for computing reachability strategies that will both ensure worst case time-bounds *as well as* provide (near-) minimal expected cost. Our method extends the synthesis algorithms of the synthesis tool Uppaal-Tiga with suitable adapted reinforcement learning techniques, that exhibits several orders of magnitude improvements w.r.t. previously known automated methods.

1 Motivation

Sparse time and resources are common problems to projects in almost any domain, ranging from manufacturing to office work-flow and program parallelization. In a real world setting, the duration of a process is dependent on the tasks it is composed of. The durations and arrival pattern of tasks are not static, but uncertain by nature. Furthermore, tasks are often solved by different agents running in parallel, creating races for shared resources. A scheduler is needed to handle these conflict situations.

The above type of scheduling problem may conveniently be represented as a timed game (TG) [27], being a two-player quantitative game involving an adversary (modeling the environment – here the tasks) which is assumed to be completely antagonistic. Classical analysis consists in the synthesis of strategies ensuring safety or time-bounded reachability objectives. In all cases, decidability for TGs are obtained from the existence of equivalent finite-state games constructed using the classical notion of regions for timed automata [3]. Moreover, efficient symbolic on-the-fly algorithms using have been developed and implemented as found in Uppaal-Tiga [4]. The assignment of resources to tasks incurs a cost – e.g. energy-consumption. This naturally leads to the extended setting of *priced* timed games [10,11] (PTG), for which – unfortunately – the corresponding synthesis problem of cost-bounded reachability strategies is undecidable in general [13], with the one-clock case being a notable exception [11].

Now, assuming a randomized environment – e.g. where the duration of tasks are stochastic – the (priced) timed game essentially defines an infinite-state Markov (reward) decision process, here named (priced) timed Markov decision processes (PTMDP). In this setting the objective is to find a strategy that will minimize the expected reachability cost, but with no guarantees on worst-case behavior.

In this paper, we provide efficient methods for synthesizing reachability strategies for PTMDPs that subject to guaranteeing a given worst case time-bound, will provide (near-) minimal expected reachability cost.

Assume a (deterministic) strategy has been synthesized guaranteeing a given time-bound, we may – as a first attempt – apply statistical model checking as found in Uppaal SMC [16], to estimate the expected reachability cost in the (priced) timed game under the given strategy. Statistical model checking [26] is a highly scalable technique which achieves its estimates by random sampling of runs, the number of which depending on the desired precision and confidence. However, there may be several strategies guaranteeing the given time-bound and we want the one with minimal expected reachability cost. For this much more ambitious goal, we apply suitable adapted reinforcement learning techniques: starting from a uniformized version of the *most permissive* strategy guaranteeing the given time bound, the learning technique iteratively improves the strategy – by observing the effect of control-choices in sampled runs – until a strategy with satisfactory expected reachability-cost is found. Crucial to the efficiency of our simulation-based synthesis method is the effective and space-efficient representation and manipulation of strategies. Besides the symbolic (zone-based) representation used for TGs, we consider a number of techniques well-known from Machine Intelligence (covariance matrices, logistic regression) as well as a new splitting data-structure of ours. The resulting method is implemented in a new version of Uppaal-Tiga that supports the statistical model-checking techniques of Uppaal. The experimental evaluation has been performed on a large collection of job-shop-like problems (so-called Duration Probabilistic Automata) demonstrating several order or magnitude improvements with respect to previous exact synthesis methods [22].

Example. Consider the PTMDP of Fig. 1 modeling a process consisting of a sequence of two uncontrollable steps (indicated by dashed edges), `r, d`, with a possible control action (indicated by full edges), `a, b, w` being taken after the first step. The first step `r` is taken between 0 and 100 time-units according to a uniform distribution[1] as can be seen by the invariant `x<=100` and the absent guard, and with cost-rate `c'==0`. In the next step, the controller may suggest to play any of the time-action pairs $(d, \mathtt{a})$, $(d, \mathtt{b})$ with $d \leq 100$ or $(100, \mathtt{w})$. These

[1] Following the stochastic semantics for timed automata components applied in Uppaal SMC.

will be in competition with the uniformly distributed choices of the environment $(e, \mathtt{d})$ with $e \in [90, 100]$. It is clear that in terms of worst-case time, the best choice for the controller is $(100, \mathtt{w})$ with 200 as worst-case overall time. In contrast, the worst choice for the controller is $(100, \mathtt{b})$ with 340 as worst-case time. For expected cost, the optimal and worst choices are $(0, \mathtt{b})$ respectively $(90, \mathtt{a})$ with $2 * 80 = 160$ respectively $4 * 90 + 3 * 90 = 630$ as expected remaining reachability cost due to the uniform distributions resolving the delays. Thus, in case there is no upper time bound to be guaranteed (or it is above 240) the cost-optimal strategy will be to choose `b` immediately, yielding an expected cost of 160.

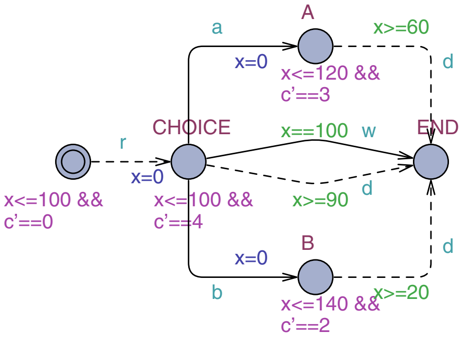

Fig. 1. A Priced Timed MDP

Now assume that the `END` location must be reached within an upper time-bound of 210. The on-the-fly method of Uppaal-Tiga (exploiting early termination) may (in fact will) produce the strategy which deterministically chooses $(100, \mathtt{w})$. This clearly meets the given upper time-bound, and yields an expected reachability cost of $4 * 95 = 380$. The most permissive strategy guaranteeing the time-bound `210` (also obtainable by Uppaal-Tiga) will have the choice depend on the time-point t when `CHOICE` is reached: if $t > 90$ only $(100, \mathtt{w})$ is a legal choice; if $70 < t \leq 90$ also $(d, \mathtt{a})$ with $d \leq 90 - t$ are legal choices, and finally if $t \leq 70$ also $(e, \mathtt{b})$ with $e \leq 70 - t$ are legal. The strategy with minimal expected reachability cost while guaranteeing the time-bound 210, will (obviously) deterministically make the "cheapest" legal choice for a given value of t, i.e. $(100, \mathtt{w})$ for $t > 90$, $(0, \mathtt{a})$ when $70 < t \leq 90$, and $(0, \mathtt{b})$ when $t \leq 70$. This yields 204 as minimum expected value

Related Work. A number of models combining continuous time and Markov decision processes have previously been proposed. We mention some of these below, and point out that they are all special cases of our proposed PTMDP formalism.

Probabilistic timed automata (PrTA) [21,24] extends the fully non-deterministic formalism of timed automata with probabilistic resolution of discrete choices, thus providing an infinite-state MDP with choices of dealys to be resolved by the strategy. Decidability for PrTA w.r.t. optimal (minimum and maximum) reachability probabilities as well as general model checking with respect to PCTL are obtained using region-constructions. Tool support for analysis of PrTAs are provided in Prism [23]. More recently, cost-bounded reachability for priced extensions of PrTAs has been considered, showing undecidability for more than 3 clocks [6] and with the semi-decision algorithmic based tool Fortuna [7].

Continuous-time Markov decision processes (CTMDPs) are also special cases of PTMDPs, where the delay choice of the environment is always made according to exponential distributions, and with choices of the strategy being instantaneous. In the setting of CTMDPs a number of bounded reachability synthesis problems has been recently addressed. In [19] multi-dimensional maximal cost-bounded reachability probability over CTMDPs are considered, offering a numerical approximation algorithm. In [8] a marriage of CTMDPs with timed automata is considered showing the existence of finite optimal schedulers for time-bounded reachability objectives. In [12], stochastic real-time games are considered where states are partitioned into environment nodes - where the behaviour is similar to CTMDPs - and control nodes - where one player chooses a distribution over actions, which induces a probability distribution for the next state. For this game model, objectives are given by deterministic timed automata (DTA), with results focusing on qualitative properties.

Our real-time synthesis problem – aiming at optimal expected cost subject to worst-case time bounds – extends the notion of *beyond* worst-case synthesis in [14] introduced for finite state MDPs, with consideration of minimizing expectation of mean-payoff (shown to be in NP∩coNP) as well reachability cost (shown to be NP-hard). The DPA formalisms considered in [22] is a proper subclass of PTMDP of this paper. In [22] exact methods for synthesizing strategies with minimal expected completion time are given and implemented. However, no worst-case guarantees are considered. As we shall demonstrate our reinforcement learning method produces identical solutions and with an order of magnitude time-improvement. [25] uses a version of Kearns algorithm to find a memoryless scheduler for expecting reward, however with no implementation provided, and no real-time consideration. Our use of statistical model checking for learning optimal strategies of PTMDPs extends that of [20] from finite-state MDPs to the setting of timed game automata based, infinite state MDPs requiring the use of symbolic strategies. Finally, statistical model checking has been used for confluent MDPs in [9].

2 Priced Timed Markov Decision Processes

Priced Timed Games [27] are two-player games played on (priced) timed automata [3,5]. Here we recall the basic results. Let $X = \{x, y, ...\}$ be a finite set of clock. We define $\mathcal{B}(X)$ as the set of clock constraints over X generated by grammar: $g, g_1, g_2 ::= x \bowtie n \mid x - y \bowtie n \mid g_1 \wedge g_2$, where $x, y \in X$ are clocks, $n \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$.

Definition 1. *A* Priced Timed Automaton *(PTA)* $\mathcal{A} = (L, \ell_0, X, \Sigma, E, P, Inv)$ *is a tuple where* L *is a finite set of locations,* $\ell_0 \in L$ *is the initial location,* X *is a finite set of non-negative real-valued clocks,* Σ *is a finite set of actions,* $E \subseteq L \times \mathcal{B}(X) \times \Sigma \times 2^X \times L$ *is a finite set of edges,* $P : L \to \mathbb{N}$ *assigns a price-rate to each location, and* $Inv : L \to \mathcal{B}(X)$ *sets an invariant for each location.*

The semantics of a PTA $\mathcal{A}$ is a *priced transition system* $S_{\mathcal{A}} = (Q, q_0, \Sigma, \to)$, where the set of states Q consists of pairs (ℓ, v) with $\ell \in L$ and $v \in \mathbb{R}_{\geq 0}^X$ such that $v \models Inv(\ell)\}$, and $q_0 = (\ell_0, 0)$ is the initial state. Σ is a finite set of actions, and $\to \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times \mathbb{R}_{\geq 0} \times Q$ is the priced transition relation defined separately for action $a \in \Sigma$ and delay $d \in \mathbb{R}_{\geq 0}$ as:

- $(\ell, v) \xrightarrow{a}_0 (\ell', v')$ if there is an edge $(\ell \xrightarrow{g,\alpha,r} \ell') \in E$ such that $v \models g$, $v' = v[r \mapsto 0]$ and $v' \models \mathit{Inv}(\ell')$,
- $(\ell, v) \xrightarrow{d}_p (\ell, v + d)$, where $p = P(\ell) \cdot d$, $v \models \mathit{Inv}(\ell)$ and $v + d \models \mathit{Inv}(\ell)$.

Thus, the price of an action-transition is 0, whereas the price of a delay transition is proportional to the delay according to the price-rate of the given location. We shall omit price-subscripts when the actual price do not matter. We shall assume that $S_{\mathcal{A}}$ is *deterministic* in the sense that any state $q \in Q$ has at most one successor q^α for any action or delay $\alpha \in (\Sigma \cup \mathbb{R}_{\geq 0})$. A *run* of a PTA $\mathcal{A}$ is an alternating sequence of priced action and delay transitions of its priced transition system $S_{\mathcal{A}}$: $\pi = q_0 \xrightarrow{d_0}_{p_0} q_0' \xrightarrow{a_0}_0 q_1 \xrightarrow{d_1}_{p_1} q_1' \xrightarrow{a_1}_0 \cdots \xrightarrow{d_{n-1}}_{p_{n-1}} q_{n-1}' \xrightarrow{a_{n-1}}_0 q_n \cdots$, where $a_i \in \Sigma$, $d_i, p_i \in \mathbb{R}_{\geq 0}$, and q_i is a state (ℓ_{q_i}, v_{q_i}). We denote the set of runs of $\mathcal{A}$ as $\mathit{Exec}_{\mathcal{A}}$, and $\mathit{Exec}^f_{\mathcal{A}}$ ($\mathit{Exec}^m_{\mathcal{A}}$) for the set of its finite (maximal) runs. For a run π we denote by $\pi[i]$ the state q_i, and by $\pi|_i$ ($\pi|^i$) the prefix (suffix) of π ending (starting) at q_i. For a finite run π, $C(\pi)$ denotes its total accumulated cost $\sum_{i=0}^{n-1} p_i$. Similarly $T(\pi)$ denotes the total accumulated time $\sum_{i=0}^{n-1} d_i$. An infinite run π is said to be cost-divergent provided $\lim_{n\to\infty} \sum_{i=0}^{n-1} p_i = +\infty$. We say that $\mathcal{A}$ is (cost-) non-Zeno provided every infinite run is time-(cost-)divergent.

Definition 2. *A* Priced Timed Game $\mathcal{G}$ *(PTG) is a PTA whose actions Σ are partitioned into controllable (Σ_c) and uncontrollable (Σ_u) actions.*

We note, that for PTAs and PTGs with $P(\ell) = 1$ in all locations ℓ, we obtain standard timed automata (TA) and timed games (TG). Given a (P)TG $\mathcal{G}$, a set of goal-locations $G \subseteq L$ and a cost- (time-) bound $B \in \mathbb{R}_{\geq 0}$, the (G, B) *cost-(time-) bounded reachability control problem* for $\mathcal{G}$ consists in finding a *strategy* σ that will enforce G to be reached within accumulated cost (time) B. Informally, a strategy σ decides to continue a run π either by a proposed controllable action $a \in \Sigma_c$ or by a delay - indicated by the symbol λ. The formal definition of this control problem is based on definitions of *strategy* and *outcome*.

Definition 3. *A* strategy σ *over a PTG $\mathcal{G}$ is a partial function from $\mathit{Exec}^f_{\mathcal{G}}$ to $\mathcal{P}(\Sigma_c \cup \{\lambda\}) \setminus \{\emptyset\}$ such that for any finite run π ending in state $q = \mathit{last}(\pi)$, if $a \in \sigma(\pi) \cap \Sigma_c$, then there must exist a transition $q \xrightarrow{a} q'$ in $S_{\mathcal{G}}$.*

Given a PTG $\mathcal{G}$ and a strategy σ over $\mathcal{G}$, the outcome $\mathit{Out}(\sigma)$ is the subset of $\mathit{Exec}_{\mathcal{G}}$ defined inductively by $q_0 \in \mathit{Out}(\sigma)$, and:

- If $\pi \in \mathit{Out}(\sigma)$ then $\pi' = \pi \xrightarrow{e} q' \in \mathit{Out}(\sigma)$ if $\pi' = \mathit{Excel}_{\mathcal{G}}$ and either one of the following three conditions hold:
  1. $e \in \Sigma_u$, or
  2. $e \in \Sigma_c$ and $e \in \sigma(\pi)$, or
  3. $e \in \mathbb{R}_{\geq 0}$ and for all $e' < e$, $\mathit{last}(\pi) \xrightarrow{e'} q'$ for some q' s.t. $\sigma(\pi \xrightarrow{e'} q') \ni \lambda$.

Let (G, B) be a cost- (time-) bounded reachability objective for $\mathcal{G}$. We say that a maximal, finite run π is *winning* w.r.t. (G, B), if $\mathit{last}(\pi) \in G \times \mathbb{R}^X_{\geq 0}$ and

$C(\pi) \leq B$. A strategy σ over $\mathcal{G}$ is a *winning strategy* if all runs in $Out(\sigma)$ are winning (w.r.t. (G, B)).

A *memoryless strategy* σ only depends on the last state of a run, e.g. whenever $last(\pi) = last(\pi')$, then $\sigma(\pi) = \sigma(\pi')$. For unbounded reachability and safety objectives for TGs, memoryless strategies suffices [27], For TGs with an additional clock `time`, which is never reset (here named *clocked* TGs), memoryless strategies even suffices for time-bounded reachability objectives.

The notion of strategy in Def. 3 is non-deterministic, thus inducing a natural order of *permissiveness*: $\sigma \preceq \sigma'$ iff $\sigma(\pi) \subseteq \sigma'(\pi)$ for any finite run π. *Deterministic* strategies – returning singleton-sets for each run – are least permissive. For safety objectives – being maximal fixed-points – strategies are closed under point-wise union, yielding (unique) *most permissive strategies.* For TGs being non-Zeno, time-bounded reachability objectives *are* safety properties.

Theorem 1. *Let $\mathcal{G}$ be a non-Zeno, clocked TG. If a time-bounded reachability objective (G, T) has a winning strategy, then it has (a) deterministic, memoryless winning strategies, and (b) a (unique) most permissive, memoryless winning strategy $\sigma^{p}_{\mathcal{G}}(G, T)$.*

The tool Uppaal-Tiga [4] provides on-the-fly, symbolic (zone-based) algorithms for computing both types of memoryless safety strategies for TGs. For PTGs, the synthesis problem for cost-bounded reachability problems is in general undecidable [13].

Priced Timed Markov Decision Processes. The definition of outcome of a strategy in the previous Section assumes that an environment behaves completely antagonistically. We will now assume a randomized environment, where the choices of delay and uncontrollable actions are stochastic according to a (delay,action)-density function for a given state.

Definition 4. *A* Priced Timed Markov Decision Process *(PTMDP) is a pair $\mathcal{M} = \langle \mathcal{G}, \mu^u \rangle$, where $\mathcal{G} = (L, \ell_0, X, \Sigma_c, \Sigma_u, E, P, Inv)$ is a PTG, and μ^u is a family of density-functions, $\{\mu^u_q : \exists \ell \exists v. q = (\ell, v)\}$, with $\mu^u_q(d, u) \in \mathbb{R}_{\geq 0}$ assigning the* density *of the environment aiming at taking the uncontrollable action $u \in \Sigma_u$ after a delay of d from state q.*

In the above definition, it is tacitly assumed that $\mu^u_q(d, u) > 0$ only if $q \xrightarrow{d,u}$ in $\mathcal{G}$. Also, we shall *wlog* for time-bounded reachability objectives assume that $\sum_u (\int_{t \geq 0} \mu^u_q(t, u) dt) = 1$[2]. In case the environment wants to perform an action deterministically after an exact delay d, μ^u_q will involve the use of Dirac delta function (see [15]).

The presence of the stochastic component μ^u makes a PTMDP a de facto infinite state Markov decision process. Here we seek strategies that will minimize the expected accumulated cost of reaching a given goal set G.

[2] For a time-bounded reachability objective (G, T), we may without affecting controllability assume that each location has each action (controllable or uncontrollable) action enabled after T.

Definition 5. *A* stochastic strategy μ^c *for a PTMDP* $\mathcal{M} = \langle \mathcal{G}, \mu^u \rangle$ *is a family of density-functions,* $\{\mu^c_q : \exists \ell \exists v. q = (\ell, v)\}$*, with* $\mu^c_q(d,c) \in \mathbb{R}_{\geq 0}$ *assigning the* density *of the controller aiming at taking the controllable action* $c \in \Sigma_c$ *after a delay of* d *from state* q*.*

Again it is tacitly assumed that $\mu^c_q(d,c) > 0$ only if $q \xrightarrow{d,c}$ in $\mathcal{G}$. Now, a PTMDP $\mathcal{M} = \langle \mathcal{G}, \mu^u \rangle$ and a stochastic strategy μ^c defines a race between the environment and the control strategy, where the outcome is settled by the two competing density-functions. More precisely, the combination of $\mathcal{M}$ and μ^c defines a probability measure $\mathbb{P}_{\mathcal{M},\mu^c}$ on (certain) sets of runs.

For $\ell_i \in L$ and $I_i = [l_i, u_i]$ with $l_i, u_i \in \mathbb{Q}$, $i = 0..n$, we denote the *cylinder set* by $\mathcal{C}(q, I_0 \ell_0 I_1 \cdots I_{n-1} \ell_n)$ consisting of all maximal runs having a prefix of the form: $q \xrightarrow{d_0} \xrightarrow{a_0} (\ell_1, v_1) \xrightarrow{d_1} \xrightarrow{a_1} \cdots \xrightarrow{d_{n-1}} \xrightarrow{a_{n-1}} (\ell_n, v_n)$ where $d_i \in I_i$ for all $i < n$. Providing the basis for a σ-algebra, we now inductively define the probability measure for such sets of runs[3]:

$$\mathbb{P}_{\langle \mathcal{G}, \mu^u \rangle, \mu^c}\big(\mathcal{C}(q, I_0 \ell_0 I_1 \cdots I_{n-1} \ell_n)\big) =$$
$$\sum_{p \in \{u,c\}} \sum_{\substack{a \in \Sigma_p \\ \ell_q \xrightarrow{a} \ell_1}} \int_{t \in I_0} \mu^p_q(t,a) \cdot \Big(\int_{\tau > t} \mu^{\overline{p}}_q(\tau) d\tau \Big) \cdot \mathbb{P}_{\langle \mathcal{G}, \mu^u \rangle, \mu^c}\big(\mathcal{C}((q^t)^a, \mathcal{C}(I_1 \cdots I_{n-1} \ell_n)\big) \, dt$$

The above definition requires a few words of explanation: the outermost sums divide into cases according to who wins the race of the first action (c or u), and which action a the winner will perform. Next, we integrate over all the legal delays the winner may choose according to the given interval I_0 using the relevant density-function. Independently, the non-winning player ($\overline{p}$) must choose a larger delay; hence the product of the probability that this will happen. Finally, the probability of runs according to the remaining cylinder $I_1 \ell_1, \cdots, I_{n-1} \ell_n$ from the new state $(q^t)^a$ is taken into account.

Now let $\pi \in \mathit{Exec}^m$ and let G be as set of goal locations. Then $C_G(\pi) = \mathit{min}\{C(\pi|_i) : \pi[i] \in G\}$ denotes the accumulated cost before π reaches G[4]. Now C_G is a random variable, which for a given stochastic strategy, μ^c, will have expected value $\mathbb{E}^{\mathcal{M}}_{\mu^c}(C_G)$ given by the Lesbegue integral $\int_{\pi \in \mathit{Exec}^m} C_G(\pi) \mathbb{P}_{\mathcal{M},\mu^c}(d\pi)$. Now, we want a (near-optimal) stochastic strategy μ^o that minimizes this expected value, subject to guaranteeing T as a worst-case reachability time-bound – or alternatively – subject to μ^o being a stochastic refinement ($\prec$[5]) of the most permissive time-bounded reachability strategy $\sigma^p(G,T)$ for $\mathcal{M}$. That is $\mathbb{E}^{\mathcal{M}}_T(C_G) = \inf \big\{ \mathbb{E}^{\mathcal{M}}_{\mu^c}(C_G) \mid \mu^c \prec \sigma^p(G,T) \big\}$. We note that letting μ^c range over deterministic strategies σ^d suffices in attaining $\mathbb{E}^{\mathcal{M}}_T(C_G)$.

[3] With the base case, e.g. $n = 0$, being 1.

[4] Note that $C_G(\pi)$ will be infinite in case π does not reach G. However, this case will never happen in our usages.

[5] $\mu^c \prec \sigma$ iff $\mu^c_q(d,a) > 0$ only if $\lambda \in \sigma(q^e)$ for all $e < d$ and $a \in \sigma(q^d)$.

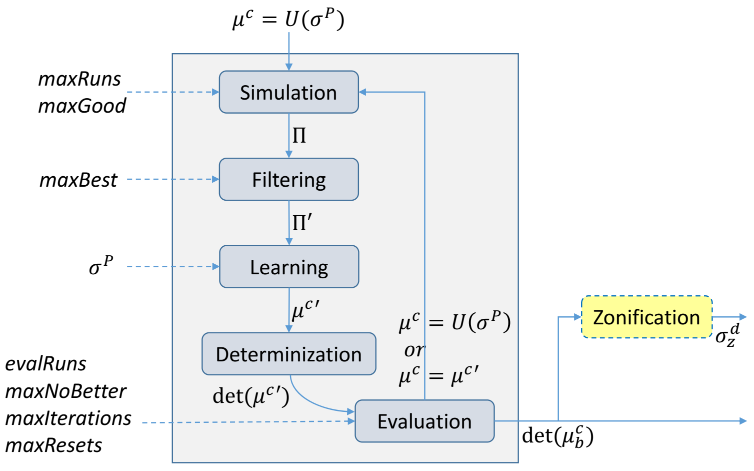

Fig. 2. Optimal scheduler approximation using reinforcement learning

3 Optimal Scheduler Approximation

Given a PTMDP $\mathcal{M}$ and a time-bounded reachability goal (G, T), we present a method for approximating $\mathbb{E}_T^{\mathcal{M}}(C_G)$ by computing a (deterministic) scheduler obtained using reinforced learning. In general, the technique of statistical model-checking (SMC) is used to generate runs according to a given stochastic semantics and then to analyze their outcomes w.r.t some property or expectation. In our context, given a PTMDP $\mathcal{M}$ and a stochastic control strategy μ^c, we use SMC to generate runs that are used both to estimate $\mathbb{E}_{\mu^c}^{\mathcal{M}}(C_G)$ and to iteratively improve μ^c towards μ^o. We combine the techniques of UPPAAL-TIGA to guarantee a given time-bound and UPPAAL SMC for expected-cost optimality. The core concept is similar to [20] but differs on the goal of the scheduler. We also differ on the termination criteria since our algorithm can reset itself to get out of local minima. We start with an overview of the procedure:

Figure 2 shows the general flow of the algorithm. The idea is to reinforce a current stochastic strategy representing distributions over controllable actions noted μ^c. This strategy is initialized to a uniform strategy noted $U(\sigma^P)$ based on a most permissive (zone-based) strategy σ^P obtained from UPPAAL-TIGA. This strategy allows all possible moves that still guarantee the controller to meet its goal within the given time-bound. The algorithm reinforces μ^c with $\mu^{c\prime}$ unless $\mu^{c\prime}$ is not improving too many times in a row, in which case it is reset to the initial uniform strategy. When it is improving and it is better than the currently *best* known strategy μ_b^c, then it replaces it. A strategy is better if it exhibits a lower expected cost.

The different steps are detailed as follows. First, the *simulation* step uses UPPAAL SMC to generate at most *maxGood good* runs that are used for learning. To do so, at most *maxRuns* runs in total are generated. The result is a set of runs

Π. It may happen that Π is empty, in which case μ^c is reset and the simulation is restarted. This is not depicted on the figure. Second, the set of runs Π is then *filtered* where at most *maxBest best* runs among those are kept. We retain the subset $\Pi' \subseteq \Pi$ of runs that have minimum cost. In practice this is done with a heap structure to keep a set of *maxBest* runs with their associated costs used for ordering the runs.

Then, central to the algorithm, comes the *learning* phase where the actual algorithm depends on the data structure used to represent μ^c. This phase computes a new $\mu^{c\prime}$ and we detail in the following section different ways to represent μ^c. Also, the strategy σ^P from Uppaal-Tiga is used here to ensure that any learned strategy still guarantees the required bound.

The resulting strategy is then *determinized* before being *evaluated.* This step uses Uppaal SMC again to evaluate the expected time (or cost) for the reinforced strategy $det(\mu^{c\prime})$ on a number of *evalRuns* runs. The resulting $\mu^{c\prime}$ may have a lower cost than μ^c, in which case we update μ^c (and possibly the best known strategy μ^c_b if it is better than this one too). However, if $\mu^{c\prime}$ is not better than μ^c *maxNoBetter* times then we reset it to $\mu^c = U(\sigma^P)$. This makes sure that the reinforcement learning does not get stuck into local minima. Finally, the algorithm loops at most *maxIteration* times if μ^c has been reset no more than *maxResets* times.

When the algorithm stops, the best known deterministic strategy $det(\mu^c_b)$ is outputted. It is then possible to "zonify" the strategy, meaning to approximate it with the zone-based representation σ^d_z used in Uppaal-Tiga, thus allowing for model checking of additional properties.

4 Strategies: Data Structures, Algorithms and Learning

Non-determistic Strategies. Crucial to our reinforcement learning algorithm Fig. 2 is the efficient representation and manipulation of control strategies. In Uppaal-Tiga, non-deterministic strategies are represented using zones, e.g. sets Z of valuations described by a guard in $\mathcal{B}(X)$. In a representation R, each location ℓ has an associated finite set $R_\ell = \{(Z_1, a_1), \ldots, (Z_k, a_k)\}$ of zone-action pairs, where $a_i \in \Sigma_c \cup \{\lambda\}$. Now R represents the strategy σ_R where $\sigma_R((\ell, v)) \ni a$ iff $(Z, a) \in R_\ell$ for some Z with $v \in Z$. In Uppaal-Tiga R is efficiently implemented as a hash-table with the location ℓ as key, and using difference bounded matrices (DBMs) [17] for representing zones.

For a non-determistic strategy σ and (l, v) a state, we write $(l, v) \xrightarrow{d}_\sigma$ to denote that σ allows a delay of d, i.e. for all $d' < d, \lambda \in \sigma(l, v + d')$. Similarly, we write $(l, v) \xrightarrow{c}_\sigma$ to denote that the controllable action c is allowed, i.e. $c \in \sigma(l, v)$. Uniformization and zonification are operations between non-deterministic and stochastic strategies. *Uniformization* is an operation that refines a non-deterministic strategy σ into a stochastic strategy μ^σ, subject to the condition that $\mu^\sigma_{(l,v)}(d, c) > 0$ if and only if $(l, v) \xrightarrow{d}_\sigma \xrightarrow{c}_\sigma$. Several implementations of uniformization may easily be obtained from the representation of a non-determistic strategy. Dually, *zonification* is an operation that abstracts a stochastic strategy μ into a non-determistic strategy σ_R, with a zone-based representation R, and subject to the condition that whenever $\mu_{(l,v)}(d,c) > 0$ then $(l,v) \xrightarrow{d}_{\sigma_R} \xrightarrow{c}_{\sigma_R}$.

Stochastic and Non-Lazy Strategies. For stochastic strategies, we shall in the following restrict our attention to so-called *non-lazy* strategies[6], μ^c, where the controller either urgently decides on an action, i.e. $\mu^c_q(d,a) = 0$ if $d > 0$, or prefer to wait until the environment makes a move, i.e. $\mu^c_{(\ell,v)}(d,a) = 0$ whenever $v(\mathtt{time}) + d \leq T$ with T being the time-bound of the reachability property in question. We shall use $\mathtt{w}$ to denote such an indefinite delay choice. Thus, for non-lazy stochastic strategies, the functionality may be recast as discrete probability distributions, i.e. $\mu^c_q : (\Sigma_c \cup \{\mathtt{w}\}) \rightarrow [0,1]$. In particular, we note that any non-lazy, stochastic strategy can trivially be transformed to a deterministic strategy by always selecting the action with the highest probability.

In the following we introduce three different data structuring and learning algorithms for stochastic strategies. Given that memoryless strategies suffices, we will learn a set of sub-strategies $\mu^c_\ell = \{\mu^c_q : \exists v.q = (\ell,v)\}$, where $\ell \in L$. The sub-strategies are then learned solely from a set of *(action,valuation)* pairs. Given a set of runs Π the relevant information for the sub-strategy μ^c_ℓ is given as In_ℓ:

$$In_\ell = \{(s_n,v) \in (\Sigma_c \cup \mathbb{R}) \times \mathbb{R}^X_{\geq 0} \mid (q_0 \xrightarrow{s_0}_{p_0} \ldots \xrightarrow{s_{n-1}}_{p_{n-1}} (\ell,v) \xrightarrow{s_n}_{p_n} \ldots) \in \Pi\}$$

Thus, in the following we only describe methods for learning sub-strategies.

Sample Mean and Covariance. For each controllable action c and location ℓ, we approximate the set of points representing clock valuations from which that action was successfully taken in ℓ by its sample mean and covariance matrix. Suppose we have N points corresponding to clock valuations $v_1,\ldots,v_N$. The sample mean vector $\overline{v}$ is the arithmetic mean, component-wise, for all the points: $\overline{v} = \frac{1}{N}\sum_{k=1}^{N} v_k$. The sample covariance matrix is defined as the square matrix $Q = [q_{ij}] = \frac{1}{N-1}\sum_{k=1}^{N}(v_k - \overline{v})(v_k - \overline{v})^T$.

Intuitively, if the sample covariance q_{ij} between two clocks x_i and x_j is positive, then bigger (resp. smaller) values of x_i correspond to bigger (resp. smaller) values of x_j. If it is negative, then the bigger (resp. smaller) values of x_i correspond to the smaller (resp. bigger) values of x_j. If it is zero then there is no such relation between the values of those two variables.

Note that the covariance matrix has size n^2 where n is the number of clocks but it is symmetric. Furthermore, for the matrix to be significant we need at least $n(n+1)/2$ sample points that correspond to the number of (potentially) different elements in the matrix, otherwise we default to using only the mean vector.

[6] In [22] it is shown that non-lazy strategies suffices for optimal scheduling of so-called DPAs.

Distribution. The purpose of this representation is to derive a distance from an arbitrary point to this "set" that is used to compute a weight for each controllable action. For a given valuation, such a distance $d(v)$ is evaluated as follows: $d(v)^2 = (u - \overline{v})^T Q^{-1}(u - \overline{v})$. If there are too few sample points then we default to using the Euclidian distance to the mean $\overline{v}$. The weight is then given by $w(v) = N \cdot e^{-d(v)}$. The weights for the different actions define a probability distribution.

Algorithm and Complexity. When generating runs using SMC, controllable actions are chosen according to the represented distribution that is initialized to be uniform. The time complexity is $O(n^2)$, n being the number of clocks. For the learning phase, the covariance matrix is computed using the filtered "best" samples. Then we need to invert it (once) before the next learning phase. The time complexity is $O(n^3)$. This is done for every action.

Logistic Regression. We consider a sub strategy μ_ℓ^c where the only options are either to take a transition (a) or wait until the environment takes a transition (w) (the case with more options is addressed later). The goal is to learn the weights $\beta_0, \beta_1, \ldots, \beta_{|X|} \in \mathbb{R}$ to use in the logistic function: Equation 4.

$$f(v) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \cdot v(x_1) + \cdots + \beta_{|X|} \cdot v(x_{|X|}))}},$$

where $x_1, \ldots, x_{|X|} \in X$. This function, combined with the learned weights $\beta_0, \beta_1, \ldots, \beta_{|X|}$, defines a stochastic sub-strategy s.t. $\mu_{(\ell,v)}^c(a) = f(v)$ and $\mu_{(\ell,v)}^c(\mathtt{w}) = 1 - f(v)$. Using Figure 3 we here give an intuition on how, given an input set In_ℓ, we learn the weights $\beta_0, \ldots, \beta_{|X|}$ (for details, see [18]). We assume that there exists only two options (a and $\mathtt{w}$) in the location ℓ, and (for simplicity and *wlog*) a single clock in the system. For each input $(s_n, v) \in In_\ell$:

– If $s_n = a$, construct a point at $(v(x), 1)$ where $x \in X$ is the clock. These are the triangles in Figure 3.
– Otherwise, construct a point at $(v(x), 0)$ where $x \in X$ is the clock. These are the circles in Figure 3.

We use L1-regularized logistic regression provided by LIBLINEAR [18] for fitting the function to the constructed points. The output of this process is the weights $\beta_0, \beta_1, \ldots, \beta_{|X|}$ and the result is shown in Figure 3. In the case of more than two options (e.g. if we also had an action b) we use the one-versus-all method. This method learns a function for each action[7].

Complexity. The complexity of fitting the points using this method is $O(|In_\ell| + i)$ [29], where i is the number of iterations before the fitting algorithm converges thus for multiple actions, the complexity for learning is $O(c \cdot (|In_\ell| + i))$ where c is the number of options. We need to store $c \cdot |X|$ weights per location, this is the space complexity.

[7] If e.g. we have three actions, a, b and w, we will learn three functions, one which is a versus b and w, one which is b versus a and w, and one which is w versus a and b.

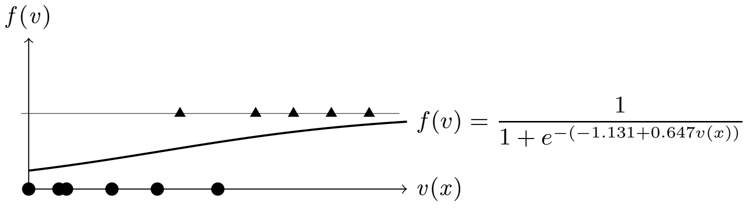

Fig. 3. Example of logistic regression with one clock x and two options a and w. For valuation v, $f(v)$ gives the probability of selecting action a (triangle) and $1-f(v)$ gives the probability of selecting action w (circle). The probabilities are equal at $v(x) = 1.747$ because $f(0.5) = 1.747$.

Splitting. Here we represent a sub-strategy as a binary tree, where an internal node is a four-tuple $(x, s, low, high)$, where *low* and *high* are either internal nodes or leaf nodes, $x \in X$ is the clock we split on and $s \in \mathbb{R}_{>0}$ is the discriminating value for the clock. A leaf node is a function W mapping actions, a, from $\Sigma_c \cup \{\mathtt{w}\}$ to weights, $W(a) \in \mathbb{R}_{>0}$. Figure 4 shows an example of a tree with a splitting for the clock x at value 2. For a given clock valuation v, the tree is traversed to the leaf node W to which it "belongs", with W represented by the pairs $(a, W(a))$ with $W(a) > 0$. This defines a stochastic sub-strategy μ^c_ℓ s.t. $\mu^c_{(\ell,v)}(a) = W(a)/\sum_{b\in\Sigma_c\cup\{\mathtt{w}\}} W(b)$ for all $a \in \Sigma_c \cup \{\mathtt{w}\}$. Initially, the tree consists of only a single leaf node assigning weight 1 to all actions. In each iteration of the learning algorithm presented in Section 3, a percentage of the leaf nodes are split on one clock according to the following algorithm:

1. *Select nodes to split.* Given a set In_ℓ, count how many of these did not perform the action with the highest weight in the corresponding leaf node. The leaf nodes over all locations with the highest counts are chosen for splitting, the remaining have their weights updated using reinforced learning.
2. *Select clock to split on.* For each node to split:
   (a) Let In^n_l be the runs from In_l which satisfy the constraints of the tree, to this leaf node.
   (b) For every clock $x \in X$:
      i. Find the minimum and maximum $v(x)$ where $(s_n, v) \in In^n_l$ and call the average of these two s.
      ii. In the set $\{(s_n, v) \in In^n_\ell \mid v(x) \le s\}$ count for each $a \in \Sigma_c \cup \{\mathtt{w}\}$ how many runs choose a. Insert these points in a vector. Make a corresponding vector for the set $\{(s_n, v) \in In^n_\ell \mid v(x) > s\}$.
      iii. Compute Euclidean distance of the vectors.
3. *Update tree.* For clock x and split s with largest Euclidian distance, split the leaf node by replacing the node itself with internal node $(x, s, low, high)$ where *low* and *high* are new leaf nodes. Compute weights using reinforced learning for the new leaf nodes.

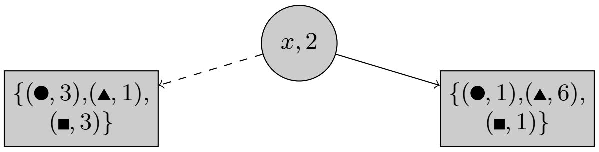
$x, 2$
$\{(\bullet,3),(\blacktriangle,1),(\blacksquare,3)\}$
$\{(\bullet,1),(\blacktriangle,6),(\blacksquare,1)\}$

Fig. 4. A binary tree with a splitting on clock x and value 2

Complexity. The complexity of step 1 is $O(nl \cdot |\Pi|)$ where nl is the current number of leafs in the tree, and Π is the number of good runs in the batch. The complexity of step 2 is $O(nl \cdot |X| \cdot |\Pi|)$ and the complexity of step 3 is $O(|\Pi|)$. This means the complete complexity of the whole learning is $O(nl \cdot |X| \cdot |\Pi|)$. In step 1 we select a percentage of the clocks to split on, thus the space complexity of this method is exponential in the worst case, however we bound this by not allowing splitting a leaf on a clock with a range shorter than a predefined (small) constant.

5 Experiments

We now present experiments for evaluating the algorithms proposed in Section 4. Table 1 shows a selection of results. The results are elaborated in the following sections. The full set of results will be available in a full version of this paper to be found on arXiv.org of Cornell University. In each of the experiments, we use 2000 runs pr. iteration. We have evaluated the learned strategies using Uppaal SMC, also with 2000 runs.

Small Examples. The first row in Table 1 shows the statistics for the motivational example in Figure 1, for obtaining the optimal strategy w.r.t. expected cost under the constraint that `END` is guaranteed to be reached within 210 time units. Recall from Section 1 that the optimal expected cost for this case is 204. An optimal strategy according to SMC is found using all the methods, achieving an expected cost very close to the optimal expected cost.

The second row shows the statistics for a Go-To-Work example from [14]. We first use Uppaal-Tiga to ensure that you are guaranteed to get to work in 60 time units and then minimize the expected *time* for going to work under this constraint. The three different methods find strategies with the same expected time, the splitting is the fastest and uses slightly less memory than the others.

Duration Probabilistic Automata. In Section 1, we initially mentioned a type of scheduling problems where tasks have uncertainty in execution time and depends no an amount of resources. This class of job-shop-like scheduling problems can be represented as Duration Probabilistic Automata (DPAs) [22]. A DPA is a multiset of resources and a set of Simple Duration Probabilistic Automata (SDPAs). An SDPA is a series of tasks which cannot be executed in parallel, be preempted and must be executed according to some order. Each task requires a multiset of resources and has a uniformly distributed duration with a

Table 1. Computed expected cost (1^{st} row) and time/memory consumption for synthesis (2^{nd} and 3^{rd} rows) for various strategies. 4^{th} row shows resets/iterations to find optimal strategy. For uniform strategies, only the computed expected cost is reported. Time/memory consumption for computing exact optimal strategies by Kempf et al. [22] reported where available.

Model	**Uniform**	**Co-variance**	**Splitting**	**Regression**	**Exact [22]**
Motivational	404.299	205.169	206.328	203.721	
example		33.78s	30.55s	42.61s	
		8.17MB	15.79MB	8.46MB	
		2/82	4/109	1/52	
	38.64	32.69	32.81	32.81	
GoWork		9.8s	24.41s	14.18s	
		7.96MB	8.38MB	8.08MB	
		5/72	0/10	4/71	
	18.08	17.53	17.34	17.38	
p0s3p1s4_4		28.84s	27.06s	51.92s	1062.77s
		8.75MB	17.25MB	8.88MB	145.47MB
		3/54	3/40	2/58	
	18.56	17.75	16.90	16.83	
p0s3p1s4_16		36.67s	46.13s	69.51s	176.15s
		8.63MB	14.38MB	9.07MB	35.60MB
		6/119	0/22	0/27	
	19.90	19.27	19.21	19.27	
p0s4p1s4_5		45.91s	47.1s	60.11s	8547.52s
		9.03MB	26.11MB	9.23MB	486.92MB
		1/31	3/86	5/115	
	3946.76	2213.35	2303.47	2218.34	
ran-4-3		196.19s	242.7s	330.64s	
		19.65MB	124.06MB	20.28MB	
		2/59	1/37	3/141	
	8068.02	4111.77	4221.75	3641.61	
ran-4-4		323.35s	281.58s	459.08s	
		34.46MB	167.66MB	28.88MB	
		5/190	6/159	5/194	
	3965.73	2765.19	2780.45	2765.77	
tiga-ran-4-3		230.06s	351.45s	337.39s	
		17.44MB	127.10MB	25.23MB	
		4/88	5/165	6/164	
	8058.78	6343.45	6307.29	6358.3	
tiga-ran-4-4		262.09s	323.7s	270.93s	
		26.37MB	170.63MB	20.76MB	
		2/32	5/121	2/30	

given time-interval. The DPA scheduling problem is now: given a configuration of an DPA, which SDPA should be allocated which resources at what time in order to minimize expected completion time? For this scheduling problem, Kempf et al. [22] provide a method for exact optimal scheduler synthesis under uncertainty. We will use their results as a benchmark. Poulsen et al. [28] provides a method for translating DPA into TAs with a uniform scheduler, it is trivial to adapt this for translating DPAs into PTMPDs.

Qualitative Comparison. Rows 3-5 show the statistics of obtaining the optimal strategy for three small models (2 processes, 2 tasks) from Kemp et al. [22][8] using our approach. If we examine the synthesized strategies, we see that these are in fact the same strategies as those found using the exact computation (with decision boundaries located within 0.5 time units of the optimal boundaries). We also observe that even though an optimal strategy is found, this does not affect the expected time significantly. Furthermore, the exact computation of [22] for these models took between 176 – 8547 seconds, whereas our method synthesize optimal strategies in less than 70 seconds in each case, thus an order of magnitude faster and less memory-consuming.

Scalability Comparison. For testing the method on larger models, we randomly generate a number of larger DPAs (3-5 processes, 3-5 tasks). On two of these DPAs we first ran Uppaal-Tiga to find the best possible worst case time guarantee (rows $8, 9$ in Table 1). Then on all four (rows $6 - 9$ in Table 1) we ran the three different methods to synthesize a strategy with near-minimal expected completion-cost subject to the possible time-bound guarantee.

Generally we observe, that it is specific to the example which method performs the best. We observe that all three methods are able to learn a strategy which is significantly better than the uniformly random strategy. We see that in the examples, where we constrain the allowed strategies using Uppaal-Tiga only little overhead is incurred, but the strategy we learn does not improve as much as otherwise. This is natural as Uppaal-Tiga constrain the choices we are allowed to learn.

6 Conclusion and Future Works

In this paper, we have presented a new technique that combines classical controller synthesis with reinforcement learning to compute strategies that provide near optimal expected cost *and* time-bound guarantees. Our experiments show very good results on the class of DPA models. The framework presented is general and not limited to neither DPAs or PTMDPs. In particular, if a time-bound is not required then we can omit the Uppaal-Tiga step and apply our technique to hybrid MDPs by utilizing Uppaal SMC's support for stochastic hybrid systems [15]. Future works include developing suitable data structures for more general classes of strategies than non-lazy ones. Though our method is guaranteed to converge to the optimal strategy (under the assumption of non-lazyness), it would be usefull if an estimation of how close a given proposed solution is to the optimal one. However, given the difficulty in obtaining such error-bounds in much simpler cases such as job-shop and task-graph scheduling [2,1], we believe that this will be very difficult.

[8] Models and results available at `http://www-verimag.imag.fr/PROJECTS/TEMPO/DATA/201304_dpa/`

References

1. Abdeddaïm, Y., Kerbaa, A., Maler, O.: Task graph scheduling using timed automata. In: IPDPS, p. 237. IEEE Computer Society (2003)
2. Abdeddaïm, Y., Maler, O.: Job-shop scheduling using timed automata. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 478–492. Springer, Heidelberg (2001)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
5. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
6. Berendsen, J., Chen, T., Jansen, D.N.: Undecidability of cost-bounded reachability in priced probabilistic timed automata. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 128–137. Springer, Heidelberg (2009)
7. Berendsen, J., Jansen, D.N., Vaandrager, F.W.: Fortuna: Model checking priced probabilistic timed automata. In: QEST, pp. 273–281. IEEE Computer Society (2010)
8. Bertrand, N., Schewe, S.: Playing optimally on timed automata with random delays. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 43–58. Springer, Heidelberg (2012)
9. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for modestly nondeterministic models. In: Schmitt, J.B. (ed.) MMB & DFT 2012. LNCS, vol. 7201, pp. 249–252. Springer, Heidelberg (2012)
10. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.G.: Synthesis of optimal strategies using hytech. Electr. Notes Theor. Comput. Sci. 119(1), 11–31 (2005)
11. Bouyer, P., Larsen, K.G., Markey, N., Rasmussen, J.I.: Almost optimal strategies in one clock priced timed games. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 345–356. Springer, Heidelberg (2006)
12. Brázdil, T., Krcál, J., Kretínský, J., Kucera, A., Rehák, V.: Measuring performance of continuous-time stochastic processes using timed automata. In: Caccamo, M., Frazzoli, E., Grosu, R. (eds.) HSCC, pp. 33–42. ACM (2011)
13. Brihaye, T., Bruyère, V., Raskin, J.F.: On optimal timed strategies. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 49–64. Springer, Heidelberg (2005)
14. Bruyère, V., Filiot, E., Randour, M., Raskin, J.F.: Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In: Mayr, E.W., Portier, N. (eds.) STACS. LIPIcs, vol. 25, pp. 199–213. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014)
15. David, A., Du, D., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. In: Bartocci, E., Bortolussi, L. (eds.) HSB. EPTCS, vol. 92, pp. 122–136 (2012)
16. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011)

17. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
18. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)
19. Fu, H.: Maximal cost-bounded reachability probability on continuous-time markov decision processes. In: Muscholl, A. (ed.) FOSSACS 2014. LNCS, vol. 8412, pp. 73–87. Springer, Heidelberg (2014)
20. Henriques, D., Martins, J., Zuliani, P., Platzer, A., Clarke, E.: Statistical model checking for markov decision processes. In: 2012 Ninth International Conference on Quantitative Evaluation of Systems (QEST), pp. 84–93 (2012)
21. Jensen, H.E., Gregersen, H.: Model checking probabilistic real time systems. Nordic Workshop on Programming Theory 7, 247–261 (1996)
22. Kempf, J.F., Bozga, M., Maler, O.: As soon as probable: Optimal scheduling under stochastic uncertainty. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 385–400. Springer, Heidelberg (2013)
23. Kwiatkowska, M.Z., Norman, G., Parker, D.: Probabilistic symbolic model checking with prism: A hybrid approach. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 52–66. Springer, Heidelberg (2002)
24. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Verifying quantitative properties of continuous probabilistic timed automata. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 123–137. Springer, Heidelberg (2000)
25. Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large markov decision processes. In: Ossowski, S., Lecca, P. (eds.) SAC, pp. 1314–1319. ACM (2012)
26. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010)
27. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
28. Poulsen, D.B., van Vliet, J.: Duration probabilistic automata, http://www.cs.aau.dk/~adavid/smc/DurationProbabilisticAutomata.pdf
29. Yuan, G.X., Ho, C.H., Lin, C.J.: An improved glmnet for l1-regularized logistic regression. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2011, pp. 33–41. ACM, New York (2011), http://doi.acm.org/10.1145/2020408.2020421