Teaching Stratego to Play Ball : Optimal Synthesis for Continuous Space MDPs

Manfred Jaeger[1], Peter Gjøl Jensen[1], Kim Guldstrand Larsen[1], Axel Legay[1,2], Sean Sedwards[3], and Jakob Haahr Taankvist[1]

[1] Department of Computer Science, Aalborg University, Denmark
[2] Université catholique de Louvain, Belgium
[3] University of Waterloo, Canada

Abstract. Uppaal Stratego facilitates optimization of quantitative measures on complex stochastic timed systems. In this paper we propose alternatives to the optimization algorithms of Uppaal Stratego, demonstrating that a significant improvement can be achieved in terms of convergence tendencies. In particular, we propose two online learning algorithms, using online partition refinement techniques, and argue for its theoretical convergence. We have implemented the proposed algorithms in Uppaal Stratego and support our claims with experiments on a range of models. We also provide the core algorithms as an Open Source library under the permissive LGPL license.

1 Introduction

Machine learning and artificial intelligence have become standard methods for controlling complex systems. For systems represented as (Hybrid) Priced Timed Markov Decision Processes (PTMDP), Uppaal Stratego [5] has to some degree demonstrated that near-*optimal* strategies can be learned [10, 12, 4]. However, as we shall demonstrate in this paper, for the rich class of PTMDPs we can still improve significantly on the existing learning techniques in terms of learning better strategies, and obtaining better convergence characteristics as a function of the data size.

In most machine learning applications one learns from real-world data that is provided in batch, or as a data stream. However, for many cyber-physical systems it is impossible to obtain sufficient data from the real-world system. On the other hand, accurate formal models of the systems may be available. We therefore base the controller synthesis on data generated from a model of the system, an approach we refer to as *in-silico* synthesis.

We extend the method of David et. al [5, 4] by developing an *online* learning method based on reinforcement learning principles. Since we are dealing with systems operating in hybrid discrete-continuous state spaces, a suitable approach to deal with continuous state spaces in a reinforcement learning setting needs to be chosen. Traditional (linear) function approximations of value functions [17] are not expressive enough to deal with the often highly non-linear, multi-modal

value functions we encounter in typical cyber-physical system models. Recent developments in which neural network representations of non-linear state-value functions are learned [14] have the drawback that the learned strategy faces similar drawbacks as any neural network in terms of verification [8], and the lack of safety guarantees. While we do not directly address safety of learned strategies in this paper, we note that the simple partition-based function approximations that we will employ are not only highly flexible regarding the types of functions that can be approximated, but also closely aligned with continuous-time model-checking techniques such as *regions* and *zones*.

In order to obtain an online method whose computational efficiency does not deteriorate over time due to the accumulation of training data, we need to base our approach on fixed-size running summaries of the data, specifically online standard deviations and running means [16]. In this paper we develop two versions of our basic partition refinement approach: one based on *Q-learning*[18] and one related to *Real Time Dynamic Programming*[15] (RTDP). While Q-learning is a so-called model-free learning-method, RTDP is a model-centric learning-method and we will thus denote the latter *M-learning*.

To demonstrate our approach we have implemented the proposed algorithms in Uppaal Stratego and investigated their performance experimentally. We also provide an Open Source `C++` implementation of the algorithms[4].

Related Work: In the area of synthesis for reactive systems, David et al. [4, 5] extended the work of Henriques et. al. [7] to continuous space systems. In the work of Henriques et al. the problem of optimization is seen as one of classification. As we shall later demonstrate experimentally, examples exist where these algorithms are not converging towards optimal controllers.

Several abstraction and partition refinement-based methods have been applied to speed up verification of finite state MDPs [2, 9, 13], utilizing underlying symbolic techniques. We distinguish ourselves from this approach by considering a richer class of MDPs, while doing an empirical partition refinement online, based on statistical tests.

The partition refinement scheme we deploy can be seen as a classical regression or classification problem. As such, alternative approaches could be based on classical decision and regression trees [1]. However, these often require a batch of data samples to be kept in memory and are thus incompatible with our online scheme.

2 Euclidean MDP and Expected Cost

In this section we introduce our formal system model and controller synthesis objective.

Definition 1 (K-Dimensional, Euclidean) Markov Decision Processes). *A MDP is a tuple $\mathcal{M} = (\mathcal{S}, Act, s_{init}, T, \mathcal{C}, \mathcal{G})$ where:*

[4] Available at `https://github.com/petergjoel/libprlearn`

- $\mathcal{S} \subseteq \mathbb{R}^K$ *is a bounded and closed subset of the Euclidean space,*
- Act *is a finite set of actions,*
- $s_{init} \in \mathcal{S}$ *is the initial state,*
- $T : \mathcal{S} \times Act \to \Delta$*, where* $\Delta : \mathcal{S} \to \mathbb{R}_{\geq 0}$*, is the transition function that, for each* $s \in \mathcal{S}$ *and* $\alpha \in Act$*, yields a probability density function over* $\mathcal{S}$ *e.g.* $\int_{s \in \mathcal{S}} \Delta(s)ds = 1$*,*
- $\mathcal{C} : \mathcal{S} \times Act \times \mathcal{S} \to \mathbb{R}$ *is a cost-function for state-action-state triples,*
- $\mathcal{G} \subseteq \mathcal{S}$ *is the set of goal states.*

A run π of an MDP is a sequence of alternating states and actions $s_1 a_1 s_2 a_2 \cdots$. We denote the set of all runs of an MDP $\mathcal{M}$ as $\Pi_{\mathcal{M}}$, and all finite runs of $\mathcal{M}$ as $\Pi^f_{\mathcal{M}}$. We use the notation $\pi|_i$ to denote the prefix of the run up to s_i, i.e. $\pi|_i = s_1 a_1 s_2 a_2 \cdots a_{i-1} s_i$. We denote the length of a trace $\pi = s_1, s_2 \cdots s_n$ as $|\pi| = n$. We let ϵ denote the empty run and by agreement let $|\epsilon| = 0$. To define the cost of a run $\pi = s_1 a_1 s_2 a_2 \cdots \in \Pi_{\mathcal{M}}$ with respect to a set of goal states $\mathcal{G}$ let $s_{i_{\min}}$ be the first state in π which is included in $\mathcal{G}$, i.e. $i_{min} = \min_{i \in \mathbb{N}}\{s_i \in \pi \mid s_i \in \mathcal{G}\}$. Then the cost of π up to $\mathcal{G}$ is the sum of costs until $i_{\min}$, defined as the random variable $\mathcal{C}_{\mathcal{G}}$:

$$\mathcal{C}_{\mathcal{G}}(\pi) = \sum_{s_i a_i s_{i+1} \in \pi|_{i_{\min}}} \mathcal{C}(s_i, a_i, s_{i+1}).$$

Definition 2 (Strategy). *A (memoryless) strategy for an MDP* $\mathcal{M}$ *is a function* $\sigma : \mathcal{S} \to (Act \to [0, 1])$*, mapping a state to a probability distribution over* Act*.*

Given a strategy σ, the expected costs of reaching a goal state is defined as the solution to a Voltaire integral equation as follows:

Definition 3 (Expected cost of a strategy). *Let* $\mathcal{G}$ *be a set of goal-states and let* σ *be a strategy. The expected cost of reaching* $\mathcal{G}$ *starting in a state* s *–* $\mathbb{E}^{\mathcal{M}}_{\sigma}(\mathcal{C}_{\mathcal{G}}, s)$ *– is the solution to the following system of equations*[5]*:*

$$\mathbb{E}^{\mathcal{M}}_{\sigma}(\mathcal{C}_{\mathcal{G}}, s) = \sum_{a \in Act} \sigma(s)(a) \cdot \int_{t \in \mathcal{S}} T(s, a)(t) \cdot \big(\mathcal{C}(s, a, t) + \mathbb{E}^{\mathcal{M}}_{\sigma}(\mathcal{C}_{\mathcal{G}}, t)\big)dt$$

when $s \notin \mathcal{G}$ *and* $\mathbb{E}^{\mathcal{M}}_{\sigma}(\mathcal{C}_{\mathcal{G}}, s) = 0$ *when* $s \in \mathcal{G}$*.*

The synthesis problem is now to find the strategy σ which minimizes $\mathbb{E}^{\mathcal{M}}_{\sigma}(\mathcal{C}_{\mathcal{G}})$.

Example 1. Consider the MDP in Fig. 1, described as a stochastic priced timed game [3] over the clock x and the cost-variable C. Here the state space $\mathcal{S}$ is given by the values of the clock x, i.e. $\mathcal{S} = [0, 10]$. The action set is $Act = \{a, b\}$.

The goal set $\mathcal{G}$ is $]9, 10]$. In the (urgent) location ℓ_0 the (probabilistic) choice between the actions a or b is made.

[5] We shall assume that the equation system has a solution for the considered MDP and goal set.

Note that for $x > 9$, neither a nor b affects the state. In the case $x \leq 9$ choosing a will reset x (at no cost), whereas b will increase the cost by 15. In both cases – now at location ℓ_1 – the new state (i.e. the new value of the clock x) will be chosen uniformly from the current value of x to 10. Thus, $T(x,a)(y) = \frac{1}{10}$ for all $y \in [0,10]$, and $T(x,b)(y) = 0$ whenever $y < x$ and $T(x,b)(y) = \frac{1}{10-x}$ when $y \geq x$. Furthermore, $\mathcal{C}(x,a,y) = 2y$ for $y \in [0,10]$, and $\mathcal{C}(x,b,y) = 2(y-x)+15$ for $y \geq x$.

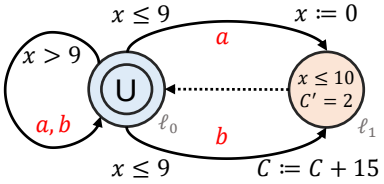

Fig. 1: MDP described as a stochastic priced timed game.

Let σ_u be the random strategy choosing uniformly between a and b for any value of x, i.e. $\sigma_u(x)(a) = \sigma_u(x)(b) = \frac{1}{2}$ for any $x \in [0,10]$. Then the expected costs $\mathbb{E}_{\sigma_u}(x)$ of reaching $\mathcal{G}$ from state x satisfy the following Voltaire integral equations system

$$\mathbb{E}_{\sigma_u}(x) = \frac{1}{2} \cdot \int_{y=0}^{10} \frac{2y + \mathbb{E}_{\sigma_u}(y)}{10} dy + \frac{1}{2} \cdot \int_{y=x}^{10} \frac{15 + 2(y-x) + \mathbb{E}_{\sigma_u}(y)}{10 - x} dy$$

when $x \leq 9$ and $\mathbb{E}_{\sigma_u}(x) = 0$ when $x > 9$. Using statistical model checking [6], we find that $\mathbb{E}_{\sigma_u}(\mathcal{G}) \approx 81.73$. Considering the deterministic strategy σ_7 with $\sigma_7(x)(a) = 1$ for $x \leq 7$ and $\sigma_7(x)(b) = 1$ for $x > 7$, we find that $\mathbb{E}_{\sigma_7}(\mathcal{G}) \approx 51.91$, thus providing an improvement over the random strategy. Using the learning methods presented in the next sections, we find that the optimal strategy σ^o has $\mathbb{E}_{\sigma^o}(\mathcal{G}) \approx 48.16$ and chooses a when $x \leq 4.8$ and b otherwise.

3 Approximation by Partitioning

The expected cost of a strategy is the solution to a Voltaire integral equation. Such equations are notoriously hard to solve, even in the case of stochastic priced timed games, as in Example 1.In this paper we offer a solution method combining reinforcement learning with an online partition refinement scheme.

First let us formally introduce the notion of a *partition* and see how it may be used symbolically to approximate the optimal expected cost. We say that $\mathcal{A} \subseteq 2^{\mathcal{S}}$ is a partition of $\mathcal{S}$ if $\mathcal{S} = \bigcup_{\nu \in \mathcal{A}} \nu$ and for any $\nu, \nu' \in \mathcal{A}$ we have $\nu \cap \nu' = \emptyset$ whenever $\nu \neq \nu'$. We call an element ν of $\mathcal{A}$ a *region* and shall assume that each such ν is Borel measurable (e.g. a k-dimensional interval). Whenever $s \in \mathcal{S}$ we denote by $[s]_{\mathcal{A}}$ the unique region $\nu \in \mathcal{A}$ such that $s \in \nu$. For $\delta \in \mathbb{R}_{>0}$ we shall say that that $\mathcal{A}$ has granularity δ if $diam(\nu) \leq \delta$ for any region $\nu \in \mathcal{A}$. We say that a partition $\mathcal{B}$ refines a partition $\mathcal{A}$ if for any $\nu \in \mathcal{B}$ there exist $\mu \in \mathcal{A}$ with $\nu \subseteq \mu$. We write $\mathcal{A} \sqsubseteq \mathcal{B}$ in this case.

Given an MDP, a partition $\mathcal{A}$ of its state space induces an abstracting Markov Interval Decision Process (MIDP) [9, 13] as follows:

Definition 4. *Let $\mathcal{M} = (\mathcal{S}, Act, s_{init}, T, \mathcal{C}, \mathcal{G})$ be an MDP, and let $\mathcal{A}$ be a finite partition of $\mathcal{S}$ consistent with $\mathcal{G}$[6]. Then $\mathcal{M}_{\mathcal{A}} = (\mathcal{S}_{\mathcal{A}}, Act, \nu_{init}, T_{\mathcal{A}}, \mathcal{C}_{\mathcal{A}}, \mathcal{G}_{\mathcal{A}})$ is the MIDP s.t.:*

- $\mathcal{S}_{\mathcal{A}} = \mathcal{A}$,
- $\nu_{init} = [s_{init}]_{\mathcal{A}}$ *is the initial state,*
- $\nu \in \mathcal{G}_{\mathcal{A}}$ *if* $\nu \subseteq \mathcal{G}$,
- $T_{\mathcal{A}} : \mathcal{A} \times Act \to (\mathcal{A} \to [0,1] \times [0,1])$ *is the transition-function,*
- $\mathcal{C}_{\mathcal{A}} : \mathcal{A} \times Act \times \mathcal{A} \to \mathbb{R} \times \mathbb{R}$ *is the transition cost-function*

where

$$T_{\mathcal{A}}(\nu, a)(\nu') = \left(\inf_{s \in \nu} \int_{\nu'} T(s,a)(t)dt,\ \sup_{s \in \nu} \int_{\nu'} T(s,a)(t)dt \right)$$

and

$$\mathcal{C}_{\mathcal{A}}(\nu, a, \nu') = \left(\inf_{s \in \nu, s' \in \nu'} \mathcal{C}(s,a,s'),\ \sup_{s \in \nu, s' \in \nu'} \mathcal{C}(s,a,s') \right)$$

For a region ν and an action a, let $\Delta(T_{\mathcal{A}}(\nu, a))$ be the set of probability functions over $\mathcal{A}$ that are consistent with $T_{\mathcal{A}}(\nu, a)$, i.e. if $p \in \Delta(T_{\mathcal{A}}(\nu, a))$ then $\sum_{\nu' \in \mathcal{A}} p(\nu') = 1$ and $p(\nu') \in T_{\mathcal{A}}(\nu, a)(\nu')$ for all ν'.

We can now define the *lower* expected costs from partitions $\nu \in \mathcal{A}$ to $\mathcal{G}$ as the least solution to the following (finite) system of equations[7]:

$$\mathbb{E}^{\min}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, \nu) = \min_{a \in Act} \inf_{p \in \Delta(T_{\mathcal{A}}(\nu,a))} \sum_{\nu' \in \mathcal{A}} p(\nu') \cdot \left(\mathcal{C}^{\inf}_{\mathcal{A}}(\nu, a, \nu') + \mathbb{E}^{\min}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, \nu') \right)$$

when $\nu \cap \mathcal{G} = \emptyset$ and $\mathbb{E}^{\min}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, \nu) = 0$ when $\nu \subseteq \mathcal{G}$. Similarly we can define the *upper* expected costs $\mathbb{E}^{\max}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, \nu)$ (simply replace the occurrences of inf with sup in the above equation). The following Theorem shows their importance for approximating the expected cost of the optimal strategy.

Theorem 1. *Let $\mathcal{M} = (\mathcal{S}, Act, s_{init}, T, \mathcal{C}, \mathcal{G})$ be an MDP and let $\mathcal{A}$ be a finite partition of $\mathcal{S}$ consistent with $\mathcal{G}$. Then for all $s \in \mathcal{S}$:*

$$\mathbb{E}^{\min}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, [s]_{\mathcal{A}}) \leq \inf_{\sigma} \mathbb{E}^{\mathcal{M}}_{\sigma}(\mathcal{C}, s) \leq \mathbb{E}^{\max}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, [s]_{\mathcal{A}})$$

We also note that, whenever $\mathcal{A} \sqsubseteq \mathcal{B}$, then $\mathcal{B}$ offers a (possible) better approximation than $\mathcal{A}$ in the sense that $\mathbb{E}^{\min}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, [s]_{\mathcal{A}}) \leq \mathbb{E}^{\min}_{\mathcal{M},\mathcal{B}}(\mathcal{C}_{\mathcal{A}}, [s]_{\mathcal{B}}) \leq \mathbb{E}^{\max}_{\mathcal{M},\mathcal{B}}(\mathcal{C}_{\mathcal{A}}, [s]_{\mathcal{B}}) \leq \mathbb{E}^{\max}_{\mathcal{M},\mathcal{B}}(\mathcal{C}_{\mathcal{A}}, [s]_{\mathcal{A}})$ for any $s \in \mathcal{S}$.

Example 1. Reconsider the MDP $\mathcal{M}$ from Fig. 1. Now consider the partition $\mathcal{A}$ of the state-space $[0, 10]$ given by the three parts $[0, 5]$, $]5, 9]$ and $]9, 10]$. Note that this partition is consistent with the goal set $]9, 10]$. The finite-state Markov Interval Decision Process in Fig. 2 provides the abstraction $\mathcal{M}_{\mathcal{A}}$. From this it may be found that $\mathbb{E}^{\min}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}}, [0, 5]) = 23.6$ obtained by the strategy $\sigma_{\min}$ with

[6] $\mathcal{A}$ is consistent with $\mathcal{G}$ if for any $\nu \in \mathcal{A}$ either $\nu \subseteq \mathcal{G}$ of $\nu \cap \mathcal{G} = \emptyset$.

[7] Here $\mathcal{C}^{\inf}_{\mathcal{A}}(\nu, a, \nu') = \inf_{s \in \nu, s' \in \nu'} \mathcal{C}(s, a, s')$.

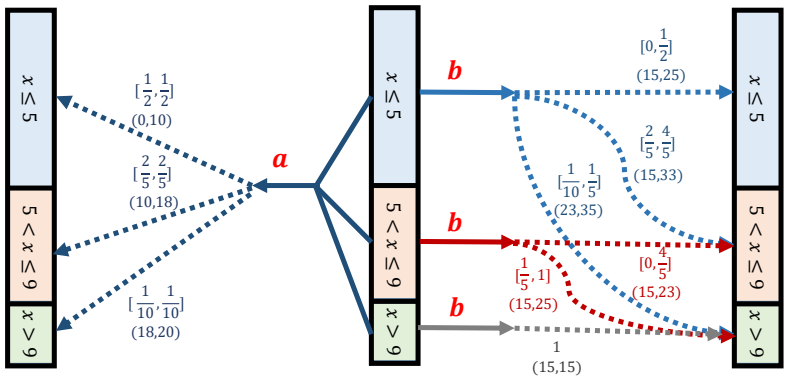

Fig. 2: Markov Interval Decision Process abstracting Fig. 1.

$\sigma_{\min}([0,5]) = a$ and $\sigma_{\min}(]5,9]) = b$. Similarly $\mathbb{E}^{\max}_{\mathcal{M},\mathcal{A}}(\mathcal{C}_{\mathcal{A}},[0,5]) = 152$ obtained by the strategy $\sigma_{\max}$ with $\sigma_{\max}([0,5]) = \sigma_{\max}(]5,9]) = b$.

A natural question to ask now is: *under which conditions may the optimal expected cost be approximated arbitrarily closely by successively refining partitions?* As we shall detail now, convergence is guaranteed for bounded horizon MDPs with continuous transition and cost functions. Luckily, this class covers all the MDPs that we consider in the evaluation of our new learning algorithms.

For an MDP $\mathcal{M} = (\mathcal{S}, Act, s_{init}, T, \mathcal{C}, \mathcal{G})$ and $N \in \mathbb{N}$, we define the induced bounded horizon MDP $\mathcal{M}^N = (\mathcal{S}^N, Act, s^N_{init}, T^N, \mathcal{C}^N, \mathcal{G}^N)$, being essentially N unfoldings of $\mathcal{M}$, i.e. $\mathcal{S}^N = \{0,\ldots,N\} \times \mathcal{S}$, $T^N((n,s),a)((n+1,t)) = T(s,a)(t)$ and $(n,s) \in \mathcal{G}^N$ if $(n = N \vee s \in \mathcal{G})$. Thus in $\mathcal{M}^N$ we will with probability 1 be in a goal state after at most N steps. Note that any partition $\mathcal{A}$ of $\mathcal{S}$ may be extended to a partition $\mathcal{A}^N$ of $\mathcal{S}^N$ by $(\{i\} \times \nu) \in \mathcal{A}^N$ whenever $\nu \in \mathcal{A}$ and $i \in \{0,\ldots,N\}$.

Theorem 2. *Let $\mathcal{M} = (\mathcal{S}, Act, s_{init}, T, \mathcal{C}, \mathcal{G})$ be an MDP with T and $\mathcal{C}$ being continuous functions. Let $\mathcal{A}_0 \sqsubseteq \mathcal{A}_1 \sqsubseteq \cdots \sqsubseteq \mathcal{A}_i \sqsubseteq \cdots$ be a refining sequence of partitions consistent with $\mathcal{G}$ and with decreasing granularity $\delta_0 \geq \delta_1 \geq \cdots \geq \delta_i \geq \cdots$ with $\lim_{i\to\infty} \delta_i = 0$. Let $\mathcal{M}^N$ be the induced bounded horizon MDP for $N \in \mathbb{N}$. Then for any $s \in \mathcal{S}$:*

$$\inf_{i\to\infty} \mathbb{E}^{\min}_{\mathcal{M}^N,\mathcal{A}_i}(\mathcal{C}^N_{\mathcal{A}_i},[(0,s)]_{\mathcal{A}_i}) = \inf_{\sigma} \mathbb{E}^{\mathcal{M}^N}_{\sigma}(\mathcal{C}^N,(0,s)) = \inf_{i\to\infty} \mathbb{E}^{\max}_{\mathcal{M}^N,\mathcal{A}_i}(\mathcal{C}^N_{\mathcal{A}_i},[(0,s)]_{\mathcal{A}_i})$$

Proof Sketch: We show by induction on $(N-n)$ that:

$$\forall \epsilon > 0. \exists i. \forall \nu \in \mathcal{A}_i.$$
$$\left|\mathbb{E}^{\max}_{\mathcal{M}^N,\mathcal{A}_i}(\mathcal{C}^N_{\mathcal{A}_i},\{N-n\}\times\nu) - \mathbb{E}^{\min}_{\mathcal{M}^N,\mathcal{A}_i}(\mathcal{C}^N_{\mathcal{A}_i},\{N-n\}\times\nu)\right| \leq \epsilon$$

Crucial for the induction step is uniform continuity of T and $\mathcal{C}$ due to compactness of $\mathcal{S}$, i.e. $\forall \epsilon > 0. \exists i. \forall \nu, \nu' \in \mathcal{A}_i. \forall a. |T_{\mathcal{A}_i}(\nu, a)(v')| \leq \epsilon$.

□

4 Algorithms

While the previous section hints that a partition refinement scheme aids in the computation of near-optimal strategies for continuous state MDPs, it assumes that minimal and maximal values of the transition and cost functions are known. It can be shown that these values, in general, are undecidable to attain for stochastic hybrid systems (which can be modeled in Uppaal SMC). We shall therefore in the sequel rely on simulation-based learning methods, using an online partition refinement scheme.

We first present the underlying learning algorithms as if a fixed partition is given, then return to the partition refinement scheme in Section 4.2.

We adopt the (adaptive, online) Q-learning and M-learning terminology from [17] and thus see both as methods for solving the Bellman equations online, commonly called *Q-values*, while guiding the search of the state space. While Q-learning is a so-called model-free learning method, directly learning the Q-values, M-learning attempts to derive the Q-values from the approximate transition-functions and cost functions. Common for both learning-methods is the online nature of the strategy synthesis, along with an online partition refinement scheme. We shall only present the pseudocode of the Q-learning variants of our algorithms and refer the reader to Appendix B for the M-learning equivalents.

4.1 Learning on Static Partitions

In the sequel the reader will encounter maps and partial functions as $X : 2^{\mathbb{R}^K} \hookrightarrow \ldots$. We use such partial functions for brevity and one can think of them as: the region $\nu \in 2^{\mathbb{R}^K}$ is decorated with some information by the X function – and ν is a specific (continuous) region of the state space. While these functions as defined as partial we implement them as maps over $2^{\mathbb{R}^K}$ in which non-updated elements have the same default value. This allows us to memorize only the (finite) number of non-default values, thus making these maps implementable.

In the sequel we let $\mathcal{A}_\alpha \subseteq 2^{\mathbb{R}^K}$ denote a partition for a given action $\alpha \in Act$. Notice that this differs slightly from the use of $\mathcal{A}$ in Section 3 where we omit the action-indexing for clarity and readability.

For both Q-learning and M-learning we shall assume the existence of the following global two modifiable functions. 1. $\mathcal{F}_\alpha : 2^{\mathbb{R}^K} \hookrightarrow \mathbb{N}_0$, for each $\alpha \in Act$, yielding an occurrence count for a given region, and 2. $Q : Act \times 2^{\mathbb{R}^K} \hookrightarrow \mathbb{R}$, yielding an approximation of the expected cost for a given action in a given region. By default, we let $\mathcal{F}_\alpha(\nu) = 0$ and $Q_\alpha(\nu) = 0$ for all $\nu \in 2^{\mathbb{R}^K}$ and all $\alpha \in Act$. Furthermore, we let the singleton set $\mathcal{A}_\alpha = \{\mathbb{R}^K\}$, for all $\alpha \in Act$, be the initial partition. While we omit the exact details of the `normalize`-function, it

Input: An MDP $\mathcal{M}$, an initial state s_{init}, a cost-function $\mathcal{C}$, a termination criterion and a set of goal states $\mathcal{G}$

Output: A near-optimal, strategy σ for the MDP $\mathcal{M}$ under the cost function $\mathcal{C}$ for the goal $\mathcal{G}$.

1 Initially let $\sigma(s)(\alpha) = \frac{1}{|Act|}$ for any $s \in \mathbb{R}^K$ and any $\alpha \in Act$
2 **while** *Termination criterion is not met* **do**
3 $\pi \leftarrow \epsilon$, $s \leftarrow s_{init}$
4 **while** $s \notin \mathcal{G}$ **do**
5 Draw α from Act according to $\sigma(s)$
6 Draw s' from $\mathcal{S}$ according to $T(s, \alpha)$
7 $\pi \leftarrow (\pi \circ (s, \alpha, s'))$, $s \leftarrow s'$
8 **for** *i from n to* 1 *with* $(s_1, \alpha_1, s_2) \dots (s_n, \alpha_n, s_{n+1}) = \pi$ **do**
9 Let $\alpha = \alpha_n$
10 Let $\nu = \mathcal{A}_\alpha[s_n]$, $\nu' = \mathcal{A}_\alpha[s_{n+1}]$
11 $\mathcal{F}_\alpha(\nu) \leftarrow \min(\mathcal{F}_\alpha(\nu) + 1, R)$
12 $Q_\alpha(\nu) \leftarrow (1 - \frac{1}{\mathcal{F}_\alpha(\nu)}) \cdot Q_\alpha(\nu) + \frac{1}{\mathcal{F}_\alpha(\nu)} \cdot (\mathcal{C}(s_n, \alpha, s_{n+1}) + \min_{\alpha' \in Act} Q_{\alpha'}(\nu'))$
13 $(Q, \mathcal{F}, \mathcal{A}) \leftarrow \texttt{Refine}_{\texttt{Q}}(Q, \mathcal{F}, \mathcal{A}, (s_n, \alpha, s_{n+1}))$
14 $\sigma \leftarrow \texttt{normalize}(Q)$
15 **return** σ

Algorithm 1: The Q-learning training algorithm.

denotes the normalization of Q-values into pseudo-probabilistic weights yielding a stochastic strategy.

Q-Learning Q-learning is a model-free learning method where the Q-values are derived directly from the samples [18]. For Q-learning, in addition to the globally defined functions, we introduce a learning-rate constant $R \in \mathbb{N}$. In Algorithm 1 we present the Q-learning training algorithm. After a uniform strategy σ is created (line 1), samples are drawn from the MDP $\mathcal{M}$ in accordance with σ (lines 3-7), backpropagated to the learning algorithm (lines 8-14) until some termination condition is met (e.g. a sample budget). Notice that we use the textbook of Q-learning update function [18] on line 12, followed by our refinement-method (line 13) and a transformation method in line 14.

M-Learning For M-learning (otherwise known as RTDP [15]) the aim is to produce an approximation of the transition- and cost-function of a finite-state MDP, where each state comes from $2^{\mathbb{R}^K}$. We therefore need the additional bookkeeping function $\hat{\mathcal{C}}_\alpha : 2^{\mathbb{R}^K} \times 2^{\mathbb{R}^K} \hookrightarrow \mathbb{R}$ tracking the empirically obtained cost-function. While Algorithm 1 learns the Q-function directly form samples, the equivalent M-learning algorithm (Appendix 5) infers the Q-function based on the sampled cost and frequency function.

4.2 Refinement Functions

To continuously improve on the refinement, we introduce by $\texttt{R}_\alpha : \nu \hookrightarrow 2^{\mathbb{R}^K} \times 2^{\mathbb{R}^K}$ a finite set of such functions for a specific action $\alpha \in Act$. We say that ρ :

$2^{\mathbb{R}^K} \hookrightarrow 2^{\mathbb{R}^K} \times 2^{\mathbb{R}^K}$ is a refinement function iff $(\nu', \nu'') = \rho(\nu)$ then $\nu \cap \nu' = \emptyset$ and $\nu \cup \nu' = \nu''$.

Note that many such functions can exist, and the method puts no restraints on these, other than what is given above. In our implementation we restrict ourselves to refinement functions defined as single-dimensional difference constraints (as done in Figure 2 but generalized to multiple dimensions). We thus consider partition-functions of the form $\rho(\nu) = (\{p \in \nu \mid p_i \leq b\}, \{p \in \nu \mid p_i > b\})$ for some constant b and fixed dimension $0 < i \leq K$. For practical reasons our $\mathtt{R}_\alpha$-set only consists of one such refinement function for each dimension in the implementation. Also note that in a practical setting, one such refinement may contain only unreachable parts of the state space, and thus occasionally we readjust our refinement functions. We can then re-adjust a refinement by increasing or decreasing b according to the number of samples observed in the two different sub-refinements. This adjustment is omitted from the presented pseudocode.

We refer the reader to the implementation for further details[8], but note that such single-dimensional partition functions (and subsequent decorations) are trivially implemented as decision trees whose leaf nodes represent the actual partitions.

Observe that this continued refinement induces an sequence of MIDP abstractions, leaning on the theoretical framework presented in Section 3.

Refinement Heuristics For each region v in the current partition $\mathcal{A}_\alpha$ and each candidate refinement $(\nu', \nu'') = \rho(\nu)$ we maintain summary statistics of the Q-values obtained at sampled states in ν, respectively ν'. In the case of M-learning, we also maintain statistics on transition frequencies between pairs of regions. For the remainder of this section we focus on Q-learning. The same principles are applied in M-learning. Our summary statistics consist of triples

$$(m, v, f) \in \Xi := \mathbb{R} \times \mathbb{R} \times \mathbb{N}_0$$

representing empirical means, variances, and frequency counts. Given a new data point $x \in \mathbb{R}$, the statistics are updated by the `US` update function (See Algorithm 3 in Appendix A), which makes use of Welford's algorithm for an online approximation of variance and otherwise updates the mean and frequency appropriately.

After each update of (m', v', f') or (m'', v'', f'') we calculate two functions `W` and `KS` as indicators for whether the distributions of Q-values in ν' and ν'' are different. The first function is inspired by the test statistic of the 2-sample t-test:

$$\mathtt{W}((m', v', f'), (m'', v'', f'')) := \frac{m' - m''}{\sqrt{f'v' + f''v''}} \sqrt{\frac{f' + f'' - 2}{1/f' + 1/f''}}$$

The `W` value provides evidence for or against equality of the two means m', m''. In order to also take possible disparities in the variances into account, we use a second function that is modeled on the Kolmogorov-Smirnov test statistic.

[8] Available at `https://github.com/petergjoel/libprlearn`

For this we interpret the given means m and variances v to be given by simple distributions with "triangular" density functions of the form

$$d(x) = \begin{cases} 0 & x < m - c \text{ or } x > m + c \\ a(x - m + c) & m - c \leq x \leq m \\ -a(m + c - x) & m \leq x \leq m + c \end{cases}$$

where a, c are uniquely determined by the conditions that $d(x)$ is a probability density function with variance v. We then compute $\texttt{KS}((m', v', f'), (m'', v'', f''))$ as the product of the maximal difference of the cumulative distribution functions of triangular distributions with means and variances (m', v'), respectively (m'', v''), and a weight factor $f'f''/(f' + f'')$. We note that triangular density functions are here used for computational convenience, not because they are assumed to be a particularly accurate model for the actual distributions of Q-values.

Given three thresholds t_l, t_u and q, with $t_l < t_u$, we now make a four-fold distinction for the obtained $\texttt{W}$ and $\texttt{KS}$ values. Abbreviating $(m', v', f') =: a$ and $(m'', v'', f'') =: b$:

$$\texttt{DIF}_{q,t}(a, b) = \begin{cases} \lightning & \text{if } \texttt{KS}(a, b) > q \text{ and } \texttt{W}(a, b) > t_u \\ \triangleleft & \text{if } a.m < b.m \text{ and } \texttt{W}(a, b) < t_l \\ \triangleright & \text{if } b.m < a.m \text{ and } \texttt{W}(a, b) < t_l \\ \bullet & \text{otherwise} \end{cases}$$

Using the outcome of $\texttt{DIF}_{q,t}(a, b)$ directly to decide whether to refine ν to ν', ν'' can lead to a somewhat fragile behavior, due to possible large fluctuations of sampled Q-values on the one hand, and dependencies between successive samples on the other. We therefore maintain a discounted running count $(x_\triangleleft, x_\triangleright, x_\lightning)$ of the three critical outcomes $\triangleleft, \triangleright, \lightning$, and perform a refinement when one of these counts exceeds a given threshold ψ.

4.3 Q-Learning on Partitions

We present the partition refinement function for Q-learning in Algorithm 2. The algorithm consists of two main parts, first the statistics is updated and then a (if deemed necessary) a refinement occurs. In lines 2-8 we update the statistics of candidate refinements. Lines 7 and 8 update the discounted counts $(x_\triangleleft, x_\triangleright, x_\lightning)$ using a simple procedure `LMS` (see Algorithm 4 in Appendix A for the details). The `LMS`-function maintains a least-mean-square like filter over each of the three significant outcomes of the `DIF`-function, essentially smoothing out jitter.

In line 9 we identify the candidate partitions that display significantly different behavior for within their regions. If one or more candidates is deemed significant, we choose one randomly and update the partition $\mathcal{A}$, the Q-function and the $\mathcal{F}$-function in an appropriate manner (lines 13-16).

Input: $\texttt{Refine}_{\texttt{Q}}(Q, \mathcal{F}, \mathcal{A}, (s, \alpha, s'))$
Output: Refined Q, $\mathcal{F}$ and $\mathcal{A}$-functions.
Let $\nu = \mathcal{A}_\alpha[s]$, $\nu' = \mathcal{A}_\alpha[s']$
$c \leftarrow \mathcal{C}_\alpha(s, s') + \min_{\alpha' \in Act} Q_{\alpha'}(\nu')$
for *all* $\rho \in \texttt{R}_\alpha(\nu)$ **do**
  Let $(\widetilde{\nu}', \widetilde{\nu}'') = \rho(\nu)$ and let $\widetilde{\nu} \in \{\widetilde{\nu}', \widetilde{\nu}''\}$ s.t. $s \in \widetilde{\nu}$
  $(m, v, f) \leftarrow \texttt{Stats}_\alpha(\widetilde{\nu})$
  $\texttt{Stats}_\alpha(\widetilde{\nu}) \leftarrow \texttt{US}(\texttt{Stats}_\alpha(\widetilde{\nu}), (1 - \frac{1}{f}) \cdot m + \frac{1}{f} \cdot c)$
  $\bowtie \leftarrow \texttt{DIF}_{\beta,p}(\texttt{Stats}_\alpha(\widetilde{\nu}'), \texttt{Stats}_\alpha(\widetilde{\nu}''))$
  $\texttt{LMSVal}^\rho_\alpha(\nu) \leftarrow \texttt{LMS}(\texttt{LMSVal}^\rho_\alpha(\nu), \bowtie)$
Let $\texttt{R}' \leftarrow \{\rho \in \texttt{R}_\alpha \mid \max(x_\triangleleft, x_\triangleright, x_\restriction) \geq \psi$ where $(x_\triangleleft, x_\triangleright, x_\restriction) = \texttt{LMSVal}^\rho_\alpha(\nu)\}$
if $\texttt{R}' \neq \emptyset$ **then**
  Pick ρ randomly from $\texttt{R}'$
  Let $(\widetilde{\nu}', \widetilde{\nu}'') = \rho(\nu)$
  $\mathcal{A}_\alpha \leftarrow (\mathcal{A}_\alpha \setminus \nu) \cup \{\widetilde{\nu}', \widetilde{\nu}''\}$
  $(m', v', f') \leftarrow \texttt{Stats}_\alpha(\widetilde{\nu}')$, $(m'', v'', f'') \leftarrow \texttt{Stats}_\alpha(\widetilde{\nu}'')$
  $Q_\alpha(\widetilde{\nu}') \leftarrow m'$, $Q_\alpha(\widetilde{\nu}'') \leftarrow m''$
  $\mathcal{F}_\alpha(\widetilde{\nu}') \leftarrow f'$, $\mathcal{F}_\alpha(\widetilde{\nu}'') \leftarrow f''$
return $\langle Q, \mathcal{F}, \mathcal{A} \rangle$

Algorithm 2: Partition refinement function for Q-learning

4.4 M-Learning on Partitions

While Algorithm 2, much in the spirit of Q-learning, infers the variance of a new hypothetical split in a model-free way, we here adopt an M-learning approach for the inference of variance over the Q-values obtained, which in turn are computed from the empirical cost ($\hat{\mathcal{C}}$) and transition-function (inferred from the frequency-function). As such, the variance of the Q-function for M-learning cannot be inferred from samples (as in Q-learning), but has to stem from an aggregate over the empiric cost and transition functions. This change introduces some types of errors compared to Q-learning (impressions in aggregation) while reducing others (sensitive to wildly fluctuating Q-values in the successor states). The full pseudocode for $\texttt{Refine}_{\texttt{M}}$ is presented in Appendix C.

5 Experimental Results

To evaluate the proposed methods we conduct a series of experiments on a number of different cases, each generated from one of four scalable models that we will shortly present in more detail. Furthermore, we provide the implementation of the presented algorithms under the LGPL license[9]. We run our experiments on an AMD Opteron 6376 processor, limited to 16 GB of RAM. The models, full table of results and the version of Uppaal Stratego implementing the proposed algorithms can be found at `http://people.cs.aau.dk/~pgj/ATVA19/results.html`.

[9] Available at `https://github.com/petergjoel/libprlearn`

For each instance we learn a strategy from varying numbers of samples (i.e. the training-budget). We measure time and memory consumption of learning. We evaluate the quality of a strategy by measuring the cost on a test run of 1000 samples. Each experiment was repeated 50 times using a different initial random seed. We here report the 25% quantile and the 50% quantile (median) of the costs obtained in the test runs.

The proposed algorithms are compared to those of David et. al.[4] (shortened D-algorithms). The D-algorithms perform batch learning and therefore require a slightly different learning regime. We train the D-algorithms on 20 batches, each of $\frac{1}{20}$ of the training budget. For each batch, we allow for an extra $\frac{1}{20}$ of the training budget to be used for strategy evaluation, to avoid overfitting. This intermediate validation step gives an advantage to the D-algorithms over Q- and M-learning, which are only evaluated after training on the full budget has completed. In addition, we consider only the best-performing of the D-algorithm for each model instance and training budget combination (measured in terms of the 25% percentile).

We note that all the presented case-studies effectively fall in the class of switch-control-systems, in which we sample only until a finite horizon, thus also residing inside the conditions of Theorem 2.

We next provide a short description of our models (for a verbose description, see Appendix D). **Bouncing Ball:** We model N balls in the physical system displayed in Figure 3. The goal is to keep the ball alive by utilizing the piston for *hitting* the ball - however, each hit comes with a slight penalty and only has an effect if the ball is above a certain height. A sample trace can be seen in Figure 4 of the ball under an untrained strategy (many hits, yet still a "dead" ball), and similar for a trained strategy (fewer and more significant hits). A visualization of the learned cost-function can be seen in Figure 3. The middle image is the cost of the hitting-action, the rightmost image is the cost of not hitting[10]. **Floorheating:** We slightly modify the case study of Larsen et. al. [11] by replacing the outdoor temperature measurements/predictions with a simplified sinusoidal curve. **Highway Control:** We model a set of different highway-scenarios for an autonomous vehicle. The goal of the controller is to avoid collisions for as long as possible (i.e. minimize the time from a crash till the time-bound of the simulation). Several environment cars are in the models, each having stochastic behavior and reacting in a non-trivial (but intuitive) way to the proximity of other cars[11]. **Mixed Integer Linear Programming:** We have obtained a series of MILP programs for optimizing a switch-control-system from one industrial partner. The purpose is to minimize the cost of energy utilization of a residence-unit by the use of buffers and on-demand smart-metering. We note that our methods are capable of delivering controllers which are suffi-

[10] A visualization of a strategy generated by M-learning for a single ball can be found online `http://people.cs.aau.dk/~pgj/ATVA19/visualize.html`

[11] A visualization of the system under control can be found at `http://people.cs.aau.dk/~pgj/ATVA19/cars.html`

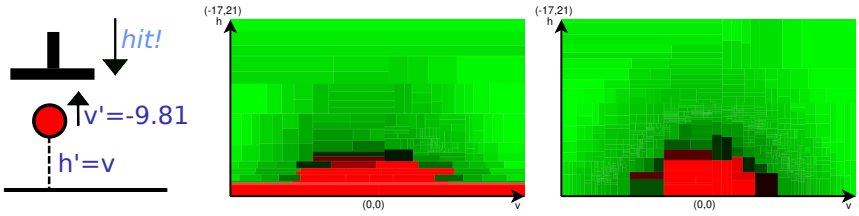

Fig. 3: A visualization of the BouncingBall model and the learned, color-coded Q-function for *hit* and $no - hit$ action (red being more expensive and green less) in over the h (height) and v (velocity) state variables.

ciently close to the optimal solution s.t. these are usable within the domain of our partner - however, often with significantly reduced computation-times.

Results Due to brevity we only present a representative subset of the experiments conducted[12].

In general we observe significantly improved convergence tendencies when applying M- and Q-learning compared to D-learning.

As can be observed in Table 1 in the BouncingBall examples, both M- and Q-learning converge towards relatively low values. We can also observe that none of the D-algorithms are able to match the Q- and M-learning approaches. In fact, the D-algorithm results appear to be oscillating in quality, not converging even with an increased training budget for $N > 1$.

In the Floorheating experiments we observe that the D-algorithms performing equally well to M- and Q-learning with a low sampling budget. However, as the sampling budget increases, Q- and M-learning eventually overtake the D-algorithms.

For the Highway experiments for the *3car* case, we observe that M-learning, seems to converge slower than Q-learning, while the opposite is true for the *4carovertake4* example. In the *3car* experiment we can see that the D-algorithms finds the optimal solution to the problem with a small budget of runs, however,

[12] All results and publishable models can be found at `http://people.cs.aau.dk/~pgj/ATVA19/results.html`

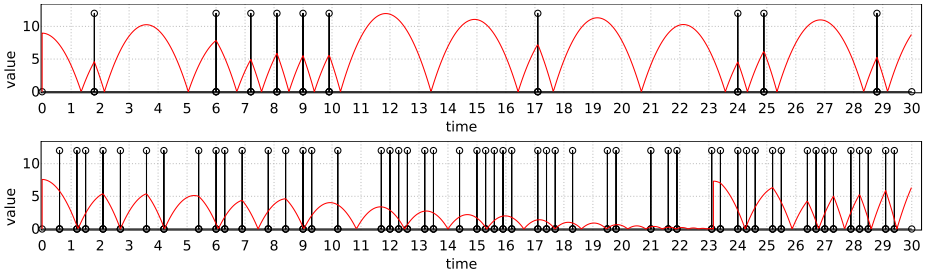

Fig. 4: A plot of the bouncing ball under control of a learned strategy (top) and a random strategy (bottom) with vertical lines indicating a *hit*.

without being unable to consistently perform with an increasing budget of runs. The D-algorithms seem to demonstrate convergence in the *4carovertake4*, however, not at the same rate as the Q- and M-learning algorithms.

In the (non-probabilistic) MILP experiments, we can see that the D-algorithms can provide quite reasonable results (*sfhd4500* and *sfhd9500*) but are eventually overtaken on the marginals by both Q- and M-learning in both cases. We observe that Q-learning requires a larger budget to converge to results comparable to M-learning, possibly due to the natural delay in the update of the Q-values and the non-probabilistic nature favoring the structured approach of M-learning.

Finally, to briefly touch upon the runtime of the different algorithms we shall here generalize via the Floorheating 1-5 experiment with a budget of 1000 simulations. We initially observe that D-algorithms are the slowest at 287 (to ranked by the best performing by the cost-function). At 205 seconds and 191 seconds of computation time, M-learning and Q-learning are significantly faster, with Q-learning winning by a small margin. We note that these relationships in running-time seem to generalize across the experiments.

6 Conclusion

Throughout this paper we have argued for the correctness and theoretical soundness of applying Q- and M-learning on a MIDP-abstraction of an Euclidean MDP. Leaning on this theoretical argumentation, we introduced an online partition refinement adaptions of Q- and M-learning, facilitating a data-driven refinement scheme. While we leave the question of convergence open for online refinement, our experiments demonstrate convergence-like tendencies for both Q- and M-learning. In particular, we observe that better convergence is attained with little or no additional overhead compared to the methods of [5]. In fact, for certain examples like the Bouncing Ball, we see that methods of [5] show no signs of convergence. We also observe that Q-learning seems to be computationally lighter, but with generally slower convergence compared to M-learning, supporting the claims of Strehl et. al. [15].

Several directions of future work are opened by this paper, including direct comparison with Neural Network alternatives, verification of the learned strategies, feature-reductions and online partition simplification. We also think that alternate refinement heuristics could improve the performance of the methods, such as using rank-based statistical tests. More complex refinement functions and function representation could also improve the proposed methods.

Bibliography

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. 1984.

[2] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reduction and refinement strategies for probabilistic analysis. In H. Hermanns and R. Segala, editors, *Process Algebra and Probabilistic Methods: Performance*

Alg	25%	50%	25%	50%	25%	50%	25%	50%	25%	50%	25%	50%	25%	50%
Runs	100		250		500		1000		2500		5000		10000	
BouncingBall-1														
M	100	188	57	73	46	53	42	46	40	41	39	39	38	39
Q	61	75	47	71	44	49	41	47	40	42	40	41	39	40
D	317	354	5044	5052	5042	5050	78	100	70	88	68	104	66	160
BouncingBall-2														
M	203	367	204	376	151	242	134	179	124	184	82	101	72	102
Q	152	236	117	189	112	153	94	134	95	114	76	86	69	76
D	361	4840	3865	8550	10096	10103	171	254	487	1624	935	3265	1244	6015
BouncingBall-3														
M	392	1032	430	1017	364	1198	402	739	197	388	187	230	154	250
Q	257	362	204	283	142	242	181	345	120	166	114	136	106	128
D	377	10635	3373	10168	15136	15155	417	1043	1952	4144	1791	6763	2595	9283
Floorheating-1-5														
M	394	419	381	391	366	380	327	341	283	289	272	276	264	267
Q	491	568	368	405	340	398	285	302	283	292	281	287	273	278
D	344	367	370	393	374	413	341	369	335	353	335	347	296	310
Floorheating-6-11														
M	395	450	357	389	343	371	323	337	300	305	285	290	277	282
Q	1059	1394	614	767	450	553	387	457	296	307	288	298	284	287
D	487	527	496	520	495	530	438	468	419	437	383	415	355	384
Highway-3car														
M	113	138	81	119	54	85	38	57	22	31	16	22	11	14
Q	108	163	70	118	25	97	11	31	9	15	8	12	5	8
D	0	0	0	3	2	10	2	13	3	9	1	5	1	6
Highway-4car-overtake4														
M	131	156	134	147	106	123	96	107	49	69	18	28	8	12
Q	126	160	97	140	108	128	80	104	58	73	31	49	17	22
D	103	104	56	100	57	94	33	67	43	67	38	63	31	71
MILP-sfhd-4500														
M	35	39	30	32	28	29	25	26	23	23	22	23	22	22
Q	228	243	82	205	37	47	32	34	29	30	28	28	26	26
D	38	40	36	40	37	40	36	39	35	38	33	37	32	35
MILP-sfhd-9500														
M	65	81	80	99	53	62	55	63	60	66	59	64	57	61
Q	77	77	56	63	55	61	56	59	56	61	58	65	61	70
D	77	77	77	77	72	77	69	76	71	77	65	71	68	75

Table 1: Comparison of $\{M, Q, D\}$-learning in terms of expected cost of the strategy synthesized. We report the 25 percentile and the median. All experiments were repeated 50 times.

Modeling and Verification, pages 57–76, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45605-6.

[3] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, and Z. Wang. Statistical model checking for networks of priced timed automata. In U. Fahrenberg and S. Tripakis, editors, *FORMATS 2011*, volume 6919, pages 80–96. Springer, 2011. ISBN 978-3-642-24309-7. doi: 10.1007/978-3-642-24310-3_7.

[4] A. David, P. G. Jensen, K. G. Larsen, A. Legay, D. Lime, M. G. Sørensen, and J. H. Taankvist. On time with minimal expected cost! In *ATVA 2014*, pages 129–145. Springer, 2014.

[5] A. David, P. G. Jensen, K. G. Larsen, M. Mikučionis, and J. H. Taankvist. Uppaal Stratego. In *TACAS 2015*, pages 206–211. Springer, 2015.

[6] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015. doi: 10.1007/s10009-014-0361-y.

[7] D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for markov decision processes. In *QEST 2012*, pages 84–93, Sept 2012. doi: 10.1109/QEST.2012.19.

[8] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In R. Majumdar and V. Kunčak, editors, *CAV 2017*, pages 3–29, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63387-9.

[9] M. Z. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for markov decision processes. In *(QEST 2006)*, pages 157–166. IEEE Computer Society, 2006. ISBN 0-7695-2665-9. doi: 10.1109/QEST.2006.19.

[10] K. G. Larsen, M. Mikučionis, and J. H. Taankvist. *Safe and Optimal Adaptive Cruise Control*, pages 260–277. Springer International Publishing, Cham, 2015. ISBN 978-3-319-23506-6. doi: 10.1007/978-3-319-23506-6_17.

[11] K. G. Larsen, M. Mikučionis, M. Muñiz, J. Srba, and J. H. Taankvist. Online and compositional learning of controllers with application to floor heating. In M. Chechik and J.-F. Raskin, editors, *TACAS 2016*, pages 244–259, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-49674-9.

[12] K. G. Larsen, A. Le Coënt, M. Mikučionis, and J. H. Taankvist. Guaranteed control synthesis for continuous systems in Uppaal Tiga. Mar. 2019. URL `https://hal.archives-ouvertes.fr/hal-02076824`. working paper or preprint.

[13] Y. Z. Lun, J. Wheatley, A. D'Innocenzo, and A. Abate. Approximate abstractions of markov chains with interval decision processes. In A. Abate, A. Girard, and M. Heemels, editors, *ADHS 2018*, volume 51 of *IFAC-PapersOnLine*, pages 91–96. Elsevier, 2018. doi: 10.1016/j.ifacol.2018.08.016.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529, 2015.

[15] A. L. Strehl, L. Li, and M. L. Littman. Incremental model-based learners with formal learning-time guarantees. *CoRR*, abs/1206.6870, 2012. URL `http://arxiv.org/abs/1206.6870`.

[16] L. Sun, Y. Guo, and A. Barbu. A Novel Framework for Online Supervised Learning with Feature Selection. *arXiv e-prints*, art. arXiv:1803.11521, Mar. 2018.

[17] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[18] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

A Bookkeeping Algorithms

We here present Algorithm 3 for updating mean, variance and frequency-count, with the running-variance approximation computed using Welfords algorithm and Algorithm 4 for maintaining the discounted outcomes of the `DIF`-check using a least-mean-squares like filter.

Input: A tuple $(m, v, f) \in \Xi$ and a new sample-point $x \in \mathbb{R}$
Output: An updated tuple $(\hat{m}, \hat{v}, \hat{f}) \in \Xi$

$\hat{f} \leftarrow f + 1$
$\hat{m} \leftarrow \frac{x + m \cdot f}{\hat{f}}$
$\hat{v} \leftarrow \frac{(|m-x| \cdot |\hat{m}-x| - v) + v \cdot f}{\hat{f}}$
return $(\hat{f}, \hat{m}, \hat{v})$

Algorithm 3: The `US` function update.

Input: $\texttt{LMS}((x_\triangleleft, x_\triangleright, x_\dagger), d)$
Output: New values $(x'_\triangleleft, x'_\triangleright, x'_\dagger)$.

if $d = \bullet$ **then**
  return $(x_\triangleleft, x_\triangleright, x_\dagger)$
$x'_d \leftarrow x_d + (1 - x_d) \cdot \nabla$
for $d' \in \{\triangleleft, \triangleright, \dagger\} \setminus \{d\}$ **do**
  $x'_{d'} \leftarrow x_{d'} + (0 - x_{d'}) \cdot \nabla$
return $(x'_\triangleleft, x'_\triangleright, x'_\dagger)$

Algorithm 4: The `LMS` algorithm for maintaining discounted counts.

B M-Learning

Initially, let us re-cast the frequency-count function $\mathcal{F}$ to accomodate the needs ot the `M`-learning approach. We let $\mathcal{F}_\alpha : 2^{\mathbb{R}^K} \times \mathbb{N}_0$, for each $\alpha \in Act$, yield an occurrence count for a given region pair. Using this new function, we can introduce a short-hand function for the probability-estimates.

$$\hat{\mathbb{P}}_\alpha(\nu, \nu') = \frac{\mathcal{F}_\alpha(\nu, \nu')}{\sum_{\nu'' \in \mathbb{A}_\alpha} \mathcal{F}_\alpha(\nu, \nu'')}$$

In addition, we introduce the following definition of "the historical union of all partitions".

– $\mathbb{A}_\alpha \subseteq 2^{\mathbb{R}^K}$ denoting the union of all prior partitions, initially $\mathbb{A}_\alpha = \mathcal{A}_\alpha$, and.

Input: An MDP $\mathcal{M}$, an initial state s_{init}, a cost-function $\mathcal{C}$, a termination criterion and a set of goal states $\mathcal{G}$

Output: A near-optimal, strategy σ for the MDP $\mathcal{M}$ under the cost function $\mathcal{C}$ for the goal $\mathcal{G}$.

1 Initially let $\sigma(s)(\alpha) = \frac{1}{|Act|}$ for any $s \in \mathbb{R}^K$ and any $\alpha \in Act$
2 **while** *Termination criterion is not met* **do**
3 $\quad \pi \leftarrow \epsilon$, $s \leftarrow s_{init}$
4 $\quad$ **while** $s \notin \mathcal{G}$ **do**
5 $\quad\quad$ Draw α from Act according to $\sigma(s)$
6 $\quad\quad$ Draw s' from $\mathcal{S}$ according to $T(s, \alpha)$
7 $\quad\quad$ $\pi \leftarrow (\pi \circ (s, \alpha, s'))$, $s \leftarrow s'$
8 $\quad$ **for** *i from n to* 1 *with* $(s_1, \alpha_1, s_2) \dots (s_n, \alpha_n, s_{n+1}) = \pi$ **do**
9 $\quad\quad$ Let $\alpha = \alpha_n$
10 $\quad\quad$ Let $\nu = \mathcal{A}_\alpha[s_n]$, $\nu' = \mathcal{A}_\alpha[s_{n+1}]$
11 $\quad\quad$ $\mathcal{F}_\alpha(\nu, \nu') \leftarrow \mathcal{F}_\alpha(\nu, \nu') + 1$
12 $\quad\quad$ $\hat{\mathcal{C}}_\alpha(\nu, \nu') \leftarrow \frac{\mathcal{C}(s_n, \alpha, s_{n+1}) + \hat{\mathcal{C}}_\alpha(\nu, \nu') \cdot (\mathcal{F}_\alpha(\nu, \nu') - 1)}{\mathcal{F}_\alpha(\nu, \nu')}$
13 $\quad\quad$ $Q_\alpha(\nu) \leftarrow \sum_{\nu'' \in \mathbb{A}_\alpha} \langle (\hat{\mathcal{C}}_\alpha(\nu, \nu'') + \min_{\alpha' \in Act} \langle Q_{\alpha'}(\nu'') \rangle) \cdot \hat{\mathbb{P}}_\alpha(\nu, \nu'') \rangle$
14 $\quad\quad$ $(Q, \mathcal{F}, \hat{\mathcal{C}}, \mathcal{A}) \leftarrow \texttt{Refine}_{\mathtt{M}}(Q, \mathcal{F}, \hat{\mathcal{C}}, \mathcal{A}, (s_n, \alpha, s_{n+1}))$
15 $\quad\quad$ $\mathbb{A}_\alpha \leftarrow \mathbb{A}_\alpha \cup \mathcal{A}_\alpha$
16 $\quad\quad$ For all $\alpha'' \in \arg\min_{\alpha' \in Act \setminus \{\alpha\}} Q_{\alpha'}(\nu''')$ where $\nu''' \in \mathcal{A}_{\alpha''}$ and $s_n \in \nu'''$
$\quad\quad\quad$ $Q_{\alpha''}(\nu''') \leftarrow \sum_{\nu'' \in \mathbb{A}_\alpha} \langle (\hat{\mathcal{C}}_{\alpha''}(\nu''', \nu'') + \min_{\alpha' \in Act} \langle Q_{\alpha'}(\nu'') \rangle) \cdot \hat{\mathbb{P}}_\alpha(\nu''', \nu'') \rangle$
17 **return** σ

Algorithm 5: The M-learning training algorithm. The algorithm is similar to Algorithm 1 except for the body of the reversing loop in line 8.

– $\mathbb{A} = \biguplus_{\alpha \in Act} \mathbb{A}_\alpha$.

Initially we let $\hat{\mathcal{C}}_\alpha(\nu, \nu') = 0$ for any $\alpha \in Act$ and any $\nu, \nu' \in 2^{\mathbb{R}^K}$ and define $\mathbb{P}$ as follows where we by agreement let division by zero yield zero. If we consider a fixed partition and assume that the refinement step in line 14 of Algorithm 7 as the identity-function, we can see that the only difference with the algorithms original construction [15] is the update of alternative in lines 16 of Algorithm 7. We observed that the introduction of these extra backpropagations improved the performance of the method without a major performance hit.

C Refinement Function for M-Learning

In M-learning we have to compute aggregates over several cost-functions (and their futures), in the form of elements of Ξ. We therefore introduce the in-line statistics-combinator $\oplus : \xi \times \xi \to \xi$.

$$\oplus((m, v, f), (m', v', f')) = (m + m', \max(v, v'), f)$$

We also define a statistics aggregator over multiset of statistics in Algorithm 6.

Input: A multiset X of elements from Ξ.
Output: An aggregated normal distribution.

$f \leftarrow \sum_{(m',v',f')\in X} f'$
$m \leftarrow \frac{1}{f} \sum_{(m',v',f')\in X} m'f'$
$v \leftarrow 0$
for *all* (m', v', f') *from* X **do**
  $s' \leftarrow \sqrt{v'}$
  $v \leftarrow v + \frac{f\cdot((m-(m'+s'))^2+(m-(m'-s'))^2)}{2}$
return $(m, \frac{v}{f}, f)$

Algorithm 6: The $\otimes$ algorithm for aggregating a multiset of elements of Ξ.

Furthermore, we assume that Algorithm 5 stores the computed variance along with the Q-value; i.e. we assume that $Q_\alpha(\nu) \in \Xi$ for any $\alpha \in Act$ and $\nu \in 2^{\mathbb{R}^K}$ (This computation can be deduced from the above aggregation-functions and the Algorithm 7).

We can now present the main pseudo-code of $\texttt{Refine}_\texttt{M}$. Algorithm 7 has three main parts; first the hypothetical model (for each hypothetical partition) is updated (lines 4-9). Then, if one or more refinement-functions are found significant (line 10), we update the partition-function and populate the transition and cost functions (lines 15-19) of the new model with the values previously computed (ξ_ρ). Lastly we update the "ancestor" regions of ν (omitted for brevity) as incoming empiric transitions might still exist leading to these. In the ancestor update on line 21 we let the ancestor assume the highest Q-value of its immediate children, update recursively in a direct line from ν. We again refer the reader to the implementation for details[13].

D Model Description

The Bouncing Ball: N balls are bouncing off the ground, each with a (small) random offset, and a (small) random change in speed every time it hits the ground or hits the bat of the controller. Over time, the ball looses energy, and will eventually be out of the range for the bat, at which point the ball is declared "dead" and will be thrown anew.

The controller will now incur a heavy cost every time a ball reaches the "dead" state, and a small cost every time the bat is swung. An optimal controller will thus use as few swings as possible to keep the balls alive. All balls within range are hit when the bat is swung and the bat can only be swung on clock ticks. A visualization of a strategy generated by M-learning for a single ball can be found online `http://people.cs.aau.dk/~pgj/ATVA19/visualize.html`. In Figure 4 we can see two simulations of a bouncing ball; one computed using a random controller and one computed using a learned controller. We observe that even though the random strategy hits with a higher rate than the learned, the

[13] Available at `https://github.com/petergjoel/libprlearn`

Input: $\texttt{Refine}_{\texttt{M}}(Q, \mathcal{F}, \hat{\mathcal{C}}, \mathcal{A}, (s, \alpha, s'))$
Output: New Q, $\mathcal{F}$, $\hat{\mathcal{C}}$ and $\mathcal{A}$-functions

Let $\nu = \mathcal{A}_\alpha[s]$, $\nu' = \mathcal{A}_\alpha[s']$
Let $\xi'_\rho = \xi''_\rho = (0, 0, 0)$ for all $\rho \in \texttt{R}_\alpha(\nu)$
for *all* $\rho \in \texttt{R}_\alpha(\nu)$ **do**
    Let $(\widetilde{\nu}', \widetilde{\nu}'') = \rho(\nu)$ and let $\widetilde{\nu} \in \{\widetilde{\nu}', \widetilde{\nu}''\}$ s.t. $s \in \widetilde{\nu}$
    $\texttt{Stats}_\alpha(\widetilde{\nu}, \nu') \leftarrow \texttt{US}(\texttt{Stats}_\alpha(\widetilde{\nu}, \nu'), \mathcal{C}(s, \alpha, s'))$
    $\xi'_\rho \leftarrow \otimes \left(\biguplus_{\nu'' \in \mathbb{A}} \{\texttt{Stats}_\alpha(\widetilde{\nu}', \nu'') \oplus \min_{\alpha' \in Act}(Q_{\alpha'}(\nu''))\} \right)$
    $\xi''_\rho \leftarrow \otimes \left(\biguplus_{\nu'' \in \mathbb{A}} \{\texttt{Stats}_\alpha(\widetilde{\nu}'', \nu'') \oplus \min_{\alpha' \in Act}(Q_{\alpha'}(\nu''))\} \right)$
    $\bowtie \leftarrow \texttt{DIF}_{\beta,p}(\xi', \xi'')$
    $\texttt{LMSVal}^\rho_\alpha(\nu) \leftarrow \texttt{LMS}(\texttt{LMSVal}^\rho_\alpha(\nu), \bowtie)$
Let $\texttt{R}' \leftarrow \{\rho \in \texttt{R}_\alpha \mid \max(x_\triangleleft, x_\triangleright, x_\dagger) \geq \psi \text{ where } (x_\triangleleft, x_\triangleright, x_\dagger) = \texttt{LMSVal}^\rho_\alpha(\nu)\}$
if $\texttt{R}' \neq \emptyset$ **then**
    Pick ρ randomly from $\texttt{R}'$
    Let $(\widetilde{\nu}', \widetilde{\nu}'') = \rho(\nu)$
    $(m', v', f') \leftarrow \xi'_\rho$, $(m'', v'', f'') \leftarrow \xi''_\rho$
    $Q_\alpha(\widetilde{\nu}') \leftarrow m'$, $Q_\alpha(\widetilde{\nu}'') \leftarrow m''$
    for *all* $\nu'' \in \mathbb{A}$ **do**
        $(m', v', f') \leftarrow \texttt{Stats}_\alpha(\widetilde{\nu}', \nu'')$, $(m'', v'', f'') \leftarrow \texttt{Stats}_\alpha(\widetilde{\nu}'', \nu'')$
        $\hat{\mathcal{C}}_\alpha(\widetilde{\nu}', \nu'') \leftarrow m'$, $\hat{\mathcal{C}}_\alpha(\widetilde{\nu}'', \nu'') \leftarrow m''$
        $\mathcal{F}_\alpha(\widetilde{\nu}', \nu'') \leftarrow \frac{f'}{2}$, $\mathcal{F}_\alpha(\widetilde{\nu}'', \nu'') \leftarrow \frac{f''}{2}$
    $\mathcal{A}_\alpha \leftarrow (\mathcal{A}_\alpha \setminus \{\nu\}) \cup \rho(\nu)$
Do a recursive update on the ancestors of ν
return $\langle Q, \mathcal{F}, \hat{\mathcal{C}}, \mathcal{A} \rangle$

Algorithm 7: The partition refinement algorithm for M-learning

ball eventually "dies" (at timestep 23) - which is not the case for the learned strategy.

Floorheating: We have modified the model of the Floorheating case study of Larsen et. al. [11], to optimize the control over a 24-hour period. We model a simple outdoor weather sequence, letting the temperature range over (approximately) 0-10 degrees in a sinusoidal form during the day. Both the parameters of the sinusoid and the configuration of the doors are influenced by stochastics. At the same time, the controller will only manage roughly 50% of the heating system at any time, with the remainder being under the control of a bang-bang controller. We use the original cost measure of comfort, which has to be minimized, but report scaled-down results (by a factor of 100) for brevity.

Highway Control: In this example, the controller is tasked with steering a car on a highway. Next to the controller, on a two track highway, $N-1$ cars will be placed. Each other car will have a simple, but stochastic, hardcoded strategy for their behavior, such that they react to the movement of the controller and the other cars. At each clock tick, the controller (and the other cars) can choose to either accelerate or decelerate in accordance with nine predefined vectors. As input to the controller we only give the measures of 8 different proximity sensors (and their delta since the last clock tick) along with the current velocity vector of the car (18 features in total).

If the controller crashes into another car, it will be punished heavily, similarly if it drives off the road. In contrast, we give a small reward to the controller for each time unit it remains without crashes.

A good controller is one that stays on the road, avoids the other cars and has a reasonable speed. We evaluate the controller on its ability to minimize the time remaining of an episode (210 seconds) after a crash has occurred. [14]

Mixed Integer Linear Programming: Our fourth case study is given as a Mixed Integer Linear Program, modeling the heating system of a modern one-family house.The model arises from a collaboration with an industrial partner. While an optimization problem formalized as a MILP can be solved analytically, such a computation might be too slow for the given application area. The system differs from that of Larsen et. al.[11] in multiple ways; no probabilities are present, the thermodynamics of the building are modeled in greater detail and the overall setup of the heating system is radically different. As it is out of the scope of this paper, we refrain from discussing the quality of the approximate solutions compared to their analytic counterpart. While we make the Bouncing Ball and Highway models available online, the MILP problems cannot be published per request of our industrial partner.

[14] A visualization of selected strategies can be found at `http://people.cs.aau.dk/~pgj/ATVA19/cars.html`