

# Runtime Verification of Learning Properties for Reinforcement Learning Algorithms

Tommaso Mannucci

Julio de Oliveira Filho

Intelligent Autonomous Systems

TNO – Netherlands Organisation for Applied Scientific Research  
The Hague, The Netherlands

tommaso.mannucci@tno.nl

julio.deoliveirafilho@tno.nl

Reinforcement learning (RL) algorithms interact with their environment in a trial-and-error fashion. Such interactions can be expensive, inefficient, and timely when learning on a physical system rather than in a simulation. This work develops new runtime verification techniques to predict when the learning phase has not met or will not meet qualitative and timely expectations. This paper presents three verification properties concerning the quality and timeliness of learning in RL algorithms. With each property, we propose design steps for monitoring and assessing the properties during the system’s operation.

## 1 Introduction

Reinforcement learning (RL) [16] is a bio-inspired approach to machine learning which formalizes the notion of “trial-and-error” and “learn-by-doing”. RL enables systems to learn during operation based on sequential interactions with the environment. During their learning phase, RL algorithms encourage decisions that led to good results in the past while avoiding detrimental choices. This simple concept is at the base of some stunning results in robotics automation [14], natural language processing [7], and computerised gaming, such as the Atari [10], StarCraft [19] video games, and the ancient tabletop games of Chess, Shogi, and Go [13].

Due to the runtime and interactive nature of RL algorithms, there has been an increasing demand for guarantees about their learning; e.g., that it will be concluded within a certain amount of time or interactions when done in an operational environment. It is also necessary to guarantee the agent learned its solution space well enough during the learning phase. Offering such guarantees for RL algorithms is a challenging task. Traditional testing is often not possible due to the difficulty of acquiring a representative set of operating conditions [6]. Formal testing methods do not yet scale well, and many RL algorithms use “black box” components [4], such as artificial neural networks. The underlying models for these algorithms are uninformative and highly dimensional; and the individual effect of their many parameters on the overall performance is not apparent nor easy to assess.

This work proposes new *Runtime Verification* (RV) techniques for checking properties of the learning phase of RL algorithms. RV is an engineering discipline concerned with checking a system behaviour during its execution [2]. Many RL algorithms’ properties require such a runtime verification approach. Safety, timeliness, and robustness properties, for example, can become invalid when RL algorithms engage in learning during operation. This happens because properties observed in the system during design time might no longer hold as the system changes by learning from new data. Timeliness properties, such as the duration of the learning phase, depend on the order and variety of interactions presented to the RL agent.

The specific contributions of this paper are:

M. Farrell, M. Luckcuck, M. Schwammberger, and  
M. Gleirscher (Eds): Fifth International Workshop on  
Formal Methods for Autonomous Systems (FMAS 2023)  
EPTCS 395, 2023, pp. 205–219, doi:10.4204/EPTCS.395.15

© T. Mannucci, J. de Oliveira Filho

- We propose formal specifications for three verification properties related to the learning phase of RL algorithms:

**Quality of learning** This property measures *how well has the agent learned its environment*. It is related to the variety and frequency of experiences presented to an agent during the learning phase.

**Distance to optimal policy** This property assesses *how far the current learned policy is from the optimal policy*.

**Time to learn** A property that estimates *the amount of interactions the learning process will need to evaluate a (new) policy*.

- Along with each property, we propose the design of an RV monitor able to assess the property from observations and during the learning phase. We discuss which information should be observable from the learning phase, how to collect it systematically, and how to use observations to assess the properties. We propose ideas for the monitor’s implementation and how it can be efficiently instrumented in an RL-based system.

The paper is organized as follows. Section 2 provides a short review of related and relevant work. Section 3 introduce basic concepts of RL and RV we need to derive the verification properties. Section 4 derives formal specifications for the properties, proposes monitoring techniques and examples. Section 5 discusses our conclusions and further work.

## 2 Related Work

Verification of RL safety properties has received the main priority in the literature because RL algorithms obviously need to explore in a safe way [3, 11, 22] if they are to learn in real-world setups. Pathak et al. [11] and Zhu et al. [22] go beyond system checks and specify how the result of their verification procedure can be used to enforce the safety constraints after a system re-design. Other specific RL frameworks [1, 5, 9] use monitors to guide the system preventing the agent from violating the properties specified. Safety is an important aspect and has received significant attention in research. In this work however, we focus on other two important runtime properties of RL algorithms: learning quality and timeliness.

Verification of properties for quality of learning has received less attention than safety properties in research. There are guarantees of convergence for specific algorithms, such as Q-learning [20] and SARSA [15]. But this only means that such an algorithm will eventually learn. We differ in which we provide explicit ways to assess how much a system already learned after a set of experiences. Like us, Van Wesel and Goodloe [18] propose off-line and online verification techniques for quality-of-learning properties. However, their approach does not leverage from knowledge of the inner structure of the algorithm. Xin et al. [21] introduce the concept of *exploration entropy* to guide the learning until the final policy is of sufficient quality; their approach differs from proper verification in that it steers, and thus interferes with, the learning process.

Verification of timeliness of the learning phase is even less prominent. Szepesvari [17] investigates the rate of convergence of Q-learning, and Potapov and Ali [12] analyze the influence of learning parameters on the convergence speed. But none of them provide an approach to verify if an RL algorithm will be able to learn within a desired number of interactions. This work differs from the aforementioned previous work by (1) providing formal specifications for quality and timeliness of the RL learning process and (2) providing monitoring techniques to check such properties at runtime. The monitors we propose do not modify the behaviour of the learning phase.

This work drives from the analysis of Markov decision processes in Mannor et al. [8]. We use their approach on the calculation of estimates for the RL value function and its bias and variance estimates. We extend many of their results to define formal verification properties. And we show how to monitor the RL algorithm during the learning phase to check these properties on-line.

### 3 Fundamentals

#### 3.1 Reinforcement learning

Reinforcement Learning is a class of machine learning (ML) algorithms that solves control and decision problems. This work targets RL variants which can be modelled as finite Markov decision processes (MDP) such as Q-learning [20]. We use the finite MDP problem structure to formally derive verification properties, and later, to design the verification monitor. An MDP problem is defined by a tuple  $\{S, A, T, R, \gamma\}$ , where  $S$  is a set possible of environment states.  $A$  is a set of actions an agent can take at each state and  $T := S \times A \times S \rightarrow [0, 1]$  is a probabilistic state transition function.  $R := S \times A \times S \rightarrow \mathbb{R}$  is a reward function attributing a payoff for each state transition and action.  $\gamma \in [0, 1)$  is a discount factor over past rewards.

During the execution of the RL algorithm, an agent operates in a sequence of distinct steps. During the  $n^{th}$  step, the agent observes the current state  $s_n \in S$ , chooses and performs an action  $a \in A$ . The action is chosen according to a (probabilistic) policy  $\pi := S \times A \rightarrow [0, 1]$ . This causes the environment to transition to a subsequent state  $s_{n+1} \in S$  according to  $T$ . For its action and the new state achieved, the agent receives an instantaneous reward  $r$  according to  $R$ . Rewards obtained from state  $s$  following policy  $\pi$  are accrued into the so-called value function:

$$V^\pi(s) := E\left[\sum_{n=0}^N \gamma^n R(s_n, \pi(s_n), T(s_n, a))\right] \quad (1)$$

where  $\gamma^n$  is the  $n^{th}$  power of  $\gamma$ , and  $s_n$  is the  $n^{th}$  state encountered after starting in  $s = s_0$ . A policy is optimal (indicated as  $\pi^*$ ) if it maximizes  $V^{\pi^*}(s)$  for all states. Note that the expectation operator in Eq. 1 is due to the potential stochasticity of  $\pi$  and  $T$ . In many cases, it is convenient to define the action value function

$$Q^\pi(s, a) = E[R(s, a, T(s, a)) + \gamma V^\pi(T(s, a))] \quad (2)$$

indicating the value obtainable in  $s$  by taking action  $a$  and following the policy thereafter.

Temporal difference(TD) [16] is a method to solve RL problems when functions  $T$  and  $R$  are unknown. In this method, the value function is randomly initialized and a policy  $\pi$  is followed. The reward observed at every transition is used to correct the value function, with a chosen learning rate  $\alpha$  dictating the speed of correction. TD learning is proven to converge to a fixed value function  $V^\pi$ , given the agent has had enough and representative interactions with the environment. In section 4, we will use this fact as an intuitive notion for the quality of learning.

To define RV properties and monitors, we will use two results from Mannor et al. [8]. First, estimates  $\hat{T}$  and  $\hat{R}$  (of transition and reward functions  $T$  and  $R$ , respectively) can be reconstructed from observing transitions during the learning phase. As a consequence, it is also possible to produce a value function estimate  $\hat{V}^\pi$  directly via Eq. 1. Second, estimators for the bias and variance of the this value function estimate can be obtained as follows. Under the assumption that all state action combinations are visited at least once:

$$\text{bias}(\hat{V}^\pi) = \gamma^2 X Q V^\pi + \gamma X B + o\left(\frac{1}{\min_{s,a} N(s,a)}\right) \approx \gamma^2 X Q \hat{V}^\pi + \gamma X B; \quad (3)$$

$$\text{cov}(\hat{V}^\pi) = X W X^T + o\left(\frac{1}{\min_{s,a} N(s,a)}\right) \approx X W X^T \quad (4)$$

where  $X$ ,  $Q$ ,  $W$  and  $B$  are matrices computed from transitions and from  $\hat{T}$  and  $\hat{R}$ . The derivation and interpretation of these matrices is out of scope for this paper; the interested reader is referred to [8]. That being said, this result provides the bias and variance of the value function, which reflect the uncertainty of the agent due to lack of data. This is confirmed by the fact that  $\text{bias}(\hat{V}^\pi)$  and  $\text{cov}(\hat{V}^\pi)$  tend to zero by construction [8] if  $\min_{s,a} N(s,a) \rightarrow \infty$ . Thus estimates  $\hat{T}$ ,  $\hat{R}$  and  $\hat{V}^\pi$  will converge to their true value with more transitions.

The value function  $V^\pi$  can be used to iteratively improve the agent policy via the so-called *policy improvement*:

$$\pi_{k+1}(s) = \operatorname{argmax}_a E[R(s,a,T(s,a)) + V^{\pi_k}(T(s,a))] = \operatorname{argmax}_a Q^{\pi_k}(s,a) \quad (5)$$

which converges during a proper learning experience to the optimal policy  $\pi^*$  yielding the optimal value function  $V^{\pi^*}$ .

Replacing  $V^{\pi_k}$  with  $\hat{V}^{\pi_k}$  will yield an estimate  $\hat{V}^{\pi^*}$  of the optimal value function  $V^{\pi^*}$ . However, such an estimate will be biased, as optimization will favor actions for which the expected cumulative reward is overestimated. [8] recognizes the problem and proposes to divide the set of all transitions into a calibration set and a validation set to mitigate this inconvenience. With this, compute calibrated estimates  $\hat{T}_{\text{cal}}$ ,  $\hat{R}_{\text{cal}}$  and  $\hat{V}^{\pi_{\text{cal}}}$ , and apply policy improvement to obtain  $\pi_{\text{cal}}^*$ . From the validation set, obtain the transition and reward function estimates  $\hat{T}_{\text{val}}$  and  $\hat{R}_{\text{val}}$ . Finally, compute  $\hat{V}^{\pi_{\text{cal}}}$  via Eq. 1.

### 3.2 Runtime verification

Runtime Verification(RV) is an engineering discipline that combines (semi-)formal methods and monitoring of the system operation to check if a system's behaviour conforms to requirements. *Monitors* assess the system based on carefully collected observations of the system behaviour – called *traces*. Traces must conform to formally specified *properties*. Therefore, Bartocci et al. [2] indicate three steps to define an RV technique:

1. First, it is necessary to describe the property under verification using an unambiguous specification. Mathematical or logical formulations which can be assessed on system traces are the most common.
2. Second, it is necessary to design a monitor, which is a component able to collect and to assess traces of the system. Assessment here means any analysis steps necessary to evaluate the trace against the specified property.
3. Third, it is necessary to *instrument* the system. That is, insert observation mechanisms for correctly collecting the system traces. Good instrumentation minimally interferes with the system behaviour and performance.

This work follows these three steps for each of the proposed properties. For each property, we derive a property specification, and provide monitoring steps to observe the system and calculate the property. The monitor can be implemented to assess all the three properties concurrently and based on the same observed traces.

### 3.3 Use case: police patrol scheduling

We sketch a fictional but typical example for an RL-based learning system. In the remainder of the paper, we will use this example to illustrate the defined properties and discuss aspects relevant to the RV monitors and instrumentation.

A police department wants to use a new scheduling system for police night shifts in a city with frequent crime. This system will use an RL module that learns the most effective patrol schedules between three risk areas: the docks, the slums, and the bus station. Specifically, every hour between 00:00 am and 06:00 am, a patrol car is assigned to one of the three areas, for a total of six shifts per night. The RL algorithm for such a system has a state set  $S := \{(t, loc) | loc \in \{\text{docks, slums, station}\}, t \in [0, 5]\}$  and an action set  $A := \{\text{docks, slums, station}\}$ .

For our approach, the underlying problem structure ( $S$  and  $A$ ) must be known to the designer of the RV monitor. Neither the system nor the monitor designer knows the transition function  $T$  and the reward function  $R$  to be used. The transition function is not known because when a patrol car is sent to a location, it may take more or less time to complete the patrol. As a consequence it may miss a shift or terminate a shift early. The reward function cannot be estimated in advance as it is unknown which criminal activities can be prevented and where. However, it is decided to assign a reward between 0 and 3 in proportion to the severity of the spotted criminal activity, with 0 corresponding to no crime and 3 corresponding to a very severe crime.

## 4 Runtime Verification for quality and timeliness of RL Learning

In this section, we propose formal specifications for three verification properties related to the learning phase of RL-algorithms: *quality of learning*, *distance to optimal policy*, and *time to learn*.

### 4.1 How well has the agent learned its environment?

The first question is how to estimate if the agent has learned “enough” from its environment. Intuitively, a TD learning agent has learned enough if the current value function  $V^\pi(s)$  is close to its converged value (for all states). This choice is justified by the fact that the value function is related to both environmental stochastic functions  $T$  and  $R$ , as well as to the fact that correctly estimating  $V^\pi$  means correctly estimating the performance of the agent’s policy as well <sup>1</sup>. Unfortunately, due to the stochasticity of both policy and environment, analyzing the value function error in time can lead to premature convergence assessments. Instead, we propose using bias and covariance of the value function estimate  $\hat{V}$ . Since these reflect the lack of gathered data of the agent, they can be used to assess when enough transitions have been accumulated by the agent to learn from, even if the agent does not make direct use of the estimates  $\hat{T}$  and  $\hat{R}$ , but relies on another method to solve the MDP problem, e.g., TD learning. The procedure, based on Eq. 3 and Eq. 4, is as follows:

1. query the policy of the system  $\pi$ ;
2. read traces, assumed in the form  $\{s, a, r, s'\}$ , i.e., the MDP transitions;
3. compute off-policy estimates  $\hat{T}$  and  $\hat{R}$  based on observed transitions, as well as on-policy estimates  $\hat{T}^\pi(s, s') := \hat{T}(s, \pi(s), s')$  and  $\hat{R}^\pi(s) := \hat{R}(s, \pi(s), \hat{T}(s, \pi(s)))$ ;

---

<sup>1</sup>Assuming that  $\pi$  and  $T$  act ergodically concerning  $S$  and  $A$ , i.e., that all state-action combinations are visited with non-zero probability.

4. compute value  $\hat{V}^\pi$  via Eq. 1.
5. compute matrices  $X$ ,  $W$ ,  $B$  and  $Q$  from  $\hat{T}$  and  $\hat{R}$ ;
6. compute bias and covariance, ignoring in first approximation the  $o(\frac{1}{\min_{s,a} N(s,a)})$  term;
7. compute relative bias  $bias_{rel} := bias(\hat{V}^\pi(s))/\hat{V}^\pi(s)$  and relative variance  $\sigma_{rel} := \sigma(\hat{V}^\pi(s))/\hat{V}^\pi(s)$ ;
8. compare relative bias and variance with predefined upper thresholds; if for all states the two are below their respective thresholds, the property is satisfied.

The procedure is straightforward to follow and implement but presents a few limitations as well. First, the monitor must have the memory to store past traces to be able to recompute the estimates  $\hat{T}^\pi$  and  $\hat{R}^\pi$ . Second, the monitor must have access to the policy of the system. This is different than observing a signal trace in that the policy is not a signal, but a function utilized internally by the system. In case the policy is not observable within the system, it would be recommendable to use an estimate  $\hat{\pi}$  from the observed actions in the trace. In this case, however, some error in the estimated bias and covariance can be expected given that the policy is an estimate in itself. Third, the procedure described here is not incremental, i.e., it does not provide for a method to update the estimates of bias and covariance at time  $k+1$  given the trace at time  $k+1$  and previous estimates of bias and covariance at time  $k$ ; however, the calculations can be repeated by storing the previous traces. Finally, the method as presented requires that all state-action combinations are visited at least once (i.e.,  $\min_{s,a} N(s,a) > 0$ ). If this condition is not verified, then the bias and covariance cannot be computed.

### Example

Looking back at the example sketched in Sec.3.3, imagine the RL scheduling system has been provided with an initial exploratory policy assumed to perform decently, so as not to waste the patrolling effort while the RL system gathers information. How long should information be gathered with this policy? To answer this question, a monitor is designed, following the given procedure, to verify the property

$$\max_s bias_{rel}(s) < 0.05 \bigwedge \max_s \sigma_{rel}(s) < 0.02 \quad (6)$$

which indicates that the bias and variance are small, respectively 2% and 5% of the estimated value, and therefore the epistemic uncertainty on the value function is low (note that these thresholds are for illustration purposes). The monitor shall inform whether this property is violated, or unverified<sup>2</sup>, or satisfied at each moment. Note that the monitor will not be able to predict when the property will be satisfied, it can only say if it is so at the current time. Furthermore, the fact that the property is initially unsatisfied does not mean that it cannot be satisfied eventually.

After a sufficient amount of traces is collected, so that  $\min_{s,a} N(s,a) > 0$ , the initial estimates of bias and variance can be generated. However,  $\hat{T}$  and  $\hat{R}$  will initially be poor estimates of  $T$  and  $R$ , so the corresponding bias and variance are likely to be outside of the ranges provided by Eq. 6. Therefore, the monitor will produce a “property violated” result. The longer the trace, however, the more  $\hat{T}$  and  $\hat{R}$  will resemble the true matrices. Accordingly, bias and variance will reduce, until eventually Eq. 6 will be true. The monitor will then produce a “property satisfied” response.

<sup>2</sup>It might appear unsound that the monitor shall be able to report that the property is unverified since the inequality formulation of the property can in theory be always verified. However, at the start of the exploration, the condition  $\min_{s,a} N(s,a) > 0$  will not be verified for the applicability of the method. Therefore, the monitor will produce a “property unverified” response.

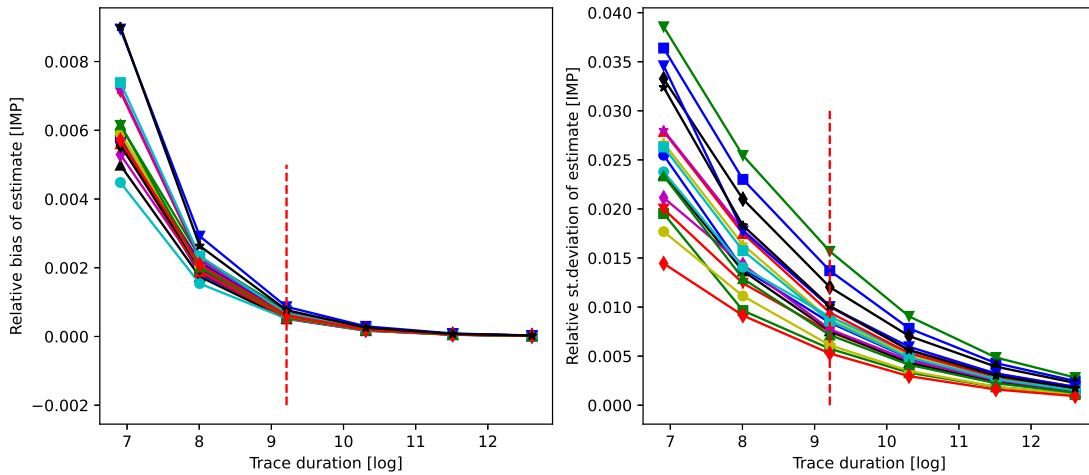


Figure 1: Evolution of relative bias and variance in the RL value function (estimates). The vertical dashed line indicates the moment of convergence after which all  $bias_{rel}$  and  $\sigma_{rel}$  are below the limits set by the property, and the property is satisfied.

Figure 1 shows the relative bias and standard deviation of Eq. 6 (plotted in logarithmic scale). It can be seen how both bias and standard deviation decay to zero, in agreement with the theory. The vertical dashed line indicates the iteration at which the property is satisfied. It is possible to empirically verify the correctness of the monitor response by confronting the value function obtainable from the actual matrices  $T$  and  $R$  versus the one that can be computed from estimates  $\hat{T}$  and  $\hat{R}$  at different iterations of the monitoring. Figure 2 shows the relative error  $\frac{V(s) - \hat{V}(s)}{\hat{V}(s)}$  for all 18 states. It can be seen that this error is initially very high, indicating that  $\hat{T}$  and  $\hat{R}$  are bad estimates. However, the error reduces sensibly with the increase in iterations. At the iteration for which the property is positively validated, it can be seen that the absolute error at such iteration is lower than the estimated bias, confirming the indication of the monitor that both  $T$  and  $R$  are reasonably learned.

## 4.2 How far from the optimum is the current policy?

If the optimal value function  $V^{\pi^*}$  was known, one could compute how well  $\pi$  is faring compared to  $\pi^*$ . Unfortunately, the optimal value function is not available before learning is concluded. However, it is possible to use the estimate  $\hat{V}^{\pi^*}$  as given in Sec. 3 in first approximation.

To simplify the exposition, assume the case of positive definite reward:  $R(s, a, s') \geq 0$ . In this case, both  $\hat{V}^{\pi}$  and  $V^{\pi^*}$  are positive by construction, so that an *optimality ratio*  $\eta(s) := \frac{\hat{V}^{\pi}(s)}{V^{\pi^*}(s)}$  can be defined: if the ratio is sufficiently high for all states, this indicates that the policy is “almost optimal”.

Under the assumption that both value functions  $\hat{V}^{\pi}$  and  $\hat{V}^{\pi_{cal}} \approx \hat{V}^{\pi^*}$  are normally distributed, it is possible to bound the optimality ratio of  $\pi$ . Given that the  $i^{th}$  diagonal elements  $\sigma^2$  of the covariance matrix  $cov(\hat{V}^{\pi})$  coincide with the variance of the value  $\hat{V}^{\pi}(s_i)$  for the  $i^{th}$  state  $s_i$ , a 95% confidence interval in  $V^{\pi}$  can be computed.

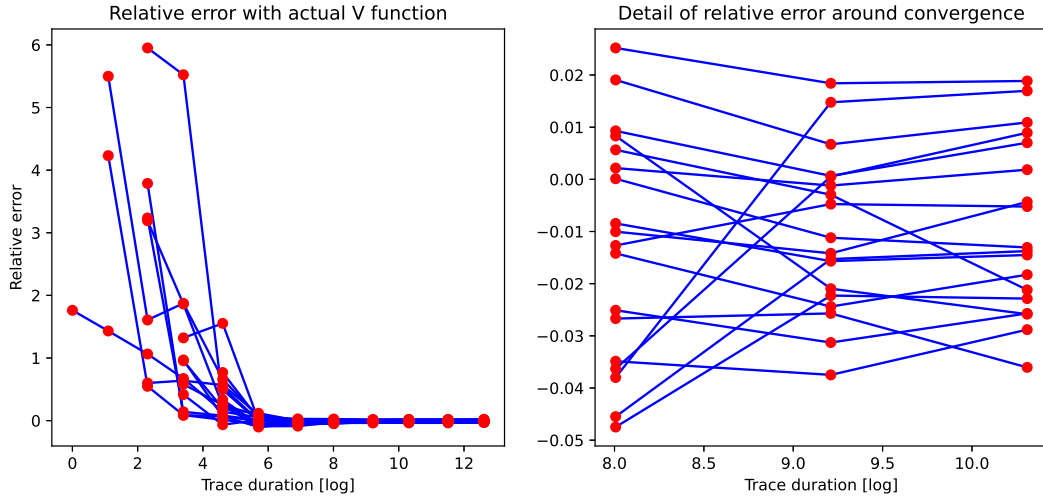


Figure 2: Relative error between  $V(s)$  and  $\hat{V}(s)$  (left); detail when the property is verified (right). This relative error is an indicator of how well the agent has learned the transition function  $T$  and the reward function  $R$ .

$$\lfloor V^\pi \rfloor := \max(0, \hat{V}^\pi - \text{bias}(\hat{V}^\pi) - 2\sigma(\hat{V}^\pi)); \lceil V^\pi \rceil := \max(0, \hat{V}^\pi - \text{bias}(\hat{V}^\pi) + 2\sigma(\hat{V}^\pi)), \quad (7)$$

and similarly for  $\hat{V}^{\pi^*}$ :

$$\lfloor V^{\pi^*} \rfloor := \max(0, \hat{V}^{\pi^*} - \text{bias}(\hat{V}^{\pi^*}) - 2\sigma(\hat{V}^{\pi^*})); \lceil V^{\pi^*} \rceil := \max(0, \hat{V}^{\pi^*} - \text{bias}(\hat{V}^{\pi^*}) + 2\sigma(\hat{V}^{\pi^*})), \quad (8)$$

and thus  $\eta$  is bounded as  $\underline{\eta} \leq \eta \leq \bar{\eta}$ , with

$$\underline{\eta} = \lfloor V^\pi \rfloor / \lceil V^{\pi^*} \rceil; \bar{\eta} = \min(1, \lceil V^\pi \rceil / \lfloor V^{\pi^*} \rfloor), \quad (9)$$

again within a 95% confidence interval. Omitting the dependency from  $s$  for legibility, the procedure is as follows.

1. divide the trace into a calibration set and a validation set;
2. utilize the calibration set to obtain the transition and reward function estimates  $\hat{T}_{\text{cal}}$  and  $\hat{R}_{\text{cal}}$ ;
3. obtain the optimal policy  $\pi_{\text{cal}}^*$  via iterated policy improvement;
4. utilize the validation set to obtain the transition and reward function estimates  $\hat{T}_{\text{val}}$  and  $\hat{R}_{\text{val}}$ ;
5. compute the value function  $\hat{V}^{\pi_{\text{cal}}^*}$ , as well as the bias  $\text{bias}(\hat{V}^{\pi_{\text{cal}}^*})$  and covariance matrix  $\text{cov}(\hat{V}^{\pi_{\text{cal}}^*})$  substituting  $\hat{T}_{\text{val}}$  and  $\hat{R}_{\text{val}}$  for  $\hat{T}$  and  $\hat{R}$ ;
6. compute upper and lower bounds  $\lfloor V^\pi \rfloor$ ,  $\lceil V^\pi \rceil$  and  $\lfloor V^{\pi^*} \rfloor$ ,  $\lceil V^{\pi^*} \rceil$ , for all states;
7. compute upper and lower bounds on the optimality ratio  $\underline{\eta}$  and  $\bar{\eta}$ , again for all states;
8. compare  $\underline{\eta}$  and  $\bar{\eta}$  with predefined lower thresholds; if for all states the two are above their respective thresholds, the property is satisfied.



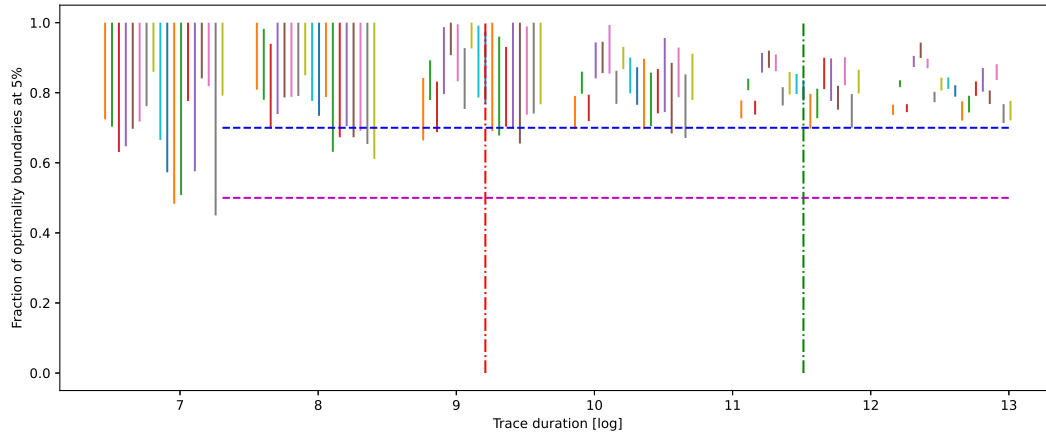


Figure 3: Optimality boundaries  $\underline{\eta}$  and  $\bar{\eta}$ , including property thresholds (horizontal dashed lines) and convergence iterations (vertical dot-dashed lines).

### Example

Consider once more the recurring use case example of police patrol. Initially, the system has been provided with a sensible exploratory policy. This is not likely to be the optimal one, but it could be close enough to optimality to not be worth changing. Conversely, it could be so suboptimal to warrant a change of policy. This loose intuition of “distance from optimum” is encoded in the property:

$$\underline{\eta}(s) \geq 50\% \wedge \bar{\eta}(s) \geq 70\% \forall s \quad (10)$$

which guarantees that  $\pi$  is not too far from the optimum (based on  $\underline{\eta}$ ) as well as indicating that  $\pi$  is potentially close to the optimum (based on  $\bar{\eta}$ ). To this must be added the condition

$$\forall s \max bias_{rel}(s) < 0.05 \wedge \max \sigma_{rel}(s) < 0.02, \quad (11)$$

on both  $V^\pi$  and  $V^{\pi^*}$ , to ensure that all estimates of  $T$  and  $R$ , which are necessary to compute the bounds on  $\eta$ , are accurate.

To verify this property (under this condition), it is necessary to first estimate the range of the optimum  $V^{\pi^*}$ . In this example, the monitor utilizes a random calibration set equal to 5% of the total traces to estimate the optimum policy. The remainder of 95% of the traces are utilized to estimate the value function  $V^{\pi^*}$  as well as the bias and variance following the procedure introduced in this section. After that, it is possible to compute the optimality ratio boundaries  $\underline{\eta}$  and  $\bar{\eta}$ .

Figure 3 shows the optimality ratio boundaries computed at different times during a sample run of the system. The optimality boundaries are indicated via vertical error bars. The dashed horizontal lines indicate the 50% and 70% optimality thresholds of in Eq. 10. The vertical lines indicate at which iteration the value function bias and standard deviation satisfy the property in Eq. 6 for the policy value function  $V^\pi$  (left) and for the optimal value function  $V^{\pi^*}$  (right) respectively. The example shows that Eq. 10 is punctually verified after Eq. 11 is satisfied.

As can be seen from Figure 3, bounds on  $\eta$  shrink with the trace duration; this is due both to the reduction in bias and variance of the estimated optimal value function, as well as due to an actual im-

provement of the calibrated policy  $\pi_{cal}$  due to additional learning. This continuous improvement also explains why error bars at a given iteration are not always contained within the error bars of the previous iterations.

### 4.3 How long will it take to evaluate a (new) policy?

We consider three scenarios when an RL system must learn or update a policy:

1. the system attempts to introduce a new policy  $\pi'$  for the same functionality;
2. the system functionality change, so that a new unknown reward function  $R'$  takes effect;
3. the system environment changes, so that a new unknown transition function  $T'$  takes effect.

Estimating how long this will take might be relevant, especially if there is a finite amount of resources to do so. This section determines an upper bound to the required number of iterations. For all cases, we assume that both policy and learning rate  $\alpha$  are stationary. We also assume that the reward function  $R$  is bounded and deterministic. To derive this property, we assume an on-policy RL algorithm is used. And our monitor design needs an estimate of the transition function  $\hat{T}$ . We discuss this assumption in details when illustrating the third case.

#### 4.3.1 New policy

We analyze first the case when the system desires to implement a different policy, but neither the functionality nor the environment is any different. Estimates  $\hat{T}$  and  $\hat{R}$  are still valid and refer to a well-learned environment. To predict the learning time, it is necessary to predict how long  $V^\pi$  will take to converge to its new value. Consider first the case in which the entire value function  $V^\pi$  or action value function  $Q^\pi$  can be re-estimated in *updates*, with each update covering the full state space or state-action space, until convergence. This can be done if both  $R$  and  $T$  are known. In this case, it can be demonstrated that for an on-policy value update, such as a SARSA [16], the expected consecutive difference  $\Delta := Q_{k+1}^\pi - Q_k^\pi$  before and after an update decays by a factor  $\hat{\gamma} = 1 - \alpha(1 - \gamma)$  at each update iteration (see Appendix for details). Furthermore, assuming  $R$  is not stochastic, the decay bound holds at every iteration. Defining convergence as  $\|\Delta\| < \varepsilon$ , it can be seen that for an on-policy update,  $Q$  will converge in  $M_u$  updates, if

$$\hat{\gamma}^{M_u} \|\bar{\Delta}\| < \varepsilon \Rightarrow M_u = \text{ceil}\left(\frac{\log(\frac{1}{\varepsilon} \|\bar{\Delta}\|)}{\log \frac{1}{\hat{\gamma}}}\right), \quad (12)$$

where  $\text{ceil}(x)$  is the ceiling function of  $x$ , and  $\bar{\Delta}$  is an upper estimate of the initial consecutive difference. This is maximum in case  $V_0^\pi(s) = Q_0^\pi(s, a) = R_{min}/(1 - \gamma)$  and  $V_1^\pi(s) = Q_1^\pi(s, a) = R_{max}/(1 - \gamma)$ ,  $\forall s, a$ . Therefore, a rigorous upper bound on convergence can be found when this is re-estimated via full state or state-action updates<sup>3</sup>.

However, such updates seldom apply in practice, with state visits determined by the stochasticity of  $\pi$  and  $T$ . In this case, it is necessary to reduce the above estimation in terms of  $M_u$  updates to a different boundary  $M_t$  in terms of *state transitions*. During updates all states or state-action combinations are visited equally often, which is not true for state transitions. However, one could make the rough assumption that if  $M_u$  updates are necessary to converge then the same convergence can be reached when

---

<sup>3</sup>Note that, even though the derivation of  $M_u$  makes use of both  $T$  and  $R$ , we do not imply that the RL agent being monitored will make direct use of these quantities.

each state is visited  $M_u$  times (whether or not this assumption is valid will be discussed in due time). Following this first assumption, it is possible to convert the previous bound into

$$M_t = M_u \max_{s,a} \tau(s,a|T, \pi, s_{\text{init}}), \quad (13)$$

where  $\tau(s,a)$  indicates the period, i.e., the number of transitions between two consecutive visits of state  $s$  in which action  $a$  is selected (also known as the *revisit time*). It is immediate to see that in the most general case, the revisit time is an unbounded stochastic variable; a first approximation of  $\tau$  can nonetheless be obtained from the steady-state state probabilities  $p$  of the stochastic transition matrix  $T$ . This can be found as the solution to the following eigenvector problem:

$$\mathbf{p} := \{p(s_0), \dots, p(s_N)\} \Rightarrow \mathbf{p} = T\mathbf{p}, \sum_{i=0}^N p(s_i) = 1, p(s_i) \geq 0. \quad (14)$$

Note that to compute such steady-state probability it is necessary to know  $T$ . That being said, Eq. 13 can be rewritten in first approximation as:

$$M_t = M_u \max_{s,a} [p(s)\pi(s,a)]^{-1}. \quad (15)$$

Note that the naive assumption of Eq. 13 that  $M_u$  visits at each state-action pair are equivalent to  $M_u$  updates does not hold in general. Indeed, updates not only guarantee an equal visit count among all states but also an equal “mixing” of visits, so that the entire value or action value function is re-estimated evenly. That being said, in the absence of pathological graphs induced by  $T$  (e.g., with absorbing states), one can utilize Eq. 15 as a first upper bound estimate.

Also note that Eq. 15 yields  $M_t \rightarrow \infty$  in case either  $p(s) \rightarrow 0$  or  $\pi(s,a) \rightarrow 0$ , i.e., if a state is not reachable or an action for a reachable state is never selected by the policy. In both cases, the limitation is more theoretical than practical. Consider the case  $p(\tilde{s}) = 0$  and assume for simplicity that the environment is not in  $\tilde{s}$  at the start of exploration: since  $\tilde{s}$  (or  $(\tilde{s}, a)$ ) cannot be reached, this never affects the value computation in Eq. 1 except for  $V(\tilde{s})$  (or  $Q(\tilde{s}, a)$ ), and  $\tilde{s}$  can be effectively ignored. The same considerations apply for  $\pi(s,a) = 0$ . By extension, a state-action couple for which  $p(s)\pi(s,a)$  is smaller than an arbitrarily small negligibility threshold  $\omega \geq 0$  will also have a negligible effect on the policy evaluation and can also be ignored. Hence, the new upper bound for convergence in terms of state transitions is

$$M_t = M_u \max_{(s,a) \in \Omega} [p(s)\pi(s,a)]^{-1}, \text{ where } \Omega := \{(s,a) | p(s)\pi(s,a) \geq \omega\}. \quad (16)$$

Finally, Eq. 16 estimates the transitions required to evaluate the new policy. Selecting a proper  $\omega > 0$  is not an immediate choice but requires some insight into the problem at hand. In the absence of this, a conservative estimate can be made by selecting  $\omega = 0$ , thus ensuring that  $M_t$  will be finite.

Summarizing, the procedure is as follows:

1. select arbitrarily small coefficients  $\varepsilon$  and  $\omega$ ;
2. compute bound  $M_u$ ;
3. solve eigenvalue problem of Eq. 14 given  $T$ ;
4. compute bound  $M_t$  from  $M_u$ ,  $\mathbf{p}$  and  $\pi$ ;
5. if  $M_t$  is lower than a predefined upper threshold, the property is verified.

### 4.3.2 New reward

The case of a new reward is identical to the case of a new policy. Indeed, Eq. 16 can be directly reutilized, with the caveat that  $M_u$  must be estimated according to the new values  $R_{min}$  and  $R_{max}$  if these differ from the previous.

### 4.3.3 New environment

When considering a new environment,  $T$  can change from what was previously estimated by the monitor. In principle, this means that the approach investigated in this section is no longer valid due to the unknown state probability  $p(s)$ . Therefore, the monitor will first need to reestimate  $\hat{T}$  (as per the first property discussed). Before that, no estimation can be provided.

That being said, a very conservative, worst-case estimation can still be given if one were to provide a positive negligibility  $\omega$  on state-action visits. In that case, a worst-case estimation can be provided in the form:

$$\overline{M}_t = M_u \frac{1}{\omega} \geq M_t. \quad (17)$$

To apply the previous estimate  $M_t$  for runtime monitoring, it is important to make a few observations. First, the convergence and negligibility constants  $\varepsilon$  and  $\omega$  must be provided to the monitor (to use in Eq. 12 and Eq. 16). Second, the monitor computes the steady-state probability  $\mathbf{p}$  of Eq. 14 by replacing  $T$  with  $\hat{T}$ . If such an estimate is missing, the monitor can either provide an immediate response using the worst-case estimate of Eq. 17 or return an unverified response until the property in Eq. 6 is verified.

### Example

Consider once more the police patrol case, and assume that informants notify police authorities of rumoured changes in the location and time of criminal activities. As a result, the initial policy  $\pi$  might no longer be satisfactory, due to the unspecified change of the reward function  $R$ , and reestimating its value will take time. Police officials want to know if the time it will take for the RL system to reestimate correctly the value of  $\pi$  is acceptable, i.e., if the following property holds:

$$M_t \leq M_t^{\max}, \quad (18)$$

where  $M_t^{\max}$  can be derived from, e.g., a time constraint. A monitor is then deployed to evaluate the property: this must have access to the estimate  $\hat{T}$  (since  $T$  is unchanged, the estimate is still valid); as well as to the potentially new quantities  $R_{min}$  and  $R_{max}$ .

Thus, the monitor can employ Eq. 16 to estimate  $M_t$  and therefore to verify the property of Eq. 18 before each policy improvement step. As an example, consider the case in which  $\pi$  dictates a fixed patrol schedule in the form

$$\pi(t, loc) = \begin{cases} \text{slums} & \text{if } t \leq 1 \\ \text{station} & \text{if } t = 2 \\ \text{docks} & \text{else.} \end{cases} \quad (19)$$

The monitor can now estimate the number of transitions necessary to evaluate this policy. Given learning parameters  $\alpha = 0.75$ ,  $\gamma = 0.5$ , constants  $\varepsilon = 0.05$  and  $\omega = 0$ , and assuming  $R_{min}$  and  $R_{max}$  to be 0 and 3 once more, the monitor estimates 62 transitions necessary for convergence.

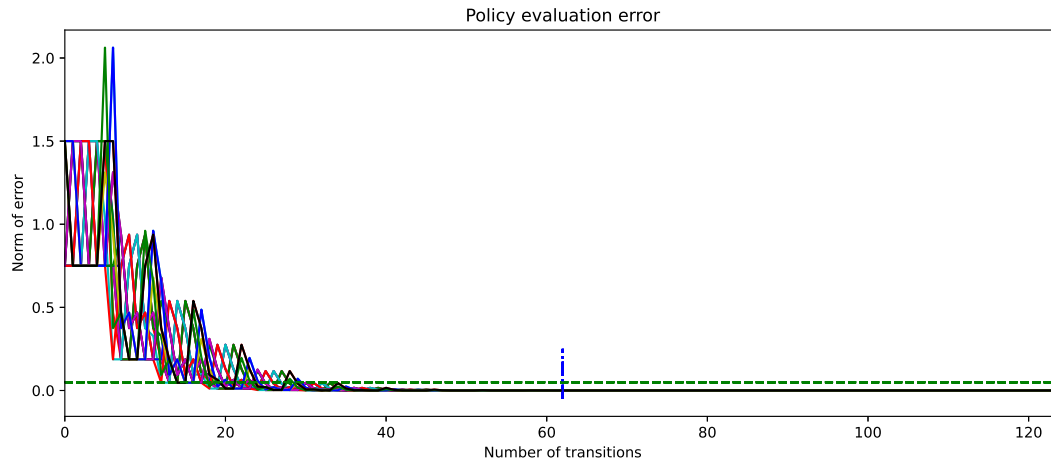


Figure 4: Norm of value function error  $\|\Delta\|$  for all initial states. The horizontal dotted line indicates the convergence threshold  $\epsilon$ , while the vertical dot-dashed line indicates the expected convergence time.

Figure 4 shows the actual amount of transitions necessary to estimate the policy value. The continuous lines indicate the change in value function error  $\|\Delta\|$  for each of the 18 initial states. It can be seen that  $\|\Delta\|$  reduces as the number of transitions increases, albeit not monotonically. It can also be seen that approximately at 30 transitions, the error for all initial states is below the threshold of  $\epsilon = 0.05$ , indicated by the dashed horizontal line. Thus, the value  $M_t$  for this example was indeed a conservative estimate.

## 5 Conclusions

In this paper, we propose three verification properties for the learning phase of reinforcement learning (RL) agents. The first property relates to the quality of the learning phase. It expresses whether or not the agent has learned from sufficiently enough and sufficiently varied experiences, to have a suitable representation of its environment. We devise a runtime verification monitoring technique to assess this property by estimating the variance and bias of the learned value function. This property enables the verification of the second and third properties.

The second property measures the actual learned policy relative to the ideal optimum. We extend our monitoring techniques to derive an optimality ratio  $\eta$ . The verification monitor uses confidence intervals of  $\eta$  to indicate upper and lower bounds on optimality. Our RV monitor can check whether the current policy guarantees minimal satisfactory behaviour and whether the policy could potentially be close to optimal.

The third property checks if the learning time falls within a desired number of interactions. It can be used when a new policy is put in place, or when a known policy is applied to a new transition function  $T$  or a new reward function  $R$ . Verification of such property is relevant to systems that require estimating the value induced by a new policy or new environment within a limited time. In this case, the evaluation time is a rough estimate, due to the assumption of Eq. 13 on the equivalence between updates and state transitions. In future work, we will improve this estimate by examining the mixing properties of the graph induced by  $\pi$  on the transition matrix  $T$ . For this property, we discuss the monitoring techniques necessary to assess the property using the system's runtime observations.

Our future work includes incorporating these techniques into a runtime verification tool for autonomous robots. We will also expand the formulation of our properties and monitoring techniques towards Deep Reinforcement Learning algorithms (DRL). DRL algorithms typically use continuous state- and action spaces, which our approach cannot yet cover.

## References

- [1] Greg Anderson, Abhinav Verma, Isil Dillig & Swarat Chaudhuri (2020): *Neurosymbolic Reinforcement Learning with Formally Verified Exploration*. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS'20*, Curran Associates Inc., Red Hook, NY, USA, doi:10.48550/arXiv.2009.12612.
- [2] Ezio Bartocci, Yliès Falcone, Adrian Francalanza & Giles Reger (2018): *Introduction to runtime verification*. In: *Lectures on Runtime Verification*, Springer, pp. 1–33, doi:10.1007/978-3-319-75632-5.
- [3] Davide Corsi, Enrico Marchesini & Alessandro Farinelli (2021): *Formal verification of neural networks for safety-critical tasks in deep reinforcement learning*. In: *Uncertainty in Artificial Intelligence*, PMLR, pp. 333–343, doi:10.48448/tj1d-sk77.
- [4] Rudiger Ehlers (2017): *Formal verification of piece-wise linear feed-forward neural networks*. In: *International Symposium on Automated Technology for Verification and Analysis*, Springer, pp. 269–286, doi:10.1007/978-3-319-68167-2\_19.
- [5] Nathan Hunt, Nathan Fulton, Sara Magliacane, Trong Nghia Hoang, Subhro Das & Armando Solar-Lezama (2021): *Verifiably safe exploration for end-to-end reinforcement learning*. In: *HSCC '21: 24th ACM International Conference on Hybrid Systems: Computation and Control, Nashville, Tennessee, May 19-21, 2021*, ACM, pp. 14:1–14:11, doi:10.1145/3447928.3456653.
- [6] Zachary Kenton, Angelos Filos, Owain Evans & Yarin Gal (2019): *Generalizing from a few environments in safety-critical reinforcement learning*. doi:10.48550/arXiv.1907.01475.
- [7] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley & Jianfeng Gao (2016): *Deep Reinforcement Learning for Dialogue Generation*. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Austin, Texas, pp. 1192–1202, doi:10.18653/v1/D16-1127. Available at <https://aclanthology.org/D16-1127>.
- [8] Shie Mannor, Duncan Simester, Peng Sun & John N Tsitsiklis (2007): *Bias and variance approximation in value function estimates*. *Management Science* 53(2), pp. 308–322, doi:10.1287/mnsc.1060.0614.
- [9] George Mason, Radu Calinescu, Daniel Kudenko & Alec Banks (2017): *Assured Reinforcement Learning with Formally Verified Abstract Policies*. In: *Proceedings of the 9th International Conference on Agents and Artificial Intelligence, ICAART 2017, Volume 2, Porto, Portugal, February 24-26, 2017*, SciTePress, pp. 105–117, doi:10.5220/0006156001050117.
- [10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra & Martin Riedmiller (2013): *Playing atari with deep reinforcement learning*. doi:10.48550/arXiv.1312.5602.
- [11] Shashank Pathak, Luca Pulina & Armando Tacchella (2018): *Verification and repair of control policies for safe reinforcement learning*. *Appl. Intell.* 48(4), pp. 886–908, doi:10.1007/s10489-017-0999-8.
- [12] Alex Potapov & MK Ali (2003): *Convergence of reinforcement learning algorithms and acceleration of learning*. *Physical Review E* 67(2), p. 026706, doi:10.1103/PhysRevE.67.026706.
- [13] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan & Demis Hassabis (2017): *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. *CoRR* abs/1712.01815, doi:10.48550/arXiv.1712.01815.
- [14] Bharat Singh, Rajesh Kumar & Vinay Pratap Singh (2021): *Reinforcement learning in robotic applications: a comprehensive survey*. *Artificial Intelligence Review* 55(2), pp. 945–990, doi:10.1007/s10462-021-09997-9.

- [15] Satinder Singh, Tommi S. Jaakkola, Michael L. Littman & Csaba Szepesvári (2000): *Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms*. *Mach. Learn.* 38(3), pp. 287–308, doi:10.1023/A:1007678930559.
- [16] Richard S Sutton & Andrew G Barto (2018): *Reinforcement learning: An introduction*. MIT press, doi:10.1109/TNN.1998.712192.
- [17] Csaba Szepesvári (1997): *The Asymptotic Convergence-Rate of Q-learning*. In: *Advances in Neural Information Processing Systems 10, [NIPS Conference]*, The MIT Press, pp. 1064–1070, doi:10.5555/3008904.3009053.
- [18] Perry Van Wesel & Alwyn E Goodloe (2017): *Challenges in the verification of reinforcement learning algorithms*. Technical Report. Available at <https://ntrs.nasa.gov/citations/20170007190>.
- [19] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev et al. (2019): *Grandmaster level in StarCraft II using multi-agent reinforcement learning*. *Nature* 575(7782), pp. 350–354, doi:10.1038/s41586-019-1724-z.
- [20] Christopher J.C.H. Watkins & Peter Dayan (1992): *Machine Learning* 8(3/4), pp. 279–292, doi:10.1023/a:1022676722315.
- [21] Bo Xin, Haixu Yu, You Qin, Qing Tang & Zhangqing Zhu (2020): *Exploration entropy for reinforcement learning*. *Mathematical Problems in Engineering* 2020, doi:10.1155/2020/2672537.
- [22] He Zhu, Zikang Xiong, Stephen Magill & Suresh Jagannathan (2019): *An inductive synthesis framework for verifiable reinforcement learning*. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, ACM, pp. 686–701, doi:10.1145/3314221.3314638.

## Appendix

First, observe that the action value update can be rewritten as

$$Q_{k+1}^\pi(s, a) = (1 - \alpha)Q_k^\pi(s, a) + \alpha \sum_{s' \in S} T(s, a, s') [r(s, a, s') + \gamma \sum_{a'} \pi(s', a') Q_k^\pi(s', a')].$$

Define now the matrix  $\Pi \in \mathbb{R}^{|S| \times |A| \times |S|}$ , whose entries are defined as:

$$\Pi_{i,j} = \begin{cases} \pi(s_j, a_{i-(j-1)|A|}) & \text{if } (j-1)|S||A| < i \leq j|S||A| \\ 0 & \text{otherwise.} \end{cases}$$

$\Pi$  is the matrix obtained by staking  $|S|$  copies of the vector-wise policy  $\pi \in \mathbb{R}^{|S| \times |A|}$ , and setting to zero each element of the resulting  $j^{\text{th}}$  column which does not contain the probabilities for the  $j^{\text{th}}$  state. Denote as  $\mathbf{Q} \in \mathbb{R}^{|S| \times |A|}$  the column vector of values  $Q^\pi(s, a)$ , as  $\mathbf{R} \in \mathbb{R}^{|S| \times |A|}$  the column vector of the expected rewards  $E_{s' \in S}[r(s, a, s')]$ , and as  $\mathbf{T} \in \mathbb{R}^{|S| \times |A| \times |S|}$  the matrix corresponding to the transition function  $T$ . Then the value update can be written in matrix form as

$$\mathbf{Q}_{k+1} = \alpha \mathbf{R} + [\alpha \gamma \mathbf{T} \Pi^T + (1 - \alpha) \mathbf{I}] \mathbf{Q}_k = \alpha \mathbf{R} + \mathbf{B} \mathbf{Q}_k,$$

where  $\mathbf{I}$  is the identity matrix. Note now that the square matrix  $\mathbf{B} := [\alpha \gamma \mathbf{T} \Pi^T + (1 - \alpha) \mathbf{I}]$  has norm  $\|\mathbf{B}\| \leq \alpha \gamma + 1 - \alpha = 1 - \alpha(1 - \gamma)$  by construction, because  $\alpha, \gamma \leq 1$ , and  $\mathbf{T}, \Pi$  are stochastic matrices.

If we define the consecutive error vector as  $\Delta_k := \mathbf{Q}_{k+1} - \mathbf{Q}_k$ . Then it follows that  $\Delta_{k+1} = \mathbf{B} \Delta_k$  indicates a contraction since  $1 - \alpha(1 - \gamma) \leq 1$ .