Journal of Machine Learning Research 17 (2016) 1-31 Submitted 1/14; Revised 8/15; Published 4/16

Addressing Environment Non-Stationarity by Repeating Q-learning Updates*

Sherief Abdallah SHARIO@IEEE.ORG
The British University in Dubai, P.O.Box 345015, Block 11, DIAC, Dubai, United Arab Emirates
University of Edinburgh, United Kingdom

Michael Kaisers KAISERS@CWI.NL
Centrum Wiskunde & Informatica, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Maastricht University, The Netherlands

Editor: Jan Peters

Abstract

Q-learning (QL) is a popular reinforcement learning algorithm that is guaranteed to converge to optimal policies in Markov decision processes. However, QL exhibits an artifact: in expectation, the effective rate of updating the value of an action depends on the probability of choosing that action. In other words, there is a tight coupling between the learning dynamics and underlying execution policy. This coupling can cause performance degradation in noisy *non-stationary* environments.

Here, we introduce Repeated Update Q-learning (RUQL), a learning algorithm that resolves the undesirable artifact of Q-learning while maintaining simplicity. We analyze the similarities and differences between RUQL, QL, and the closest state-of-the-art algorithms theoretically. Our analysis shows that RUQL maintains the convergence guarantee of QL in stationary environments, while relaxing the coupling between the execution policy and the learning dynamics. Experimental results confirm the theoretical insights and show how RUQL outperforms both QL and the closest state-of-the-art algorithms in noisy non-stationary environments.

Keywords: reinforcement learning, Q-learning, multi-agent learning, non-stationary environments

1. Introduction

Q-Learning (Watkins and Dayan, 1992), QL, is one of the most widely-used and widely-studied learning algorithms due to its ease of implementation, intuitiveness, and effectiveness over a wide-range of problems (Sutton and Barto, 1999). Although QL was originally designed for single-agent domains, it has also been used in multi-agent settings with reasonable success (Claus and Boutilier, 1998). In addition, several gradient-based learning algorithms, which were invented specifically for multi-agent interactions, rely on Q-learning as an internal component to estimate the values of different actions (Abdallah and Lesser, 2008; Bowling, 2005; Zhang and Lesser, 2010). In this paper, we propose a novel algorithm that addresses important limitations of the Q-learning algorithm while retaining Q-learning's attractive simplicity. We show that our algorithm outperforms Q-learning and closely related state-of-the-art algorithms in several noisy and *non-stationary* environments.

*. An earlier version of this paper was presented at the International conference on Autonomous Agents and Multi-Agent Systems (Abdallah and Kaisers, 2013).

The main problem that our algorithm addresses is what we refer to as the *policy-bias* of the action value update. The policy-bias problem appears in Q-learning because the value of an action is only updated when the action is executed. Consequently, the *effective* rate of updating an action value directly depends on the probability of choosing the action for execution. If, hypothetically, the reward information for all actions (at every state) were available at every time step, then the update rule could be applied to all actions and the policy-bias would disappear. Let us refer to this hypothetical update as the *QL-ideal-update*.[1]

It is important to note that the policy-bias problem is different from the exploration-exploitation problem. To solve the exploration-exploitation problem, researchers optimize the execution policy in order to balance exploration vs. exploitation. However, the policy-bias refers to the dependency of the *rate of updating* an action's value on the *probability of selecting* the corresponding action; this dependency is not resolved by changing the policy. As previous research showed, the policy bias may cause a temporary decrease in the probability of choosing optimal actions with higher expected payoffs (Leslie and Collins, 2005; Kaisers and Tuyls, 2010; Wunder et al., 2010). This effect can be more severe, as we show later, in noisy non-stationary environments where the environment dynamics change over time.

Figure 1 illustrates the policy-bias problem using a very simple multi-armed-bandit domain. The domain is stateless with only two actions, one giving the reward of -1 and the other giving the reward of 1. For simplicity, let us assume the best value for the learning rate α, is 0.01 (as we highlight in our experiments later, the best value of the learning rate depends on how noisy the specific domain is). To measure the deviation between a learning algorithm and the QL-ideal-update, we use RMSE: the root mean square error of the Q values (for all actions) compared to the Q values of the QL-ideal-update over 1000 time steps. Figure 1 plots the RMSE of QL for different learning rates and over different execution policies (the probability of choosing the first action), which is fixed throughout the 1000 time steps.[2] We can observe from the figure that the RMSE of QL is minimal when the probability of choosing both actions are equal (pure exploration). However, as the execution policy deviates from the uniform distribution (toward exploitation), the RMSE of QL grows rapidly. It is also important to note that higher learning rate (α) will not lower the RMSE of QL across different execution policies. The RMSE of our proposed algorithm, RUQL, is lower across the different execution policies. In other words, even though our algorithm RUQL, similar to QL, updates only the selected action, the dynamics of learning are very similar to QL-ideal-update and therefore the Q values are close to the ideal values (as if all the actions are updated at every time step).

The algorithm we propose here addresses the policy bias problem. The main idea of the Repeated Update Q-learning (RUQL) is to repeat the update of an action inversely proportional to the probability of choosing that action. For example, if an action is to be chosen for execution only 10% of the time, then the update of that action (when finally chosen) shall be repeated 1/0.1 or 10 times. Consequently, every action will, in expectation, be updated an equal number of times. RUQL can be approximated by a closed-form expression that removes the need to actually repeat the updates,

1. In practice, QL-ideal-update may lead to horrible performance (since we are executing suboptimal actions every time step) and may be even impossible to achieve (in multi-state domains executing an action would change the state), but hypothetically QL-ideal-update would lead to ideal estimates of the action values (Q).
2. More specifically, QL-ideal-update was run for 1000 time steps. Then QL with each of the various execution policies was run for 1000 time steps. Then the squared error of Q-values were averaged over all actions, the root computed, and then averaged again over all time steps.

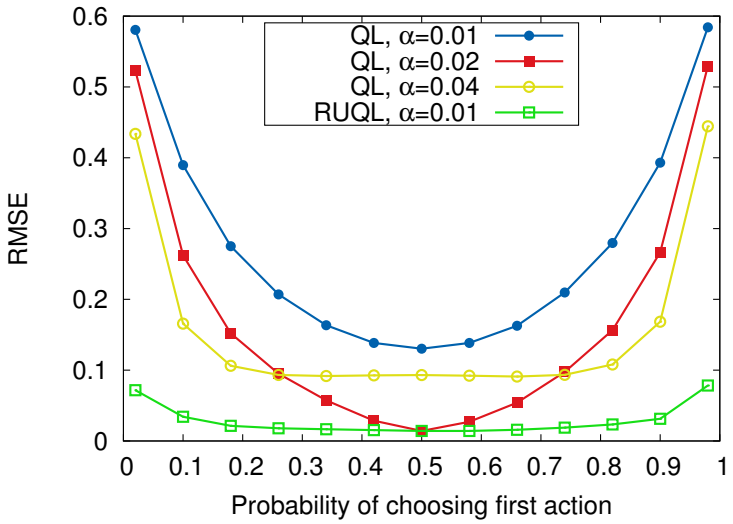


Figure 1: RMSE of QL for different values of α, and RUQL plotted against the probability of choosing the first action $\pi(1)$. RUQL, approximated as in Theorem 1, has the lowest RMSE across different policies.

as shown in Section 3. We show both theoretically and experimentally that the dynamics of RUQL approximate the dynamics of QL-ideal-update (as if all the actions were updated every time step). More importantly, RUQL outperforms both QL and the closest state-of-the-art, FAQL (Kaisers and Tuyls, 2010), in noisy non-stationary settings.

In summary, the main contributions of this paper are:

- A new reinforcement learning algorithm, RUQL, which builds on the Q-learning algorithm but does not suffer from the policy-bias problem.
- A mathematical derivation of a closed form of RUQL that makes the RUQL algorithm computationally feasible.
- Theoretical analysis that shows how RUQL maintains the convergence guarantee of QL (in stationary environments, and for tabular Q values), while having learning dynamics that are better suited for non-stationary environments.
- Theoretical analysis that shows the similarities and differences between RUQL and the closely-related state-of-the-art algorithm, FAQL (Kaisers and Tuyls, 2010).
- An extensive experimental study that confirms the theoretical analysis and compares the performance of RUQL to QL, FAQL, and two other learning algorithms.

The following section presents the required background concepts covering Q-learning and related algorithms. We then introduce the algorithm RUQL, followed by theoretical and experimental analysis. Finally, a summary and a discussion of the contributions conclude the article.

2. Background

The Q-learning algorithm (Watkins and Dayan, 1992) is a model-free[3] reinforcement learning algorithm that is guaranteed to find the optimal policy for a given Markov Decision Process (MDP). An MDP is defined by the tuple $\langle S, A, P, R \rangle$, where S is the set of states representing the system and $A(s)$ is the set of actions available to the agent at a given state s. The function $P(s, a, s')$ is the transition probability function and quantifies the probability of reaching state s' after executing action a at state s. Finally the reward function, $R(s, a, s')$, gives the average reward an agent acquires if the agent executes action a at state s and reaches state s'. The goal is then to find the optimal policy $\pi^*(s, a)$ which gives the probability of choosing every action a at every state s in order to maximize the expected total discounted rewards. In other words, $\pi^*(s, a) = \arg\max_{\pi(s,a)} E\{\sum_t \gamma^t r_t | s_{t=0} = s, a_{t=0} = a\}$, where the parameter γ denotes the discount factor that controls the relative value of future rewards. To solve the model, the action value function, Q, is introduced where $Q(s, a) = \sum_{s'} P(s, a, s') \left(R(s, a, s') + \gamma \max_{a'} Q(s', a')\right)$. The Q function represents the expected total discounted reward if the agent starts at state s, executes action a, and then follows the optimal policy thereafter. Intuitively, the function $Q^t(s, a)$ represents what the agent believes, at time t, to be the worth of action a at state s. The optimal policy can then be defined as $\pi^*(s) = \arg\max_a Q(s, a)$. The Q-learning algorithm incrementally improves the action-value function Q using the update equation

$$Q^{t+1}(s, a) \leftarrow Q^t(s, a) + \alpha \left(r + \gamma \max_{a'} Q^t(s', a') - Q^t(s, a) \right). \tag{1}$$

The parameter α is called the learning rate and controls the speed of learning. The variables r and s' refer to the immediate reward and the next state, both of which are observed after executing action a at state s. Algorithm 1 illustrates how the Q-learning update equation is typically used in combination with an exploration strategy that generates a policy from the Q-values.

Algorithm 1: Q-learning Algorithm

```
begin
    Initialize function Q arbitrarily.
    Observe the current state s.
    repeat
        Compute the policy π(s, a) from Q(s, a), balancing exploration and exploitation.
        Choose an action a according to agent policy π(s, a).
        Execute action a and observe the resulting reward r and the next state s'.
        Update Q(s, a) using Equation 1.
        Set s ← s'.
    until done
end
```

The agent needs to choose the next action considering that its current Q-values may still be erroneous. The belief of an action to be inferior may be based on a crude estimate, and there may be a chance that updating that estimate reveals the action's superiority. Therefore, the function Q^t

3. Q-learning does not require knowing the underlying stochastic reward or transition function of the MDP model.

in itself does not dictate which action an agent should choose in a given state (Step 5 in Algorithm 1). The policy $\pi(s, a)$ of an agent is a function that specifies the probability of choosing action a at state s. A greedy policy that only selects the action with the highest estimated expected value[4] can result in an arbitrarily bad policy, because if the optimal action initially has a low value of Q it might never be explored. To avoid such premature convergence on suboptimal actions, several definitions of π have been proposed that ensure all actions (including actions that may appear sub-optimal) are selected with non-zero probability. These definitions of π are often called *exploration* strategies (Sutton and Barto, 1999). The two most common exploration strategies are ϵ-greedy exploration and Boltzmann exploration. The ϵ-greedy exploration simply chooses a random action with probability ϵ and otherwise (with probability 1-ϵ) chooses the action with the highest Q-value greedily. The Boltzmann exploration strategy defines the policy $\pi(s, a)$ as a function of Q-values and τ

$$\pi^t(s,a) = \frac{e^{\frac{Q^t(s,a)}{\tau}}}{\sum_{a'} e^{\frac{Q^t(s,a')}{\tau}}}, \tag{2}$$

where the tunable parameter τ is called the temperature.

We use Boltzmann exploration in our experiments, but with an additional parameter π_{min}, which defines a threshold below which the probability of choosing an action can not fall. In a sense, this combines the nice continuity of the Boltzmann exploration with the exploration guarantee of ϵ-greedy exploration. This is implemented using a projection function (Zinkevich, 2003; Abdallah and Lesser, 2008), which projects the policy to the closest valid policy with minimum exploration.

Regardless of the exploration strategy, the agent needs to gain information about the value of each action at every state in order to become more certain over time that the action with the highest estimate truly is the optimal action. However, an agent can only execute one action at a time and the agent only receives feedback (reward) for the action that was actually executed. As a result, in Q-learning, the rate of updating an action relies on the probability of choosing that action. The theoretical comparison of updating one vs. all actions at each time step reveals that Q-learning counter-intuitively decreases the probability of optimal actions under some circumstances, which leads to drawbacks in non-stationary environments (Kaisers and Tuyls, 2010; Wunder et al., 2010).

Scaling the learning rate inversely proportional to the policy has been proposed to overcome this limitation, thereby approximating the simultaneous updating of all actions every time a state is visited. This concept has been initially studied to modify fictitious play (Fudenberg and Levine, 1998), and inspired two equivalent modifications of Q-learning named Individual Q-learning (Leslie and Collins, 2005) and Frequency Adjusted Q-learning (FAQL) (Kaisers and Tuyls, 2010). FAQL used the modified Q-learning update rule

$$Q^{t+1}(s,a) \leftarrow Q^t(s,a) + \frac{1}{\pi(s,a)}\alpha\Big(r + \gamma \max_{a'} Q^t(s',a') - Q^t(s,a)\Big).$$

Thus, FAQL scales the learning rate for an action a to be inversely proportional to the probability of choosing action a at a given state. This simple modification to the Q-learning update rule suffers from a practical concern: the update rate becomes unbounded (approaches ∞) as $\pi(s, a)$ approaches zero. Therefore, a safe-guard condition has to be added in practice (Kaisers and Tuyls, 2010)

$$Q^{t+1}(s,a) \leftarrow Q^t(s,a) + \min(1, \frac{\beta}{\pi(s,a)})\alpha\Big(r + \gamma \max_{a'} Q^t(s',a') - Q^t(s,a)\Big). \tag{3}$$

4. A greedy policy can be formally defined as $\pi^t(s, a) = 1$ iff $a = \arg\max_{a'} Q^t(s, a')$ and $\pi^t(s, a) = 0$ otherwise.

where β is a tuning parameter that safeguards against the cases where $\pi(s,a)$ is close to zero. The resulting algorithm is similar to Algorithm 1 but with Line 8 modified to use Equation 3 instead of Equation 1.

While introducing parameter β does make FAQL applicable in practical domains, the introduction of β also results in two undesirable properties. The first undesirable property is reducing the effective learning rate (instead of α it is now $\alpha\beta$) and therefore reducing the responsiveness of FAQL.[5] The second and more important undesirable property is what we refer to as the β-limitation: Once the probability of choosing an action, $\pi(s,a)$, goes below β, FAQL behaves identical to the original QL. In other words, FAQL suffers from the same policy-bias problem when the probability of choosing an action becomes lower than β. This is problematic because the lower the probability of choosing an action, the more important it is to adjust the learning rate (to account for the infrequency of choosing that action). Our proposed algorithm RUQL does not suffer from these limitations and we show in the experiments how this can improve performance in noisy and non-stationary settings.

3. Repeated-Update Q-learning, RUQL

Our proposed algorithm is based on a simple intuitive idea. If an action is chosen with low probability $\pi(s,a)$ then instead of updating the corresponding action value $Q(s,a)$ once, we repeat the update $\frac{1}{\pi(s,a)}$ times. Algorithm 2 shows the naive implementation of this idea. Line 8 is the only difference between Algorithm 2 and Algorithm 1.

Algorithm 2: RUQL (Impractical Implementation)

```
1  begin
2      Initialize function Q arbitrarily.
3      Observe the current state s.
4      for each time step do
5          Compute the policy π using Q.
6          Choose an action a according to agent policy π.
7          Execute action a and observe the resulting reward r and the next state s'.
8          for i : 1 ≤ i ≤ ⌊1/π(s,a)⌋ do
9              Update Q(s,a) using Equation 1.
10         end
11         Set s ← s'.
12     end
13 end
```

This simple modification to the QL algorithm addresses the policy-bias, but it has two limitations: (1) The repetition only works with $\frac{1}{\pi(s,a)}$ an integer, and (2) as $\pi(s,a)$ gets lower, the number of repetitions increases and quickly becomes unbounded as $\pi(s,a)$ approaches zero. Unlike FAQL, here the computation *time* (not the *value* of the update) is what becomes unbounded as $\pi(s,a)$ approaches zero. In the remainder of this section we will derive a closed-form expression for the

5. In our experiments we take the effective learning rate into account when comparing FAQL to our algorithm RUQL, setting $\alpha_{FAQL} = \beta_{FAQL} = \sqrt{\alpha_{RUQL}}$.

RUQL algorithm. The closed-form expression removes the need for actually repeating any update and therefore makes RUQL computationally feasible.

Theorem 1 *The RUQL update rule can be approximated with an error of $O(\alpha^2)$ as an instance of QL with learning rate (step size) z given by the equation*

$$z_{\pi^t(s,a)} = 1 - [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor},$$

where α is a constant, denoting the learning rate of the underlying QL update equation being repeated.

Proof

For convenience, let $Q_i(s,a)$ refer to the intermediary value of the Q function after i iterations of the loop in Steps 8-10 in Algorithm 2. Before starting the loop, we have $Q_0(s,a) \leftarrow Q^t(s,a)$, i.e. before executing any iterations the value of $Q(s,a)$ is set to the value of the previous time step t. Upon finishing the loop we get $Q^{t+1}(s,a) = Q_{\lfloor \frac{1}{\pi(s,a)} \rfloor}(s,a)$, or the new value of $Q(s,a)$ at time step $t+1$. Let us now trace the computation of $Q^{t+1}(s,a) = Q_{\lfloor \frac{1}{\pi(s,a)} \rfloor}$. After one iteration we have

$$Q_1(s,a) \;=\; [1-\alpha]Q_0(s,a) + \alpha[r + \gamma \max_{a'} Q_0(s',a')].$$

The second iteration results in

$$\begin{aligned} Q_2(s,a) &= [1-\alpha]\left([1-\alpha]Q_0(s,a) + \alpha[r+\gamma \max_{a'} Q_0(s',a')]\right) + \alpha[r+\gamma \max_{a'} Q_1(s',a')] \\ &= [1-\alpha]^2 Q_0(s,a) + [1-\alpha]\alpha[r+\gamma \max_{a'} Q_0(s',a')] + \alpha[r+\gamma \max_{a'} Q_1(s',a')]. \end{aligned}$$

Similarly, after $\lfloor \frac{1}{\pi(s,a)} \rfloor$ iterations

$$\begin{aligned} Q_{\lfloor \frac{1}{\pi(s,a)} \rfloor}(s,a) \;=\; & [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q_0(s,a) + \\ & [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor - 1}\alpha[r+\gamma \max_{a'} Q_0(s',a')] + \\ & [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor - 2}\alpha[r+\gamma \max_{a'} Q_1(s',a')] + \\ & \ldots \\ & \alpha[r+\gamma \max_{a'} Q_{\lfloor \frac{1}{\pi(s,a)} \rfloor - 1}(s',a')]. \end{aligned}$$

It is important to keep in mind that all the above repeated updates occur at the *same time step*. It is also worth noting that the updates are monotonic (i.e., either increasing or decreasing) between t and $t+1$. In the following, the floor notation is dropped for notational convenience. There are four cases that we need to consider (derivations are given in the Appendix):

Case 1, $s' \neq s$**, or** $\left(s' = s \text{ and } a \neq \arg\max_{a'} Q^t(s,a')\right)$ **throughout the** $\frac{1}{\pi(s,a)}$ **iterations:**

$$Q^{t+1}(s,a) = [1-\alpha]^{\frac{1}{\pi(s,a)}} Q^t(s,a) + \left[1 - (1-\alpha)^{\frac{1}{\pi(s,a)}}\right][r + \gamma \max_{a'} Q^t(s',a')] \quad (4)$$

Case 2, $s' = s$ **and** $a = \arg\max_{a'} Q^t(s, a')$ **throughout the** $\frac{1}{\pi(s,a)}$ **iterations:**

$$Q^{t+1}(s,a) = [1-\alpha(1-\gamma)]^{\frac{1}{\pi(s,a)}} \left(Q^t(s,a) - \frac{r}{1-\gamma} \right) + \frac{r}{1-\gamma} \tag{5}$$

Case 3, $s' = s$ **and** $a \neq \arg\max_{a'} Q^t(s, a')$ **until iteration** $i_\top$**:** Since the update is monotonic, iteration $i_\top$ can be determined, at which $Q_i(s,a)$ becomes larger than $\max_{a'} Q^t(s,a')$. Subsequently, we can compute $Q^{t+1}(s,a)$ by applying the remaining updates $\frac{1}{\pi(s,a)} - i_\top$ to the value of $\max_{a'} Q^t(s,a')$ according to Equation 5, since the updated action bears the maximum value for the remaining updates, i.e.,

$$Q^{t+1}(s,a) = [1-\alpha+\alpha\gamma]^{\frac{1}{\pi(s,a)}-i_\top} \left(\max_{a'} Q^t(s,a') - \frac{r}{1-\gamma} \right) + \frac{r}{1-\gamma}.$$

Case 4, $s' = s$ **and** $a = \arg\max_{a'} Q^t(s, a')$ **until iteration** $i_\perp$**:** Similar to Case 3, let $i_\perp$ be the iteration at which $Q_{i_\perp}(s,a)$ drops below $\max_{a'} Q^t(s,a')$. As a result, we can compute $Q^{t+1}(s,a)$ by applying the remaining updates $\frac{1}{\pi(s,a)} - i_\perp$ to the value of $\max_{a'} Q^t(s,a')$ according to Equation 4, i.e.

$$Q^{t+1}(s,a) = [1-\alpha]^{\frac{1}{\pi(s,a)}-i_\perp} \max_{a'} Q^t(s,a') + \left[1-(1-\alpha)^{\frac{1}{\pi(s,a)}-i_\perp}\right] [r+\gamma \max_{a'} Q^t(s',a')].$$

The computation time complexity of $i_\top$, $i_\perp$ and the update rule of RUQ-Learning is $O(1)$, a significant improvement over the naive implementation in Algorithm 2 with unbounded time complexity. However, the four cases and the corresponding update equations are not as simple as the original Q-learning update equation (Equation 1) or even the FAQL update equation (Equation 3). The following derivation will elucidate the difference between Equation 4 and Equation 5 and show that it is insignificant for small learning rates, such that solely Equation 4 can be used as an approximation in all cases. Consider the Taylor series expansion of both equations at $\alpha = 0$, using $(1-\alpha)^c = 1 - c\alpha + O(\alpha^2)$. In Case 2, when $Q^t(s,a) = \max_{a'} Q(s,a')$, Equation 4 becomes

$$\begin{aligned} Q^{t+1}(s,a) &= \left(1 - \frac{\alpha}{\pi(s,a)}\right) Q^t(s,a) + \frac{\alpha}{\pi(s,a)}[r + \gamma Q^t(s,a)] + O(\alpha^2) \\ &= \left(1 - \frac{\alpha(1-\gamma)}{\pi(s,a)}\right) Q^t(s,a) + \frac{\alpha}{\pi(s,a)} r + O(\alpha^2), \end{aligned}$$

while Equation 5 becomes

$$\begin{aligned} Q^{t+1}(s,a) &= \left(1 - \frac{\alpha(1-\gamma)}{\pi(s,a)}\right) \left(Q^t(s,a) - \frac{r}{1-\gamma} \right) + \frac{r}{1-\gamma} + O(\alpha^2) \\ &= \left(1 - \frac{\alpha(1-\gamma)}{\pi(s,a)}\right) Q^t(s,a) + \frac{\alpha}{\pi(s,a)} r + O(\alpha^2). \end{aligned}$$

In brief, Case 1 requires to apply Equation 4 anyway, and other cases only require to change to Equation 5 if $Q^t(s,a) = \max_{a'} Q(s,a')$. However, given $Q^t(s,a) = \max_{a'} Q(s,a')$ the lines above show that the difference between the equations is $O(\alpha^2)$, and hence negligible for small α,

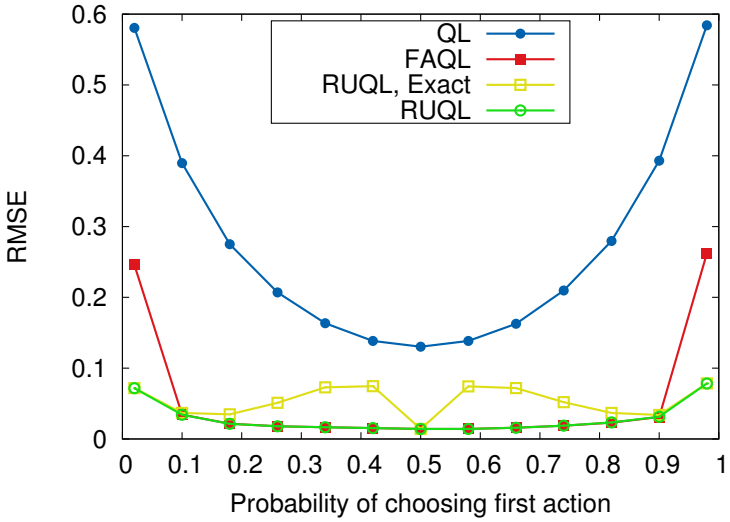


Figure 2: RMSE of QL, FAQL, RUQL (exact) and RUQL (approximate) plotted against the probability of choosing the first action $\pi(1)$. RUQL, approximated as in Theorem 1, has the lowest RMSE across different policies. The learning parameters are set as follows: $\alpha = 0.01$ for QL and RUQL and $\alpha_{FAQL} = \beta = \sqrt{\alpha} = 0.1$.

since higher order terms become insignificant. Therefore, Equation 4 can be applied in all four cases. In other words, RUQL can be approximated by

$$Q^{t+1}(s,a) = [1-\alpha]^{\frac{1}{\pi(s,a)}} Q^t(s,a) + \left[1-(1-\alpha)^{\frac{1}{\pi(s,a)}}\right][r + \gamma \max_{a'} Q^t(s',a')].$$

This equation can then replace Lines 8-10 in Algorithm 2 to produce a simple and efficient implementation of RUQL as an instance of QL with substitute learning rate $z_{\pi^t(s,a)} = 1-[1-\alpha]^{\frac{1}{\pi^t(s,a)}}$. ■

Algorithm 2 uses the floor notation, since fractional repetitions are not possible. However, the approximation alleviates this limitation, hence we adopt $z_{\pi^t(s,a)} = 1-[1-\alpha]^{\frac{1}{\pi(s,a)}}$ without flooring for both our experimental and theoretical analyses. The continuous and smooth definition of z_π will simplify the theoretical analysis we conduct later. More importantly, such smooth definition will handle fractions of $1/\pi$ more accurately. To illustrate this point we recall here the simple domain we used in the introduction section. The RUQL update equation is trying to imitate the QL update equation if (hypothetically) the agent can execute and update *every action at every time step*, or what we referred to as QL-ideal-update in the introduction. Figure 2 plots the RMSE of QL, FAQL, RUQL (exact) and RUQL (approximate) when compared to QL-ideal-update and using the same settings of the multi-armed bandit problem we explained in the introduction section.

It is clear from the figure that RUQL (approximate) has the lowest RMSE across different values of $\pi(1)$. In other words, independent of the underlying execution (exploration) strategy, RUQL results in action values that are the closest to the QL-ideal-update. Furthermore, as $\pi(1)$ goes beyond β in either direction the β limitation hits FAQL. Another interesting observation is that the RMSE of RUQL (exact) is higher than the RMSE of RUQL (approximate). The reason is the

inability of RUQL (exact) to handle partial iterations (i.e. if the ratio $\frac{1}{\pi}$ is not an integer). This is also why the performance of RUQL (exact) is best when $\pi(1) = 0.02, 0.1$, 0.5, 0.9, and 0.98 (with an almost perfect match with RUQL-approximate) in a clear verification of the validity of our approximation. At these points the boost in the learning rate that is given to the less probable action (with probability 0.02,0.1, and 0.5) is based on $\frac{1}{\pi}$ which is an integer in these cases. The further the probabilities are from these values, the further the RMSE of RUQL-exact from the RMSE of RUQL-approximate.

4. Theoretical Analysis

This section analyzes RUQ-Learning by comparing it with two closely-related algorithms: the original Q-learning and FAQL, the closest state of the art.

4.1 RUQL and QL

As stated by Theorem 1, RUQL can be approximated as an instance of QL with an effective learning rate $z_{\pi^t(s,a)} = 1 - [1-\alpha]^{\frac{1}{\pi^t(s,a)}}$. However, while (traditionally) the learning rate of Q-learning, α, is independent of the policy, the *effective* learning rate of RUQL, z, is a function of the policy at the particular state s and action a at the given time t. For example, a common definition of the learning rate $\alpha(s,a,t) = \frac{1}{visits(s,a,t)}$, where $visits(s,a,t)$ is the number of times the learning agent visited state s and executed action a until time t. This is different from RUQL where the effective learning rate z includes the policy. The inclusion of the policy in z creates a feedback loop: the effective learning rate affects the learning which affects the policy which affects the effective learning rate.[6] As we show later in the experimental analysis, this results in significantly different learning dynamics (between RUQL and traditional QL) in non-stationary environments. In particular, we show that increasing the learning rate α even by 100 folds does not change the dynamics of QL significantly. On the other hand, using RUQL results in significantly different dynamics.

The dependence of the effective learning rate on the policy raises concerns of whether the convergence of Q-learning in stationary environments still holds for RUQL. We revisit here the main conditions for Q-learning convergence (Sutton and Barto, 1999), and consider whether the conditions still hold for RUQL and under what assumptions. For the original Q-learning two conditions concerning the learning rate need to be satisfied for convergence in MDPs: $\sum_t \alpha_t = \infty$ and $\sum_t (\alpha_t)^2 < \infty$. The second condition requires that the learning rate decays over time, while the first condition ensures that such decay is slow enough to allow the agent to learn the optimal Q values. One possible definition of α that satisfies both conditions is $\alpha_t = \frac{1}{t}$.

Extending the two conditions to RUQL, we substitute z for α_t in the learning rate conditions to get $\sum_t z_{\pi^t} = \infty$ and $\sum_t z^2_{\pi^t} < \infty$. Expanding the first condition yields

$$\sum_t \left(1 - [1-\alpha_t]^{\frac{1}{\pi^t(s,a)}}\right) = \infty.$$

Using polynomial expansion of z

$$z_{\pi^t} = 1 - [1-\alpha_t]^{\frac{1}{\pi^t(s,a)}} = 1 - \left(1 + \frac{1}{\pi^t(s,a)}(-\alpha_t) + O(\alpha_t^2)\right) = \frac{\alpha_t}{\pi^t(s,a)} - O(\alpha_t^2).$$

6. In the case of a multi-agent system, this feedback loop is affected by other agents (because other agents affect the individually learned policy).

Subsequently, omitting the higher order terms[7] denoted by $O(\alpha_t^2)$ and substituting in the first condition leads to

$$\sum_t \left[\alpha_t \frac{1}{\pi^t(s,a)}\right] = \infty.$$

It is worth noting that $\pi^t(s,a)$ is bound away from zero since the sequence of updates proceeds over solely those actions that are selected with positive probability. This implies that for any sequence α_t for which $\sum_t \alpha_t = \infty$, the first condition also holds for the effective learning rate z_π of RUQL, since each summand is at least as large as before. We now move to the second condition:

$$\sum_t \left(1 - [1-\alpha_t]^{\frac{1}{\pi^t(s,a)}}\right)^2 < \infty.$$

Again using the polynomial expansion yields

$$\sum_t \left[\alpha_t \frac{1}{\pi^t(s,a)}\right]^2 < \infty.$$

Here we observe a clear distinction between RUQL and the original QL. Unlike the original Q-learning, we need to impose restrictions on the policy to ensure the above condition. Let us denote the minimum probability of choosing an action at a given time with $\epsilon^t = \min_{s,a} \pi^t(s,a)$ (which reflects the exploration rate). We then need to ensure that

$$\sum_t \left[\frac{\alpha_t}{\epsilon^t}\right]^2 < \infty.$$

To clarify how the above conditions can be used, consider the following example. Suppose we are using ϵ-greedy exploration strategy and $\alpha_t = \frac{1}{t}$, then one possible exploration rate that satisfies the second condition is $\epsilon^t = \frac{1}{\lg t}$.[8] Similarly, we can ensure minimum exploration using Boltzmann exploration through updating the temperature τ using the learned action values (Achbany et al., 2008).

It is worth noting that in practice, and particularly in non-stationary environments, the exploration rate parameter (which is ϵ in ϵ-greedy and τ in Boltzmann) is usually held to small constant values. In such cases, the second condition is trivially satisfied,[9] which means RUQL is guaranteed to converge to the optimal Q values. Our experiments confirm that RUQL works well in these environments.

4.2 RUQL and FAQL

Both RUQL and FAQL provide an algorithmic implementation to address the policy-bias of Q-learning. The two algorithms resolve this problem in different ways: FAQL normalizes the learning rate of the Q-learning update while RUQL repeats the Q-learning update rule. Despite their difference in implementation, RUQL learning dynamics are similar to FAQL learning dynamics when the

7. Omitting $O(\alpha_t^2)$ is justified because α_t is assumed to decay over time to establish convergence in stationary environments, and for small α_t the first order term dominates diminishing higher order terms.

8. The series $\sum_t \left[\frac{\lg t}{t}\right]^2$ is convergent using the Cauchy condensation test.

9. For constant exploration rates, ϵ^t is bound away from zero and can be replaced by a constant that represents the minimum positive value it assumes. RUQL thus satisfies the condition for the same α^t as QL.

probability of choosing actions are larger than β, otherwise FAQL learning dynamics deviate from RUQL learning dynamics. This can be shown through the following arguments.

Let $\Delta Q^t_{\text{algorithm}}(s,a) = Q^{t+1}(s,a) - Q^t(s,a)$ denote the update step for each of the algorithms FAQL and RUQL. Then the difference between the updates of FAQL and RUQL is $d_{\text{FAQL,RUQL}} = \Delta Q^t_{\text{FAQL}}(s,a) - \Delta Q^t_{\text{RUQL}}(s,a) = (w_{\pi^t(s,a)} - z_{\pi^t(s,a)})\mathbf{e}^t(s,a)$, where the Q-value error is denoted by $\mathbf{e}^t(s,a) = \big[r + \gamma Q^t(s', a_{\max}) - Q^t(s,a)\big]$, $w_{\pi^t(s,a)} = \min(1, \frac{\beta}{\pi(s,a)})\alpha_{FAQL}$ represents FAQL's effective learning rate, and $z_{\pi^t(s,a)} = 1 - [1 - \alpha_{RUQL}]^{\frac{1}{\pi^t(s,a)}}$ represents RUQL's effective learning rate. Further working out the Taylor expansion at $\alpha = 0$ of the term $(1-\alpha_{RUQL})^{\frac{1}{\pi(s,a)}}$ from RUQL's update equation yields

$$[1 - \alpha_{RUQL}]^{\frac{1}{\pi(s,a)}} = 1 - \frac{\alpha_{RUQL}}{\pi(s,a)} + O(\alpha^2_{RUQL}).$$

Therefore, the learning rate of RUQL can be simplified to.

$$\begin{aligned} z_{\pi^t(s,a)} &= 1 - \left(1 - \frac{\alpha_{RUQL}}{\pi(s,a)} + O(\alpha^2_{RUQL})\right) \\ &= \frac{\alpha_{RUQL}}{\pi(s,a)} + O(\alpha^2_{RUQL}). \end{aligned}$$

If $\mathbf{e}^t(s,a) = 0$, then both algorithms keep the Q-values unchanged, i.e.,

$$\Delta Q^t_{\text{FAQL}}(s,a) = \Delta Q^t_{\text{RUQL}}(s,a) = 0.$$

Otherwise, the difference in updates can be rewritten for small learning rates α_{RUQL} as

$$\frac{d_{\text{FAQL,RUQL}}}{\mathbf{e}^t(s,a)} = w_{\pi^t(s,a)} - z_{\pi^t(s,a)} = \min(1, \frac{\beta}{\pi(s,a)})\alpha_{FAQL} - \frac{\alpha_{RUQL}}{\pi(s,a)}.$$

For $\beta \cdot \alpha_{FAQL} = \alpha_{RUQL}$ the two algorithms yield equal updates if $\pi(s,a) \geq \beta$. However, when $\pi(s,a) < \beta$ the learning rates differ by $\frac{\alpha_{RUQL}}{\beta} - \frac{\alpha_{RUQL}}{\pi(s,a)}$, which is due to the β limitation of FAQL as we mentioned before.

5. Related Work

Several modifications and extensions to Q-learning were proposed, such as individual Q-learning (Leslie and Collins, 2005), the CoLF and CK heuristics (de Cote et al., 2006), and the utility-based Q-learning (Moriyama et al., 2011). Some of the proposed extensions aimed at improving the performance of Q-learning in specific domains (such as promoting cooperation in public good games (de Cote et al., 2006; Moriyama et al., 2011)), rather than addressing the policy-bias limitation of Q-learning. There also have been some works on modifying the learning rate α to speed up convergence (George and Powell, 2006; Dabney and Barto, 2012; Mahmood et al., 2012). Most of the proposed approaches defined different decaying functions for the learning rate (for example, as a function of time or the number of visits to a state) that are *independent* of the underlying policy (George and Powell, 2006). More recent approaches proposed more sophisticated methods for

adapting the learning rate based on the error (Dabney and Barto, 2012; Mahmood et al., 2012). None of the proposed approaches addressed the policy-bias problem.

Some learning algorithms specifically targeted non-stationary environments (da Silva et al., 2006; Noda, 2010; Kaisers and Tuyls, 2010). One technique devised a mechanism for updating the learning rate based on gradient descent (to minimize error) (Noda, 2010). Aside from being more complex than our simple update equation, the approach could only handle stateless domains. The RL-Context-Detection (RL-CD) algorithm (da Silva et al., 2006) was model-based and assumed finite set of stationary contexts that the environment switches among. For each stationary context a model was learned. This can be contrasted to our approach, which is model-free and tracks the non-stationarity of the environment without assuming a finite set of contexts. This is important when the non-stationarity is a result of an adaptive opponent (with potentially infinite contexts). The more recently proposed Frequency Adjusted Q-Learning (Kaisers and Tuyls, 2010) aimed to address the policy-bias limitation of Q-learning, but as we have shown, FAQL suffers from the β-limitation.

Q-learning was originally designed for single-agent environment, but yet performed adequately in multi-agent environments. HyperQ-learning was proposed as an extension to Q-learning in order to support multi-agent contexts (Tesauro, 2004). Unlike our approach, HyperQ attempts to learn the opponents' strategies without adapting the learning rate. We believe it is possible to integrate RUQL with HyperQ, but it is beyond the scope of this paper and remains an interesting future direction. Studying the dynamics of learning algorithms in multi-agent contexts has been an active area of research (Abdallah and Lesser, 2008; Babes et al., 2008; Bowling and Veloso, 2002; Claus and Boutilier, 1998; de Cote et al., 2006; Eduardo and Kowalczyk, 2009; Lazaric et al., 2007; Leslie and Collins, 2005; Moriyama, 2009; Tuyls et al., 2006; Vrancx et al., 2008; Wunder et al., 2010; Galstyan, 2013). The dynamics of Q-learning with Boltzmann exploration received attention earlier due to the continuous nature of the Boltzmann exploration. One of the earliest analyses of Q-learning assumed agents keep record of previous interactions and used Boltzmann exploration strategies (Sandholm and Crites, 1996). More recent analysis of Q-learning with Boltzmann exploration used evolutionary (replicator) dynamics (Kaisers and Tuyls, 2010; Lazaric et al., 2007; Tuyls et al., 2006). Similarly, Q-learning with ϵ-greedy exploration has been analyzed theoretically in stateless games (Eduardo and Kowalczyk, 2009; Wunder et al., 2010).

Another relevant and somewhat recent area of research is adversarial MDPs, where an oblivious adversary may choose a new reward function (and possibly the transition probability function as well) at each time step (Yu et al., 2009; Abbasi et al., 2013; Neu et al., 2010). A central assumption in that work is that the agent observes the full reward function (not only the current reward sample), and possibly the transition probability function once a change in the environment occurred. Furthermore, some of the approaches could not handle deterministic environments, as a mixed policy is assumed (Abbasi et al., 2013). Our approach (similar to Q-learning) does not make such strong assumptions.

Aside from Q-learning and its variations and extensions, several reinforcement learning algorithms have been proposed to optimize the exploration and achieve more efficient learning (Strehl et al., 2009; Jaksch et al., 2010). Aside from being more complex than Q-learning and RUQL, the algorithms that optimize exploration do not address the policy-bias problem. We view such algorithms as complementary to RUQL, which reduces the coupling between the learning dynamics and the underlying exploration strategy. Similar to the possible integration between RUQL and gradient-based multi-agent learning algorithms, RUQL can also be integrated with algorithms that optimize the exploration strategy.

The remainder of this section explains the two algorithms DynaQ and Double Q in more detail. These algorithms are more relevant to RUQL and serve as a benchmark comparison in our experiments.

5.1 Dyna-Q (DynaQ)

The Dyna-Q learning algorithm (Sutton, 1990) integrated the use of a model with the traditional Q-learning update. In its simplest form (which was used in the original paper experiments, as well as our experiments), Dyna-Q can be expressed as shown in Algorithm 3. Note that we intentionally framed DynaQ in such a way that highlights the similarities and differences with RUQL. From the algorithm, we note that similar to the original RUQL, DynaQ repeats the Q-learning update using the learned model. However, there are crucial differences. DynaQ repeats the updates *independent* of the underlying policy, in clear contrast to RUQL which repeats the update inversely proportional to the policy. Also there is no closed form equation that approximates DynaQ, and therefore it can be orders of magnitudes slower than QL (depending on the value of parameter k). Our intuition is that DynaQ is equivalent to uniformly increasing the effective learning rate, but we are not aware of any thorough theoretical analysis of DynaQ that established such connection (unlike our theoretical analysis of RUQL, which highlighted how RUQL affects the effective learning rate).

Algorithm 3: Dyna-Q

begin
  Initialize function Q arbitrarily and list L to empty.
  Observe the current state s.
  for *each time step* **do**
    Compute the policy π using Q.
    Choose an action a according to agent policy π.
    Execute action a and observe the resulting reward r and the next state s'.
    Update Q using Equation 1 and the information $\langle s, a, s', r\rangle$.
    Add the experience tuple $\langle s, a, s', r\rangle$ to L
    for $i : 1 \leq i \leq k$ **do**
      retrieve an experience tuple $\langle s, a, s', r\rangle$ uniformly at random from list L
      Update Q using Equation 1 and the information $\langle s, a, s', r\rangle$.
    end
    Set $s \leftarrow s'$.
  end
end

5.2 Double Q-Learning (DoubleQ)

An interesting recent variation of Q-learning is Double Q-learning (DQL) (Hasselt, 2010). DQL attempts to address how Q-learning may overestimate action values due to the max operator in the Q-learning equation. DQL counters overestimation of action values by maintaining two separate action values estimates, Q_A and Q_B, for each state-action pair. Whenever an action value is to be updated, either Q_A or Q_B is chosen uniformly at random to be updated. The update is done

using the traditional QL update equation, Equation 1, with only one difference: the max operator is applied to the action values of the other table. For example, suppose we chose Q_A is to be updated, then we apply Equation 6 as follows:

$$Q_A^{t+1}(s,a) \leftarrow Q_A^t(s,a) + \alpha \left(r + \gamma\, Q_B^t(s', a^*) - Q_A^t(s,a)\right), \tag{6}$$

where $a^* = \arg\max_{a'} Q_A^t(s', a')$. In other words, for the next state s', the best action is determined using the action value table to be updated. However, the *value* of the best action is determined using the action value in the other table. It is important to note that DoubleQ is *identical* to QL in stateless domains, where $\gamma = 0$. Therefore, the curve corresponding to DoubleQ in Figure 2 would be identical to the QL curve (DoubleQ suffers from the same policy bias as QL, at least for stateless domains).

6. Experimental Analysis

To ensure thorough evaluation of the RUQL algorithm, our analysis spans over several domains. The first two domains focus on verifying the theoretical arguments that we have made in Section 4 using simple, single-state, and non-stationary domains. The first domain (Section 6.1) uses a variation of the multi-armed-bandit problem to expose the effect of QL limitations in dynamic environments. The second domain (Section 6.2) aims at verifying the equivalence of RUQL and FAQL for sufficiently small values of the learning rate α. The subsequent three domains evaluate RUQL in multi-state, non-stationary, and noisy domains.

6.1 Multi-Armed Bandit With Switching Means

In order to tease out the differences between RUQL and FAQL, we propose a simple variation of the multi-armed bandit (MAB) problem (Robbins, 1952). Consider the basic MAB problem: an agent can choose between two actions. Each action i has a mean payoff $\mu_i \in \{0, 1\}$ with some random noise. Since the agent is not aware which action is the best (with expected payoff of 1), the agent needs to explore the two actions while learning the expected returns for each action.

We made two modifications to the traditional MAB problem. The first modification is *switching* the value of μ for the two actions every D time steps. In other words, having $D = 500$ means that from time 0 to time 499 $\mu_0 = 0$ and $\mu_1 = 1$ then from time 500 to time 999 $\mu_0 = 1$ and $\mu_1 = 0$ and so on. Allowing μ to switch models a non-stationary environment. The second modification is, instead of the usual Gaussian noise, we use an asymmetric noise signal. Let r_0 be the sample reward of the action with mean payoff of 0, and r_1 be the sample reward of the action with mean payoff of 1. The sample rewards r_0 and r_1 are defined as

$$r_0 = \begin{cases} \frac{-1}{1-p_0}, & \text{with probability } 1 - p_0. \\ \frac{1}{p_0}, & \text{otherwise (with probability } p_0), \end{cases}$$

$$r_1 = \begin{cases} \frac{2}{1-p_1}, & \text{with probability } 1 - p_1. \\ \frac{-1}{p_1}, & \text{otherwise (with probability } p_1), \end{cases}$$

where both p_0 and p_1 are parameters for controlling noise. The first thing to note here is that the expected payoffs remain 0 and 1, respectively. The value of p_1 determines how misleading the optimal action can be. For instance, with $p_1 = 0.01$ the optimal action provides (with very low

probability) a payoff of -100, a payoff that is much lower than the expected payoff of the sub-optimal action. This type of noise targets the policy bias weakness of Q-learning. Once the optimal action receives such negative sample reward, and unless the learning rate is very small, the optimal action will not be tried for a long time. However, if the learning rate is very small, then Q-learning will respond poorly to changes in the environment. On the other hand, the value of p_0 determines how misleading the sub-optimal action can be. For example, with $p_0 = 0.01$ the sub-optimal action provides (with very low probability) a payoff of 100, which is much higher than the expected payoff of the actual optimal action. This type of noise targets the weakness of the RUQL algorithm, which gives a boost in the learning rate for infrequent actions.

The modified domain allows us to study two desired properties of learning algorithms: responding (quickly) to genuine change in the payoff *mean* of a particular action while resisting hasty response to stochastic payoff samples. Our definition of noise also resembles realistic situations. For example, the action of investing one's own money in gambling resembles a sub optimal action that has negative expected reward, while (very rarely) producing very high reward samples (very low p_0). Similarly, the action of investing money in stocks has positive expected reward with rare significant losses. It is worth noting that several techniques were proposed specifically to solve the non-stationary multi-armed bandit problem (Garivier and Moulines, 2011; Besbes et al., 2014). We are using the multi-armed bandit problem as a simple domain to understand the benefits and limitations of our approach. Our evaluation, therefore, include domains other than the multi-armed bandit problem (some of which are multi-state domains).

We conducted the experiments over a range of parameter values, including $D \in \{250, 500, \infty\}$, $p_0, p_1 \in \{0, 0.003, 0.01, 0.03, 0.1\}$, $\tau \in \{0.1, 0.3, 1\}$, $\alpha \in \{0.001, 0.01, 0.1, 1\}$,[10] and $\gamma = 0$. For each combination we collected the average payoff over 2000 consecutive time steps. The initial Q values are individually initialized randomly to either 0 or 1. Finally, statistical significance of the difference in the mean is computed over 100 independent simulation runs, using two-tail two-sample t-tests, where the null hypothesis is rejected at p-value 0.05.

In the absence of noise ($p_0 = p_1 = 0$), the only reason for change in a sampled payoff is the switch of μ. Therefore, setting $\alpha = 1$ achieves the best performance across the three algorithms, rendering all of them equivalent. Interestingly, increasing the value of p_0 (while the value of p_1 remains zero) does not cause the setting $\alpha = 1$ to be inferior. The reason is to be found in the QL policy bias. Increasing p_0 means that the suboptimal action will produce exceptionally high reward very rarely. With $\alpha = 1$, QL will naively, and from only one sample, think that the suboptimal action is the optimal one. As a result, QL will almost certainly choose the suboptimal action the following time step, which will most likely result in a disappointment. Therefore, QL will almost immediately correct its expectation. The same goes for RUQL and FAQL with $\alpha = 1$.

Table 1 shows the results for $p_1 = 0.01$ and $p_0 = 0.01$, for different values of D and α. For each algorithm, we report the best average payoff it achieved across the different values of the exploration parameter τ. Bold entries confirm the statistical significance of the outperforming algorithm(s). For example, if only RUQL is bold, then this means RUQL outperforms both QL and FAQL with statistical significance. If both RUQL and FAQL are bold, then this means both algorithms outperform QL with statistical significance but there is no statistical significance regarding the difference between FAQL and RUQL. Notice here the significant drop in performance for high α. The drop is more severe for stationary environments, which may appear counter intuitive.

10. Note that these values of α are for QL and RUQL. For FAQL, we used the square root of these α values for both α_{FAQ} and β_{FAQ}, so that FAQL has an effective learning rate $\alpha_{FAQ} \cdot \beta_{FAQ} = \alpha$.

D \ α	0.001	0.01	0.1	1
∞: QL	0.98	0.69	0.09	0.06
RUQL	0.97	**0.89**	0.09	0.06
FAQL	0.97	0.82	0.08	0.06
500: QL	0.51	0.58	0.37	0.53
RUQL	0.54	**0.72**	0.37	0.54
FAQL	0.53	**0.70**	0.37	0.53
250: QL	0.52	0.55	0.48	0.53
RUQL	0.52	**0.62**	0.48	0.53
FAQL	0.51	0.57	0.48	0.53

Table 1: The mean payoff for the MAB-switching game, with $p_0 = p_1 = 0.01$, and for different values of D and α. The reported value in each cell is the best value we obtained across the different values of the exploration rate τ. The first observation is the significant drop in performance when α increases beyond 0.01. The policy bias results in the optimal action being rarely tried after the optimal action receives a big penalty. For stationary environments ($D = \infty$), RUQL's performance is comparable to QL with no statistically significant difference for $\alpha = 0.001$. As alpha increases to 0.01, the policy bias affects QL significantly, even in stationary environments. RUQL is affected as well, but to a lesser degree. As alpha increases to 0.1 and above, the effect of noise becomes so significant that all algorithms perform poorly. Furthermore, RUQL achieves superior performance in non-stationary environments when $D < \infty$.

What happens here is that when the optimal action receives the rare negative reward, if α is high, then the corresponding Q value becomes much smaller than the Q-value of the suboptimal action, making it very unlikely that the optimal action will ever be tried. When the environment becomes non-stationary, QL's performance improves simply because the suboptimal action becomes optimal after the switch. RUQL reduces the severity of policy bias, which results in better performance in non-stationary cases ($D = 500$ and $D = 250$). It is important to note also that in the presence of noise, and for non-stationary environments, α should neither be too small nor too large. In this experiment, $\alpha = 0.01$ resulted in the most robust performance.

To have a closer look at the learning dynamics, Figure 3 illustrates a sample run with $\alpha = 0.01, D = 500, p_1 = p_0 = 0.01$ and $\tau = 0.3$. The figure compares the accumulated payoff for the three algorithms, using the same random seed (hence the synchronized noise). The three algorithms quickly learn to select Action 1, with payoff given by r_1 (hence the steady increase in the accumulated payoff). The first sudden drop in the accumulated payoff (around Time 50) is due to the noise defined by r_1. Notice slight gain of RUQL as a result of this noise. The reason is that the sudden drop reduces the probability of choosing the optimal action. The reduction in the probability results in subsequent small boost in the learning rate for RUQL (and FAQL), hence the quicker recovery of both RUQL and FAQL against QL. Now consider Time 500, when the first switch of the actions' expected payoff occurs and now Action 1 generates payoff according to r_0. We can see that both RUQL and FAQL initially continue to choose Action 1 (initial drop in the accumulated payoff). The probability of choosing Action 0, which is now the optimal action, is very low. RUQL quickly recovers while FAQL (due to the β-limitation) takes longer time to recover. This results in a bigger gain for RUQL.

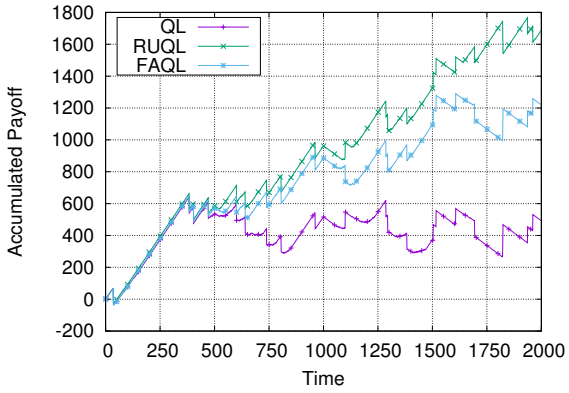


Figure 3: Comparison of the accumulated payoff for the three algorithms in a sample run (with the same random seed), using $\alpha = 0.01, D = 500, p_1 = p_0 = 0.01$ and $\tau = 0.3$.

D	$p_0 = p_1$ / Alg.	0.003	0.01	0.03	0.1
∞	QL	0.96	0.98	0.95	1
	RUQL	0.98	0.97	0.97	1
	FAQL	0.98	0.97	0.97	1
500	QL	0.56	0.58	0.55	0.72
	RUQL	**0.65**	**0.72**	**0.6**	0.71
	FAQL	**0.62**	**0.70**	**0.6**	0.71
250	QL	0.51	0.55	0.52	0.65
	RUQL	**0.58**	**0.62**	**0.56**	0.64
	FAQL	**0.57**	0.57	0.55	0.66

Table 2: The mean payoff for the MAB-switch game, for different values of D, p_0 and p_1. Each cell represents the average payoff, maximized over α and τ.

Table 2 compares the performance of the three algorithms for different levels of noise (p_i) and non-stationary environments (D). The performance reported for each algorithm is the maximum over α and τ.[11] We can observe here that for stationary environment, RUQL performs as well as QL, with no statistical significance for the difference between the three algorithms. As the environment becomes stationary and noisy, RUQL outperforms QL and FAQL with statistical significance for several settings. Even for the cases where RUQL does not outperform QL (or FAQL), RUQL has comparable performance with no-statistical significance of the difference.

6.2 Prisoner's Dilemma Game

This section compares the (experimental) dynamics of Q-learning, FAQL, and RUQL in the Prisoner's Dilemma game (PD). We are using the PD domain primarily for benchmarking, as it was

11. It is worth noting that for the majority of the cells, $\alpha = 0.01$ and $\tau = 0.3$ resulted in best performance for all three algorithms.

used before in evaluating FAQL (Kaisers and Tuyls, 2010). Each agent is oblivious to the existence of the other agent, therefore the domain can be viewed as a non-stationary environment from the perspective of each individual agent. For all the three algorithms (QL, FAQL, RUQL) the temperature for Boltzmann exploration was set to $\tau = 0.1$. The discount factor γ was set to 0. The learning rate α was set at 10^{-6} for both Q-learning and RUQL, while for FAQL both α and β were set to 10^{-3}. Table 3 shows the payoff values for the PD game. The above setting was used before in the literature (Kaisers and Tuyls, 2010) and we use the same setting here for comparability.

	c	d
c	3,3	0,5
d	5,0	1,1

Table 3: The prisoner's dilemma game with actions cooperate (c) and defect (d).

Figure 4 shows the dynamics of the three algorithms for three different initial value levels of Q. The dynamics of Q-learning and FAQL are consistent with the results reported before (Kaisers and Tuyls, 2010). It is worth noting that under pessimistic initial Q-values (centered around zero, top row), Q-learning approaches the Pareto optimal joint action, and hence a strictly dominated action, simply because it is initialized near this joint action, updates are self-reinforcing and low initial values lead to extremely low exploration probabilities. Given infinite time, the algorithms would converge to the Nash equilibrium, since the defection action would eventually catch up due to its factual higher value against cooperation. In contrast, the dynamics of both FAQL and RUQL show consistent behavior across different initializations, and they are very similar and close to the dynamical system of the infinitesimal limit (approximating that all actions were updated at every iteration), despite the conceptually different update equations. A plot of the infinitesimal limit is omitted since it would be indistinguishable from FAQL and RUQL, but illustrations can be found in related work (Kaisers and Tuyls, 2010).

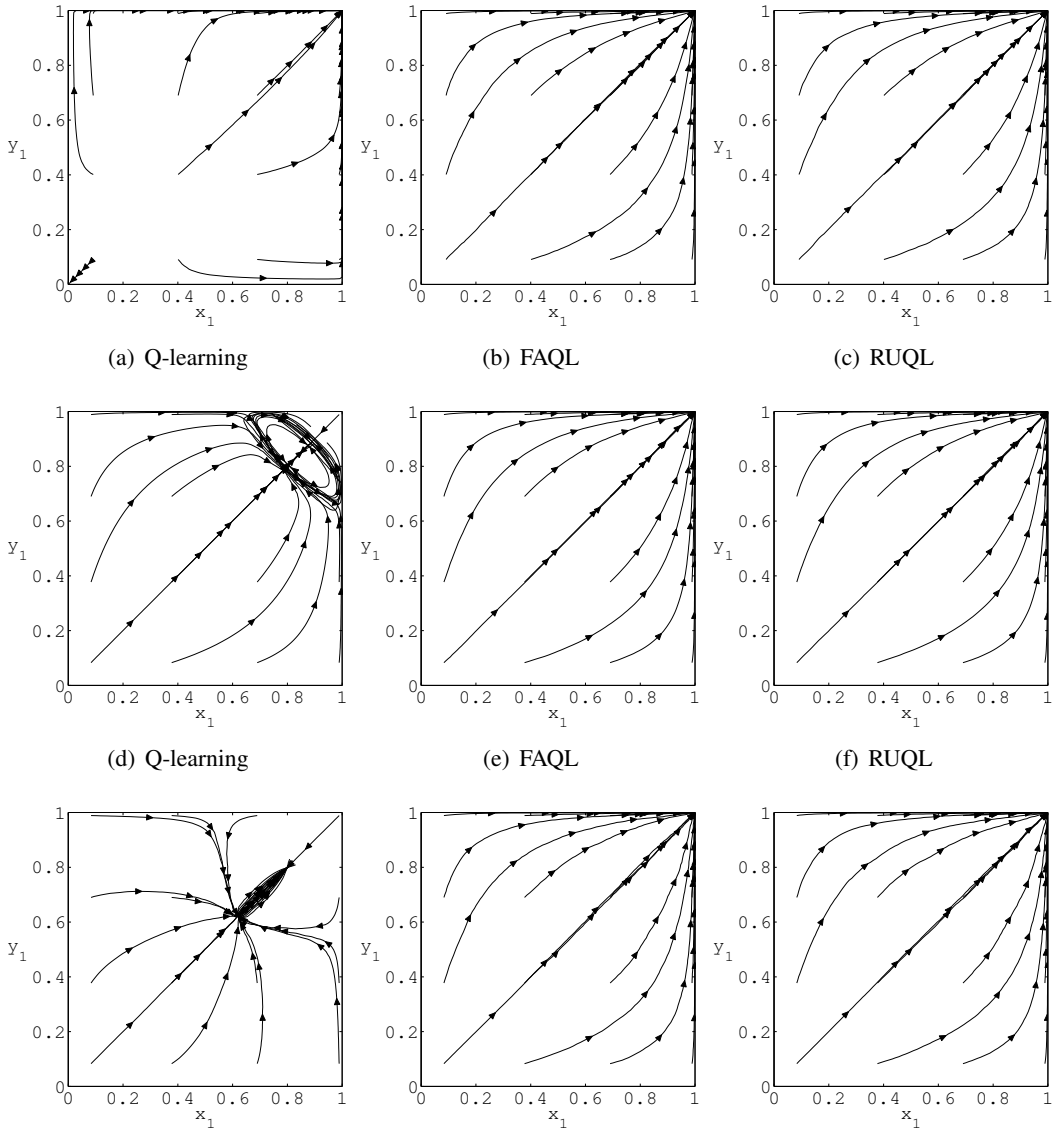
To demonstrate that the effect of RUQL is not equivalent to simply scaling the learning rate of Q-learning, Figure 5 shows the behavior dynamics of Q-learning when increasing the learning rate up to 100 times. As we can see, the general gross features of the dynamics remain the same. Increasing the learning rate only increases the step size in policy space without changing the expected direction.

6.3 1-Row Domain

This section presents a simple multi-state domain to test RUQL's multi-state performance. The domain is depicted in Figure 6 [a]. There are 10 total states, which are organized in one row (hence the name). There are two actions: left and right. The agent starts at a random state (which is not goal) and uses the two actions to reach the goal state depicted by 'G'. Each step yields a reward of -1, and the expected reward for reaching the goal is

$$r_g = \begin{cases} \frac{11}{1-P}, & \text{with probability } 1-P \\ \frac{-1}{P}, & \text{otherwise (with probability } P\text{).} \end{cases}$$

The domain is episodic. An episode starts from any random state that is not a terminal state. A terminal state is either the goal, or the cell opposite from the goal. To study non-stationarity in the

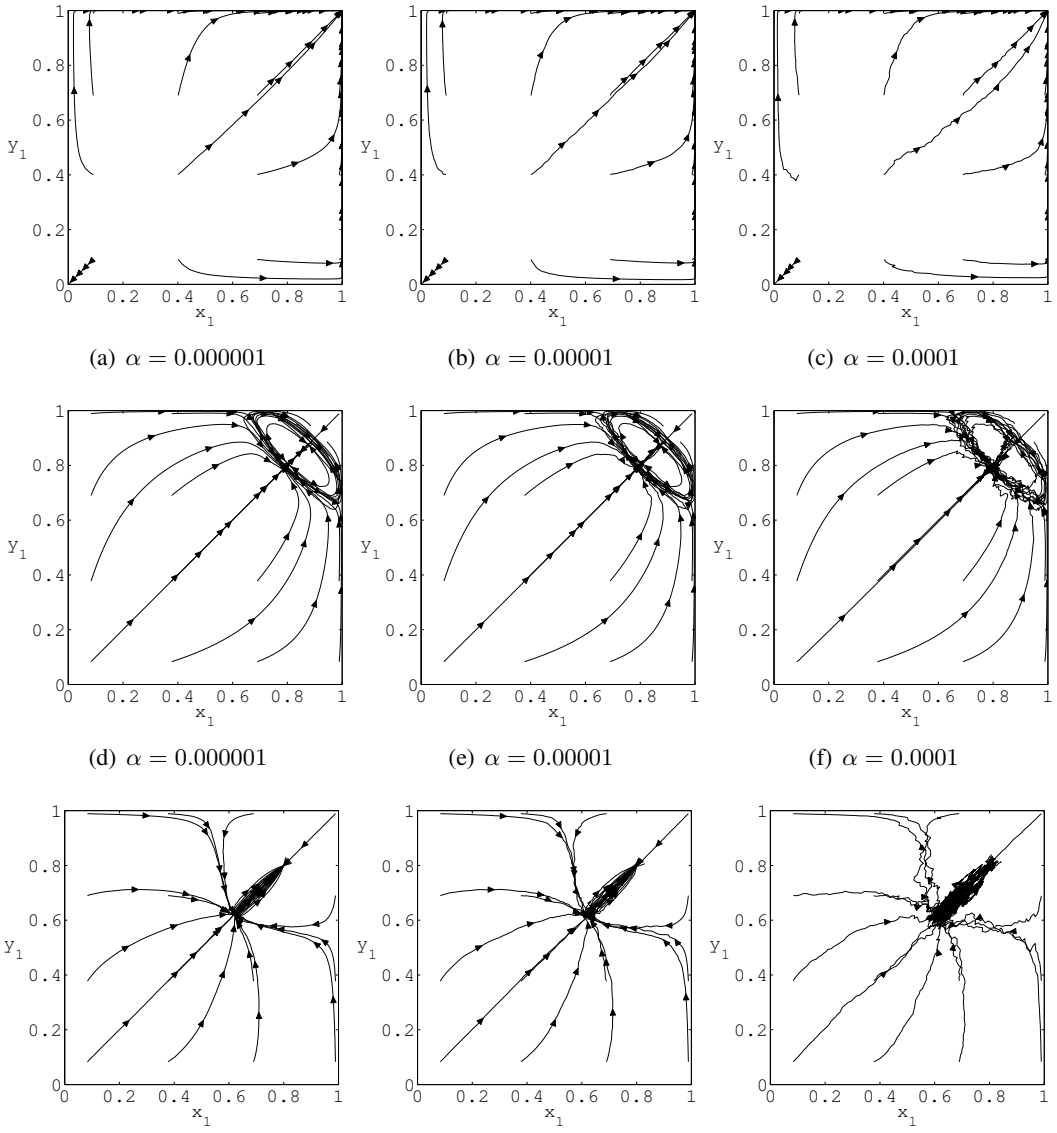


(a) Q-learning (b) FAQL (c) RUQL

(d) Q-learning (e) FAQL (f) RUQL

(g) Q-learning (h) FAQL (i) RUQL

Figure 4: The learning dynamics of Q-learning (left), FAQL (middle), and RUQL (right) for the PD game. Each figure plots the probability of defecting for Player 1 (x-axis) against the probability of defecting for Player 2 (y-axis). The rows show the dynamics when the initial Q-values are centered around 0, 2.5 and 5 (top, middle, bottom). For all figures, the algorithms were allowed to run for 5 million time steps. The learning rate α was set at 10^{-6} for both Q-learning and RUQL, while for FAQL both α and β were set to 10^{-3}. RUQL has dynamics very similar to FAQL, while Q-learning has erratic dynamics due to its policy bias.



(a) $\alpha = 0.000001$ (b) $\alpha = 0.00001$ (c) $\alpha = 0.0001$

(d) $\alpha = 0.000001$ (e) $\alpha = 0.00001$ (f) $\alpha = 0.0001$

(g) $\alpha = 0.000001$ (h) $\alpha = 0.00001$ (i) $\alpha = 0.0001$

Figure 5: The learning dynamics of Q-learning for different values of the learning rate α: 0.000001 (left), 0.00001 (middle), and 0.0001 (right) for the PD game. Each figure plots the probability of defecting for Player 1 (x-axis) against the probability of defecting for Player 2 (y-axis). The top row shows the dynamics when the initial Q-values are around 0, the middle row shows the dynamics when the initial Q-values are around 2.5, and the bottom row shows the dynamics when the initial Q-values are around 5. For all figures, the algorithms were allowed to run for $\frac{5}{\alpha}$ time steps.

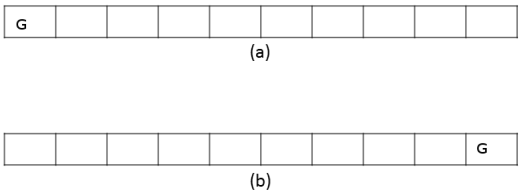


Figure 6: The two settings of the 1-row domain. G indicates the goal state.

domain, there are two settings (shown as [a] and [b] in Figure 6), where parameter D controls the duration before the environment switches from one setting to another.

We conducted the experiments over a range of parameter values, including $D \in \{5000, 10000, \infty\}$, $P \in \{0, 0.01, 0.1\}$, $\tau \in \{0.008, 0.04, 0.2, 1, 5\}$, $\pi_{min} \in \{0, 0.01, 0.001\}$, $\gamma = 1$, and $\alpha \in \{0.01, 0.04, 0.16, 0.64\}$.[12] For each combination we collected the average payoff over 20000 consecutive time steps. The initial Q values are initialized to 0. Finally, statistical significance of the difference in the mean is computed over 10 independent simulation runs, using two-tail two-sample t-test, where the null hypothesis is rejected at p-value 0.05.

Table 4 compares the performance of the three algorithms for different levels of noise (P) and non-stationary environments (D). The performance reported for each algorithm is the best over the different values of π_{min}, α and τ. We can observe similar results to the stateless domain. For noise-free and stationary environments, RUQL performs as well as the other algorithms, with no statistical significance for the difference. For noise-free non-stationary environments, DynaQ performs significantly better. The reason is that by repeating the updates, DynaQ is effectively increasing the learning rate for all actions, which is beneficial in noise-free environments. As the environment becomes non-stationary and noisy, RUQL consistently outperforms other algorithms, with FAQL and DoubleQ as close seconds. Dyna-Q fails in noisy environments because Dyna-Q effectively boosts the learning rate for *all* the actions (therefore, the Q-value of an optimal action may drop significantly upon receiving noisy negative reward).

6.4 Taxi Domain with Switching Stations

To investigate how RUQL works in multi-state environments we use a variation of the taxi domain (Dietterich, 2000). Figure 7 illustrates the domain. The world is a 5x5 grid that has 4 stations (represented by the letters) and walls (represented by the thick lines). A customer appears at a source station and is to be delivered to a destination station. Both the source and the destination stations are picked uniformly at random and can be the same. A taxi can pick a customer up, drop a customer, or navigate through the world, which results in a total of 6 actions: UP, DOWN, LEFT, RIGHT, PICK, and DROP. The state here is the taxi location (25 values) the customer location (5 values: 4 stations and in taxi) and destination (4 values) for a total of 500 states. Each navigation action receives a reward of -1. Hitting the wall results in no change in the state and the same reward of -1. Picking up or dropping off a customer receives a reward of 20 if at the right location, otherwise receives a reward of -10. We made two main modifications:

12. Note that these values of α are for QL, DoubleQ, DynaQ, and RUQL. For FAQL, we used the square root of these α values for both α_{FAQ} and β_{FAQ}, so that FAQL has an effective learning rate $\alpha_{FAQ}.\beta_{FAQ} = \alpha$.

D	Alg. \ P	0.0	0.01	0.1
5000	QL	0.77	0.74	0.83
	DynaQ	**1.31**	-0.01	0.02
	DoubleQ	0.85	0.75	0.80
	FAQL	0.81	**0.80**	**0.85**
	RUQL	0.84	**0.81**	**0.90**
10000	QL	0.81	0.74	0.84
	DynaQ	**1.34**	-0.16	0
	DoubleQ	0.95	0.74	**0.95**
	FAQL	0.88	**0.80**	0.86
	RUQL	0.90	**0.85**	**0.95**
∞	QL	1.18	**1.16**	**1.19**
	DynaQ	1.2	-0.17	0.36
	DoubleQ	1.20	**1.15**	**1.2**
	FAQL	1.18	**1.16**	**1.18**
	RUQL	1.18	**1.16**	**1.18**

Table 4: The mean payoff for the 1-Row Domain, for different values of D and P. Each cell represents the average payoff, maximized over α and τ.

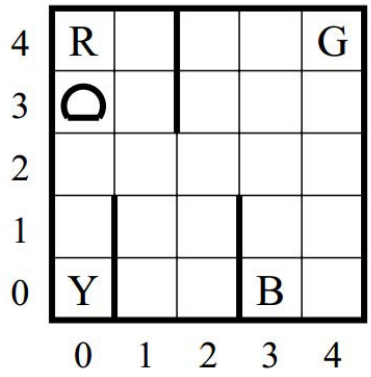


Figure 7: The Taxi domain (Dietterich, 2000).

- Rare but severe noise: PICK and DROP actions fail with probability p and result in penalty $\frac{R_p}{p}$.[13]
- Non-stationary domain: every duration D the stations (R,G,Y,B) rotate their locations (station R takes the location of station G which in turn takes the location of station Y, etc.).

We conducted the experiments over range of parameter values, including $D \in \{300000, 600000, \infty\}$, $\tau \in \{0.03, 0.1, 0.3, 1\}$, $\pi_{min} \in \{0, 0.001, 0.01\}$, $\alpha \in \{0.001, 0.01, 0.1\}$,[14] $p \in \{0, 0.01\}$, $R_p =$

13. For the navigation actions, the noise remains similar to the original Taxi domain: a navigation action fails with probability p and results in skidding to one of the sides with reward of -1.
14. Again the values of α are for QL and RUQL. For FAQL, we used the square root of these α values for both α_{FAQ} and β_{FAQ}, so that FAQL has an effective learning rate $\alpha_{FAQ}.\beta_{FAQ} = \alpha$.

D	Alg. \ p	0	0.01
∞	QL	**2.35**	-0.61
	DynaQ	**2.34**	0.19
	DoubleQ	2.27	-1.23
	RUQL	**2.37**	**0.45**
	FAQL	**2.38**	0.29
600000	QL	**1.79**	-1.31
	DynaQ	**1.83**	**-0.42**
	DoubleQ	1.17	-1.91
	RUQL	**1.77**	**-0.47**
	FAQL	**1.80**	-0.58
300000	QL	0.64	-1.46
	DynaQ	**1.38**	-0.79
	DoubleQ	0.11	-1.92
	RUQL	1.19	**-0.70**
	FAQL	0.77	-0.94

Table 5: The mean payoff for the Taxi domain, for different values of p and D.

-6, and $\gamma = 1$. For each combination we collected the average payoff over 1200000 consecutive time steps. All Q-values are initialized to zeros. Finally, statistical significance of the difference in the mean is computed over 100 independent simulation runs, using two-tail two-sample t-test, where the null hypothesis is rejected at p-value 0.05.

Table 5 shows the results for different values of D and p. For each algorithm, we report the best average payoff it achieved across the different parameter values. Bold entries confirm the statistical significance of the outperforming algorithm, similar to the results shown in Section 6.1. We notice similarities to the previous domain. In noise-free environments, again DynaQ outperforms other algorithms. As the environment becomes noisy and non-stationary, the performance of DynaQ drops significantly, while RUQL outperforms all other algorithms. Interestingly, DoubleQ performs the worst in this domain, across the different settings. This is perhaps due to the reported underestimation of DoubleQ for the Q values (Hasselt, 2010). It is also worth noting that even in stationary but noisy setting, RUQL outperforms other algorithms in the taxi domain. The reason for this is that RUQL can boost the learning rate more effectively than Q-learning and other algorithms. For example, suppose the best action in a particular state is to drop the customer. However, due to noise, the payoff will be highly negative. This will result in sudden drop in the Q-value for the DROP action in this particular state, and consequently the probability of choosing DROP. However, RUQL will boost the learning rate for the DROP action once it is tried and succeeds, therefore recovering from the noise effect faster. This results in a cascading effect across states. Also because RUQL's boost diminishes as the action is chosen more frequently, the learning becomes stable in the face of the noise (which can not be achieved via simple increase of the learning rate for QL, or using DynaQ). This is consistent with previous work, which experimentally showed that converging to a policy is in some cases inferior to continuous learning (tracking) even in stationary environments (Sutton et al., 2007). As the environment becomes more dynamic ($D = 600000$ and $D = 300000$) the performance of all algorithms drop (as expected) but RUQL maintains the superior performance.

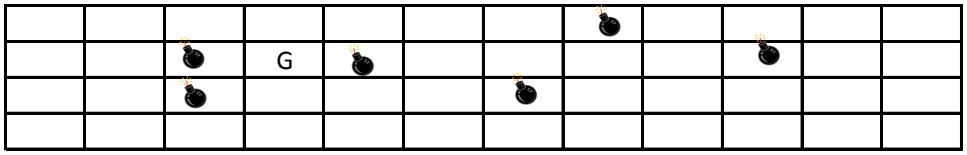


Figure 8: The Mines domain. The agent can navigate through the 4x16 grid, while avoiding the scattered mines and trying to reach the goal.

6.5 Mines Domain

The mines domain is illustrated by Figure 8 and is adopted from the RL-GLUE software (Tanner and White, 2009). The world consists of 4x16 grid, where the agent can navigate up, down, left and right (4 actions) and receives a reward of -1 unless it encounters a mine, in which case the reward is -100, or it reaches the goal state and receives the reward of 10. In this simple domain there are only 64 states. 6 mines in addition to the goal are scattered randomly in the grid. We modified the domain slightly from the original as follows:

- Rare but severe noise: reaching the goal state will fail with probability p and result in reward of $-1/p$, otherise will succeed and produce reward of $11/(1-p)$ (expected reward remains 10).
- Non-stationary domain: every duration D the goal state is randomly re-assigned.

We conducted the experiments over a range of parameter values, including $D \in \{25 \cdot 10^4, 50 \cdot 10^4, \infty\}$, $\tau \in \{0.008, 0.04, 0.2, 1, 5\}$, $p \in \{0, 10^{-2}\}$, $\alpha \in \{0.0025, 0.01, 0.04, 0.16\}$, $\pi_{min} \in \{0, 10^{-2}, 10^{-3}\}$, and $\gamma = 1$.[15] For each combination, we collected the average payoff over 10^6 consecutive time steps, and reported the best performance for each algorithm over the tested parameter values. All Q-values are initialized to zeros. Finally, statistical significance of the difference in the mean is computed over 20 independent simulation runs, using two-tail two-sample t-test, where the null hypothesis is rejected at p-value 0.05. Table 6 summarizes the results for different values of p and the duration D. We observe similar qualitative results to the previous domains. DynaQ is better in noise-free settings, with RUQL outperforming other algorithms as the environment becomes noisy and non-stationary. FAQL is close second in noisy and non-stationary environments as well, albeit RUQL is clearly superior in the most frequently changing benchmark environment with D=250,000.

7. Discussion of RUQL

Looking at the previous experimental results we can identify a few interesting observations. The main benefit of RUQL is reducing the policy-bias, which is reflected in better performance in non-stationary environments. In noise-free environments, however, the benefit of RUQL is diminishing, since for noise-free environments the learning rate can be as high as 1. As RUQL reduces the policy bias by boosting the learning rate for actions with low probability of being chosen, such

15. Again the values of α are for QL and RUQL. For FAQL, we used the square root of these α values for both α_{FAQ} and β_{FAQ}, so that FAQL has an effective learning rate $\alpha_{FAQ}.\beta_{FAQ} = \alpha$.

D	Alg. \ P	0	0.01
∞	QL	0.79	0.6
	DynaQ	0.78	0.35
	DoubleQ	0.78	0.54
	RUQL	0.79	**0.63**
	FAQL	0.79	**0.63**
500000	QL	0.70	0.43
	DynaQ	**0.77**	0.34
	DoubleQ	0.69	0.40
	RUQL	0.70	**0.49**
	FAQL	0.72	**0.48**
250000	QL	0.67	0.27
	DynaQ	**0.75**	0.29
	DoubleQ	0.63	0.39
	RUQL	0.65	**0.43**
	FAQL	0.66	0.33

Table 6: The mean payoff for the Mines domain, for different values of p and D.

benefit fades as the basic learning rate increases. In contrast, if the environment is noisier and changes more frequently, the benefits of RUQL become more apparent. This is consistent across the domains we have studied. Overall, RUQL retains the simplicity of QL, while showing robust performance across various experimental settings.

8. Conclusions

In this paper we proposed and evaluated the Repeated Update Q-learning algorithm, RUQL, which addresses the policy bias of Q-learning. Unlike the closest state-of-the-art algorithm, RUQL's behavior remains consistent even for policies with arbitrarily small probabilities of choosing actions. We show theoretically that our algorithm is guaranteed to converge to the optimal Q-values in single-agent stationary environments, while having better response to changes in non-stationary environments. We show experimentally how this results in superior performance in non-stationary and noisy environments, compared to 4 other algorithms.

Given its merits in non-stationary environments, RUQL may find interesting applications in multi-agent environments. There is a growing collection of algorithms that were designed specifically for multi-agent environments. The Win-or-Learn-Fast heuristic (Bowling and Veloso, 2002; Bowling, 2005) resulted in the first gradient-ascend-based (GAB) multi-agent learning algorithms that successfully converged to mixed Nash Equilibrium in small general-sum games with minimum knowledge of the underlying game (only the player's own payoff). This was followed by the more recent GAB algorithms (Abdallah and Lesser, 2008; Zhang and Lesser, 2010). Other multi-agent algorithms that assumed knowledge of the underlying game were also proposed and were able to converge in larger games (Hu and Wellman, 2003; Conitzer and Sandholm, 2007). All of these algorithms could benefit from improved value approximation techniques like the one presented in this paper. The integration of RUQL in specialized multi-agent learning algorithms is a promising avenue of future research.

Finally, it is important to note that RUQL is *not* intended as a replacement of Q-learning. In stationary environments with stochastic outcomes, QL may outperform RUQL, since the latter may slightly boost the learning rate of rarely chosen action. However, if the environment is actually changing over time (non-stationary and thus violating the core assumption of QL), then RUQL's boost may yield superior performance.

Acknowledgments

We would like to acknowledge support for this project from the Emirates Foundation Ref No: 2010-107 Sience & Engineering Research grant and British University in Dubai INF009 Grant.

Appendix A. Theoretical Derivations of Exact RUQL

Here we derive the different quantities that are needed for each of the four cases of the exact RUQL.

Case 1:

$$\begin{aligned} Q_{\lfloor \frac{1}{\pi(s,a)} \rfloor}(s,a) &= [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q_0(s,a) + \alpha[r + \gamma \max_{a'} Q_0(s',a')] \\ &\quad \left(1 + (1-\alpha) + (1-\alpha)^2 + ... + (1-\alpha)^{\lfloor \frac{1}{\pi(s,a)} \rfloor - 1}\right) \\ &= [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q_0(s,a) + \alpha[r + \gamma \max_{a'} Q_0(s',a')] \frac{[1-(1-\alpha)^{\lfloor \frac{1}{\pi(s,a)} \rfloor}]}{1-(1-\alpha)} \\ &= [1-\alpha]^{\lfloor \frac{1}{\pi(s,a)} \rfloor} Q_0(s,a) + [1-(1-\alpha)^{\lfloor \frac{1}{\pi(s,a)} \rfloor}][r + \gamma \max_{a'} Q_0(s',a')] \end{aligned}$$

We can remove the floor notation for a better generalization, and using the equalities $Q_0(s,a) = Q^t(s,a)$ and $Q_{\lfloor \frac{1}{\pi(s,a)} \rfloor}(s,a) = Q^{t+1}(s,a)$ we get (RUQL's main update rule):

$$Q^{t+1}(s,a) = [1-\alpha]^{\frac{1}{\pi(s,a)}} Q^t(s,a) + \left[1-(1-\alpha)^{\frac{1}{\pi(s,a)}}\right][r + \gamma Q^t(s',a_{max})]$$

Case 2:

$$\begin{aligned} Q_1(s,a) &= [1-\alpha]Q^t(s,a) + \alpha\left(r + \gamma Q^t(s,a)\right) \\ &= [1-\alpha+\alpha\gamma]Q^t(s,a) + \alpha r \\ Q_2(s,a) &= [1-\alpha+\alpha\gamma]Q_1(s,a) + \alpha r \\ &= [1-\alpha+\alpha\gamma]\left([1-\alpha+\alpha\gamma]Q^t(s,a) + \alpha r\right) + \alpha r \\ Q_n(s,a) &= [1-\alpha+\alpha\gamma]^n Q^t(s,a) + \alpha r\left(1 + [1-\alpha+\alpha\gamma] + \ldots + [1-\alpha+\alpha\gamma]^{n-1}\right) \\ &= [1-\alpha+\alpha\gamma]^n Q^t(s,a) + \alpha r \frac{1-[1-\alpha+\alpha\gamma]^n}{1-[1-\alpha+\alpha\gamma]} \\ &= [1-\alpha+\alpha\gamma]^n Q^t(s,a) + r \frac{1-[1-\alpha+\alpha\gamma]^n}{1-\gamma} \\ &= [1-\alpha+\alpha\gamma]^n Q^t(s,a) + (1-[1-\alpha+\alpha\gamma]^n)\frac{r}{1-\gamma} \end{aligned}$$

$$= [1-\alpha+\alpha\gamma]^{n}\left(Q^t(s,a)-\frac{r}{1-\gamma}\right)+\frac{r}{1-\gamma}$$
$$Q^{t+1}(s,a) = [1-\alpha+\alpha\gamma]^{\frac{1}{\pi(s,a)}}\left(Q^t(s,a)-\frac{r}{1-\gamma}\right)+\frac{r}{1-\gamma}$$

Case 3 using $Q^t(s,a'') = Q^t(s,a)$, where $a'' = \arg\max_{a'\in A\setminus a} Q^t(s,a')$:

$$Q^t(s,a'') = [1-\alpha+\alpha\gamma]^{i_\top}\left(Q^t(s,a)-\frac{r}{1-\gamma}\right)+\frac{r}{1-\gamma}$$
$$Q^t(s,a'')-\frac{r}{1-\gamma} = [1-\alpha+\alpha\gamma]^{i_\top}\left(Q^t(s,a)-\frac{r}{1-\gamma}\right)$$
$$\frac{Q^t(s,a'')-\frac{r}{1-\gamma}}{Q^t(s,a)-\frac{r}{1-\gamma}} = [1-\alpha+\alpha\gamma]^{i_\top}$$
$$i_\top = \frac{\log\left(\frac{Q^t(s,a'')-\frac{r}{1-\gamma}}{Q^t(s,a)-\frac{r}{1-\gamma}}\right)}{\log\left(1-\alpha+\alpha\gamma\right)}$$

Case 4 using $Q^t(s,a'') = Q^t(s,a)$, where $a'' = \arg\max_{a'\in A} Q^t(s,a')$:

$$Q^t(s,a'') = [1-\alpha]^{i_\perp}Q^t(s,a)+\left[1-(1-\alpha)^{i_\perp}\right]\left[r+\gamma Q^t(s,a'')\right]$$
$$Q^t(s,a'')-r-\gamma Q^t(s,a'') = [1-\alpha]^{i_\perp}Q^t(s,a)-(1-\alpha)^{i_\perp}[r+\gamma Q^t(s,a'')]$$
$$(1-\gamma)Q^t(s,a'')-r = [1-\alpha]^{i_\perp}\left[Q^t(s,a)-r-\gamma Q^t(s,a'')\right]$$
$$\frac{(1-\gamma)Q^t(s,a'')-r}{Q^t(s,a)-r-\gamma Q^t(s,a'')} = [1-\alpha]^{i_\perp}$$
$$\frac{(1-\gamma)Q^t(s,a'')-r}{Q^t(s,a)-\gamma Q^t(s,a'')-r} = [1-\alpha]^{i_\perp}$$
$$i_\perp = \frac{\log\left(\frac{(1-\gamma)Q^t(s,a'')-r}{Q^t(s,a)-\gamma Q^t(s,a'')-r}\right)}{\log\left(1-\alpha\right)}$$

References

Yasin Abbasi, Peter L Bartlett, Varun Kanade, Yevgeny Seldin, and Csaba Szepesvari. Online learning in markov decision processes with adversarially chosen transition probability distributions. In *Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 2508–2516. Curran Associates, Inc., 2013.

Sherief Abdallah and Michael Kaisers. Addressing the policy-bias of q-learning by repeating updates. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1045–1052, 2013.

Sherief Abdallah and Victor Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33:521–549, 2008.

Youssef Achbany, François Fouss, Luh Yen, Alain Pirotte, and Marco Saerens. Tuning continual exploration in reinforcement learning: An optimality property of the boltzmann strategy. *Neurocomputing*, 71(13):2507–2520, 2008.

Monica Babes, Enrique Munoz de Cote, and Michael L. Littman. Social reward shaping in the prisoner's dilemma. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1389–1392, 2008.

Omar Besbes, Yonatan Gur, and Assaf Zeevi. Optimal exploration-exploitation in a multi-armed-bandit problem with non-stationary rewards. *Available at SSRN 2436629*, 2014.

Michael Bowling. Convergence and no-regret in multiagent learning. In *Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 209–216, 2005.

Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002. ISSN 0004-3702.

Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *National Conference on Artificial intelligence/Innovative Applications of Artificial Intelligence*, pages 746–752, 1998.

Vincent Conitzer and Tuomas Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1-2):23–43, 2007. ISSN 0885-6125.

Bruno C. da Silva, Eduardo W. Basso, Ana L. C. Bazzan, and Paulo M. Engel. Dealing with non-stationary environments using context detection. In *International Conference on Machine Learning (ICML)*, pages 217–224. ACM, 2006.

William Dabney and Andrew G Barto. Adaptive step-size for online temporal difference learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

Jose Enrique Munoz de Cote, Alessandro Lazaric, and Marcello Restelli. Learning to cooperate in multi-agent social dilemmas. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 783–785, 2006.

Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Rodrigues Gomes Eduardo and Ryszard Kowalczyk. Dynamic analysis of multiagent q-learning with ϵ-greedy exploration. In *International Conference on Machine Learning (ICML)*, pages 369–376. ACM, 2009.

Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. MIT press, 1998.

Aram Galstyan. Continuous strategy replicator dynamics for multi-agent q-learning. *Autonomous Agents and Multi-Agent Systems*, 26(1):37–53, 2013. ISSN 1387-2532.

Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. In *International Conference on Algorithmic Learning Theory*, pages 174–188, 2011.

Abraham P George and Warren B Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine learning*, 65(1):167–198, 2006.

Hado V Hasselt. Double q-learning. In *Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 2613–2621, 2010.

Junling Hu and Michael P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003. ISSN 1533-7928.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 99:1563–1600, 2010.

Michael Kaisers and Karl Tuyls. Frequency adjusted multi-agent q-learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 309–316, 2010. ISBN 978-0-9826571-1-9.

Alessandro Lazaric, Jose Enrique Munoz de Cote, Fabio Dercole, and Marcello Restelli. Bifurcation analysis of reinforcement learning agents in the selten's horse game. In *Workshop on Adaptive Agents and Multi-Agents Systems*, pages 129–144, 2007.

David S. Leslie and Edmund J. Collins. Individual q-learning in normal form games. *SIAM J. Control and Optimization*, 44(2):495–514, 2005.

Ashique Rupam Mahmood, Richard S Sutton, Thomas Degris, and Patrick M Pilarski. Tuning-free step-size adaptation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2121–2124. IEEE, 2012.

Koichi Moriyama. Utility based q-learning to facilitate cooperation in prisoner's dilemma games. *Web Intelligence and Agent Systems*, 7:233–242, August 2009.

Koichi Moriyama, Satoshi Kurihara, and Masayuki Numao. Evolving subjective utilities: Prisoner's dilemma game examples. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 233–240, Richland, SC, 2011.

Gergely Neu, Andras Antos, András György, and Csaba Szepesvári. Online markov decision processes under bandit feedback. In *Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 1804–1812, 2010.

Itsuki Noda. Recursive adaptation of stepsize parameter for non-stationary environments. In MatthewE. Taylor and Karl Tuyls, editors, *Adaptive and Learning Agents*, volume 5924 of *Lecture Notes in Computer Science*, pages 74–90. Springer Berlin Heidelberg, 2010.

Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

Tuomas W. Sandholm and Robert H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37(1–2):147 – 166, 1996.

Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10:2413–2444, 2009.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning (ICML)*, pages 216–224, 1990.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1999.

Richard S Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *International Conference on Machine learning (ICML)*, pages 871–878. ACM, 2007.

Brian Tanner and Adam White. RL-Glue: language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, September 2009.

Gerald Tesauro. Extending q-learning to general adaptive multi-agent systems. In *Annual Conference on Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2004.

Karl Tuyls, Pieter Jan 't Hoen, and Bram Vanschoenwinkel. An evolutionary dynamical analysis of multi-agent learning in iterated games. *Autonomous Agents and Multi-Agent Systems*, 12(1): 115–153, 2006.

Peter Vrancx, Karl Tuyls, and Ronald Westra. Switching dynamics of multi-agent learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 307–313, 2008. ISBN 978-0-9817381-0-9.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Michael Wunder, Michael L. Littman, and Monica Babes. Classes of multiagent q-learning dynamics with ϵ-greedy exploration. In *International Conference on Machine Learning (ICML)*, pages 1167–1174, 2010.

Jia Yuan Yu, Shie Mannor, and Nahum Shimkin. Markov decision processes with arbitrary reward processes. *Mathematics of Operations Research*, 34(3):737–757, 2009.

Chongjie Zhang and Victor Lesser. Multi-agent learning with policy prediction. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 927–934, 2010.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *International Conference on Machine Learning (ICML)*, pages 928–936, 2003.