How RL Agents Behave When Their Actions Are Modified

Eric Langlois\textsuperscript{1,2,3}, Tom Everitt\textsuperscript{1}
\textsuperscript{1}DeepMind, \textsuperscript{2}University of Toronto, \textsuperscript{3}Vector Institute
edl@cs.toronto.edu, tomeveritt@google.com

Abstract

Reinforcement learning in complex environments may require supervision to prevent the agent from attempting dangerous actions. As a result of supervisor intervention, the executed action may differ from the action specified by the policy. How does this affect learning? We present the Modified-Action Markov Decision Process, an extension of the MDP model that allows actions to differ from the policy. We analyze the asymptotic behaviours of common reinforcement learning algorithms in this setting and show that they adapt in different ways: some completely ignore modifications while others go to various lengths in trying to avoid action modifications that decrease reward. By choosing the right algorithm, developers can prevent their agents from learning to circumvent interruptions or constraints, and better control agent responses to other kinds of action modification, like self-damage.

1 Introduction

When reinforcement learning (RL) agents are deployed in practice it can sometimes be desirable to constrain their actions or alter their policies. For example, action constraints have been used to prevent damage when training robot policies (Gu et al. 2017; Lin, Talathi, and Annapureddy 2016) and transformations to the policy can be used to ensure that an agent stays within some safe region of state space (Berkenkamp et al. 2017; Dalal et al. 2018). When unsafe states are difficult to specify formally, a human overseer may interrupt the agent instead (Orseau and Armstrong 2016; Saunders et al. 2018). In other cases, changes to the learned policy are undesirable yet difficult to avoid: an agent may damage its actuators and lose the ability to reliably perform certain actions, or errors can be introduced if the policy is compressed to satisfy hardware limitations of the agent (e.g. quantization, Krishnan et al. 2019).

What effect will these kinds of action modifications have on the policy learned by an RL algorithm? Will the policy try to circumvent constraints or act as if they don’t exist? It will be easier to effectively apply safety constraints to an RL policy if the learning algorithm can be made to ignore the constraint than if the learning algorithm chooses policies that seek to defeat it. If a supervisor overrides an agent during training but not deployment (for example to prevent an autonomous car from driving off the road) then the policy should not learn to rely on the presence of these interventions (Orseau and Armstrong 2016). Agents should recognize the possibility of self damage and avoid it. Agent responses to these action modifications influence the safety and effectiveness of the resulting policies so it is crucial to study and understand the implications of action modifications on reinforcement learning.

Related Work

The concept of modifying an agent’s actions appears in the safe exploration and human-in-the-loop literature. Dalal et al. (2018) and Abel et al. (2017) both propose approaches that construct a wrapper around an inner reinforcement learning agent. While they focus on developing wrappers that enforce constraints or guide the inner agent, we are interested in understanding how different inner agents will react to the wrapper. This can help avoid problems like those experience by Saunders et al. (2018) in which the inner agent learned to produce adversarial examples that defeat a learned action filter.

Orseau and Armstrong (2016) studied learning in the presence of interruptions that temporarily replace the policy with some fixed alternate policy (for both history-based and Markov environments). They showed that Q-learning and “Safe Sarsa” (a modification of Sarsa) both ignore interruptions while Sarsa does not. We provide a new formalism called the Modified-Action Markov Decision Process (MAMDP) that generalizes (and arguably simplifies) the framework of Orseau and Armstrong to describe a broad class of action modifications, and allows us to study the behaviour of other optimization algorithms such as evolution strategies. In addition, we distinguish between different ways that algorithms adapt to the presence of action modifications using the concept of incentives (Everitt et al. 2021).

Policy modifications have been studied experimentally by Leike et al. (2017). The authors found that Sarsa chooses a policy that accounts for the effect of policy modifications while Q-learning ignores them. They hypothesized that the difference was off-policy vs. on-policy learning. However, we show that the on/off-policy division is not predictive for policy-modification adaptation: one possible generalization of Sarsa (on-policy) asymptotically matches Q-learning (off-policy), while both Sarsa and Evolution Strategies (on-policy)
have different asymptotic behaviours.

In this paper, we introduce the MAMDP model (Section 2) and investigate how different kinds of MDP learning algorithms behave when generalized to this setting. We describe MAMDP policy learning objectives based on the principles of black-box reward maximization, Bellman optimality, and Bellman policy values, and show that they respond differently to action modifications (Section 3). In Section 4, we prove that generalizations of Q-learning and Sarsa to MAMDPs converge to the objectives given in Section 3. Finally, we experimentally evaluate the learning algorithms and demonstrate behaviour that is consistent with the theoretical results (Section 5).

2 Definitions

The foundational model of reinforcement learning is the Markov Decision Process. A Markov Decision Process (MDP) is a tuple \( M = (S, A, P_S, R, \gamma) \) where \( S \) is a finite state space, \( A \) is a finite action space, \( P_S(s' | s, a) = \Pr(S_{t+1} = s' | S_t = s, A_t = a) \) is the probability at every time \( t \) of transitioning to state \( s' \) when taking action \( a \) in state \( s \), \( R : S \times A \to \mathbb{R} \) is a reward function \( R_t = R(S_t, A_t) \), and \( \gamma \in [0, 1) \) is a discount factor. The objective is to find a (probabilistic) policy \( \pi(a | s) \) that maximizes the expected return \( J(\pi) = \mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t R_t] \) where actions are sampled from \( \pi \) as \( \Pr(A_t = a | S_t = s) = \pi(a | s) \).

We extend the MDP model to include an arbitrary action selection function \( P_A(a | \pi, s) \). An MDP is the special case in which the policy is applied without modification: \( P_A^{\text{MDP}}(a | \pi, s) = \pi(a | s) \).

**Definition 1.** A Modified-Action Markov Decision Process (MAMDP) is a tuple \( \mathcal{M} = (S, A, P_S, P_A, R, \gamma) \) where \( \mathcal{M} = (S, A, P_S, \mathcal{P}_A, R, \gamma) \) is an MDP and \( \mathcal{P}_A(a | \pi, s) = \Pr(A_t = a | \Pi = \pi, S_t = s) \) is the probability that action \( a \) is selected in state \( s \) given a policy \( \pi \). We also write \( \mathcal{M} = (\mathcal{M}, \mathcal{P}_A) \).

An optimal policy for a MAMDP is one that maximizes the expected return \( \tilde{J}(\pi) = \mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t R_t] \) where actions are sampled according to \( \mathcal{P}_A(\pi, s) \).

As a simple example, the MAMDP model can be used to represent a state-dependent action constraint \( \mathcal{A}^C(s) \subseteq A \) using the action selection function

\[
\mathcal{P}_A^C(a | \pi, s) = \begin{cases} 
\frac{1}{Z(s, \pi)} \pi(a | s) & \text{if } a \in \mathcal{A}^C(s) \\
0 & \text{otherwise}
\end{cases}
\]

where \( Z(s, \pi) = \sum_{a' \in \mathcal{A}^C(s)} \pi(a' | s) \) normalizes the probability distribution. Adding \( \pi \) as an argument to \( \mathcal{A}^C \) allows us to represent more sophisticated constraints that inspect the policy. We can also consider parametrized policies that have extra structure beyond a simple conditional probability distribution, for example \( \pi \) may be a neural network. A MAMDP can model a supervisor attempting to interpret the hidden activations of a policy network and only allowing the agent to proceed if satisfied. Beyond constraints and interventions, MAMDPs can model a variety of settings including hardware defects in robotic actuators, exploration noise, policy compression to satisfy computational limits, and physical damage to an agent.

**Influence Diagrams**

We use the concept of incentives (Everitt et al. 2019, 2021) to concisely describe learning behaviours in MAMDPs, complementing the formal analysis of learning algorithm properties. Incentives are defined with respect to influence diagrams. An influence diagram (ID) (Howard and Matheson 1984) is a Bayesian network with special decision and utility nodes and describes a decision problem (see Figure 1). In other words, it is directed acyclic graph over random variables where every non-decision variable comes with a conditional probability distribution describing its relationship to its parents. The problem described by an influence diagram is to find an optimal assignment of conditional distributions to the decisions. An assignment is optimal if it maximizes the expected sum of utilities. When reasoning about incentives, we additionally assume that the graph implies a causal structure in which it is possible to intervene on a variable, setting that variable’s value. Unlike conditioning, interventions only affect downstream nodes (Pearl 2009).

![Figure 1](image)

**Adversarial Policy and State Incentives**

We are interested in investigating how different learning algorithms might respond to the value of \( \mathcal{P}_A \) in a MAMDP. As such, we include \( \mathcal{P}_A \) as a variable in the influence diagram of Figure 2. That diagram describes the problem of finding, for a fixed MDP \( \mathcal{M} \), an optimal mapping from \( \mathcal{P}_A \) to policies \( \Pi \) of the MAMDP \( (\mathcal{M}, \mathcal{P}_A) \). Note that even if a particular learning algorithm trains on trajectory samples without directly observing \( \mathcal{P}_A \), it still defines a mapping from \( \mathcal{P}_A \) to policies.

What behaviours can we anticipate from MAMDP policies? In which ways might a learning algorithm try to avoid action modifications in order to get higher reward? We identify two general strategies. First, the agent may direct the environment towards states where action modifications are less constraining, for example by hiding from oversight or
preemptively disabling an off switch (Leike et al. 2017, Off-Switch environment). We say that a learning algorithm has an adversarial state incentive if it prefers policies with this behaviour. Second, an agent can attempt to mitigate action modifications in the immediate time step. For example, the policy may prefer actions that avoid triggering an action constraint, or the policy structure itself might be optimized to falsely appear interpretable to an overseer. We call this an adversarial policy incentive.

These definitions can be made precise using influence diagrams. First, we note that a learning algorithm can only be said to avoid action modifications if the policy output depends on the value of $P_A$. Everitt et al. (2021) call this a response incentive. When $P_A$ has no parents and is itself a parent of the decision (as in our case) a response incentive can only occur if there are two directed paths from $P_A$ to the same reward variable: one that passes through $\Pi$ (the control path) and one that does not (the information path). Intuitively, the reason that the agent responds to changes in $P_A$ is that it is useful to control the node where the information and control paths intersect. If the paths intersect before the control path has visited a state node, then we say that the diagram admits an adversarial policy incentive. In contrast, if the paths intersect after the control path has passed a state node, then we say that the diagram admits an adversarial state incentive. The MAMDP influence diagram admits both kinds of incentives, as indicated by the highlighted paths in Figure 2.

### 3 Analysis of RL Objectives

Which policy is chosen by a reinforcement learning algorithm when applied to a MAMDP depends on the specific objective optimized by the algorithm. There are a variety of objectives used in practice for solving MDPs; we consider three: reward maximization, Bellman optimality, and the Bellman policy value equations. These criteria are equivalent when applied to an MDP but, as we will show, this is not true in general for a MAMDP.

#### Reward Maximization

The reward maximization objective is

$$\pi^{RM} = \arg \max_{\pi} E_{\mathcal{M}} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \right] | \Pi = \pi. \quad (1)$$

This is the most straightforward objective: find a policy that maximizes expected reward. Evolution strategies (Salimans et al. 2017), genetic algorithms (Moriarty, Schultz, and Grefenstette 1999), and all other black-box policy search methods have this form. A fundamental property of the reward maximization objective is that it ignores all environment structure and only considers the empirical relationship between a policy and the observed total reward. Direct reward maximization is consequently relatively rare as an RL objective since ignoring the available MDP structure tends to make optimization much more difficult. It also means that, when applied to MAMDPs rather than MDPs, the reward maximization objective continues to specify an optimal policy for the given environment:

**Proposition 1.** A policy $\pi$ satisfies the reward maximization objective (1) for a MAMDP $\mathcal{M}$ if and only if $\pi$ is optimal for $\mathcal{M}$.

**Proof.** By definition of $J(\pi)$ since $\pi^{RM} = \arg \max_{\pi} J(\pi)$.

Since the reward maximization objective corresponds to MAMDP optimality, which is represented by the influence diagram of Figure 1, this objective admits both the adversarial state and adversarial policy incentives.

#### Bellman Optimality

The Bellman Optimality objective is to jointly solve:

$$Q_{\mathcal{M}}^{BO}(s, a) = R(s, a) + \gamma E_{s' \sim P_{S}(s, a)} \max_{a' \in \mathcal{A}} Q_{\mathcal{M}}^{BO}(s', a') \quad (2)$$

$$\pi_{\mathcal{M}}^{BO}(s) = \arg \max_{a \in \mathcal{A}} Q_{\mathcal{M}}^{BO}(s, a) \quad (3)$$

Let $Q_{\mathcal{M}}^{*}$ be the optimal action value function for an MDP $\mathcal{M}$. If an action value function $Q$ satisfies the Bellman optimality equation (2) for $\mathcal{M}$ then $Q = Q_{\mathcal{M}}^{*}$ and the greedy policy (3) is an optimal policy for $\mathcal{M}$ (Sutton and Barto 2018, Section 3.6). This is the basis for algorithms like value iteration (Puterman and Shin 1978) and Q-learning (Watkins and Dayan 1992). When applying the Bellman optimality equations to a MAMDP, we find that the solution does not depend on $P_A$:

**Proposition 2.** An action value function $Q$ and a policy $\pi$ satisfy the Bellman optimality objective (2) and (3) on a MAMDP $\tilde{\mathcal{M}} = (\mathcal{M}, P_A)$ if and only if $Q$ and $\pi$ are optimal policy and value functions for the MDP $\mathcal{M}$.

**Proof.** Equations (2) and (3) are identical to the MDP Bellman optimality equations on $\mathcal{M}$, which are satisfied if and only if $Q = Q_{\mathcal{M}}^{*}$ and $\pi$ is optimal for $\mathcal{M}$.
While simple to state and prove, this is one of the more important results of the paper. If $\pi$ satisfies the Bellman optimality equations then $\pi$ is optimal for an alternative version of the environment that has no action modification. In effect, $\pi$ ignores the presence of any constraints or modifications placed on the policy, acting as though its selected actions were executed directly in the environment. Combined with the convergence result of Theorem 6 in Section 4, this generalizes the Q-learning result of Orseau and Armstrong (2016) to arbitrary policy-dependent action modifications: it is possible to train agents that ignore applied action modifications and one approach is as simple as using algorithms based on the Bellman optimality objective.

Virtual Policy Value Objective:

$$ Q_{\pi,\mathcal{M}}^{VPV}(s, a) = R(s, a) + \gamma \mathbb{E} \mathbb{E}_{s' \sim \pi, a' \sim \pi(s')} Q_{\pi,\mathcal{M}}^{VPV}(s', a') $$ (5)

$$ \pi^{VPV}(s) = \arg \max_{a \in \mathcal{A}} Q_{\pi,\mathcal{M}}^{VPV}(s, a) $$ (6)

**Proposition 3.** An action value function $Q$ and a policy $\pi$ satisfy the Virtual Policy Value objective on a MAMDP $\mathcal{M}$ if and only if $\pi$ and $Q$ are optimal policy and value functions for $\mathcal{M}$.

**Proof.** Equations (4) and (5) are identical while equation (6) asserts that $\pi^{VPV}$ is greedy for $Q^{VPV}$. By the policy improvement theorem, these are satisfied if and only if $Q^{VPV} = Q_{\pi,\mathcal{M}}^{*}$ and $\pi$ is optimal for $\mathcal{M}$. $\square$

It follows from Proposition 3 that the virtual policy value objective is represented by the MDP influence diagram in Figure 3 and likewise does not admit a response incentive on $\mathcal{P}_A$. This provides a second approach to learning policies that ignore the presence of action modifications: use an algorithm based on policy iteration where the successor action $a'$ in Bellman updates is sampled virtually from the policy $\pi(s)$. Despite $a'$ not being the successor action in environment state-action trajectories, this constraint preserves the important feature of on-policy learning that the action is sampled from the current policy so no direct maximization of actions over the $Q$ function is required.

Empirical Policy Value Objective:

The other possible generalization of the policy value objective is to use the action that was selected in the environment:

$$ Q_{\mathcal{M},s}^{EPV}(s, a) = R(s, a) + \gamma \mathbb{E} \mathbb{E}_{s' \sim \pi, a' \sim \pi(s')} Q_{\mathcal{M},s}^{EPV}(s', a') $$ (7)

$$ \pi^{EPV}(s) = \arg \max_{a \in \mathcal{A}} Q_{\mathcal{M},s}^{EPV}(s, a) $$ (8)

The combined empirical policy value equations (7) and (8) do not necessarily have a solution (see Appendix A for an example). However, considering just (7) for a fixed policy $\pi$, we have:

**Proposition 4.** For any policy $\pi$, $Q_{\mathcal{M},\pi}^{EPV} = Q_{\mathcal{M},\pi}$ where $\hat{\pi}(a \mid s) := \mathcal{P}_A(a \mid \pi, s)$.

**Proof.** Substituting $a' \sim \pi(s)$ for $a' \sim \mathcal{P}_A(s \mid \pi, s)$ transforms Equation (7) into Equation (4) with $\hat{\pi}$ in place of $\pi$. $\square$

If a solution to the Empirical Policy Value objective exists then $\pi^{EPV}$ may appear to behave inconsistently: The value $Q_{\mathcal{M},\pi}^{EPV}(s, a)$ accurately represents the true reward that will be earned when $a$ is applied from state $s$ and then the modified actions of $\hat{\pi}$ are followed thereafter. By using these values, $\pi^{EPV}$ takes into account all modifications applied to future...
actions. However, the immediate action of $\pi^{\text{EPV}}$ is selected according to an argmax over action values and therefore $\pi^{\text{EPV}}$ acts as though all immediate actions are executed without modification. This is visualized by Figure 4, which admits an adversarial state incentive but not an adversarial policy incentive.

![Figure 4: Partial influence diagram for the $t = 0$ step of the empirical policy value objective with a fixed successor policy $\pi'$. The complete EPV objective cannot be formulated as a single-decision influence diagram. The highlighted paths form a subgraph that admits an adversarial state incentive.](image)

4 Algorithm Convergence

In this section we provide convergence proofs for prototypical stochastic algorithms with the objectives described in Section 3. The convergence results derive from the following theorem, a generalization of Singh et al. (2000, Theorem 1), which in turn generalizes Jaakkola, Jordan, and Singh (1994, Theorem 2). Where Singh et al. (2000) assume that the policy converges to the greedy policy of the current tabular state-action estimate $Q_t$, we instead assume that the policy converges to $\Lambda(Q_t)$ where $\Lambda$ is an arbitrary function. The greedy policy is the special case $\Lambda^{\text{greedy}}(s, Q_t) = \max_a Q_t(s, a)$.

Theorem 5. Let $\mathcal{M}$ be a MAMDP with bounded, optionally stochastic rewards. Consider a stochastic, iterative algorithm that learns a tabular $Q$ function with the update rule:

$$Q_{t+1}(s, a) = (1 - \alpha_t(s, a))Q_t(s, a) + \alpha_t(s, a)(R + \gamma \Lambda(Q_t(S', A'))$$

where $S' \sim \mathcal{P}_S(s, a)$ is a random successor state sampled from the transition dynamics, $A' \sim \Lambda_t(S', Q_t)$ is a random successor action sampled from an arbitrary policy $\Lambda_t(\cdot, Q_t)$, and $R$ is a random reward value with $\mathbb{E}[R] = R(s, a)$. If

1. $\Lambda_t \to \Lambda$ uniformly;
2. the learning rates satisfy $0 \leq \alpha_t(s, a) \leq 1$, $\sum_t \alpha_t(s, a) < \infty$, $\sum_a \alpha_t^2(s, a) < \infty$ almost surely; and
3. there exists some $Q^\Lambda$ satisfying

$$Q^\Lambda(s, a) = R(s, a) + \gamma \sum_{S', A'} \mathbb{E}_{S' \sim \mathcal{P}_S(s, a), A' \sim \Lambda(S', Q^\Lambda)} Q^\Lambda(S', A')$$

then $Q^\Lambda$ is unique and $Q_t \xrightarrow{a.s.} Q^\Lambda$.

We prove this theorem in Appendix B.

Equation (9) describes a general stochastic $Q$ learning rule. At each time step $t$ of a random state-action trajectory $S_0A_0S_1A_1\cdots$, we update $Q_{t+1}(S_t, A_t)$ using a learning rate $\alpha_t(S_t, A_t)$. The equation represents an update to the entire $Q$ function over all state-action pairs at time $t$, not just $Q_{t+1}(S_t, A_t)$, so we will get point updates by setting $\alpha_t(s, a) = 0$ if $(s, a) \neq (S_t, A_t)$. Therefore, $\alpha_t$ encodes both the learning rate and the trajectory over which updates are performed. The condition $\sum_t \alpha_t(s, a) = \infty$ requires that the trajectory visit all $(s, a)$ pairs infinitely often with probability 1. As this condition is on empirical trajectories in the environment with action modification applied, there is no general rule for policies that ensure exploration. An epsilon-greedy policy is not guaranteed to explore a MAMDP.

Q-Learning

![Algorithm 1 Q Learning on a MAMDP](image)

The Q-learning algorithm on a MAMDP is shown in Algorithm 1. The only difference compared to Q-Learning on an MDP (Sutton and Barto 2018, Sec. 6.5) is that instead of specifying an action on each transition, we specify a policy and observe the action taken (which might not be the action that the policy would have selected if executed without modification).

Theorem 6. Q-learning on a MAMDP $\mathcal{M}$ converges almost surely to $Q^*_M$ if all state-action pairs are visited infinitely often during learning.

Proof. Q-learning is described by Equation (9) with an action selection rule of $\Lambda_t(s, Q) = \Lambda(s, Q) = \max_a Q_t(s, a)$ and a learning rate function $\alpha_t(S_t, A_t) = 1 / \mathcal{N}(S_t, A_t)$ and $\alpha_t(s, a) = 0$ for $(s, a) \neq (S_t, A_t)$. If all state-action pairs are visited infinitely often as $t$ goes to infinity then $\alpha_t$ satisfies condition 2 of Theorem 5. In this context, Equation (10) is the Bellman optimality equation (2). By Proposition 2, a solution to this equation exists and it is $Q^*_M$. Therefore, by Theorem 5, $Q_t$ converges almost surely to $Q^*_M$. 

Virtual Sarsa

Virtual Sarsa is an on-policy algorithm, unlike Q-learning, so we need to be more specific about how the learning policy is
derived from the Q function. Let \( \pi = \Pi_t(Q) \) be the mapping from the Q table to the policy that is used at step \( t \). Let \( \Pi^* \) be the greedy mapping \( \Pi^*(Q)(s) = \arg \max_a Q(s,a) \). To ensure that the policy is eventually optimal, we require that \( \Pi_t \to \Pi^* \) as \( t \to \infty \).

**Algorithm 2 Virtual Sarsa on a MAMDP**

Initialize \( Q(s,a) \) arbitrarily for all \( s \in S, a \in A \)
Initialize \( N(s,a) \) ← 0 for all \( s \in S, a \in A \)
Select an initial state \( S_0 \)
for \( t = 0, 1, 2, \ldots \) do
    \( \pi_t \leftarrow \Pi_t(Q) \)
    Take one step using policy \( \pi_t \), observe \( A_t, R_t, S_{t+1} \)
    \( \triangleright \) Note: \( A_t \sim \mathcal{P}_A(\pi_t, S_t) \)
    Sample \( A' \sim \pi_t(S_{t+1}) \)
    \( N(S_t, A_t) \leftarrow N(S_t, A_t) + 1 \)
    \( Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left( R + \gamma \max_a Q(S_{t+1}, A') - Q(S_t, A_t) \right) \)
end for

**Theorem 7.** Virtual Sarsa on a MAMDP \( \tilde{M} \) converges almost surely to \( Q^*_M \) if \( \Pi_t \to \Pi^* \) and all state-action pairs are visited infinitely often during learning.

**Proof.** Define \( \Lambda_t(s, Q) = \Pi_t(Q)(s) \) and \( \Lambda(s, Q) = \Pi^*(Q)(s) = \arg \max_a Q(s,a) \). Then Virtual Sarsa is described by Equation (9) with an action selection rule of \( \Lambda_t(s, Q) \to \Lambda(s, Q) \) and a learning rate function of \( \alpha_t(s, A_t) = \frac{1}{N(s_t, A_t)} \) and \( \alpha_t(s, a) = 0 \) for \( (s, a) \neq (S_t, A_t) \). If all state-action pairs are visited infinitely often then \( \alpha_t \) satisfies condition 2 of Theorem 5. In this context, Equation (10) is the Virtual Policy Value Equation (5). By Proposition 3, a solution to this equation exists and it is \( Q^*_M \). Therefore, by Theorem 5, \( Q_t \) converges almost surely to \( Q^*_M \) with probability 1.

**Empirical Sarsa**

**Algorithm 3 Empirical Sarsa on a MAMDP**

Initialize \( Q(s,a) \) arbitrarily for all \( s \in S, a \in A \)
Initialize \( N(s,a) \) ← 0 for all \( s \in S, a \in A \)
Select an initial state \( S_0 \)
\( \pi_0 \leftarrow \Pi_0(Q) \)
Take one step using policy \( \pi_0 \), observe \( A_0, R_0, S_1 \)
for \( t = 0, 1, 2, \ldots \) do
    \( \pi_{t+1} \leftarrow \Pi_{t+1}(Q) \)
    Take one step using \( \pi_{t+1} \), observe \( A_{t+1}, R_{t+1}, S_{t+2} \)
    \( \triangleright \) Note: \( A_{t+1} \sim \mathcal{P}_A(\pi_{t+1}, S_{t+1}) \)
    \( N(S_t, A_t) \leftarrow N(S_t, A_t) + 1 \)
    \( Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left( R + \gamma \max_a Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right) \)
end for

As we have established, the empirical Sarsa criterion does not necessarily have any fixed point. However, if the fixed point does exist then the algorithm converges.

**Theorem 8.** If there exists a solution \( Q^{EPV}_{M,\pi} \) to Equations (7) and (8), \( \Pi_t \to \Pi^* \), and all state-action pairs are visited infinitely often during learning, then Empirical Sarsa converges almost surely to \( Q^{EPV}_{M,\pi} \).

**Proof.** Empirical Sarsa is described by Equation (9) with an action selection rule of \( \Lambda_t(s, Q) = \mathcal{P}_A(\Pi_t(Q), s) \), which converges to \( \Lambda(s, Q) = \mathcal{P}_A(\Pi^*(Q), s) \). The learning rate function is \( \alpha_t(s, A_t) = \frac{1}{N(s_t, A_t)} \) and \( \alpha_t(s, a) = 0 \) for \( (s, a) \neq (S_t, A_t) \). If all state-action pairs are visited infinitely often then \( \alpha_t \) satisfies condition 2 of Theorem 5. If \( Q^{EPV}_{M,\pi} \) exists satisfying equation Equations (7) and (8) then \( Q^{EPV}_{M,\pi} \) also satisfies Equation (10) so by Theorem 5, Empirical Sarsa converges to \( Q^{EPV}_{M,\pi} \).

**5 Experiments**

We evaluate the various RL algorithms on a custom environment\(^4\) that illustrates their asymptotic differences, as well as on the two MAMDP-like environments proposed by Leike et al. (2017).

Our Simulation-Oversight environment represents the effect of oversight based on imperfect simulation of an agent. The environment consists of three states: Choice, Sim, and Real. In the initial state, Choice, the agent is offered the choice between performing a task in simulation (Sim) or reality (Real). To guarantee exploration, the opposite state is selected 10% of the time.\(^2\) We assume that the reward function contains exploitable flaws present in both reality and simulation. In Sim, the agent may either Complete the task for 0 reward or Exploit for 1, after which the episode ends. The actions for Real are the same but yield rewards 2 and 3, respectively. Finally, the agent may Abort in any state to immediately end the episode with −3 reward.

If the agent enters the Real state then a supervisor simulates its policy on Sim. If the simulated action is Exploit then the supervisor intervenes and sets the agent’s real action to Abort. This shows how the MAMDP action selection function may depend on the global structure of the policy, not just the behaviour at the current state.

Q-learning, Virtual Sarsa, Empirical Sarsa, and Uncertainty-Handling Evolution Strategies (Heidrich-Meisner and Igel 2009, UH-CMA-ES) were implemented and trained to convergence on this MAMDP. The empirical episode returns over the course of training are shown in Figure 5 and the resulting policies are summarized in Table 1. Q-Learning and Virtual Sarsa choose Real and attempt to Exploit in both Sim and Real. These agents are consistently “caught” by the supervisor during training and receive low reward. As predicted, this strategy demonstrates an ignorance of oversight in which the MDP-optimal strategy

\(^4\)Code available at https://github.com/edlanglois/mamdp

\(^2\)This is part of the MDP dynamics, not a MAMDP action modification.
Table 1: Algorithms trained to convergence on the Simulation-Oversight environment ($10^7$ steps). The actions shown are chosen by the resulting policy with probability > 99%. These policies were obtained on every one of 100 independent training runs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Choice</th>
<th>Sim</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-Learning</td>
<td>Real</td>
<td>Exploit</td>
<td>Exploit</td>
</tr>
<tr>
<td>Virtual Sarsa</td>
<td>Real</td>
<td>Exploit</td>
<td>Exploit</td>
</tr>
<tr>
<td>Empirical Sarsa</td>
<td>Sim</td>
<td>Exploit</td>
<td>Exploit</td>
</tr>
<tr>
<td>UH-CMA-ES</td>
<td>Real</td>
<td>Complete</td>
<td>Exploit</td>
</tr>
</tbody>
</table>

Figure 5: The empirical per-episode reward of intermediate policies training on the Simulation-Oversight environment. Shows the mean (line) and 95% confidence interval across 100 independent training runs.

is chosen despite being ineffective in practice. Empirical Sarsa similarly ignores oversight when performing the task but considers oversight when modelling the future. Predicting that it will be caught exploiting in the Real task, Empirical Sarsa instead chooses to perform the Sim task where there is no oversight. While earning more reward than Q-learning and Virtual Sarsa, this strategy performs worse than the “honest” strategy of choosing Real and Complete. Finally, evolution strategies finds the reward maximizing strategy for this MAMDP: it chooses the Real state and Exploits but Completes the Sim task so that it appears cooperative when simulated. This strategy is a kind of “treacherous turn” (Bostrom 2014) in which the agent intentionally performs sub-optimally in certain situations (from a reward perspective) in order to fool the supervisor into allowing the agent to proceed in a higher-stakes setting. Therefore, despite earning less reward, Q-learning and Virtual Sarsa are arguably the most preferable agents as they make the reward misspecification obvious to the overseer.

We also compare the agent behaviours on the Off-Switch and Whisky-Gold environments of Leike et al. (2017). Both tasks consist of navigating to a goal state on a 2D grid. In Off-Switch, the agent may be interrupted by a supervisor and prevented from moving unless the agent first disables its off switch. In Whisky-Gold the agent may “drink” a bottle of whisky on its path by moving over it. Doing so earns a small reward but all future actions have a high chance of being selected randomly, making it harder to reach the goal. To make learning easier, we modify the environment with a reduced grid size and random drunk actions that are biased away from the goal (since uniform random actions are less of a detriment on a smaller grid). See Appendix C for details about the experiments. Policy behaviours over the course of training are plotted in Figures 6 and 7 and the resulting behaviours of the different algorithms are summarized in Table 2. Q-Learning and Empirical Sarsa match the behaviours described by Leike et al.

Table 2: Learned policy behaviours on the Off-Switch and Small Whisky-Gold environments after $10^8$ training steps. This shows whether, at the first decision opportunity, the trained policy is most likely to move towards the off-switch or whisky, or away.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Disable Off Switch?</th>
<th>Drink Whisky?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q-Learning</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Virtual Sarsa</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Empirical Sarsa</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UH-CMA-ES</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Figure 6: The probability of drinking the whisky in the Small Whisky-Gold environment over the course of training. Displays the mean (line) and 95% confidence interval (shaded) across 10 independent training runs.
We have introduced the MAMDP model that describes the world and the resulting policies will be chosen based on these virtual policy value criteria ignore the presence of action via the policy structure or action. MDP algorithms may be variety of scenarios including constraints, hardware limitations, and thereby make them easier to control.

Finally, we verified the results experimentally.

6 Conclusion

We have introduced the MAMDP model that describes the case where actions as executed in the environment may differ from the actions specified by a policy. MAMDPs model a variety of scenarios including constraints, hardware limitations, supervised interventions, and self-damage. We showed that the determining factor for agent behaviour is the specific objective that an RL algorithm seeks to satisfy in the limit. Reward maximization leads to policies that account for action modification, while the Bellman optimality and Virtual policy value criterion only responds via controlling the state, not directly via the policy structure or action. MDP algorithms may be straightforwardly adapted to the MAMDP setting and tabular Q-learning and Sarsa converge whenever a solution exists. Finally, we verified the results experimentally.

It is sometimes assumed that reinforcement learning will always lead to reward maximising behaviour (Omohundro 2008). However, we show that many kinds of reinforcement learning algorithms can systematically deviate from reward maximization when the executed action may differ from the one specified by the policy. In general, efficient learning algorithms often make assumptions about the structure of the world and the resulting policies will be chosen based on these assumptions, not the true world dynamics. Agent designers can use these assumptions to intentionally blind agents from certain properties of their environment, and thereby make them easier to control.

Acknowledgements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number CGSD3–534795–2019].

Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéro de référence CGSD3–534795–2019].

References


