High Confidence Policy Improvement

Philip S. Thomas PTHOMAS@CS.UMASS.EDU
University of Massachusetts Amherst

Georgios Theocharous THEOCHAR@ADOBE.COM
Adobe Research

Mohammad Ghavamzadeh GHAVAMZA@ADOBE.COM
Adobe Research & INRIA

Abstract

We present a batch reinforcement learning (RL) algorithm that provides probabilistic guarantees about the quality of each policy that it proposes, and which has no hyper-parameters that require expert tuning. The user may select any performance lower-bound, ρ_-, and confidence level, δ, and our algorithm will ensure that the probability that it returns a policy with performance below ρ_- is at most δ. We then propose an incremental algorithm that executes our policy improvement algorithm repeatedly to generate multiple policy improvements. We show the viability of our approach with a simple gridworld and the standard mountain car problem, as well as with a digital marketing application that uses real world data.

1. Introduction

Most *reinforcement learing* (RL) algorithms do not provide any guarantees that they will work properly with high confidence. This plays a major role in precluding their use for applications in which execution of a bad policy could be costly (Jonker et al., 2004) or even dangerous (Moore et al., 2010). To overcome this limitation, rather than focusing on learning speed, sample complexity, or computational complexity, we focus on ensuring "safety". In this paper, we mean by *safety* that the probability that our algorithm returns a policy with performance below a baseline ρ_- is at most δ, where both ρ_- and δ are chosen by the user to specify how much risk is reasonable for the application at hand.

We present batch and incremental policy improvement algorithms that provide safety guarantees for every policy they propose. Our algorithms do not have any hyper-parameters that require expert tuning (there is one hyper-parameter that should be set based on runtime limitations). The primary drawbacks of our algorithms are their high computational complexity and that, in order to provide their guarantees, they tend to require more data to produce a policy improvement than ordinary RL algorithms.

To the best of our knowledge, *conservative policy iteration* (CPI) (Kakade & Langford, 2002) and its derivatives (Pirotta et al., 2013) are the only RL algorithms with monotonically improving behavior. However, CPI was not intended to be implemented as a safe algorithm—the analysis that shows it is safe was meant to provide insight into how it scales to larger problems, not a practical guarantee of safety. This is evidenced by its use of big-O notation when specifying how many trajectories to generate before updating the policy (Kakade, 2003).[1] We found that CPI requires an impractical number of trajectories to ensure safety—for the gridworld that we use it requires more trajectories than the span of our plots to make a single change to the policy.

The remainder of this paper is organized as follows. We review relevant material in Section 2 before formalizing the problem in Section 3. We present a batch version of our algorithm in Section 4 and an incremental version in Section 5. In Section 6 we provide case studies including a digital marketing study using data from a Fortune 20 company to show the viability of our approach before concluding.

2. Preliminaries

We assume that the environment can be modeled as a *Markov decision process* (MDP) or *partially observable Markov decision process* (POMDP) (Sutton & Barto,

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

[1] CPI requires $m + k$ trajectories for each change to the policy. Sham Kakade's thesis specified m using big-O notation in Lemma 7.3.4, and k can be derived from the appendix of the work of Kakade & Langford (2002).

1998). Although we adopt MDP notation hereafter, this work can be immediately translated to the POMDP setting with state-free policies (Kaelbling et al., 1996, Sections 7.1 and 7.2) by replacing states with observations. Let $\mathcal{S}$ and $\mathcal{A}$ denote the state and action sets, $r_t \in [r_{\min}, r_{\max}]$ be the bounded reward at time t, and $\gamma \in [0,1]$ be a discount factor. Let $\pi(a|s)$ be the probability (density) of taking action a in state s when using *policy* π. A *trajectory* of length T is an ordered set of states, actions, and rewards: $\tau = \{s_1^\tau, a_1^\tau, r_1^\tau, s_2^\tau, a_2^\tau, r_2^\tau, \ldots, s_T^\tau, a_T^\tau, r_T^\tau\}$. We define the (normalized and discounted) *return* of a trajectory to be $R(\tau) := \left(\left(\sum_{t=1}^T \gamma^{t-1} r_t^\tau\right) - R_-\right) / (R_+ - R_-) \in [0,1]$, where R_- and R_+ are upper and lower bounds on $\sum_{t=1}^T \gamma^{t-1} r_t^\tau$.[2] Let $\rho(\pi) := \mathbf{E}[R(\tau)|\pi]$ denote the performance of policy π, i.e., the expected normalized and discounted return when using policy π. We assume that all trajectories are of length at most T. We denote by $\mathcal{D}$ a set of n trajectories, $\{\tau_i\}_{i=1}^n$, each labeled by the policy that generated it, $\{\pi_i\}_{i=1}^n$, i.e., $\mathcal{D} := \{(\tau_i, \pi_i) : i \in \{1,\ldots,n\}, \tau_i \text{ generated using } \pi_i\}$. We call the policies, $\{\pi_i\}_{i=1}^n$, that generated the available data, $\mathcal{D}$, *behavior policies*. We write $|\mathcal{D}|$ to denote the number of trajectories in $\mathcal{D}$, $\pi_i^\mathcal{D}$ to denote the i^{th} behavior policy in $\mathcal{D}$, and $\tau_i^\mathcal{D}$ to denote the i^{th} trajectory in $\mathcal{D}$.

We define a *mixed policy*, $\mu_{\alpha,\pi_0,\pi}$, to be a mixture of π_0 and π with mixing parameter $\alpha \in [0,1]$. As α increases, the mixed policy becomes more like $\pi(a|s)$, and as α decreases it becomes more like $\pi_0(a|s)$. Formally, $\mu_{\alpha,\pi_0,\pi}(a|s) := \alpha\pi(a|s) + (1-\alpha)\pi_0(a|s)$.

2.1. Stationarity

Because we will use data from the past to make predictions about the future, we must assume that the future will somehow reflect the past. By assuming the problem can be exactly formulated as a POMDP, we have assumed that the initial state distribution and the mechanisms that produce the rewards, observations, and state transitions, are all stationary. Although such stationarity assumptions are commonplace in machine learning, it is important to note that, if an application is not exactly modeled as a POMDP, our probabilistic guarantees are only approximate. So, practitioners should strive to formulate the problem in a way that is as close to a POMDP as possible, and should be aware of the risks that stem from this assumption.[3]

[2] We normalize the returns because $\rho_-^{\text{CI}}(\mathbf{X}, \delta, m)$, which will be introduced later, requires $\mathbf{X}$ to be positive.

[3] For example, it is common in industry to retrain the policy for selecting user-specific advertisements every day, specifically to mitigate non-stationarity.

2.2. High-Confidence Off-Policy Evaluation (HCOPE)

High-confidence off-policy evaluation (HCOPE) involves taking a set of trajectories, $\mathcal{D}$, generated using some behavior policies, and using them to lower-bound the performance of another policy, π_e, called the *evaluation policy*. In this section we present three approaches to HCOPE that we use later. When discussing policy improvement later, it will be important that we can not just lower-bound the performance of a policy given some trajectories, but that we can predict from some trajectories what the lower bound would be if computed later with more trajectories. We therefore present each approach as an algorithm that predicts what the lower-bound would be if computed using a different number of samples.

All of the approaches are based on *importance sampling* (Precup et al., 2000), which can be used to produce an unbiased estimator of $\rho(\pi_e)$ from a trajectory τ generated by a behavior policy, π_b.[4] This estimator is called the *importance weighted return*, $\hat{\rho}(\pi_e|\tau,\pi_b)$, and is given by: $\hat{\rho}(\pi_e|\tau,\pi_b) := R(\tau)w(\tau,\pi_e,\pi_b)$, where $w(\tau,\pi_e,\pi_b)$ is the *importance weight*: $w(\tau,\pi_e,\pi_b) := \prod_{t=1}^T \frac{\pi_e(a_t^\tau|s_t^\tau)}{\pi_b(a_t^\tau|s_t^\tau)}$. The three approaches to bounding $\rho(\pi_e)$ are all based on first computing the importance weighted returns, which are then used to produce a high confidence lower-bound. For generality, we present each method in terms of a set of n random variables, $\mathbf{X} = \{X_1, \ldots, X_n\}$, which are independent and which all have the same expected value, $\bar{x} := \mathbf{E}[X_i]$. Later we will use the importance weighted returns, $\hat{\rho}(\pi_e|\tau_i,\pi_i)$, as the X_i, and so $\bar{x} = \rho(\pi_e)$.

2.3. A Concentration Inequality (CI) for HCOPE

Thomas et al. (2015) presented a concentration inequality that is particularly well-suited to using importance weighted returns to lower-bound the performance of the evaluation policy. Let $\hat{\rho}_-(\mathbf{X}, \delta, m, c)$ be a prediction, computed from $\mathbf{X}$, of their $1-\delta$ confidence lower-bound on $\bar{x}$, if the lower bound were to be computed using m random variables rather than n (m trajectories rather than n), and if the provided hyperparameter value c were used (Thomas et al., 2015, cf. Equation 7):

$$\hat{\rho}_-(\mathbf{X},\delta,m,c) := \frac{1}{n}\left(\sum_{i=1}^n \star\right) - \frac{7c\ln(2/\delta)}{3(m-1)} - \sqrt{\frac{2\ln(2/\delta)}{mn(n-1)}\left(n\left(\sum_{i=1}^n \star^2\right) - \left(\sum_{i=1}^n \star\right)^2\right)},$$

where $\star$ is shorthand for $\min\{X_i, c\}$.

Algorithm 1 shows how their concentration inequality-based approach can be used to compute an estimate, $\rho_-^{\text{CI}}(\mathbf{X}, \delta, m)$, of what the $1-\delta$ confidence lower bound on

[4] If the support of π_e and π_b differ, then importance sampling produces an *underestimate* of $\rho(\pi_e)$ (Thomas et al., 2015).

$\bar{x}$ would be if computed using m trajectories. Intuitively, this algorithm approximates the optimal value for the parameter c, and then computes the prediction with this value. If $n = m$, then $\rho_-^{\text{CI}}(\mathbf{X}, \delta, m)$ is a $1 - \delta$ confidence lower bound on $\bar{x}$. If $n \neq m$, then $\rho_-^{\text{CI}}(\mathbf{X}, \delta, m)$ is an estimate of what the lower bound would be if it were computed using m random variables rather than n.[5] Note that in this method it is assumed that each $X_i \geq 0$. This approach is similar to that of Bottou et al. (2013), but it does not discard data and it optimizes the hyperparameter, c, in a more principled manner.

Algorithm 1 $\rho_-^{\text{CI}}(\mathbf{X}, \delta, m)$: Predict what the $1 - \delta$ confidence lower bound on $\mathbf{E}[X_i]$ would be if $\mathbf{X}$ contained m random variables rather than n.

1: Place $1/20$ of $\mathbf{X}$ in $\mathbf{X}_{\text{pre}}$ and the remainder in $\mathbf{X}_{\text{post}}$
2: $c^\star \in \arg\max_{c \in [1,\infty)} \hat{\rho}_-(\mathbf{X}_{\text{pre}}, \delta, m, c)$
3: **return** $\hat{\rho}_-(\mathbf{X}_{\text{post}}, \delta, m, c^\star)$

The primary benefit of this approach, which we call the *concentration inequality* (CI) approach, is that it introduces no false assumptions. The drawback of the CI approach is that, because it makes no simplifying assumptions, it tends to be overly conservative. In the following two subsections we present approaches to HCOPE that introduce simplifying assumptions that are typically false, but which allow for tighter lower bounds.

2.4. Student's t-Test for HCOPE

Each of the X_i are unbiased estimators of $\bar{x}$. So, the sample mean of these estimators, $\hat{X} := \frac{1}{n}\sum_{i=1}^n X_i$, is also an unbiased estimator of $\bar{x}$. As $n \to \infty$, given mild assumptions, the *central limit theorem* (CLT) says that the distribution of $\hat{X}$ approximates a normal distribution, and so it is appropriate to use a one-sided Student's t-test to get a $1 - \delta$ confidence lower bound on $\bar{x}$. We present in Algorithm 2 an algorithm similar to Algorithm 1, but using a one-sided Student's t-test in place of the concentration inequality. Here $t_{1-\delta,\nu}$ denotes the $100(1-\delta)$ percentile of the Student's t distribution with ν degrees of freedom (i.e., $\text{tinv}(1-\delta, \nu)$ in MATLAB).

The benefit of using the t-test is that it usually produces tighter bounds than the CI approach because it introduces the additional assumption that $\hat{X}$ is normally distributed. When n is sufficiently large (using a sufficiently large number of trajectories), this assumption becomes quite reasonable. However, the importance weighted returns often come from distributions with heavy upper tails (Thomas et al., 2015), which can make the t-test overly conservative even for seemingly large data sets, as shown in Fig. 1.

[5] There are additional stipulations on how trajectories are partitioned into $\mathcal{D}_{\text{pre}}$ and $\mathcal{D}_{\text{post}}$ when there are multiple behavior policies (Thomas et al., 2015).

Notice, however, that this is not a real guarantee that the t-test will be overly conservative—in some cases it may still result in an error rate greater than δ. Still, if the domain is non-stationary, then the false assumption that $\hat{X}$ is normally distributed may be minor in comparison to the false assumption that the environment is a POMDP.

Algorithm 2 $\rho_-^{\text{TT}}(\mathbf{X}, \delta, m)$: Predict what the $1 - \delta$ confidence lower bound on $\mathbf{E}[X_i]$ would be if $\mathbf{X}$ contained m random variables rather than n.

1: $\hat{X} \leftarrow \frac{1}{n}\sum_{i=1}^n X_i$
2: $\sigma \leftarrow \sqrt{\frac{1}{n-1}\sum_{i=1}^n \left(X_i - \hat{X}\right)^2}$
3: **return** $\hat{X} - \frac{\sigma}{\sqrt{m}} t_{1-\delta,m-1}$

2.5. A Bootstrap Confidence Interval for HCOPE

Rather than assume that the sample mean, $\hat{X}$, is normally distributed, we can use bootstrapping to estimate the true distribution of $\hat{X}$, which can then be used to produce a lower-bound. A popular method for this sort of bootstrapping is *Bias Corrected and accelerated* (BCa) bootstrap (Efron, 1987). We present in Algorithm 3 an algorithm similar to Algorithm 1, but using BCa in place of the concentration inequality. We choose to use $B = 2000$ resamplings as suggested by practitioners (Efron & Tibshirani, 1993; Davison & Hinkley, 1997). In the pseudocode, Φ, is the cumulative distribution function of the normal distribution, $\mathbf{1}_A$ is one if A is true and 0 otherwise, and $\#\xi_i < \hat{X}$ denotes the number of ξ_i that are less than $\hat{X}$. Our pseudocode is derived from that of Carpenter & Bithell (2000), with notation changed as necessary to avoid conflicts with our other notation.

Using the bootstrap estimate of the distribution of $\hat{X}$, BCa can correct for the heavy tails in our data to produce lower-bounds that are not overly conservative, as shown in Figure 1. As with the t-test, for some distributions the lower-bounds produced by BCa may have error rates larger than δ. However, the bounds produced by BCa are reliable enough that their use is commonplace in high-risk applications such as analyzing the results of medical research (Champless et al., 2003; Folsom et al., 2003).

3. Problem Formulation

Given a (user specified) lower-bound, ρ_-, on the performance and a confidence level, δ, we call an RL algorithm *safe* if it ensures that the probability that a policy with performance less than ρ_- will be proposed is at most δ. The only assumption that a safe algorithm may make is that the underlying environment is a POMDP. Moreover, we require that the safety guarantee must hold regardless of how any hyperparameters are tuned.

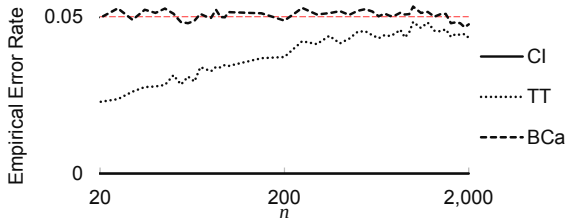


Figure 1. Empirical error rates when estimating a 95% confidence lower-bound on the mean of a gamma distribution (shape parameter $k = 2$ and scale parameter $\theta = 50$) using $\rho^{\dagger}_{-}$, where the legend specifies the value of †. This gamma distribution has a heavy upper-tail similar to that of importance weighted returns. The logarithmically scaled horizontal axis is the number of samples used to compute the lower bound (from 20 to 2000) and the vertical axis is the mean empirical error rate over 100,000 trials. Note that CI is overly conservative, with zero error in all the trials (it is on the x-axis). The t-test is initially conservative, but approaches the allowed 5% error rate as the number of samples increases. BCa remains around the correct 5% error rate regardless of the number of samples.

We call an RL algorithm *semi-safe* if it would be safe, except that it makes a false but reasonable assumption. Semi-safe algorithms are of particular interest when the assumption that the environment is a POMDP is significantly stronger than the (other) false assumption made by the algorithm, e.g., that the sample mean of the importance weighted returns is normally distributed when using many trajectories.

We call a policy, π, (as opposed to an algorithm) safe if we can ensure that $\rho(\pi) \geq \rho_{-}$ with confidence $1-\delta$. Note that "a policy is safe" is a statement about our belief concerning that policy given the observed data, and not a statement about the policy itself.

If there are many policies that might be deemed safe, then the policy improvement mechanism should return the one that is expected to perform the best, i.e.,

$$\pi' \in \arg\max_{\text{safe } \pi} g(\pi|\mathcal{D}), \tag{1}$$

where $g(\pi|\mathcal{D}) \in \mathbb{R}$ is a prediction of $\rho(\pi)$ computed from $\mathcal{D}$. We use a lower-variance, but biased, alternative to ordinary importance sampling, called *weighted* importance sampling (Precup et al., 2000), for g, i.e., $g(\pi|\mathcal{D}) := \left(\sum_{i=1}^{|\mathcal{D}|} \hat{\rho}(\pi|\tau_i^{\mathcal{D}}, \pi_i^{\mathcal{D}})\right) / \left(\sum_{i=1}^{|\mathcal{D}|} \hat{w}(\tau_i^{\mathcal{D}}, \pi, \pi_i^{\mathcal{D}})\right)$. Note that even though Eq. 1 uses g, our safety guarantee is uncompromising—it uses the true (unknown and often unknowable) expected return, $\rho(\pi)$.

In this paper, we present batch and incremental policy improvement algorithms that are safe when they use the CI approach to HCOPE and semi-safe when they use the t-test or BCa approaches. Our algorithms have no hyperparameters that require expert tuning.

Algorithm 3 $\rho^{\text{BCa}}_{-}(\mathbf{X}, \delta, m)$: Predict what the $1-\delta$ confidence lower bound on $\mathbf{E}[X_i]$ would be if $\mathbf{X}$ contained m random variables rather than n.

$\hat{X} \leftarrow \frac{1}{n}\sum_{i=1}^{n} X_i$
$B \leftarrow 2000$;
for $i = 1$ **to** B **do**
  Randomly sample m elements of $x \in \mathbf{X}$, with replacement.
  Set ξ_i to be the mean of these m samples.
end for
Sort the vector $\xi = (\xi_1, \xi_2, \ldots, \xi_B)$ such that $\xi_i \leq \xi_j$ for $1 \leq i < j \leq B$.
$z_0 \leftarrow \Phi^{-1}\left(\frac{\{\#\xi_i < \hat{X}\}}{B}\right)$
for $i = 1$ **to** n **do**
  Set y_i to be the mean of $\mathbf{X}$ excluding the i^{th} element: $y_i \leftarrow \frac{1}{n-1}\sum_{j=1}^{n} \mathbf{1}_{(j\neq i)} X_j$
end for
$\bar{y} \leftarrow \frac{1}{n}\sum_{i=1}^{n} y_i$
$a \leftarrow \frac{\sum_{i=1}^{n}(\bar{y}-y_i)^3}{6\left[\sum_{i=1}^{n}(\bar{y}-y_i)^2\right]^{3/2}}$
$z_L \leftarrow z_0 - \frac{\Phi^{-1}(1-\delta)-z_0}{1+a(\Phi^{-1}(1-\delta)-z_0)}$
$Q \leftarrow (B+1)\Phi(z_L)$
$l \leftarrow \min\{\lfloor Q \rfloor, B-1\}$
return $\hat{X} + \frac{\Phi^{-1}(\frac{Q}{B+1})-\Phi^{-1}(\frac{l}{B+1})}{\Phi^{-1}(\frac{l+1}{B+1})-\Phi^{-1}(\frac{l}{B+1})}(\xi_Q, \xi_{Q+1})$

In the following, we use the † symbol as a placeholder for either CI, TT, or BCa. We also overload the symbol $\rho^{\dagger}_{-}$ so that it can take as input a policy, π, and a set of trajectories, $\mathcal{D}$, in place of $\mathbf{X}$, as follows:

$$\rho^{\dagger}_{-}(\pi, \mathcal{D}, \delta, m) := \rho^{\dagger}_{-}\Big(\underbrace{\bigcup_{i=1}^{|\mathcal{D}|} \left\{\hat{\rho}\left(\pi|\tau_i^{\mathcal{D}}, \pi_i^{\mathcal{D}}\right)\right\}}_{\mathbf{X}}, \delta, m\Big). \tag{2}$$

For example, $\rho^{\text{BCa}}_{-}(\pi, \mathcal{D}, \delta, m)$ is a prediction made using the data set $\mathcal{D}$ of what the $1-\delta$ confidence lower-bound on $\rho(\pi)$ would be, if computed from m trajectories by BCa.

4. Safe Policy Improvement

Our proposed batch (semi-)safe policy improvement algorithm, PolicyImprovement$^{\dagger}_{\ddagger}$, takes as input a set of trajectories labeled with the policies that generated them, $\mathcal{D}$, a performance lower bound, ρ_{-}, and a confidence level, δ, and outputs either a new policy or No Solution Found (NSF). The meaning of the ‡ subscript will be described later.

When we use $\mathcal{D}$ to both search the space of policies and perform safety tests, we must be careful to avoid the *multiple comparisons problem* (Benjamin & Hochberg, 1995). To make this important problem clear, consider what would happen if our search of policy space included only two poli-

cies, and used all of $\mathcal{D}$ to test both of them for safety. If at least one is deemed safe, then we return it. HCOPE methods can incorrectly label a policy as safe with probability at most δ. However, the system we have described will make an error whenever either policy is incorrectly labeled as safe, which means its error rate can be as large as 2δ. In practice the search of policy space should include many more than just two policies, which would further increase the error rate.

We avoid the multiple comparisons problem by setting aside data that is only used for a single safety test that determines whether or not a policy will be returned. Specifically, we first partition the data into a small training set, $\mathcal{D}_{\text{train}}$, and a larger test set, $\mathcal{D}_{\text{test}}$. The training set is used to search for which single policy, called the *candidate policy*, π_c, should be tested for safety using the test set. This policy improvement method, POLICYIMPROVEMENT$^{\dagger}_{\ddagger}$, is reported in Algorithm 4. To simplify later pseudocode, POLICYIMPROVEMENT$^{\dagger}_{\ddagger}$ assumes that the trajectories have already been partitioned into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$. In practice, we place $1/5$ of the trajectories in the training set and the remainder in the test set. Also, note that POLICYIMPROVEMENT$^{\dagger}_{\ddagger}$ can use the safe concentration inequality approach, $\dagger$ = CI, or the semi-safe t-test or BCa approaches, $\dagger \in \{$ TT, BCa$\}$.

POLICYIMPROVEMENT$^{\dagger}_{\ddagger}$ is presented in a top-down manner and makes use of the GETCANDIDATEPOLICY$^{\dagger}_{\ddagger}(\mathcal{D}, \delta, \rho_-, m)$ method, which searches for a candidate policy. The input m specifies the number of trajectories that will be used during the subsequent safety test. Although GETCANDIDATEPOLICY$^{\dagger}_{\ddagger}$ could be any batch RL algorithm, like LSPI or FQI (Lagoudakis & Parr, 2001; Ernst et al., 2005), we propose an approach that leverages our knowledge that the candidate policy must pass a safety test. We will present two versions of GETCANDIDATEPOLICY$^{\dagger}_{\ddagger}$, which we differentiate between using the subscript $\ddagger$, which may stand for None or k-fold.

Before presenting these two methods, we define an objective function $f^{\dagger}$ as:

$$f^{\dagger}(\pi, \mathcal{D}, \delta, \rho_-, m) := \begin{cases} g(\pi|\mathcal{D}) & \text{if } \rho^{\dagger}_{-}(\pi, \mathcal{D}, \delta, m) \geq \rho_-, \\ \rho^{\dagger}_{-}(\pi, \mathcal{D}, \delta, m) & \text{otherwise.} \end{cases}$$

Intuitively, $f^{\dagger}$ returns the predicted performance of π if the predicted lower-bound on $\rho(\pi)$ is at least ρ_-, and the predicted lower-bound on $\rho(\pi)$, otherwise.

Consider GETCANDIDATEPOLICY$^{\dagger}_{\text{None}}$, which is presented in Algorithm 5. This method uses all of the available training data to search for the policy that is predicted to perform the best, subject to it also being predicted to pass the safety test. That is, if no policy is found that is predicted to pass the safety test, it returns the policy, π, that it predicts will have the highest lower bound on $\rho(\pi)$. If policies are found that are predicted to pass the safety test, it returns one that is predicted to perform the best (according to g).

The benefits of this approach are its simplicity and that it works well when there is an abundance of data. However, when there are few trajectories in $\mathcal{D}$ (e.g., cold start), this approach has a tendency to overfit—it finds a policy that it predicts will perform exceptionally well and which will easily pass the safety test, but actually fails the subsequent safety test in POLICYIMPROVEMENT$^{\dagger}_{\text{None}}$. We call this method $\ddagger$ = None because it does not use any methods to avoid overfitting.

Algorithm 4 POLICYIMPROVEMENT$^{\dagger}_{\ddagger}(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}, \delta, \rho_-)$
Either returns NO SOLUTION FOUND (NSF) or a (semi-)safe policy. Here $\dagger$ can denote either CI, TT, or BCa.

1: $\pi_c \leftarrow$ GETCANDIDATEPOLICY$^{\dagger}_{\ddagger}(\mathcal{D}_{\text{train}}, \delta, \rho_-, |\mathcal{D}_{\text{test}}|)$
2: **if** $\rho^{\dagger}_{-}(\pi_c, \mathcal{D}_{\text{test}}, \delta, |\mathcal{D}_{\text{test}}|) \geq \rho_-$ **then return** π_c
3: **return** NSF

Algorithm 5 GETCANDIDATEPOLICY$^{\dagger}_{\text{None}}(\mathcal{D}, \delta, \rho_-, m)$
Searches for the candidate policy, but does nothing to mitigate overfitting.

1: **return** $\arg\max_{\pi} f^{\dagger}(\pi, \mathcal{D}, \delta, \rho_-, m)$

In machine learning, it is common to introduce a regularization term, $\alpha\|w\|$, into the objective function in order to prevent overfitting. Here w is the model's weight vector and $\|\cdot\|$ is some measure of the complexity of the model (often L_1 or squared L_2-norm), and α is a parameter that is tuned using a model selection method like cross-validation. This term penalizes solutions that are too complex, since they are likely to be overfitting the training data.

Here we can use the same intuition, where we control for the complexity of the solution policy using a regularization parameter, α, that is optimized using k-fold cross-validation. Just as the squared L_2-norm relates the complexity of a weight vector to its squared distance from the zero vector, we define the complexity of a policy to be some notion of its distance from the initial policy, π_0. In order to allow for an intuitive meaning of α, rather than adding a regularization term to our objective function, $f^{\dagger}(\cdot, \mathcal{D}_{\text{train}}, \delta, \rho_-, |\mathcal{D}_{\text{test}}|)$, we directly constrain the set of policies that we search over to have limited complexity.

We achieve this by only searching the space of mixed policies, $\mu_{\alpha,\pi_0,\pi}$, where α is the fixed regularization parameter, π_0 is the fixed initial policy, and where we search the space of all possible π. Consider, for example what happens to the probability of action a in state s when $\alpha = 0.5$. If $\pi_0(a|s) = 0.4$, then for any π, we have that $\mu_{\alpha,\pi_0,\pi}(a|s) \in [0.2, 0.7]$. That is, the mixed policy

can only move 50% of the way towards being deterministic (in either direction). In general, α denotes that the mixed policy can change the probability of an action no more than $100\alpha\%$ towards being deterministic. So, using mixed policies results in our searches of policy space being constrained to some *feasible set* centered around the initial policy, and where α scales the size of this feasible set.

While small values of α can effectively eliminate overfitting by precluding the mixed policy from moving far away from the initial policy, they also limit the quality of the best mixed policy in the feasible set. It is therefore important that α is chosen to balance the tradeoff between overfitting and limiting the quality of solutions that remain in the feasible set. Just as in machine learning, we use k-fold cross-validation to automatically select α.

This approach is provided in Algorithm 6, where $\text{CROSSVALIDATE}^\dagger(\alpha, \mathcal{D}, \delta, \rho_-, m)$ uses k-fold cross-validation to predict the value of $f^\dagger(\pi, \mathcal{D}_{\text{test}}, \delta, \rho_-, |\mathcal{D}_{\text{test}}|)$ if π were to be optimized using $\mathcal{D}_{\text{train}}$ and regularization parameter α. $\text{CROSSVALIDATE}^\dagger$ is reported in Algorithm 7. In our implementations we use $k = \min\{20, \frac{1}{2}|\mathcal{D}|\}$ folds.

Algorithm 6 $\text{GETCANDIDATEPOLICY}^\dagger_{k\text{-fold}}(\mathcal{D}, \delta, \rho_-, m)$
Searches for the candidate policy using k-fold cross-validation to avoid overfitting.

$\alpha^\star \leftarrow \arg\max_{\alpha\in[0,1]} \text{CROSSVALIDATE}^\dagger(\alpha, \mathcal{D}, \delta, \rho_-, m)$
$\pi^\star \leftarrow \arg\max_\pi f^\dagger(\mu_{\alpha^\star,\pi_0,\pi}, \mathcal{D}, \delta, \rho_-, m)$
return $\mu_{\alpha^\star,\pi_0,\pi^\star}$

Algorithm 7 $\text{CROSSVALIDATE}^\dagger(\alpha, \mathcal{D}, \delta, \rho_-, m)$

Partition $\mathcal{D}$ into k subsets, $\mathcal{D}_1, \ldots, \mathcal{D}_k$, of approximately the same size.
result $\leftarrow 0$
for $i = 1$ **to** k **do**
  $\widehat{\mathcal{D}} \leftarrow \bigcup_{j\neq i} \mathcal{D}_j$
  $\pi^\star \leftarrow \arg\max_\pi f^\dagger(\mu_{\alpha,\pi_0,\pi}, \widehat{\mathcal{D}}, \delta, \rho_-, m)$
  result $\leftarrow$ result $+ f^\dagger(\mu_{\alpha,\pi_0,\pi^\star}, \mathcal{D}_i, \delta, \rho_-, m)$
end for
return result$/k$

5. Daedalus

The $\text{POLICYIMPROVEMENT}^\dagger_\ddagger$ algorithm is a batch method that can be applied to an existing data set, $\mathcal{D}$. However, it can also be used in an incremental manner by executing new safe policies whenever they are found. The user might choose to change ρ_- at each iteration, e.g., to reflect an estimate of the performance of the best policy found so far or the most recently proposed policy. However, for simplicity in our pseudocode and experiments, we assume that the user fixes ρ_- as an estimate of the performance of the initial policy. This scheme for selecting ρ_- is appropriate when trying to convince a user to deploy an RL algorithm to tune a currently fixed initial policy, since it guarantees with high confidence that it will not decrease performance.

Our algorithm maintains a list, $\mathcal{C}$, of the policies that it has deemed safe. When generating new trajectories, it always uses the policy in $\mathcal{C}$ that is expected to perform best. $\mathcal{C}$ is initialized to include a single initial policy, π_0, which is the same as the baseline policy used by $\text{GETCANDIDATEPOLICY}^\dagger_{k\text{-fold}}$. This online safe learning algorithm is presented in Algorithm 8.[6] It takes as input an additional constant, β, which denotes the number of trajectories to be generated by each policy. If β is not already specified by the application, it should be selected to be as small as possible, while allowing $\text{DAEDALUS}^\dagger_\ddagger$ to execute within the available time. We name this algorithm $\text{DAEDALUS}^\dagger_\ddagger$ after the mythological character who promoted safety when he encouraged Icarus to use caution.

Algorithm 8 $\text{DAEDALUS}^\dagger_\ddagger(\pi_0, \delta, \rho_-, \beta)$
Incremental policy improvement algorithm.

$\mathcal{C} \leftarrow \{\pi_0\}$
$\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{test}} \leftarrow \emptyset$
while true do
  $\widehat{\mathcal{D}} \leftarrow \mathcal{D}_{\text{train}}$
  $\pi_\star \leftarrow \arg\max_{\pi\in\mathcal{C}} g(\pi|\widehat{\mathcal{D}})$
  Generate β trajectories using $\pi^\star$ and append $\lceil \beta/5 \rceil$ to $\mathcal{D}_{\text{train}}$ and the rest to $\mathcal{D}_{\text{test}}$
  $\pi_c \leftarrow \text{POLICYIMPROVEMENT}^\dagger_\ddagger(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}, \delta, \rho_-)$
  $\widehat{\mathcal{D}} \leftarrow \mathcal{D}_{\text{train}}$
  if $\pi_c \neq$ NSF **and** $g(\pi_c|\widehat{\mathcal{D}}) > \max_{\pi\in\mathcal{C}} g(\pi|\widehat{\mathcal{D}})$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \pi_c$
    $\mathcal{D}_{\text{test}} \leftarrow \emptyset$
  end if
end while

The benefits of $\ddagger = k$-fold are biggest when only a few trajectories are available, since then $\text{GETCANDIDATEPOLICY}^\dagger_{\text{None}}$ is prone to overfitting. When there is a lot of data, overfitting is not a big problem, and so the additional computational complexity of k-fold cross-validation is not justified. In our implementations of $\text{DAEDALUS}^\dagger_{k\text{-fold}}$, we therefore only use $\ddagger = k$-fold until the first policy is successfully added to $\mathcal{C}$, and $\ddagger$ = None thereafter. This provides the early benefits of k-fold cross-validation without incurring its full computational complexity.

The $\text{DAEDALUS}^\dagger_\ddagger$ algorithm ensures safety with each newly proposed policy. That is, during each iteration of the while-loop, the probability that a new policy, π, where $\rho(\pi) < \rho_-$, is added to $\mathcal{C}$ is at most δ. The multiple comparison problem is not relevant here because this guarantee is per-iteration. However, if we consider the safety guarantee over multiple iterations of the while-loop, it applies and means

[6] If trajectories are available *a priori*, then $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$, and $\mathcal{C}$ can be initialized accordingly.

that the probability that at least one policy, π, where $\rho(\pi) < \rho_-$, is added to $\mathcal{C}$ over k iterations is at most $\min\{1, 2\delta\}$.

We define DAEDALUS2$^\dagger_\ddagger$ to be DAEDALUS$^\dagger_\ddagger$ but with line 12 removed. The multiple hypothesis testing problem does *not* affect DAEDALUS2$^\dagger_\ddagger$ more than DAEDALUS$^\dagger_\ddagger$, since the safety guarantee is per-iteration. However, a more subtle problem is introduced: the importance weighted returns from the trajectories in the testing set, $\hat{\rho}(\pi_c|\tau_i^{\mathcal{D}_{\text{test}}}, \pi_i^{\mathcal{D}_{\text{test}}})$, are not necessarily unbiased estimates of $\rho(\pi_c)$. This happens because the policy, π_c, is computed in part from the trajectories in $\mathcal{D}_{\text{test}}$ that are used to test it for safety. This dependence is depicted in Figure 2. We also modify DAEDALUS2$^\dagger_\ddagger$ by changing lines 4 and 8 to $\widehat{\mathcal{D}} \leftarrow \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$, which introduces an additional minor dependence of π_c on the trajectories in $\mathcal{D}^j_{\text{test}}$.

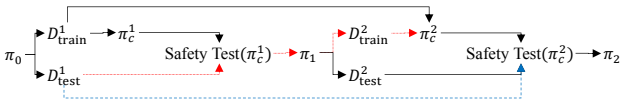


Figure 2. This diagram depicts influences as DAEDALUS2$^\dagger_\ddagger$ runs. First, π_0 is used to generate sets of trajectories, $\mathcal{D}^1_{\text{train}}$ and $\mathcal{D}^1_{\text{test}}$, where superscripts denote the iteration. Next $\mathcal{D}^1_{\text{train}}$ is used to select the candidate policy, π^1_c. Next, π^1_c is tested for safety using the trajectories in $\mathcal{D}^1_{\text{test}}$ (this safety test occurs on line 2 of POLICYIMPROVEMENT$^\dagger_\ddagger$). The result of the safety test influences which policy, π_1, will be executed next. These policies are then used to produce $\mathcal{D}^2_{\text{train}}$ and $\mathcal{D}^2_{\text{test}}$ as before. Next, both $\mathcal{D}^1_{\text{train}}$ and $\mathcal{D}^2_{\text{train}}$ are used to select the candidate policy, π^2_c. This policy is then tested for safety using the trajectories in $\mathcal{D}^1_{\text{test}}$ and $\mathcal{D}^2_{\text{test}}$. The result of this test influences which policy, π_2, will be executed next, and the process continues. Notice that $\mathcal{D}^1_{\text{test}}$ is used when testing π^2_c for safety (as indicated by the dashed blue line) even though it also influences π^2_c (as indicated by the dotted red path). This is akin to performing an experiment, using the collected data ($\mathcal{D}^1_{\text{test}}$) to select a hypothesis (π^2_c is safe), and then using that same data to test the hypothesis. DAEDALUS$^\dagger_\ddagger$ does not have this problem because the dashed blue line is not present.

Although our theoretical analysis applies to DAEDALUS$^\dagger_\ddagger$, we propose the use of DAEDALUS2$^\dagger_\ddagger$ because the ability of the trajectories, $\mathcal{D}^i_{\text{test}}$, to bias the choice of which policy to test for safety in the future (π^j_c, where $j > i$) towards a policy that $\mathcal{D}^i_{\text{test}}$ will deem safe, is small. However, the benefits of DAEDALUS2$^\dagger_\ddagger$ over DAEDALUS$^\dagger_\ddagger$ are significant—the set of trajectories used in the safety tests increases in size with each iteration, as opposed to always being of size β. So, in practice, we expect the over-conservativeness of ρ^{CI}_- to far outweigh the error introduced by DAEDALUS2$^\dagger_\ddagger$. Notice that DAEDALUS2$^{\text{CI}}_\ddagger$ is safe (not just semi-safe) if we consider its execution up until the first change of the policy, since then the trajectories are always generated by π_0, which is not influenced by any of the testing data.

6. Case Studies

We present three case studies: a discrete 4×4 gridworld, the canonical Mountain Car domain (Sutton & Barto, 1998), which has continuous state variables, and a digital marketing domain. The digital marketing domain involves optimizing a policy that targets advertisements towards each user that visits a webpage, and uses real data collected from a Fortune 20 company. The mountain car and digital marketing domains are almost identical to those used by Thomas et al. (2015). The only difference is that the digital marketing domains use data collected from different companies. Due to space restrictions, we refer the reader to Thomas et al. (2015) for additional details about these two domains.

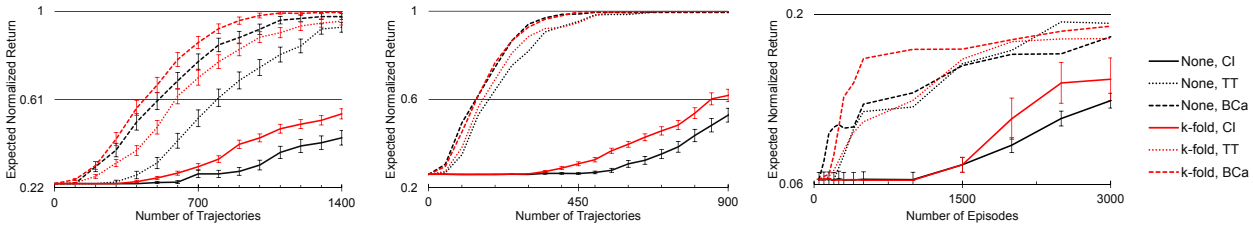
The gridworld domain uses four actions: up, down, left, and right, which deterministically move the agent in the specified direction. The top left and bottom right states are the initial and terminal states, respectively, and a rewards are always -1. We use $T = 10$ and a decent handcrafted initial policy that leaves room for improvement.

In all of the experiments that we present, we selected ρ_- to be an empirical estimate of the performance of the initial policy and $\delta = 0.05$. We used CMA-ES (Hansen, 2006) to solve all $\arg\max_\pi$, where π was parameterized by a vector of policy parameters using linear softmax action selection (Sutton & Barto, 1998) with the Fourier basis (Konidaris et al., 2011).

First we ran POLICYIMPROVEMENT$^\dagger_{\text{None}}$ on the Mountain Car domain for all three values of $\dagger$. The results are provided in Fig. 3. As expected, the CI approach requires more data to ensure safe policy improvement than the TT approach, and BCa requires the least data. All three methods jump to an optimal policy given 5,000 trajectories. Notice that the CI approach, which is safe (not just semi-safe) is able to produce safe policy improvement using 900 trajectories. In results that are not shown here, using POLICYIMPROVEMENT$^{\text{CI}}_{k\text{-fold}}$ reduces this to only 400 trajectories.

Next, for each domain, we executed DAEDALUS2$^\dagger_\ddagger$ with $\dagger \in \{\text{CI, TT, BCa}\}$ and $\ddagger \in \{\text{None}, k\text{-fold}\}$. Ideally, we would use $\beta = 1$ for all domains. However, as β decreases, the runtime increases. We selected $\beta = 100$ for the gridworld, $\beta = 50$ for Mountain Car, and $\beta \in [50, 100, 500]$ for the digital marketing domain. β increases with the number of trajectories in the digital marketing domain so that the plot can span the number of trajectories required by the CI approach without requiring too many calls to the computationally demanding POLICYIMPROVEMENT$^{\text{BCa}}_{k\text{-fold}}$ method. We did not tune β for these experiments—it was set solely to limit the runtime.

The performance of DAEDALUS2$^\dagger_\ddagger$ on these three domains



(a) 4×4 gridworld results. (b) Mountain Car results. (c) Digital marketing results.

Figure 4. Performance of DAEDALUS2$^{\dagger}_{\ddagger}$. The legend specifies ‡, †.

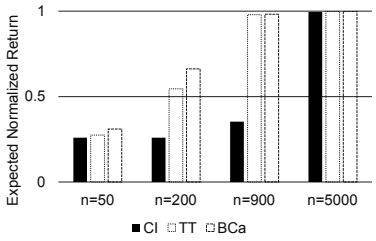


Figure 3. Mean performance (100 trials) of the policies produced by POLICYIMPROVEMENT$^{\dagger}_{\text{None}}$ for Mountain Car, where † is specified in the legend and the number of trajectories, $n = |\mathcal{D}|$, is specified on the horizontal axis. Using † = CI with $n = 50$ never resulted in a change to the policy, and so it specifies the performance of the initial policy.

is provided in Figure 4. The expected normalized returns in Figures 4(a), 4(b), and 4(c) are computed using 100,000, 10,000, and 20,000 Monte Carlo rollouts, respectively. The curves are also averaged over 100, 100, and 10 trials, respectively, with standard error bars provided when they do not cause too much clutter.

First, consider the different values for †. As expected, the CI approaches (solid curves) are the most conservative, and therefore require the most trajectories in order to guarantee improvement. The BCa approaches (dashed lines) perform the best, and are able to provide high-confidence guarantees of improvement with as few as 50 trajectories. The TT approach (dotted lines) perform in-between the CI and BCa approaches, as expected (since the t-test tends to produce overly conservative lower bounds for distributions with heavy upper tails).

Next, consider the different values of ‡. Using k-fold cross-validation provides an early boost in performance by limiting overfitting when there are few trajectories in the training set. Although the results are not shown, we experimented with using ‡ = k-fold for the entire runtime (rather than just until the first policy improvement), but found that while it did increase the runtime significantly, it did not produce much improvement.

Next, we computed the empirical error rates—the probability that a policy was incorrectly declared safe. In the gridworld and digital marketing domains, the empirical error rates were zero (no errors were made in any trials). In the Mountain Car domain the empirical error rates were below 0.6% for all † and ‡, which is far below the allowed 5% error rate. Also, although there are often concerns about the performance of off-policy evaluation methods as policies become deterministic, we observed no complications or instabilities as the policies produced by DAEDALUS2$^{\dagger}_{\ddagger}$ approached the deterministic optimal policies.

Lastly, notice that using n trajectories with POLICYIMPROVEMENT$^{\dagger}_{\text{None}}$ tends to result in a worse policy than using DAEDALUS2$^{\dagger}_{\text{None}}$ for n trajectories. For example, when $\dagger \in \{\text{TT}, \text{BCa}\}$ and $n = 200$, POLICYIMPROVEMENT$^{\dagger}_{\text{None}}$ achieves a mean normalized return of approximately 0.6 (see Fig. 3), while DAEDALUS2$^{\dagger}_{\text{None}}$ achieves a mean normalized return between 0.7 and 0.86 (see Fig. 4(b)). This is because DAEDALUS2$^{\dagger}_{\ddagger}$ collects trajectories from increasingly good regions of policy space, which allows it to produce tighter lower bounds on the performance of even better policies.

7. Conclusion

We have presented batch and incremental policy improvement algorithms that provide (exact and approximate) statistical guarantees about the performance of policies that they propose. These guarantees can be tuned by the user to account for the acceptable level of risk in each application. We showed on a real world digital marketing problem that our algorithms can use a realistic amount of data to provide guaranteed policy improvements with confidence 95%.

Acknowledgments

We would like to thank everyone who assisted with this work, particularly Bruno Scherrer, Alessandro Lazaric, Andrew Barto, Sridhar Mahadevan, Lucas Janson, and the reviewers.

References

Benjamin, Y. and Hochberg, Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society*, 57 (1):289–300, 1995.

Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D. X., Chickering, D. M., Portugaly, E., Ray, D., Simard, P., and Snelson, E. Counterfactual reasoning and learning systems: The example of computational advertising. *Journal of Machine Learning Research*, 14:3207–3260, 2013.

Carpenter, J. and Bithell, J. Bootstrap confidence intervals: when, which, what? a practical guide for medical statisticians. *Statistics in Medicine*, 19:1141–1164, 2000.

Champless, L. E., Folsom, A. R., Sharrett, A. R., Sorlie, P., Couper, D., Szklo, M., and Nieto, F. J. Coronary heard disease risk prediction in the atherosclerosis risk in communities (ARIC) study. *Journal of Clinical Epidemiology*, 56(9):880–890, 2003.

Davison, A. C. and Hinkley, D. V. *Bootstrap Methods and their Application*. Cambridge University Press, Cambridge, 1997.

Efron, B. Better bootstrap confidence intervals. *Journal of the American Statistical Association*, 82(397):171–185, 1987.

Efron, B. and Tibshirani, R. J. *An Introduction to the Bootstrap*. Chapman and Hall, London, 1993.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Folsom, A. R., Chambless, L. E., Duncan, B. B., Gilbert, A. C., and Pankow, J. S. Prediction of coronary heart disease in middle-aged adults with diabetes. *Diabetes Care*, 26(10):2777–2784, 2003.

Hansen, N. The CMA evolution strategy: a comparing review. In Lozano, J.A., Larranaga, P., Inza, I., and Bengoetxea, E. (eds.), *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pp. 75–102. Springer, 2006.

Jonker, J., Piersma, N., and den Poel, D. Van. Joint optimization of customer segmentation and marketing policy to maximize long-term profitability. *Expert Systems with Applications*, 27(2):159–168, 2004.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

Kakade, S. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 267–274, 2002.

Konidaris, G. D., Osentoski, S., and Thomas, P. S. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pp. 380–395, 2011.

Lagoudakis, M. and Parr, R. Model-free least-squares policy iteration. In *Neural Information Processing Systems: Natural and Synthetic*, pp. 1547–1554, 2001.

Moore, B., Panousis, P., Kulkarni, V., Pyeatt, L., and Doufas, A. Reinforcement learning for closed-loop propofol anesthesia: A human volunteer study. In *Innovative Applications of Artificial Intelligence*, pp. 1807–1813, 2010.

Pirotta, M., Restelli, M., Pecorino, A., and Calandriello, D. Safe policy iteration. In *Proceedings of the 30^{th} International Conference on Machine Learning*, 2013.

Precup, D., Sutton, R. S., and Singh, S. Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 759–766, 2000.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. High confidence off-policy evaluation. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence*, 2015.