Better Education Through Improved Reinforcement Learning

Travis Scott Mandel

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Zoran Popović, Chair

Emma Brunskill, Chair

Dan Weld

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Better Education Through Improved Reinforcement Learning

Travis Scott Mandel

Co-Chairs of the Supervisory Committee:
Professor Zoran Popović
University of Washington

Assistant Professor Emma Brunskill
Stanford University

When a new student comes to play an educational game, how can we determine what content to give them such that they learn as much as possible? When a frustrated customer calls in to a helpline, how can we determine what to say to best assist them? When an ill patient comes in to the clinic, how do we determine what tests to run and treatments to give to maximize their quality of life?

These problems, though diverse, are all a seemingly natural choice for reinforcement learning, where an AI agent learns from experience how to make a sequence of decisions to maximize some reward signal. However, unlike many recent successes of reinforcement learning, in these settings the agent gains experience solely by interacting with humans (e.g. game players or patients). As a result, although the potential to directly impact human lives is much greater, intervening to collect new data is often expensive and potentially risky.

Therefore, in this thesis I present several methods that allow us to evaluate candidate learning approaches offline using previously-collected data instead of actually deploying them. First, I present an unbiased evaluation methodology based on importance sampling that allows us to compare policies built on very different representations. I show how this approach enables us to improve student achievement by over 30% on a challenging and important

educational games problem with limited data but 4,500 features. Next, I examine the understudied problem of offline evaluation of algorithms that learn online. In the simplified case of bandits, I present a novel algorithm that is (often vastly) more efficient than the previously state-of-the-art approach. Next, for the first time I examine the more general reinforcement learning case, developing several new evaluation approaches, each with fairly strong theoretical guarantees. Using actual student data, we show that each method has different empirical tradeoffs and is useful in different settings.

Further, I present new learning algorithms which ensure that, when we do choose to deploy algorithms to humans, the data we gather is maximally useful. I first examine the important real-world problem of delayed feedback in the bandit case. I present an exploration algorithm which is theoretically on par with the state-of-the-art but much more attractive empirically, as evaluated on real-world educational games data. I show how one can incorporate arbitrary heuristics to further improve reward without harming theoretical guarantees. Next I present Thompson Clustering for Reinforcement Learning (TCRL), a Bayesian clustering algorithm which addresses the key twin problems of exploration and generalization in a computationally-efficient and data-efficient manner. TCRL has gained traction in industry, being used by an educational startup to serve literacy content to students.

Finally, I explore how reinforcement learning agents should best leverage human expertise to gradually extend the capabilities of the system, a topic which lies in the exciting area of Human-in-the-Loop AI. Specifically, I develop Expected Local Improvement (ELI), an intuitive algorithm which carefully directs human effort when creating new actions (e.g. new lines of dialogue). I show that this approach performs extremely well across a variety of simulated domains. I then conclude by launching a large-scale online reinforcement learning system, in which ELI is used to direct actual education experts to improve hint quality in an math word problems game. Our preliminary results, based on live student data, indicate that ELI shows good performance in this setting as well.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to sincerely thank my co-advisor Professor Zoran Popović, who, in addition to providing invaluable resources and guidance, provided me with a sense of a larger mission, without which this thesis would not have been possible. I am tremendously grateful to my co-advisor Professor Emma Patricia Brunskill, who has helped me grow immensely as a scholar through her constant support and advice, both technical and higher level. I am incredibly grateful to my mentor, project advisor, and good friend Professor Jens Mache, for always being available to help in any way I needed, whether it be encouragement, good humor, or advice. Without his intelligence, compassion, and support I would not be anywhere near where I am today. I am forever grateful to my friend Dr. Yun-En Liu, for his constant well-reasoned advice, and persistent playful optimism, even when I could not give much back in return. And to Professor Dan Sabey Weld, for his willingness to help me grow as an artificial intelligence scholar and educator, and his unwavering enthusiasm for my work.

I wish to warmly thank my fellow Center For Game Science students:

Dr. Dun-Yu Hsiao, for his humility, perseverance, and friendship,

Yvonne Chen, for giving me the honor of being able to collaborate with her,

Seong-Jae Lee, for his enthusiasm for machine learning,

Aaron Bauer, for pushing me towards excellence in teaching as well as research,

Rahul Banerjee, for his constant encouragement,

and Eric Butler, for being an awesome guy to work with for the past 10 years.

To Roy Szeto, who truly went above and beyond to help with my experiments, as well as past CGS developers Aaron Whiting and Tim Pavlik. Thank you for your invaluable help.

To the numerous other faculty who have helped me through my graduate studies: Pro-

fessors Mausam, Mari Ostendorf, Sergey Levine, Jeff Ely, Min Li, and others, thank you!

I wish to acknowledge the funding which made the work in this thesis possible, especially the NSF BIGDATA grant No. DGE-1546510, the Office of Naval Research grant N00014-12-C-0158, the Bill and Melinda Gates Foundation grant OPP1031488, the Hewlett Foundation grant 2012-8161, Adobe, Google, and Microsoft.

A hearty thank you Pigs Peace Sanctuary for all the support throughout my time in Seattle.

Thanks to my sister Wrye, for always cheering me on.

Lastly, I am thankful beyond words to my parents, Bob and Annette Mandel, who have been incredibly supportive throughout this long journey.

Chapter 1

INTRODUCTION

In recent years, artificial intelligence and machine learning techniques have advanced tremendously. Just a short time ago, having AI drive cars in the wild, beat the world champion at the game of Go, and learn from raw pixels how to expertly play Atari games all seemed very far off. And in addition to achieving these large-scale successes, there are countless smaller-scale machine learning success stories, as machine learning is being used to interpret images, text, and sound in numerous applications without the need for massive amounts of hand-labeling or difficult feature engineering. Robotics has also moved forward by leaps and bounds, with robots constantly becoming more and more autonomous, skilled, and dexterous.

The bulk of these success have been made possible by "big data": being able to play countless Go games, simulate robot control policies, and scrape petabytes of multi-modal content from the web. Combining these large datasets with state of the art work in deep learning has proven incredibly effective. From a naïve perspective, it seems that big data is king, and "small data" scenarios are a relic of the past.

However, some of the most interesting and potentially impactful applications of AI come from close interaction with humans. Consider a doctor agent treating patients, a teacher agent helping students learn a difficult concept, or an AI salesperson trying to sell a new product to potential customers. These application domains don't fit the typical mold of "big data", as we need to learn how to navigate a large space of potential interactions using only limited human interaction data, which is expensive to collect. Additionally these domains pose other challenges, as launching a new system to humans could be potentially risky, especially if these humans are vulnerable populations such as patients or children.

These problems,in which an agent needs to learning to near-optimally interact with humans, are all a seemingly natural choice for reinforcement learning, in which an AI agent learns from experience how to make a sequence of decisions to maximize some reward signal. Reinforcement learning has seen a recent surge in popularity due to successes in the domains of robotics and game-playing [107, 63, 130], "big data" settings where systems can gather large amounts of data through simulation. However, when RL systems need to use limited data to learn how to best interact with humans, a number of new challenges arise.

The central claim of this thesis is that in order for reinforcement learning to become a practical and effective approach for human-focused domains, we need algorithms that accurately evaluate reinforcement learning approaches offline, algorithms that efficiently explore the environment, and algorithms that effectively make use of human expertise. This thesis identifies and addresses important challenges in these three areas, and applies them to real-world problems in the domain of education.

Specifically, I investigate three major challenges:

Evaluation In real-world domains we need to be able to iterate on different reinforcement learning approaches, and it is vital to be able to determine if an approach performs well enough to merit deployment. Unfortunately, collecting new data for the purpose of evaluation is often expensive or risky. Therefore, in this thesis I investigate the understudied problem of how to use previously-collected data to evaluate reinforcement learning approaches offline.

In Chapter 3 I propose a methodology for using unbiased offline evaluation techniques to compare policies across very different policy representations. The key idea is to adapt importance sampling, a technique previously used in the off-policy learning setting, to the problem of evaluating learned policies in a way that is agnostic to representation and addresses nonstationarity in the environment. I was the first to show how offline evaluation can help one cope with a challenging high-dimensional real-world reinforcement learning problem where expert intuition is limited. Specifically, I used

Figure 1.1: Refraction.

a game called Refraction (Figure 1.1), played by over 500k students worldwide, to study the problem of how one should select fractions concepts in a personalized manner in order to increase student achievement. The proposed methodology allowed us to develop a novel hybrid neural-network POMDP approach which substantially outperformed other techniques for our problem. And, when this policy was deployed as part of a 2000-person experiment, it improved our measure of student achievement by over 30% compared to random and expert baselines.

Although powerful, one significant limitation of that work was that it was restricted to evaluating fixed policies, whereas in reality we wish to deploy algorithms that continually learn to improve their performance. Therefore, in Chapter 4 I study the more general (and difficult) problem of evaluating RL algorithms that learn and evolve over time. I first studied this problem in the simplified setting of bandits, where I propose an evaluation method which used a method based on storing samples from the reward distributions in queues to outperform (sometimes vastly) prior methods while retaining the same guarantees. In addition, I address this important problem for the first time in the much more complex setting of general reinforcement learning. I develop three novel evaluation approaches: one an extension of the queue-based idea to the RL setting, and two which involve a novel application of rejection sampling to recreate a history of interactions between the agent and the environment. Theoretically, I show that all three have guarantees that the samples they draw come from the true distribution given the history, and I develop a variant which has stronger guarantees of full unbiasedness (which had not been previously shown even in the bandit setting). Using a real-world dataset I collected which involved thousands of students playing an educational game, I illustrate that the three methods have a variety of interesting tradeoffs and each is useful under different conditions.

Efficient Exploration It is important that when we do launch a reinforcement learning algorithm on such a problem, we ensure that it explores efficiently so as to quickly perform well, but not at the expense of optimal long-term performance. Therefore, in Chapter 5 I study exploration in the face of delayed feedback, a problem that arises in many real-world

applications but is not accounted for in standard problem formulations. Prior methods for handling delay in multi-armed bandit problems were attractive theoretically but were highly inefficient empirically. I present a method which carefully modifies the distribution over actions at each step to gain the benefits of using stochastic bandit algorithms such as Thompson Sampling while being robust to delayed feedback. I then evaluate it offline on a dataset I collected from Treefrog Treasure (Figure 1.2), a numberline game which has been played by over a million students. The goal was to determine which numberline settings best improved student learning. The proposed approach had similar guarantees to prior work but substantially improved performance in simulation and on this real-world dataset. Additionally, I study how one can use general-purpose heuristics to modify the distribution to improve performance even further in the face of delay without worsening guarantees.

In real-world domains, it is too inefficient to learn the dynamics of each state independently; we must generalize across states in order to quickly learn to perform well. Yet the addition of automatic generalization substantially complicates the problem of exploration, as in order to explore effectively we need to consider how exploration will reduce not just the uncertainty over state parameters but also the uncertainty over state representation. Therefore, in Chapter 6 I develop a family of approaches, Thompson Clustering for Reinforcement Learning (TCRL), which are computationally efficient, require no parameter tuning, and (under mild conditions) converge to acting optimally. TCRL takes a Bayesian clustering approach to aggregate dynamics, but leverages structure in the state space to efficiently search through the space of clusterings, avoiding the computational expense of methods like Markov Chain Monte Carlo (MCMC). In a variety of domains taken from the literature, we showed that TCRL substantially outperforms state of the art methods which learn each state independently and time-limited MCMC methods. After developing the algorithm, I worked with an educational startup to integrate TCRL with their systems, in order to deliver better content in the domain of

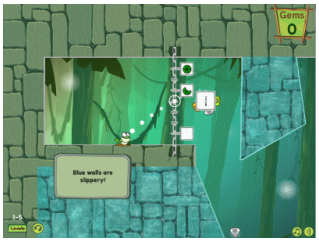


Figure 1.2: Treefrog Treasure.

literacy education. As a result of these efforts, TCRL was deployed to thousands of students.

Interactively Directing Human Effort to Create New Actions Finally, in settings where the agent interacts with humans, the action space (all possible levels in an educational game, or all possible treatments in a medical setting) is too vast to explore autonomously. Typically experts expend substantial effort to handcraft a useful action space. However, constructing the best action space for a reinforcement learning agent is a challenging task, and the human-designed action space is likely to be highly imperfect. Therefore, I propose asking experts to extend the capabilities of the system by adding actions (e.g. writing new line of dialogues or hints) to the system over time.

Given limited expert availability, one wishes to be certain that experts are focused on the most important situations. Therefore, in Chapter 7 I study automatically determining *where* (at what state) the human should create the next action. This Human-in-the-Loop framework is visualized in Figure 1.3. I develop a new algorithm, Expected Local Improvement (ELI), which intelligently selects states to query in order to make best use of human effort. ELI carefully identifies where the source of uncertainty comes from (low data in a state or insufficient actions in a state), and combines this with a Bayesian approach to select states in a way that does not require making restrictive assumptions on the way experts generate actions. I empirically evaluate ELI using simulated humans in in a variety of domains taken from the literature, some with over a million ground actions. I find that ELI greatly outperforms random selection and a number of other baselines, even when the simulated humans change over time or the simulated experts are very poor. Finally, I evaluate ELI with real humans on a challenging real-world problem: how to give personalized hints that allow students to reach mastery faster in Riddle Books, a mathematical word problems game which has been played

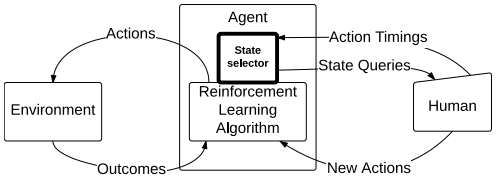


Figure 1.3: Proposed Human-in-the-Loop RL framework, in which a human provides new actions in response to state queries.

by thousands of students. The real-world experimental results are promising, and indicate that directing human effort using ELI may be helpful in navigating this challenging problem.

Although this thesis takes the first steps, there is much left to be done. I am excited to continue working towards the goal of creating reinforcement learning systems that learn how to tailor their behavior to meet the needs of each individual. This problem is extremely challenging, and so to solve it we must think beyond traditional RL algorithms, and instead develop new approaches that keep experienced humans in the loop. If we can achieve this, we unlock the potential to improve the lives of people around the world, particularly disadvantaged individuals for which personalization (in the form of private tutors or frequent medical check-ups) has typically been out of reach.

Chapter 2
BACKGROUND

Much of machine learning focuses on the problem of identifying certain statistical regularities in data that allow us to predict future data. The most popular areas of machine learning are *supervised learning*, which focuses on learning to predict future labels given labeled training data, and *unsupervised learning* which focuses on discovering structure (e.g. clusters) in unlabeled data. One typically uses these predictions or structure to drive some from of decision-making, whether it be displaying an alert to a user or moving a message from an inbox to spam. However, this traditional view of machine learning leaves out the important problem of automating this decision-making process; that is, using data to directly determine how to act rather than relying on manual decision-making or brittle rules and heuristics.

2.1 Multi-armed Bandits

2.1.1 Framework & Motivation

Multi-armed bandits [148, 86] are in some sense the simplest form of learning to make decisions. The name comes from casinos, where one has many different slot machine arms to pull, and wishes to pull arms in such a way as to maximize their payout. Formally, there are N arms, labeled $1, \ldots, N$. Associated with each arm i is an unknown probability distribution P_i with support in $[0, 1]$. At each timestep $1, \ldots, T$ the agent picks an arm i to pull and receives a scalar reward drawn i.i.d from P_i. The goal of the agent is to maximize the sum of rewards it receives. If the P_i were known the problem would be trivial: simply pick the distribution with highest mean and pull the associated arm forever. But, since the P_i are unknown, the agent must balance exploring to learn about the distributions, with exploiting

the arm(s) that yield high reward.

Despite its extreme simplicity, this problem has immense practical importance. This framework encompasses diverse problems, such as determining which treatment is most effective on a particular disease, what ad to show to maximize profits, or what teaching style to use to helps students learn most effectively. Although one could argue that bandits are too simple a model for many real-world problems, the basic formulation serves as a good starting point for developing extensions that allow us to better understand the exploration problem (some of which are considered in section 5). But despite its simplicity, even the basic problem is quite difficult to solve in a way that is attractive both practically and theoretically.

2.1.2 A/B testing (ϵ-first)

The simplest approach is to split the exploration and exploitation phases: for the first k steps, pick an arm uniformly at random to explore them, then exploit by selecting the best arm. This approach has widely been adopted in practice by internet companies seeking to optimize their content, likely due to two main reasons: the simplicity of the method, and the fact that using this method statistical comparisons between the N conditions are very clean.[1]. Additionally, there's only one parameter to choose, k. However, the best value of k is unclear, and more importantly it seems like if an arm is yielding very low reward we should be quick to stop pulling it in favor of high-reward arms. This approach, however, will continue pulling very bad arms during the exploration phase, potentially costing us a lot in terms of reward.

2.1.3 ϵ-greedy

Another popular approach, widely used in the robotics and reinforcement learning communities, gets away from separate exploration and exploitation time periods. Instead, this approach chooses an arm uniformly at random with probability ϵ, and with probability $1-\epsilon$

[1]For example, change over time is much less likely to confound results because each arm is equally likely to be pulled at each step.

greedily exploits the best arm so far.[2] Of course, one major challenge with this approach is determining how to set ϵ to properly balance exploration/exploitation for the problem at hand. Often "standard" values like $\epsilon = 0.1$ are used, which overexplore for some problems and underexplore for others. Alternatively, variants are used which decrease ϵ as time goes on, to correspond to the intuition that exploration should diminish as we gather more data. However, we run into a similar problem, where it is not clear how best to decrease ϵ for the problem at hand. More importantly, uniform exploration of all empirically suboptimal arms (without regard to how close to empirically optimal they are) seems wasteful.

2.1.4 Gittins Index

Gittins index is an Bayesian approach which is purportedly "optimal" for selecting arms [58]. However, the notion of optimality used is discounted reward, where the reward received on timestep t is multiplied by a factor of γ^t for some $\gamma \in (0, 1)$. However, discounting inherently causes the algorithm to prefer immediate rewards. Therefore, the algorithm can (and does) decide to stop pulling arms which have generated low rewards, because the unlikely (in a Bayesian sense) case that we might discover the arm might actually be good is outweighed by the fact that we could instead be getting high rewards from a known good arm [125]. This phenomenon is known as "incomplete learning", and seems intuitively undesirable that, even after an infinite amount of time, our algorithm could still have settled on a significantly suboptimal arm.

2.1.5 UCB

UCB (Upper Confidence Bound) [11] is a popular algorithm which is guaranteed to be asymptotically optimal. The basic idea derives from the general idea of "optimism under uncertainty": we place confidence intervals around the empirical mean of each arm, and pick the arm with the highest upper confidence interval. The confidence intervals themselves are

[2]Note that one important implementation detail concerns how arm values are initialized, and how ties are broken. These design decisions can greatly impact performance, especially early on.

derived from the Chernoff-Hoeffding bounds for bounded random variables, and have the from $\mu_i + \sqrt{\frac{\log t}{n_i}}$, where n_i is the number of times arm i has been pulled so far.One attractive feature about UCB is that it does not need to be given a budget of timesteps T a priori, and has nice guarantees that it learns at a near-optimal rate. However, to be practical in most situations, one needs to multiply the confidence intervals by some $\alpha < 1$ to boot exploitation. Tuning this parameter is very difficult and erodes theoretical guarantees.

2.1.6 Thompson Sampling

Thompson Sampling [148] is a state-of-the art Bayesian bandit algorithm. The basic idea is extremely simple: compute posterior distributions for each of the arm rewards based on past pulls, and to choose an arm sample from each distribution and take the maximum. Although proposed in 1933, this idea only recently gained traction, due to is combination of state-of-the-art theoretical guarantees [3] and excellent empirical performance without needing parameter tuning [29]. An additional attractive feature of Thompson Sampling is that it is a stochastic algorithm, and thus performs much better than the deterministic UCB algorithm in the face of delayed reward (as we explore further in Chapter 5).

2.2 Reinforcement Learning

Reinforcement learning is an extremely general form of machine learning. In fact, there are theoretical results [69] that state basically all artificial intelligence problems can be solved by a reinforcement learning agent. Reinforcement learning studies how to learn from experience how to make a sequence of decisions to maximize some reward function in an unknown and possibly uncertain environment. Despite its vast generality, reinforcement learning has historically been adopted primarily by the robotics community, where the need to perform a sequence of actions to achieve some goal is quite natural. However, in this thesis we are primarily interested in reinforcement learning as it applies to human-focused domains, in which the framework is the same but the real-world challenges encountered are somewhat different.

2.2.1 General formulation

The system consists of two components: an **agent**, and an unknown stochastic decision process, called an **environment**, which interact as follows. The agent takes an action from some (possibly infinite) space of action $\mathcal{A}$, the environment responds with an observation drawn from a (possibly high-dimensional, continuous) space of observations $\mathcal{O}$ and a scalar reward in a known range $[r_{min}, r_{max}]$. The agent's goal is to learn over time which actions to take to maximize the sum of rewards received.

Typically we assume set of available actions is small, discrete, and known. We investigate approaches that deal with situations where this is not the case in chapter 7.

One difficulty with this formulation is that it represents one long continuous interaction with goes on forever. But in many practical situations, experience comes in sequences which "reset". For instance, multiple users may arrive to use a system, and each one has a sequential interaction with the system before leaving. Therefore, we assume that experience comes in episodes, that is sequences in the form $o_0, a_0, r_0, o_1, a_1, r_1, \ldots, o_\ell, a_\ell, r_\ell$. For theoretical results in this thesis we typically assume that the distribution over episodes is stationary, more precisely that each "reset" consists of drawing a new (possibly high-dimensional, continuous, and partially hidden) initial observation i.i.d. from some unknown distribution, and then future observations and rewards are drawn iid given the history (possibly including high-dimensional, continuous, and partially hidden observations). This corresponds to assuming that each user or student is drawn iid from some distribution over users or students, instead of the population shifting over time.

In this work we deal primarily with finite-horizon tasks, where one knows a bound T such that $\ell \leq T$ for all episodes. This is a natural fit for many education domains, where there is a maximum number of problems or lessons a student might see. We make typically make the natural "episodic reset" assumption: that each episodes starts with some high-dimensional (possibly partially hidden) observation drawn i.i.d. from some distribution (e.g. each new player, with their personality and life experiences, being drawn i.i.d. from a fixed distribution

of players).

A **policy** is a function that maps from episode prefixes, called histories, to a distribution over actions. This gives us a complete sense of how to act in every situation. Note that there is a key difference between a policy, which is a fixed specification of how to act, a learning algorithm, which should refine the policy it uses based on the experience it gathers.

2.2.2 MDP with outcomes simplification

Although the previous structure is quite general, it is also extremely challenging, as it requires learning from large, high-dimensional histories with very little data. In certain cases we are willing to assume additional structure, in order to make the learning problem more tractable.

The most commonly used simplifying assumption in reinforcement learning is the Markov assumption: we assume we know a function that maps histories of actions, observations, and rewards to discrete states in a known state space $\mathcal{S}$, such that we can assume that the future depends only on the information contained in the state, and not the full sequential history of actions, observations, and rewards. For example, we just need to know the what the current board looks like in chess, not the full sequence of past piece moves.

Although we wish to learn how to make decisions under a Markov assumption, in this thesis we typically depart slightly from the standard Markov Decision Process (MDP) formulation by defining the dynamics in terms of relative outcomes [89, 8]. Relative outcomes are useful in situations where a small number of events can summarize the dynamics, although they can in principle encode any discrete MDP. Specifically, we assume a discrete state space $\mathcal{S}$ and an action set $\mathcal{A}$. In addition, one is given a set of relative outcomes $\mathcal{O}$ such that after taking an action $a \in \mathcal{A}$ from a state $s \in \mathcal{S}$ the agent observes an outcome $o \in \mathcal{O}$. The agent knows the reward function $R(s, a, o)$, which deterministically outputs a scalar value in $[r_{min}, r_{max}]$, and the transition function $T(s, a, o)$, which deterministically outputs a next state s'. However, it does not know the distribution over relative outcomes at each state. Some of the outcomes may be terminal, i.e. $T(s, a, o) = \emptyset$.

To better understand relative outcomes, consider as an example encoding a single-player

game of Yahtzee. In Yahtzee, the agent is given five dice, and their goal is to make those dice match certain predefined patterns. The game consists of 13 rounds, and in each round the agent gets three dice rolls. The agent starts by rolling all five dice, and then in each subsequent roll the agent picks zero or more dice up from the board and re-rolls them. At the end agent receives a score based on whether the dice match certain patterns (e.g. four-of-a-kind). Here, the relative outcome would be the results of the last dice rolled. So given the current state (timestep, score, and current dice values), the player picks the dice to re-roll, and receives the dice roll outcomes, which tells us how to (deterministically) update the state (timestep, score and dice values).

In the unknown-horizon MDP case, policies map states (instead of histories) to action distributions. However, since we know the horizon T, in order to be optimal we usually need to account for the timestep t, since as we get closer to the horizon we should start favoring short-term rewards more. Therefore, policies in this framework map state-timestep pairs to actions.

The notion of policy is important to distinguish between two key types of reinforcement learning. In **online** reinforcement learning, we wish to deploy a full-fledged learning algorithm in the actual environment, that learns to improve its behavior from its own experience. In **offline** reinforcement learning, we simply wish to learn a good policy from previously-collected data[3]. The majority of this thesis focuses on online RL, with the exception of Chapter 3.

2.2.3 Theoretical Guarantees

The most common form of guarantee we study on reinforcement learning algorithms are cumulative regret bounds. Assuming for simplicity an algorithm updates only after he cumulative regret[4] is defined as follows:

---

[3]Often known in the literature as off-policy learning

[4]A related notion, simple regret, focuses on the quality of the final policy produced by the agent.

$$R(M) = \sum_{m=1}^{M} V(\pi^*) - V(\pi_m)$$

where V, maps a policy to its expected episodic reward (value), π^* refers to the optimal policy, and π_m refers to the policy produced by the algorithm after m episodes. Note that in the special case of bandits, π^* refers to only pulling the optimal arm indefinitely.

We typically wish to upper bound regret. Sublinear regret (such as $\mathrm{O}(\sqrt{M})$) indicates the algorithm is learning to improve reward over time.[5]

For offline evaluation, we look for statistical unbiasedness as a key metric, and statistical consistency as a secondary objective. Statistical unbiasedness of an estimate $\hat{V}$ (using some dataset D) of some true value V states that $E_D[\hat{V}] = V$, that is, that the expectation of the estimate (over randomness of the data used as input) is equal to the true value. This property is very powerful and desirable, as it allows us to make strong claims about the accuracy (on average) of our estimates even given a small finite dataset.

Consistency states that, as $|D|$ approaches infinity, $\hat{V}$ will converge to V. This is a nice property to have as one would hope that if given unlimited data we could produce a highly accurate estimate, but it is less practically useful, especially in human-focused domains where datasets are rarely large.

2.2.4 Model-based Reinforcement Learning

Most of the reinforcement learning methods discussed in this thesis are model-based. Broadly defined, a model-based reinforcement learning method constructs a model of the dynamics of the environment at some point during its execution.

The typical view of a model-based reinforcement learning algorithm is that one first tries to construct a highly-accurate model of the environment (often an MDP, Partially Observable

[5]Alternative theoretical guarantees are possible, such as Knows What it Knows (KWIK) bounds, or (more popularly) Probably Approximately Correct (PAC)-style bounds or Bayesian optimality guarantees. Regret formulations do not require a prioi specification of δ and ϵ, as is common practice with PAC bounds, nor do they require setting a discount factor, as is common with Bayesian optimality guarantees.

Markov Decision Process (POMDP), or Predictive State Representation (PSR)), and then uses planning to find an optimal (or near-optimal) policy given that model. Although this approach is attractive (as it combines two independently well-studied components) it can perform very poorly as weaknesses and blindspots in the model are exacerbated during the planning phase [121, 72].

More state-of-the-art approaches (UCRL [10], PSRL [114]) seek to more holistically approach the problem to address this concern. Either the model is built in a modified way that is cognizant of the fact that the model will be used for planning, or the planning is modified since it knows the produced model is potentially inaccurate. By integrating these approaches, these methods are able to achieve near-optimal regret bounds, and (in the case of PSRL) impressive empirical results.

The benefits of model-based RL are numerous, as they naturally learn from data collected from other algorithms, they easily allow for principled efficient exploration, and they allow for naturally incorporating model knowledge into the RL algorithm to speed learning. However, model-based RL has several downsides, chief among them being that the algorithm may waste limited resources (hidden states, parameters, etc.) modeling aspects of the problem that are not relevant for decision-making, as we discuss further in Chapter 3.

2.2.5 Model-free Reinforcement Learning

Model-free methods avoid constructing an explicit model of the dynamics, with the goal of getting closer to modeling only what we actually care about (i.e. the optimal policy).

Value-based methods

There is a large literature on value-based methods, which seek to model the value function directly. The value function maps each state s to a value $v(s)$, which represents the expected future reward one would get if one started at s and acted optimally thereafter. Most work here focuses on, given a large state space, how one should best generalize values across states. Typically, states are assumed to consist of features, and function approximation

techniques (i.e. deep neural networks in DQN [107]) are used to boost efficiency. The benefit over model-based methods is that the function approximator is unlikely to expend resources modeling aspects of the problem that do not affect future reward. The major downside is that, with powerful nonlinear function approximators, these methods have little to no theoretical guarantees, and can require a lot of careful tuning to work well in practice. Additionally, one still runs the risk of modeling some aspect of the value function that is not relevant to the final policy.

Policy Search

Policy Search is gaining popularity [94, 41], and is one of the most direct reinforcement learning methods. One is given a class of policies and seeks to use previously-collected data to find the best ones. Often, the policy class is specified by a vector of continuous parameters, and the search process can be guided by the partial derivative of the objective function with respect to each parameter. This approach makes the challenge of accurately evaluating policies critical to building an effective algorithm, but accurately evaluating long-horizon policies in large spaces can be extremely challenging. Also, if the number of policy parameters becomes very large relative to the amount of data, these methods become infeasible. The main reason for the popularity of these approaches is their efficiency in "guided" settings, where one has an existing approach that that can direct data collection towards the most useful parts of the space, and constraints the policy updates to remain in that region [90].

2.3 Concluding notes

In this thesis we focus primarily on bandit algorithms and model-based reinforcement learning algorithms, although we do touch on policy search methods in chapter 3.

The focus on model-based reinforcement learning algorithms in this thesis is largely motivated by their ability to explore in a principled and highly efficient manner. The secondary reason is the ease of incorporating domain knowledge about the model, for example using the MDP with outcomes framework. These two aspects combine an speed learning considerably

compared to traditional model-free approaches.

In terms of specific algorithms, our study of bandit algorithms focuses mostly on extensions of UCB and Thompson Sampling, due to their attractive theoretical and empirical performance. In terms of model-based reinforcement learning, we mostly focus on PSRL as a baseline approach, due to its careful and elegant handling of the exploration-exploitation tradeoff.

Chapter 3

OFFLINE EVALUATION OF POLICIES ACROSS REPRESENTATIONS

This chapter is based on work the author published at AAMAS 2014.

3.1 Introduction: The Importance of Unbiased Evaluation

Consider the problem of creating an autonomous teacher agent whose goal is to sequence material such that students remain engaged while learning as much as possible. In order to perform well, the agent must observe the behavior of students in response to different orderings so as to decide on the best order. For example, when teaching fractions, introducing 7/9 before 1/3 may cause students to be frustrated, while reversing the order may cause more students to learn. Furthermore, the best strategy will likely not be a fixed sequence, but instead a policy for adapting the sequence to meet the needs of each student. This is a challenging reinforcement learning problem, since the agent must infer meaning from the complex and high-dimensional history of the student's past behavior. The problem is also partially observable, since we cannot access the true mental state of the student, only observed behaviors. And the population of students may change over time, for example due to a regional school holiday causing students in some part of the world to stop playing, meaning the agent must be careful not to overfit to one specific time period. These features make the problem challenging even if constructing a policy only over a short horizon. Similar challenges arise in other human-centric domains: for example, Shortreed [128] must solve a high-dimensional problem of determining which sequence of 2 treatment choices to give each schizophrenia patient. In such high stakes environments, trying several approaches online is undesirable. Instead we propose using existing data to learn and evaluate new policies for

future use.

When confronted with a new problem in these difficult domains, we must determine which of the many possible policy representations will perform best. Although the most general representations such as Predictive State Representations (PSRs) [98] may be capable of capturing the environment, learning all the parameters of a complex PSR or POMDP [133] model directly may not always be the best approach given limited data, e.g. due to overfitting. Ideally we could try many different representations, such as MDPs with different state spaces, POMDPs with different observation features, and a variety of direct policy representations, and pick the best one. This is similar to how researchers often try a variety of approaches (neural networks, SVMs, decision trees) when confronted with a new supervised learning problem. Ideally, we could identify the representation whose learned policy is likely to generalize the best in our domain by evaluating them offline using a single previously-collected dataset. This would allow us to experiment with new representations and policies, without the cost and risk of running them online.

Offline policy evaluation has been well-studied in domains where accurate simulators exist (such as a physics simulator in a robotic control task [110]). However, when an agent is designed for the purpose of human interaction, policy evaluation becomes challenging: experience is scarce and expensive, and the human mind is part of the environment and difficult to accurately simulate. Unless we have an extremely small number of candidate policies, we cannot run everything in the true environment. Instead, we need an evaluator which can use existing data collected from a non-deterministic policy to give us a reliable estimate of policy performance, regardless of which representation is used.

Previous work by Precup et al. [119] presented an unbiased estimator of policy reward based on importance sampling (IS). However, it has previously been used only for evaluating policies given a fixed representation, and Precup does not address the problem of evaluating policy generalization to new data, which is particularly critical in our setting where the population of students can change over time.

In this work, we present three main contributions. First, we introduce an unbiased

offline evaluation methodology. We combine Precup's importance sampling estimator with a modified cross-validation approach, and show how one can use this method to tackle a previously unaddressed, yet important problem: how to evaluate the generalization ability of different representations offline. Second, guided by our evaluation methodology, we introduce a novel feature compaction algorithm, which address the problem of high-dimensionality by combining the strengths of PCA and neural networks, and outperforms other techniques for our problem. Finally, we present a challenging real-world application, involving learning the concept selection policy that optimizes engagement in an educational game. We show that our offline evaluation methodology allows us to select a policy which improves our metric of concept completion by over 30% compared to random and expert baselines when deployed to students.

3.2 Background

We consider partially observable environments parameterized by a discrete action space $\mathcal{A}$, an action validity function V, an episode length T, and an observation space $\mathcal{O}$. We do not assume a state space is given. $\mathcal{O}$ can be multidimensional and real-valued. For each episode, the initial history h_1 consists of an initial scalar reward r_1 and an observation $o_1 \in \mathcal{O}$ provided by the environment. At each timestep $t \in \{1 \dots T\}$, we have a valid subset of actions specified by the validity function, $V_t = V(h_t) \subseteq \mathcal{A}$. The agent chooses an action $a_t \in V_t$. Then the agent receives a scalar reward r_{t+1} and an observation $o_{t+1} \in \mathcal{O}$ from the environment. The agent then appends the tuple (a_t, o_{t+1}, r_{t+1}) to the history h_t to get h_{t+1}. A stochastic policy π is defined as a probability distribution $\pi(a_i|h_t)$, denoting the probability of selecting action $a_i \in V(h_t)$ under policy π given a history h_t. Similarly, $\pi(a_1, \dots, a_t|h_t) \equiv \prod_{i=1}^{t} \pi(a_i|h_i)$.

Our objective is to find a policy π, which maximizes the reward function$\sum_{t=1}^{T} \gamma^{t-1} r_t$, where $\gamma \in [0, 1]$ is the discount rate. To find a good policy we must find a good representation, that is, a good way of specifying the (possibly non-deterministic) mapping from h_j to a_j for all possible histories h_j. A naïve representation would simply map each history to an action, but given the large (possibly infinite) number of possible histories h_j, a good representation

Figure 3.1: A level from our Refraction experiment. This level is from the Non-one Sources concept (see Section 3.8).

somehow shares data among episodes with different histories.

3.3 Problem Setup

Games have received significant interest from the education community in recent years due to their potential to motivate a wide variety of students and promote learning beyond the traditional classroom. Fractions is a particularly important topic since it is known to be a major obstacle to progress in mathematics [117]. We focus on creating an autonomous teacher agent that adaptively selects concepts (sets of 2-4 levels meant to teach a certain skill) in the educational fractions game Refraction, developed by the Center for Game Science (CGS). The goal of this puzzle game is to split lasers to direct the right amount of power into

each target spaceship (see Figure 3.1). The game involves both mathematical and spatial reasoning. Refraction has been played by over 500,000 players, both online and in classrooms. Despite this success, it has a high dropoff rate when played online, especially on child-centric sites. To reduce this attrition rate, we focus on improving the early part of the game where dropoff is most prevalent.

Choosing how to order concepts in the game to maximize engagement is difficult for game designers, even ignoring adaptivity, because the factors that cause a player to quit are complicated. For example, the mathematics may be in a reasonably engaging order, but the spatial difficulty of some of the levels may cause students to become frustrated and quit. In fact, we will see in sections 3.9 and 3.10 that the expert progression designed by game designers at CGS performs poorly, even compared to random sequencing, suggesting expert intuition may be limited.

Even though our problem is short horizon (there are 6 points where we can select the next concept), adaptively selecting concepts is still a challenging reinforcement learning problem for several reasons. First, we have a high dimensional set of 4,500 observation features, which capture each move the student has made while playing this concept and associated timing information. Some of these features are certainly irrelevant, but it is difficult to determine which a priori. Secondly, the problem is partially-observable, since we cannot observe the student's true mental state. The population of players that comes to the website to play Refraction is changing over time as well; for example on school holidays only a certain subset of students will be playing the game from home. Hence we must be careful not to overfit to one particular time period.

3.4 Approach Overview

Figure 3.2 gives an overview of our approach. Given data collected from an initial stochastic policy and an initial set of representations, we learn the representation parameters on the data, extract a policy from each representation, and evaluate the performance of each policy offline. If the estimated performance of the best policy is unsatisfactory, we create new

representations to add to the representation set. Otherwise, we take the strongest policy according to our evaluation methodology and run it online, closing the loop.

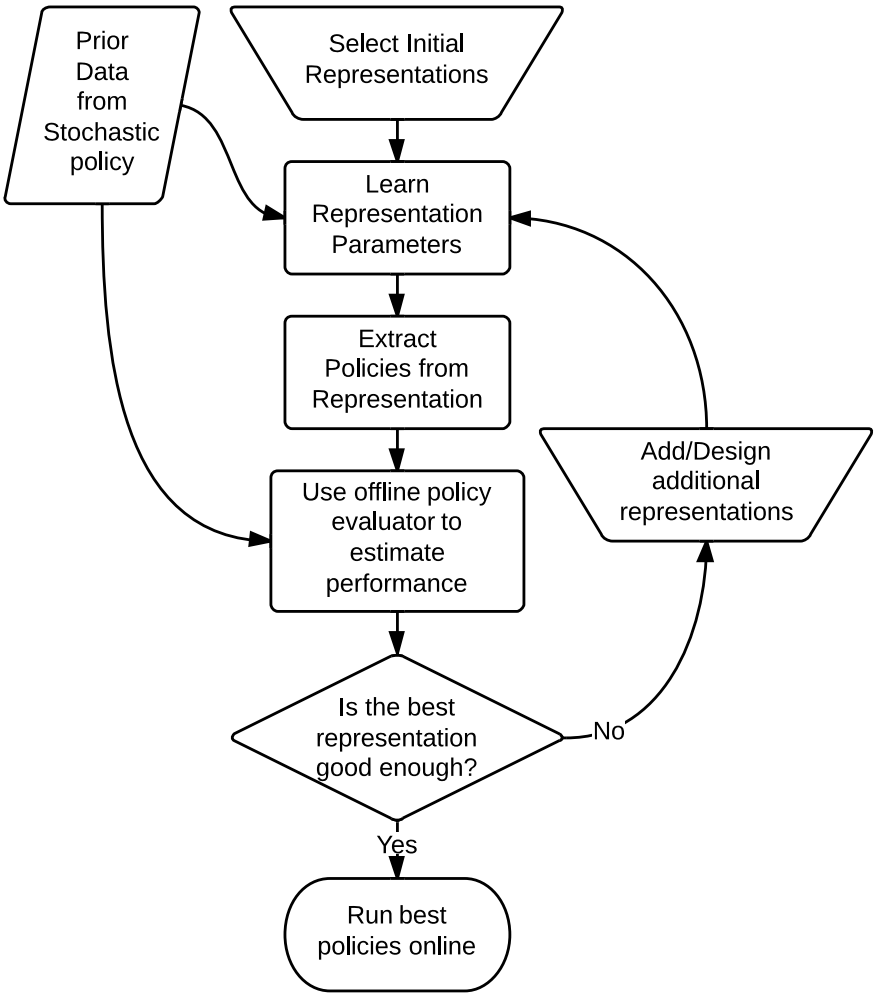


Figure 3.2: Approach overview.

3.5 Prior evaluation approaches

A key part of our approach is to calculate a good offline estimate of how well a policy learned on some representation performs.

Many approaches acquire policies by first learning the parameters of an underlying generative representation, such as an MDP or POMDP. In this case, instead of considering policy

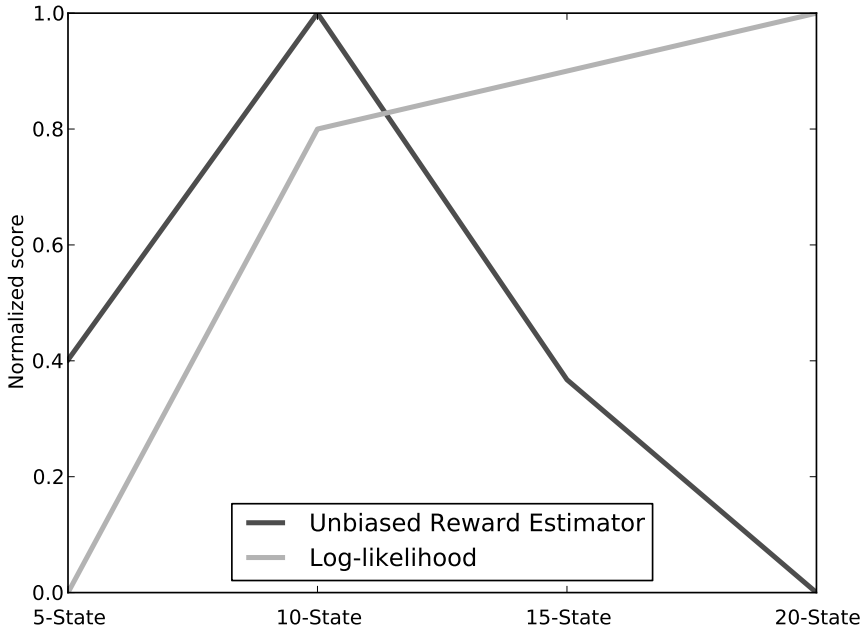


Figure 3.3: Unbiased reward estimator (Section 3.6) vs Log-Likelihood (Section 3.5). As we increase the number of hidden states in our learned PCA-NN-HMM model (see Section 3.9 for details) from 5 to 20, the cross-validated log-likelihood score increases even on held-out data, but the unbiased estimate of policy reward sharply peaks at 10 states. Because the score is cross-validated, we can see that the model is not overfitting. Instead we see that the 15 and 20 state models better capture the dynamics of the environment, but result in weaker policies. This indicates that selecting models based on likelihood alone can lead to poor policies.

performance directly, one instead optimizes model parameters using (log-)likelihood of the dataset under the model [35]. However, using likelihood to evaluate reinforcement learning representations can often be misleading, even if we compute the likelihood of held-out data to evaluate model generalization. The reason is that likelihood considers how well the system models every observation, but there may be observation features that are not important to model in order to build a good policy, but have a strong impact on the likelihood score. Even if the features are all relevant to the task, there may be aspects of continuous features that are not necessary to model. For example, a tea-making robot given temperature as a scalar feature might achieve high likelihood by modeling the exact temperature of a kettle of water at low temperatures, when all that matters is modeling whether it is above 100°C. For a concrete example of log-likelihood leading us astray in reality, see Figure 3.3.

It is not straightforward to modify log-likelihood to evaluate how well a model captures aspects of the system relevant to reward. Just restricting log-likelihood to compute $\log(p(r_1, \ldots, r_T | a_1, \ldots, a_T, \theta))$ is a poor evaluator since it ignores observations completely, and therefore cannot assess how well a model can interpret observations to better predict future rewards. Even computing $\log(p(r_1, \ldots, r_T | o_1, \ldots, o_T, a_1, \ldots, a_T, \theta))$ is a poor choice, since r_t may be easily inferable from o_t. The central issue is that a good evaluator needs to evaluate how well a model captures relevant observations needed to construct a good policy, but log-likelihood is not robust to observations that contain irrelevant information.

Another method considered for comparing representations is ECR [143], however this method is not statistically consistent, which is a significant concern for high stakes domains. See appendix A for more discussion.

3.6 Importance Sampling Method

To evaluate policies offline, we use a method called importance sampling to estimate the expected policy reward. Intuitively, this method only uses samples where the entire action history has (by chance) matched the decisions that would have been made by the policy we wish to evaluate. Importance sampling (IS) is a general technique for estimating the

expected value of a function f under a distribution p when samples are drawn from a different distribution q. Importance sampling is an attractive choice because it gives an unbiased estimate of f for any distributions p and q. This means that the expected value of the estimate is equal to the true value, which allows us a degree of trust in the estimates even on a limited amount of data, although the variance may be high. Note that some have proposed *normalized* (weighted) importance sampling [118], which has lower variance but is biased; this bias makes it unattractive for evaluation in high-stakes domains. If we let π_p be the policy we wish to evaluate, let π_q be the policy under which the data was collected, and we have a dataset of m episodes, we use IS to estimate the reward of π_p as follows [118]:

$$\hat{\mathbb{E}}\left[\sum_{t=1}^{T}\gamma^{t-1}r_t|\pi_p\right]=\frac{1}{m}\sum_{i=1}^{m}\left[\frac{\pi_p(a_{1,i},\ldots,a_{T,i}|h_{T,i})}{\pi_q(a_{1,i},\ldots,a_{T,i}|h_{T,i})}\sum_{t=1}^{T}\gamma^{t-1}r_{t,i}\right], \tag{3.1}$$

where $\pi(a_{1,i},\ldots,a_{T,i}|h_{T,i})$ denotes the probability of taking actions $a_{1,i},\ldots,a_{T,i}$ given the history in episode i, as described in Section 3.2.

This estimator will have very high variance on a small amount of data, because only episodes that completely follow a policy from beginning to end are counted. The variance can be reduced by observing that rewards only depend on previous actions [119]:

$$\hat{\mathbb{E}}\left[\sum_{t=1}^{T}\gamma^{t-1}r_t|\pi_p\right]=\sum_{t=1}^{T}\frac{1}{m}\sum_{i=1}^{m}\frac{\pi_p(a_{1,i},\ldots,a_{t,i}|h_{t-1,i})}{\pi_q(a_{1,i},\ldots,a_{t,i}|h_{t-1,i})}\gamma^{t-1}r_{t,i} \tag{3.2}$$

Intuitively, this estimator averages the per-timestep reward for the examples that match π_2 (weighted by probability), and then combines these per-timestep rewards using the discount factor. This method is still unbiased [119], but has lower variance than equation (3.1).

One potential downside of this approach is that if we have episodes that are relatively long, it is unlikely there will be many samples that exactly match a given deterministic policy at later timesteps. This will yield a high variance estimate. However, in our domain (and others, e.g. [128]) the episodes are short, so we can effectively use Equation (3.2) to evaluate policies.

Real-world reinforcement learning problems often involve severe data sparsity and sometimes nonstationarity. Because of this, evaluating policies by running this estimator on the training data will cause us to choose policies that overfit, that is, appear to perform strongly on the training data but perform poorly in practice. We must therefore determine how well the policies generalize, which is especially important when data is limited. Following supervised learning, we could divide the data randomly into validation and training datasets, learn the policies on one portion of data and evaluate them on another fold using IS. However, this does not address nonstationary environments, which may fluctuate or change over time. We want to determine whether we have found a policy that is likely to generalize well across time. To this end, we propose a "temporal" cross-validation approach, where the dataset is divided into N folds, each of which spans a contiguous span of time (e.g. a series of days). To score well using this approach, a representation must generate policies that generalize to held-out time periods, and improving on just one time period (such as a special school holiday) will not score well. The performance of the algorithm across the 5 temporal folds also gives us a sense of the variance across time. However, the value of the initial random policy may change over time as well, and thus it may be difficult to tell how much better than random a policy is by comparing the overall variance of the random policy to the overall variance of the candidate policy. For example, a policy could always perform slightly better than random, but not appear significantly different from random if the variance across time is high. We therefore subtract the value of the random policy from the score, in order to focus on how these policies do relative to the random baseline.

3.7 Candidate Representations

Our approach (Figure 3.2) takes as input a variety of possible representations suitable for high-dimensional, partially observable problems, and learns policies on these representations. Here we describe a variety of representations we evaluated, including novel representations developed after observing how previous attempts performed. The set of methods considered include feature compression methods, POMDP models, and linear policies. Although they

are inspired by our high dimensional, partially observable task, they are potentially applicable to other domains.

3.7.1 Feature Compression

Training models with thousands of potentially irrelevant observation features can be difficult, due to computational challenges, data sparsity, and overfitting. Some models may work well on the full feature space, but others (such as generative models, which model every observation feature) are not feasible on the full feature space. One approach to reduce the space from a large number of largely irrelevant observation features into a small set of relevant ones is automatic feature compression. Below, we describe two established methods for automatic feature compression, discuss their limitations, and describe a novel method that combines their strengths to address their limitations.

Principal Component Analysis (PCA) is a widely-used technique for taking a high-dimensional feature space and projecting it to a lower dimensionality. This is very useful for our purposes as the produced features are often quite rich, and are guaranteed to be orthogonal. However, doing PCA on a large feature space with a large number of features that are potentially irrelevant can lead to the relevant information being lost after the compression.

Neural networks (NN) are well-suited to compressing a large feature space to a smaller number of hidden units during a supervised prediction task, which in our case is predicting future reward from the current observation. Unlike PCA, neural networks can capture a nonlinear relationship between original and compressed features. However, neural networks often produce features which are highly correlated, which is not ideal if we want to find the most compressed representation possible.

We introduce a new feature compression method that seeks to build on the relative strengths of NN and PCA (non-linear function approximation targeting a particular prediction task, and orthogonal features, respectively). The input features are fed into a neural network with one hidden layer of 100 units. The neural network is trained with Rprop to best predict future reward (in our case, time steps until the player quits). At this point it

has found 100 non-orthogonal features that predict reward. Next, for each training example we take the output of each hidden unit and multiply it by the weight between it and the output unit:

$$\phi_i^{NN}(o) = h_i(o) * w_i$$

This causes hidden units which feature more prominently in the value of the output to also be better preserved in the compaction. Then we feed those 100 compressed features into PCA to compress further into a small number of rich, orthogonal features. We refer to this method as PCA-NN.

3.7.2 POMDP/IOHMM

We now consider how to use a small number of compacted or hand-selected features as the observations of a partially observable Markov decision process. A POMDP [133] is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \Omega, R)$, where $\mathcal{S}$ is the finite set of hidden states, $\mathcal{A}$ is the action space, $\mathcal{O}$ is a space of observations, $\mathcal{T}$ is a set of transition probabilities conditioned on each state and action, Ω is a set of observation probabilities conditioned on each state (and potentially action), and R is a set of reward probabilities conditioned on each state and action.

We can treat the POMDP as an Input Output Hidden Markov Model (IOHMM) [18], where the inputs are actions, and learn the model parameters using the Baum-Welch (EM) algorithm similar to [35]. Because EM is prone to local optima, we do 10 random restarts for each model and take the one with the highest log-likelihood.

For our application, the outputs are set to be a vector of continuous observations , each of which is modeled by a single independent Gaussian. We simplify the POMDP so that observations at each state are not conditioned on action, since we want each state in the POMDP to represent the same behavior independent of time step, although these assumptions would be easy to remove. Episodes in our domain can terminate early, so we define a separate terminal state, and binary quit observation. We initialize the observation distribution so that the terminal state is the only state that can output the $terminal = true$

observation. Furthermore, the terminal state always transitions to itself.

Many widely-used offline POMDP planning techniques such as SARSOP [84] are not applicable due to the fact that our observations are continuous. As such, we choose the QMDP [97] approximate planning algorithm for its simplicity and speed. To help prevent overfitting, we learn the policies with $\gamma = 0.5$ (determined empirically) even though the true reward signal is undiscounted. Discount factors lower than 1.0 can help the POMDP model generalize if it is not accurate enough to issue reliable long-term predictions due to inaccurate transitions caused by data sparsity. Finally, we restrict the planning algorithm to obey the domain constraints, namely that concepts cannot be repeated and certain concepts must come before others.

3.7.3 Linear Direct Policy Search

Since we perform policy evaluation using importance sampling, a straightforward policy training procedure is to directly maximize the importance sampled estimate of the expected reward with respect to the policy parameters. To this end, we constructed a linear policy that maps a history of observations $o_1, \ldots, o_t$ to action probabilities. The outputs are converted to action probabilities by exponentiating and normalizing by a sum over all legal actions:

$$\pi(a_i|o_1, \ldots, o_t) = \frac{\exp(-\theta_i^{\mathrm{T}} \psi_t)}{\sum_{a_j \in V(s)} \exp(-\theta_j^{\mathrm{T}} \psi_t)} \tag{3.3}$$

where θ_i is the policy parameter for a_i and ψ_t is the input. This input can correspond to directly to the current observations o_t, to a set of learned features ϕ_t, or to entire histories of observations or features over all preceding time steps. When using whole histories, the input vector is padded with zeros to ensure an equal length at each time step. The weights θ_i are learned by inserting Equation 3.3 into Equation 3.2, differentiating, and optimizing with LBFGS.

3.7.4 Extracting Static policies

One interesting question in human-centric domains is how much improvement (if any) can be achieved by adapting actions based on observations. In the context of educational games, this means adapting to meet the needs of each students instead of choosing the best static policy, that is, overall best sequence of concepts. In general, *static policies* are defined as a class of policies which depend only on the action history: $\pi_s(a|h_t) = \pi_s(a|a_1, \ldots, a_{t-1})$. If we have learned an adaptive policy with strong performance, we would like to know if it simply found a static policy that generalized well, or if the adaptive choices make a large difference. To answer this question, we have to extract the "most likely" static policy from the adaptive one. Our approach first simulates running the policy over players in the training data, picking actions until the action history no longer agrees with the policy's choices. From this information we can calculate the overall likelihood of choosing each action at each timestep, and then for each timestep add to the static policy the most likely action that obeys the constraints.

3.8 Experiment Details

Random	**Expert**	**Static** Only depends on action history		**(IO)HMM-n** POMDP policies with n states (sec 3.7.2)				**Direct** Direct linear policies (sec 3.7.3)			
Initial Random Policy	Expert Designed Policy	**γ = n**	**Extracted**	**Expert**	**PCA**	**PCA-NN** (feature compression sec 3.7.1)		**Full** (full feature history)		**Last** (last timestep features)	
		Exhaustive search on IS with discount factor n	Extracted (sec 3.7.4) from PCA-NN-HMM-10	Uses two hand-chosen features	Two PCA features (sec 3.7.1)	**2F** 2 PCA-NN features	**3F** 3 PCA-NN features	**All** All features	**PCA-NN** 2 PCA-NN features	**All** All features	**PCA-NN** 2 PCA-NN features

Table 3.1: Methods in Figure 3.4. The feature counts in this table refer to the feature count per timestep.

Refraction begins with two required levels constituting the introduction, which familiarize players with the basic gameplay. Next, there is a set of seven concepts we teach students in the early part of Refraction, corresponding to seven sets of 2-4 levels each. In standard

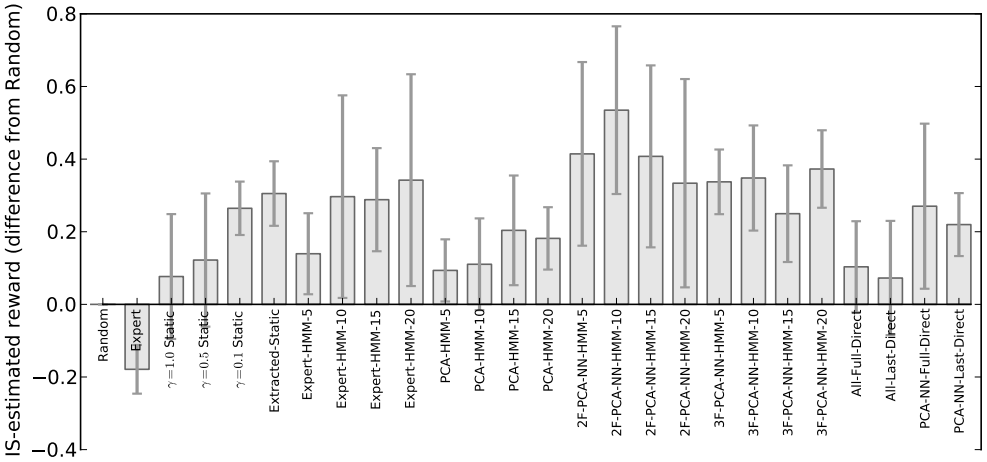


Figure 3.4: Importance sampling results. The error bars are 95% confidence intervals computed with 5-fold temporal cross validation using normal assumptions.(Specifically, the Excel CONFIDENCE function was used.)

("expert") order, they are: Spatial Reasoning(1), Half Splitting(2), Fourths(3), Third Splitting(4), Ninths(5), Sixths(6), and Non-one Sources(7). This expert ordering was designed based on the CGS game designers' intuition about what concept sequence would engage players, and was playtested significantly.

In our experiment there are only 6 time steps, so the agent is allowed to choose one concept to leave out. After all 6 have been given out there is a test level incorporating some of the more difficult concepts. Also we add a small set of hard constraints to which actions are valid at any given time, one of which is that no action may be repeated. The constraints prevent an advanced level involving a game mechanic from coming before the tutorial on that mechanic. The addition of these constraints leaves us with 560 possible static policies, and an infinite set of adaptive policies.

The agent is guided by a reward signal, which is based on concepts completed. In a voluntary game setting where players can quit at any time, measuring exit performance (and therefore total learning) is impractical. However, learning and engagement are intertwined, as

it is not possible to achieve learning without first engaging students. Additionally, completion of each set of levels requires a degree of mastery of the corresponding concept, making completion a reasonable metric.

The reward signal gives a reward of 1.0 for concepts completed after the first, and 0.0 otherwise. This is because players often quit early on due to a host of factors outside our control, such as disliking the overall genre of the game, which can cause a lot of noise. Also, due to the fact that it is possible to eventually complete most levels by trial-and-error, players who complete a single concept and quit seem less likely to have learned something than players who complete two or more concepts. The reward for completing the final concept is given only if the end test is completed. These rewards are all undiscounted.

In order to learn and evaluate policies, we had access to raw logging data collected from an experiment on BrainPop.com, a popular educational website for grades K-12. The experiment was deployed with a randomized exploration policy, meaning actions were chosen uniformly at random from the valid options to ensure broad coverage of the space. Data was collected for about 6 weeks, during which time 11,300 people made it to the first action. We used this data to search for agent policies which could be run online.

We calculate 180 base features per level, and while the user normally plays 2-4 levels between timesteps, it is possible to replay any of the 25 levels at any time, resulting in 4,500 features split across levels. These 180 base features include simple features such as total number of moves and total time, and more complicated features calculated based on the player's search graph for a level. Specifically, a playthrough of one level of Refraction consists of a human player moving through the game state space. Depending on the feature, each board configuration can be a state in this space, or we can aggregate states, for example by considering all board configurations with the same number of pieces on the board to be identical. We can think of playthroughs as directed multigraphs through a given state space, where the vertices are game states and the edges are labeled with the time the player takes to move from one state to the next. For each possible state space and associated graph, we can calculate its properties to use as features, such as the total number of nodes, the

average vertex degree, the sum of all edge weights, and so on. These serve as our 180 base playthrough features, which are then expanded to 4,500 per timestep.

3.9 Offline Evaluation Results

In this section, we evaluate all the approaches described in Section 3.7 using the importance sampling evaluator mentioned in Section 3.6. The approaches are described in detail in Table 3.1. The Random and Expert policies serve as baselines. We learn static policies (which give the same concept sequence to everyone), by performing exhaustive search on IS with various discount factors. Even though the true reward is undiscounted, learning with small discount factors can prevent overfitting since we have much more data at the beginning of the game. Note that since we are searching over policies, a discount factor of 0 is not a reasonable option since it would only consider the initial reward. We also extract a static policy from the best adaptive policy (see Section 3.7.4). We learn IOHMM/POMDP models (Section 3.7.2) with various feature compaction methodologies (Section 3.7.1), and two features were selected by hand which were predictive of quitting in other tasks. Lastly, we tried our linear direct policy search methodology (Section 3.7.3) with a variety of feature representations, investigating feature compaction and whether to include the full history of observation features or just the features from the last timestep.

One conclusion apparent from Figure 3.4 is that the variance of most approaches across folds is fairly high. This is due to a combination of temporal changes in the population, the learning methods being prone to local optima (hence they find better solutions on some folds and worse on others), and the inherent high variance of importance sampling. Next, we observe that 2F-PCA-NN-HMM-10 appears to do by far the best, indicating that the PCA-NN feature compaction had a significant benefit over the expert-chosen features. The expert features do show an improvement over PCA compaction however, illustrating that the PCA features are not very relevant to the task at hand. On the other end of the spectrum, the hand-designed expert policy does poorly, performing worse than random, suggesting that despite the best efforts of the CGS game designers, it is a challenging problem to design

a good concept sequence by hand. Interestingly, the method that attempts to optimize importance sampling directly also fares poorly, although that may be partly due to the simple linear representation.

Among the static policies, extracting static policies from the best adaptive policy (2F-PCA-NN-HMM-10) generalizes better than training them to maximize IS. This is likely because the IOHMM is compacting action histories in a way that generalizes better than looking at the full history.

3.10 Real-World Results

Since our estimator is unbiased, we would expect that with high probability each policy's true value will fall within the error bars shown in Figure 3.4. Since evaluation is expensive in our domain, it is not possible to run all policies on the game. Indeed, this is the motivation for evaluating policies offline in the first place.

However, to empirically verify that our offline evaluation methodology allowed us to discover policies that actually perform well in the game, we evaluated our two baselines (Expert and Random), the adaptive policy which appeared to do best offline (2F-PCA-NN-HMM-10) and the the best-performing offline static policy (Static Extracted). Because we saw high variance during training, out of the 5 folds we picked the 2F-PCA-NN-HMM-10 policy that had the highest difference from Random on its test fold, which we will henceforth refer to as "Adaptive", and the static policy extracted from it. The policies were run simultaneously and the experiment was stopped when we had 500 players in each condition, for 2000 total players.

The results are presented in Figure 3.5. As expected, Adaptive has the highest average reward. To determine statistical significance, we use the non-parametric Wilcoxon rank-sums test due to the non-normality of our data. We find that Adaptive is the only condition which is significantly different than random (p=0.002, W=263682.5, Z=3.17951) and Adaptive is also significantly different from expert (p¡0.0001, W=267305.5, Z=4.04048). It performs on average 32% better than the Random and Expert baselines. This is an absolute difference

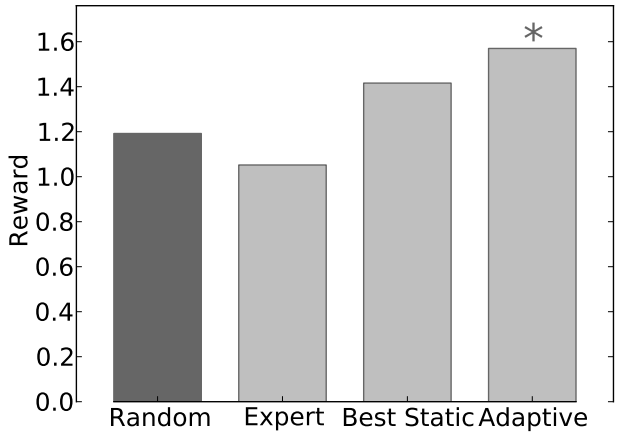


Figure 3.5: Experimental results on N=2000 players distributed equally among the four conditions. Adaptive received the highest average reward, and was the only policy that differed significantly from Random (p=0.002), as indicated by the asterisk.

of 0.38, which can be interpreted as approximately 1 level (recall that reward is measured in concepts). Expert had a slightly lower mean than Random (p=0.389, W=246684, Z=-0.862), and Best Static showed no significant improvement over Random (p=0.112, W=256878, Z= 1.58744).

We find that importance sampling gives us good estimates of actual performance, as all observed average rewards fall within their 95% confidence interval (Figure 3.4). However, this is unsurprising given that the estimator is unbiased.

The best static policy was the following: Half Splitting(2), Third Splitting(4), Spatial Reasoning(1), Non-one Sources(7), Ninths(5), Sixths(6). Compared to the expert policy in Section 3.8, it has delayed the introduction of spatial reasoning, and put the fractions which require single-splits (e.g. 1/2) before the double-splits (e.g. 1/9 = 1/3 * 1/3).

We observe that the adaptive policy chooses between two static policies roughly equally based on performance in the introduction. It seems (but is difficult to confirm precisely), that if the policy determines that the player does well on the introduction, they are given the Halves concept next, which involves mathematics, but if they do poorly they are given

the Spatial concept, which does not.

3.11 Related Work

There is a large body of work in learning policies offline, such as LSPI [85] or Fitted Q-Iteration [51]. When online evaluation of policies learned offline is expensive, successful real-world applications of reinforcement learning have generally relied on simulations or domain knowledge to guide the choice of representation. For example, several prominent RL applications, such as autonomous flight [110] or backgammon [142], relied on use of a simulator to determine the appropriate representation. In other cases, state-spaces were directly designed by experts [131]. An alternate approach is to use a complex set of input features, and use value function approximation over the feature set; however, this still requires choosing the right combination of input features and function approximator [70, 65]. In contrast, we seek to explore the space of representations without access to an accurate model of the environment, and without a priori knowledge about which representation best fits our problem.

Unbiased offline policy evaluation has been studied in the restricted domain of contextual bandits [92]. In reinforcement learning, importance sampling has been previously considered as an unbiased estimator of policy reward. Precup et al. [119] proposes importance sampling for learning the value function given an MDP state space, and Peshkin and Shelton [118] investigate using IS to guide a policy gradient algorithm. However, in contrast to our temporal cross-validation approach, neither addressed how to evaluate the generalization ability of policies to new data. Hachiya et al. [66] use a cross-validated importance sampling estimate of value function error (as estimated by the Bellman residual) to determine the best value of a single parameter of a fixed policy representation. Evaluating models based on value function error is a concern for two reasons, first, an improvement in modeling the value function does not necessarily improve policy performance, and second, this method cannot evaluate approaches which do not compute an explicit value function (such as policy gradient approaches). Our methodology does not have these restrictions, being able to directly

compare the expected reward of the policy, and making no assumptions about the presence of a value function. Lastly, all three of these methods focus on tuning a few parameters of a fixed representation rather than comparing estimated reward across representations.

There has been significant work in applying reinforcement learning to develop educational strategies in intelligent tutoring systems (ITSes), such as using MDPs for hint generation tutors [16] or POMDPs to select topics of focus [26]. Work by Chi et al. [33] also focuses on improving an ITS via reinforcement learning, and addresses a similar problem of deciding between potential state spaces offline. Their approach did achieve encouraging learning gains, but did not use a statistically consistent estimator (for the proof, see appendix A) for evaluating different representations, and only considered restricted MDP feature selection among a small set of features.

3.12 Future Work and Conclusion

We are the first to show how an existing importance sampling estimator can be used to compare the generalization of different representations, such as policy gradient approaches and POMDPs, even if the environment changes over time. We show that in a challenging educational games application, our methodology is a useful guide to allow us to choose the best representation for our domain. Without our importance sampling methodology, choosing between the many different representations, algorithms, and parameter choices would have been challenging. We would have been unlikely to design our novel PCA-NN feature compaction methodology, not knowing that it would outperform PCA in our domain. We show that our methodology allows us to select the best representations to run online, allowing us to see a 30% reward improvement in a domain where we cannot evaluate a large number of policies in the true environment. These results suggest that our evaluation methodology has the potential to improve performance in other challenging reinforcement learning tasks in which the choice of representation is unclear.

Since the variance of importance sampling estimates depends heavily on the horizon, it will take a very large amount of randomized data to evaluate long-horizon deterministic poli-

cies, and hence research is needed into better offline evaluation techniques for these problems. Another direction would be to explore how POMDPs can be trained to maximize importance sampling directly, and what effect that has on generalization. Yet another promising avenue for future work is exploring further methods of learning representations for high-dimensional problems.

Our evaluation methodology is general and can be applied to other discrete-action reinforcement learning problems where the choice of representation is unclear and an accurate simulator is not available. Similar situations arise in dialogue systems [143], healthcare [128], intelligent tutoring systems [33], and online advertisement delivery [15]. Our evaluation methodology could be applied to select appropriate representations for these domains, allowing us to better solve these challenging reinforcement learning problems.

3.13 Acknowledgements

This material is based on work supported by the National Science Foundation Graduate Research Fellowship grant No. DGE-0718124, as well as the Office of Naval Research grant N00014-12-C-0158, the Bill and Melinda Gates Foundation grant OPP1031488, the Hewlett Foundation grant 2012-8161, Adobe, Google, and Microsoft.

Chapter 4

OFFLINE EVALUATION OF ONLINE REINFORCEMENT LEARNING ALGORITHMS

This chapter is based on work the author published at AAAI 2015 and AAAI 2016.

4.1 Introduction: The Need for Algorithm Evaluation

In the previous chapter we discussed the offline policy evaluation problem, where one uses a precollected dataset to achieve high-quality estimates of the performance of a proposed policy. However, this prior work focuses only on evaluating a *fixed policy* learned from historical data. In many real-world problems, we would instead prefer to deploy a *learning algorithm* that continues to learn over time, as we expect that it will improve over time and thus (eventually) outperform such a fixed policy. But as before, in these situations we want to have a high degree of confidence in our system before we deploy it. Therefore, we wish to evaluate the whole learning algorithm as it explores, learns and evolves, and not a fixed policy.

An important related problem is developing testbeds on which we can evaluate new reinforcement learning algorithms. Historically, these algorithms have been evaluated on simple hand-designed problems from the literature, often with a small number of states or state variables. Recently, work has considered using a diverse suite of Atari games as a testbed for evaluating reinforcement learning algorithms [17]. Similarly, OpenAI has released Gym and Universe projects [24], which allow one to have reinforcement learning agents control various programs, such as videogames and UIs of varying complexity. However, it is not clear that these artificial problems accurately reflect the complex structure present in important real-world environments. These testbeds, rich as they are, do not give us any

insight into how an RL agent will learn when they must interact with humans, such as helping teach students or treat patients. And these kind of problems, in which an RL agent learns how to directly intervene to help people in need, are arguably much more important use cases than learning to play games or navigate UIs. Therefore, in this chapter we explore using precollected real-world datasets to evaluate new RL algorithms on real human interaction problems in domains such as healthcare, education, or e-commerce.

However, this new problem of evaluating a learning algorithm is very different from the problem of evaluating a fixed policy. We cannot evaluate an algorithm's ability to learn by, for example, feeding it 70% of the precollected dataset as training data and evaluating the produced policy on the remaining 30%. Online, it would have collected a different training dataset based on how it trades off exploration and exploitation. In order to evaluate the performance of an algorithm as it learns, we need to simulate running the algorithm by allowing it to interact with the evaluator as it would with the real (stationary) environment, and record the resulting performance estimates (e.g. cumulative reward). See figure 4.1.

A typical approach to creating such an evaluator is to build a model using the historical data, particularly if the environment is known to be a discrete MDP. However, this approach can result in error that accumulates at least quadratically with the evaluation length [121]. Equally important, in practice it can result in very poor estimates, as we demonstrate in our experiments section. Worse, in complex real-world domains, it is often unclear how to build accurate models. An alternate approach is to try to adapt importance sampling techniques to this problem, but the variance of this approach is unusably high if we wish to evaluate an algorithm for hundreds of timesteps [48].

In this chapter, we first start with a simplified version of the problem in the case of multi-armed bandits. We present a new queue-based algorithm that is substantially more data-efficient than the state-of-the-art prior approach.

Next, we examine the much more challenging case of fully general reinforcement learning. We present, to our knowledge, the first methods for using historical data to evaluate how an RL algorithm would perform online, which possess both meaningful guarantees on the

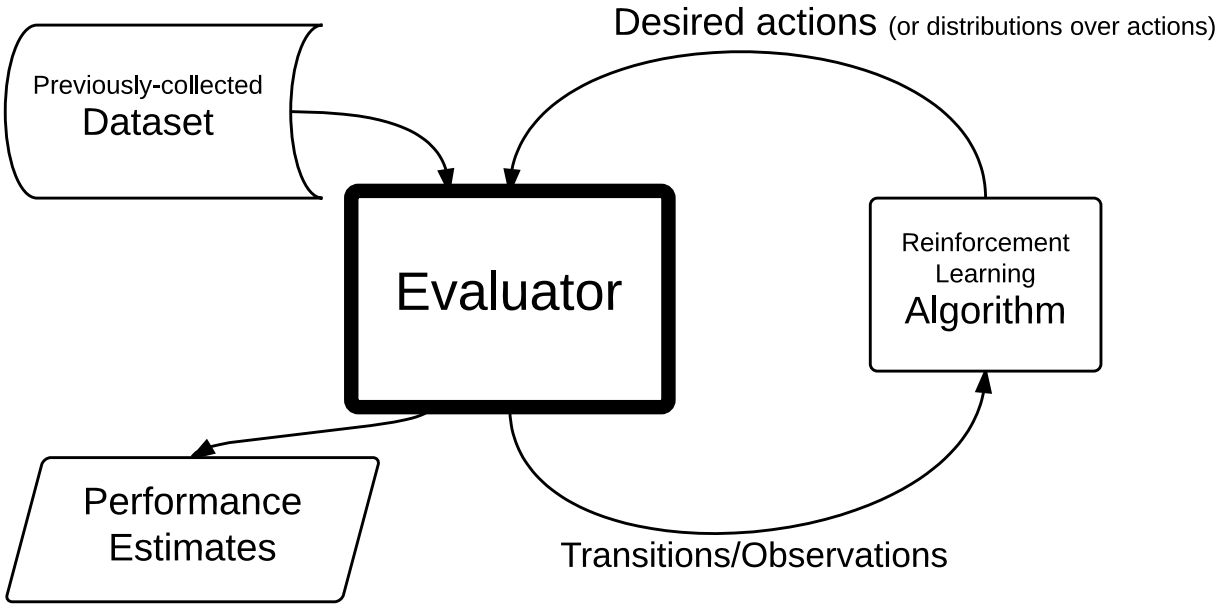


Figure 4.1: Evaluation process: We are interested in developing evaluators that use a previously-collected dataset to interact with an arbitrary reinforcement learning algorithm as it would interact with the true environment. As it interacts, the evaluator produces performance estimates (e.g. cumulative reward).

quality of the resulting performance estimates and good empirical performance. Building upon state-of-the-art work in offline bandit algorithm evaluation ([92]), as well as our novel techniques, we develop three evaluation approaches for reinforcement learning algorithms: queue-based (Queue), per-state rejection sampling (PSRS), and per-episode rejection sampling (PERS). We prove that given the current history, and that the algorithm receives a next observation and reward, that observation and reward is drawn from a distribution identical to the distribution the algorithm would have encountered if it were run in the real world. We show how to modify PERS to achieve stronger guarantees, namely per-timestep unbiasedness given a finite dataset, a property that has not previously been shown even for bandit evaluation methods. Our experiments, including those that use data from a real educational domain, show these methods have different tradeoffs. For example, some are more useful for short-horizon representation-agnostic settings, while others are better suited for long-horizon known-state-space settings. For an overview of further tradeoffs see Table 4.1. We believe this work will be useful for practitioners who wish to evaluate RL algorithms in a reliable manner given access to historical data.

4.2 Online Algorithm Evaluation in the Simplified Case of Bandits

Before moving on the the general reinforcement case in section 4.3, we first examine the simplified case of bandits. Although we were the first to study this problem in the general reinforcement learning case, there had already been investigations of this problem in the setting of (contextual) bandits.[1] However, in this section we improve upon this prior work by developing a novel approach which is often much more sample-efficient.

The state-of-the-art in this space was the replayer of Li et al. [92] developed for the more challenging case of contextual bandits. The original paper focuses on the case where the sampling policy is purely uniform. If this is not the case, Li notes that one can generalize this approach using rejection sampling, but does not give full details. The most straight-

[1]This work referred to the problem as *nonstationary policy evaluation*, but we avoid use of this term to prevent confusion, as these terms are used in many different RL contexts.

forward way would be to subsample the dataset (using rejection sampling) to transform it into a dataset that appears to come from a uniform distribution. However, this is somewhat inefficient, as if the algorithm and sampling policy become similar, we reject a lot of seemingly useful data. For example, the sampling policy samples arm 1 more frequently than average, and the algorithm discovers arm 1 is optimal and so also wishes to sample it more than average. If we subsample the data to emulate a uniform distribution we will have many fewer samples from arm 1 and may not have enough data to get a good long-term estimate of the algorithm's performance.

An alternative variant of Li's replayer (that we use in this chapter) is to keep the full stream of data, and use rejection sampling at each timestep. Specifically, assume we have a bandit problem with N arms labeled $1, \ldots, N$. We know the sampling policy $q(x)$ which defines a probability distribution over the N arms.[2] At some timestep t the algorithm produces a (possibly deterministic) distribution over arms it wishes to pull p(x). Rejection sampling first defines a constant $M = \max_{1,\ldots,N} \frac{p(x)}{q(x)}$. Then we go through each arm-reward tuple (a,r) in our dataset, accepting with probability $\frac{p(a)}{Mq(a)}$ and rejecting otherwise. Accepted samples are treated as online samples from a run of the candidate algorithm, having their rewards recorded and being fed to the algorithm for processing. In cases where $p(x)$ and $q(x)$ are very similar, this approach will be highly data-efficient and accept almost all samples, while reducing to Li's original proposal in the case that the sampling policy is uniform and the algorithm is deterministic. Therefore we compare to this enhanced approach in subsequent sections.

4.2.1 Queue-based Bandit Evaluation Algorithm

In the bandit case, our proposed evaluation process is based on a simple queue-based approach. We have one queue for each bandit arm, each initialized with samples from that arm from a previously-collected dataset D. To evaluate performance of an algorithm, we run it

[2]It is easy to extend the method to handle a changing sampling distribution, but we keep it constant for simplicity of exposition.

as normal, except that at each timestep t instead of getting a reward from a fresh pull of the requested arm i in the true environment, we simply remove a reward from arm i's queue. The intuition is that since the rewards in the queue were drawn i.i.d. from arm i's distribution, blindly storing them in queues and removing them does not change the fact that they are i.i.d samples. When the algorithm eventually samples an empty queue after some number of updates T, we terminate and return an estimate of the algorithm's performance at every timestep up to time T.

This queue-based method allows us to be much more data efficient in the special case of context-free bandits: Li's replayer will "reject" many samples, but our approach stores them in queues for later use. Furthermore, we will show that our method has an unbiasedness guarantee and does not require knowledge of the distribution from which the data was collected.

Traditional unbiasedness can be defined as, for all t:

$$\mathbb{E}\left[\hat{r_t}\right] = \sum_{s'=\{i,\ldots,j\}\in S_t} p(s'|\theta)\mu_j, \tag{4.1}$$

where $\hat{r_t}$ is the estimate of reward at time t, S_t is the list of all possible arm sequences (i.e. possible histories of arm pulls up to time t), $p(s'|\theta)$ is the probability that given the true environment our algorithm will select sequence s' at time t, and μ_j is the true mean of arm j (the last arm in sequence s'). Both our and Li's algorithm satisfy this unbiasedness property, given an infinite dataset. In practice, however, we will only have a finite dataset; for some sequences neither algorithm will be able to output an estimate for time t due to lack of data, so that neither estimator satisfies (4.1).

For our queue replay estimator, however, we can claim a slightly weaker conditional unbiasedness property for the set of sequences $U_t \subseteq S_t$ for which we can return an estimate before hitting the end of a queue. Specifically, we can claim that $\mathbb{E}\left[r_t | s \in U_t\right] = \sum_{s'=\{i,\ldots,j\}\in U_t} p(s'|s' \in U_t, \theta)\mu_j$.

Theorem 4.2.1. *Queue Replay Evaluation estimates are unbiased conditioned on the fact*

that the bandit algorithm produces a sequence of actions for which we issue an estimate. Specifically, $\mathbb{E}[r_t|s \in U_t] = \sum_{s'=\{i,\ldots,j\}\in U_t} p(s'|s' \in U_t, \theta)\mu_j$.

The proof can be found in appendix B.

The obvious issue this raises is that if our algorithm puts considerable probability mass on sequences not in U_t, having the correct expectation for sequences in U_t is meaningless. However, after one run of our algorithm, we can observe when we hit the end of a queue and only report estimates for earlier timesteps. If we want to estimate the reward over multiple randomizations of our algorithm up to some time T, if our evaluation frequently terminates earlier than T steps we can reduce T such that the probability of being over T is sufficiently high (perhaps over a certain threshold, such as 95% or 99%). This will give us a degree of confidence in the reported estimates since rewards are bounded in [0, 1], so a very small percentage of episodes cannot have a large effect on the average reward. Furthermore, note that the order in which we draw items from the queue does not impact the theoretical properties since each item is iid, so we can re-randomize between runs of the algorithm to get a better estimate of general performance.

4.2.2 Bandit Experimental Results

Although theoretically promising, how does our queue-based replayer compare to Li's rejection sampling estimator [92] empirically? To answer this we gathered a real-world dataset from an educational game and used it to compare our evaluation approaches.

Experiment Setup Treefrog Treasure (Figure 4.2) is an educational fractions game developed by the Center for Game Science. Players control a frog and jump off walls, occasionally needing to jump through correct point on a numberline. We want to select optimal settings for numberlines to promote student learning and engagement, which we model as a standard bandit problem. Our experiment has 9 numberline parameters, e.g. whether or not ticks are shown to help guide the player. Given a fixed first set of lines, we want to either increase the difficulty level of one parameter (e.g. removing tick marks) or stay

Figure 4.2: Treefrog Treasure: players guide a frog through a dynamic world, solving number line problems.

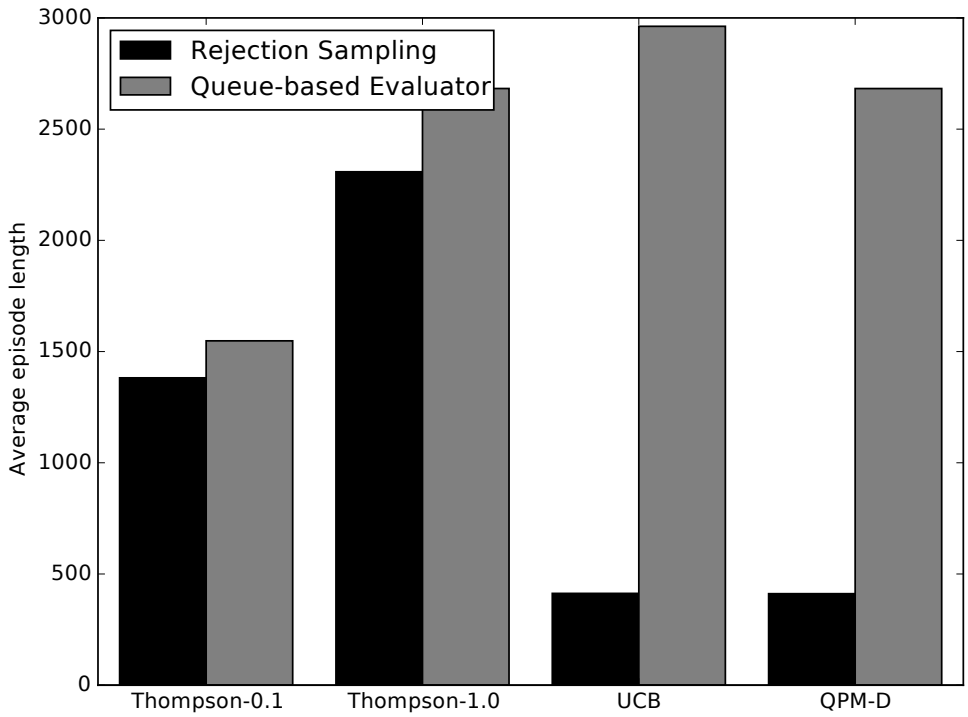
at the same difficulty, corresponding to $9 + 1 = 10$ bandit arms. Since the game randomly increases the difficulty of each parameter as time goes on, the choice of arm affects not only the immediate next line but also future lines. The data collection procedure assigned each player to an arm using a fixed (almost uniform) distribution. Our dataset was collected from BrainPOP.com, an educational website focused on school-aged children. It consisted of 4,396 students, each of whom formed a single (arm, reward) pair.

Our reward is a mix of engagement and learning. At the beginning of the game we assign a randomized in-game pretest with 4 numberlines, and include only those players in the experiment who score at most 2 out of 4. Then, we randomly pick a number of numberlines the player must complete before we give a 4-question in-game posttest. The posttest questions are randomly drawn from the same distribution as the pretest, which allows us to define a noisy metric of learning ranging from -2 to 4 based on the difference between pretest and posttest scores. The reward cannot be less than -2 since students must have gotten at least two lines incorrect on the pretest to be included. Finally, if the player quits before the posttest they are given -0.98 reward. We transform rewards to [0,1] before running our algorithms to satisfy their assumptions.

Results Now, we wish to use this dataset to compare our replayer compare to Li's rejection sampling estimator [92] given real data and various bandit algorithms. Note that if the data collection policy were unknown, it is unclear how to apply Li's approach since it requires knowing the distribution from which the data was drawn. However in our case we do know this distribution, so both approaches produce unbiased sequences of interactions with the environment. Hence the key criteria is how many pulls each approach can generate estimates for: longer sequences give us a better sense of the long-term performance of our algorithms.

Figure 4.3 compares data efficiency between our queue-based estimator and the rejection sampling based replayer of [92] on our 4,396 person dataset when running various algorithms without delay. We see that our performance is only slightly better given a stochastic policy (Thompson), since rejection sampling leverages the revealed randomness to accept many

Figure 4.3: In the bandit case, the queue-based estimator outperforms the rejection-sampling replayer, especially for deterministic bandit algorithms.



samples. However, with deterministic policies such as UCB, our queue-based estimator can evaluate policies for much longer.[3]

4.2.3 Conclusion: Bandit Case

In conclusion, in this section we developed a highly-efficient evaluator for the standard bandit case. We find it outperforms the state-of-the-art evaluation approaches on real data, especially so when the candidate algorithms are deterministic. A further advantage of this approach is that, unlike prior work, it does not need to know the sampling policy to achieve strong guarantees.

However, our method critically relies on the bandit assumption (that pulling arm i produces an i.i.d. reward from a distribution P_i) and it is unclear how to generalize this to the much more general reinforcement learning case. Indeed, prior work had not studied offline evaluation of online reinforcement learning algorithms, motivating our investigation in the rest of this chapter.

4.3 Background and Setting for the Reinforcement Learning Case

We now leave the simplified case of bandits and turn to the more general (and less studied) reinforcement learning case.

A discrete Markov Decision Process (MDP) is specified by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_I)$ where $\mathcal{S}$ is a discrete state space, $\mathcal{A}$ is a discrete action space, $\mathcal{R}$ is a mapping from state, action, next state tuples to a distribution over real valued rewards, $\mathcal{T}$ is a transition model that maps state, action, next state tuples to a probability between 0 and 1, and s_I denotes the starting state[4]. We consider an episode to end (and a new episode to begin) when the system transitions back to initial state s_I.

---

[3]This limitation of rejection sampling-based replayers was noted in [47] as an open problem, although our estimator does not directly apply to the fully-general contextual bandits setting.

[4]Our techniques could still apply given multiple starting states, but for simplicity we assume a single starting state s_I

We assume, unless otherwise specified, that the domain consists of an episodic MDP $\mathcal{M}$ with a given state space $\mathcal{S}$ and action space $\mathcal{A}$, but unknown reward model $\mathcal{R}$ and transition model $\mathcal{T}$. As input we assume a set D of N transitions, (s, a, r, s') drawn from a fixed sampling policy π_e.

Our objective is to use this data to evaluate the performance of a RL algorithm $\mathbb{A}$. Specifically, without loss of generality[5] we will discuss estimating the discounted sum of rewards obtained by the algorithm $\mathbb{A}$ for a sequence of episodes, e.g. $R^{\mathbb{A}}(i) = \sum_{j=0}^{L(i)-1} \gamma^j r_j$ where $L(i)$ denotes the number of interactions in the i^{th} episode, and γ is a discount factor between 0 (inclusive) and 1 (exclusive).

At each timestep $t = 1 \ldots \infty$, the algorithm $\mathbb{A}$ outputs a (possibility stochastic) policy π_b from which the next action should be drawn, potentially sampling a set of random numbers as part of this process. For concreteness, we refer to this (possibly random length) vector of random samples used by $\mathbb{A}$ on a given timestep with the variable χ. Let H_T be the history of (s, a, r, s') and χ consumed by $\mathbb{A}$ up to time T. Then, we can say that the behavior of $\mathbb{A}$ at time T depends only on H_T.

Our goal is to create evaluators (sometimes called replayers) that enable us to simulate running the algorithm $\mathbb{A}$ in the true MDP $\mathcal{M}$ using the input dataset D. One key aspect of our proposed evaluators is that they terminate at some timestep. To that end, let g_t denote the event that we do not terminate before outputting an estimate at timestep t (so g_t implies $g_1, \ldots, g_{t-1}$). In order to compare the evaluator to reality, let $P_{\mathcal{R}}(x)$ denote the probability (or pdf if x contains continuous rewards[6]) of generating x under the evaluator, and $P_{\mathcal{E}}(x)$ denote the probability of generating x in the true environment (the MDP $\mathcal{M}$). Similarly, $\mathbb{E}_{\mathcal{R}}[x]$ is the expected value of a random variable x under the evaluator, and $\mathbb{E}_{\mathcal{E}}[x]$ is the expected value of x under the true environment ($\mathcal{M}$).

We will shortly introduce several evaluators. Here we provide only proof sketches: full

[5]It is easy to modify the evaluators to compute other statistics of the interaction of the algorithm $\mathbb{A}$ with the evaluator, such as the cumulative reward, or the variance of rewards.

[6]In this case, sums should be considered to be integrals

	Samples true	Unbiased estimate of i-th episode performance	Allows unknown sampling distribution	Does not assume Markov	Computationally efficient
Queue	✓	×	✓	×	✓
PSRS	✓	×	×	×	✓
PERS	✓	×(Variants:✓)	×	✓	Not always

Table 4.1: Desired properties of evaluation approaches, and a comparison of the three RL evaluators introduced in this chapter. We did not include the sample efficiency, because although it is a key metric it is typically domain-dependent.

proofs are in appendix B.

What guarantees do we desire on the estimates our evaluators produce? Unbiasedness of the reward estimate on episode i is a natural choice, but it is unclear what this means if we do not always output an estimate of episode i due to termination caused by the limited size/coverage of our dataset. Therefore, we show a guarantee that is in some sense weaker, but applies given a finite dataset: Given some history, the evaluator either terminates or updates the algorithm as it would if run online. Given this guarantee, the empirical question is how early termination occurs, which we address experimentally. We now highlight some of the properties we would like an evaluator to possess, which are summarized in Table 4.1.

1. *Given some history, the (s, a, r, s') tuples provided to $\mathbb{A}$ have the same distribution as those the agent would receive in the true MDP $\mathcal{M}$.* Specifically, we desire $P_{\mathcal{R}}(s, a, r, s', \chi | H_T, g_T) = P_{\mathcal{E}}(s, a, r, s', \chi | H_T)$ so that $P_{\mathcal{R}}(H_{T+1} | H_T, g_T) = P_{\mathcal{E}}(H_{T+1} | H_T)$. As mentioned above, this guarantee allows us to ensure that the algorithm is fed on-policy samples, guaranteeing the algorithm behaves similarly to how it would online.

2. *High sample efficiency.* Since all of our approaches only provide estimates for a finite number of episodes before terminating due to lack of data in D, we want to make efficient use of data to evaluate $\mathbb{A}$ for as long as possible.

3. *Given an input i, outputs an unbiased estimate of $R^{\mathbb{A}}(i)$.* Specifically, $\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(i)] = \mathbb{E}_{\mathcal{E}}[R^{\mathbb{A}}(i)]$. Note that this is non-trivial to ensure, since the evaluation may halt before the i-th episode is reached.

4. *Can leverage data D collected using an unknown sampling distribution π_e.* In some situations it may be difficult to log or access the sampling policy π_e, for example in the case where human doctors choose treatments for patients.

5. *Does not assume the environment is a discrete MDP with a known state space $\mathcal{S}$.* In many real world problems, the state space is unknown, partially observed, or continuous, so we cannot always rely on Markov assumptions.

6. *Computationally efficient.*

4.4 Queue-based Evaluator

We first propose a *queue-based* evaluator for evaluating algorithms for episodic MDPs with a provided state $\mathcal{S}$ and action $\mathcal{A}$ space (Algorithm 1). This technique is inspired by the queue-based approach to evaluation in non-contextual bandits we introduced in section 4.2. The key idea is to place feedback (next states and rewards) in queues, and remove elements based on the current state and chosen action, terminating evaluation when we hit an empty queue. Specifically, first we partition the input dataset D into queues, one queue per (s, a) pair, and fill each queue $Q(s, a)$ with a (random) ordering of all tuples $(r, s') \in D$ s.t. $(s_i = s, a_i = a, r_i = r, s'_i = s')$ To simulate algorithm $\mathbb{A}$ starting from a known state s_k, the

Algorithm 1 Queue-based Evaluator

Input: Dataset D, RL Algorithm $\mathbb{A}$, Starting state s_I
Output: $R^{\mathbb{A}}$ s.t. $R^{\mathbb{A}}(i)$ is sum of rewards in ep. i
$Q[s,a] = Queue(RandomOrder((s_i, a_i, r, s') \in D, \text{ s.t. } s_i = s, a_i = a)), \forall s \in \mathcal{S}, a \in \mathcal{A}$
for $i = 1$ to ∞ **do**
    $s = s_I$, $t = 0, r_i = 0$
    Let π_b be $\mathbb{A}$'s initial policy
    while $\neg(t > 0$ and $s == s_I)$ **do**
        $a_b \sim \pi_b(s)$
        if $Q[s, a_b]$ is empty **then return** $R^{\mathbb{A}}$
        $(r, s') = Q[s, a_b].pop()$
        Update $\mathbb{A}$ with (s, a, r, s'), yields new policy π_b
        $r_i = r_i + \gamma^t r, s = s', t = t + 1$
    $R^{\mathbb{A}}(i) = r_i$

algorithm $\mathbb{A}$ outputs a policy π_b, and selects an action a sampled from $\pi_b(s_k)$.[7]

The evaluator then removes a tuple (r, s') from queue $Q[s_k, a]$, which is used to update the algorithm $\mathbb{A}$ and its policy π_b, and simulate a transition to the next state s'. By the Markov assumption, tuples (r, s') are i.i.d. given the prior state and selected action, and therefore an element drawn without replacement from the queue has the same distribution as that in the true environment. The evaluator terminates and outputs the reward vector $R^{\mathbb{A}}$, when it seeks to draw a sample from an empty queue.[8]

Unlike many offline evaluation approaches (such as importance sampling for policy evaluation), our queue evaluator does not require knowledge of the sampling distribution π_e used to generate D. It can even use data gathered from a deterministic sampling distribution.

[7]Note that since we only use π_b to draw the next action, this does not prevent $\mathbb{A}$ from internally using a policy that depends on more than s (for example, s and t in finite horizon settings).

[8]For details about why this is necessary, see appendix B.

Both properties are useful for many domains (for example, it may be hard to know the stochastic policy used by a doctor to make a decision).

Theorem C.1. *Assuming the environment is an MDP with state space $\mathcal{S}$ and the randomness involved in drawing from π_b is treated as internal to $\mathbb{A}$, given any history of interactions H_T, if the queue-based evaluator produces a (s, a, r, s') tuple, the distribution of this tuple and subsequent internal randomness χ under the queue-based evaluator is identical to the true distribution the agent would have encountered if it was run online. That is, $P_{\mathcal{R}}(s, a, r, s', \chi | H_T, g_T) = P_{\mathcal{E}}(s, a, r, s', \chi | H_T)$, which gives us that $P_{\mathcal{R}}(H_{T+1} | H_T, g_T) = P_{\mathcal{E}}(H_{T+1} | H_T)$.*

Proof Sketch. The proof follows fairly directly from the fact that placing an (r, s') tuple drawn from $\mathcal{M}$ in $Q[s, a]$ and sampling from Q without replacement results in a sample from the true distribution. See appendix B.

Note that theorem C.1 requires us to condition on the fact that $\mathbb{A}$ reveals no randomness, that is, we consider the randomness involved in drawing from π_b on line 8 to be considered as internal, that is (included in χ). This means the guarantee is slightly weaker than the approaches we will present in sections 4.5 and 4.6, which condition on general π_b.

4.5 Per-State Rejection Sampling Evaluator

Ideally, we would like an evaluator that can recognize when the algorithm chooses actions similarly to the sampling distribution, in order use more of the data. For example, in the extreme case where we know the algorithm we are evaluating always outputs the sampling policy, we should be able to make use of all data, or close to it. However, the queue method only uses the sampled action, and thus cannot determine directly whether or not the distribution over actions at each step (π_b) is similar to the sampling policy (π_e). This can make a major difference in practice: If π_b and π_e are both uniform, and the action space is large relative to the amount of data, we will be likely to hit an empty queue if we sample a fresh action from π_b. But, if we know the distributions are the same we can simply take the first

sampled action from π_e. Being able to take advantage of stochastic distributions in this way is sometimes referred to as leveraging *revealed randomness* in the candidate algorithm [47].

To better leverage this similarity, we introduce the Per-State Rejection Sampling (PSRS) evaluator (see Algorithm 2), inspired by approaches used in contextual bandits [92, 47]. PSRS divides data into streams for each state s, consisting of a (randomized) list of the subsequent (a, r, s') tuples that were encountered from s in the input data. Specifically, given the current state s, our goal is to sample a tuple (a, r, s') such that a is sampled from algorithm $\mathbb{A}$'s current policy $\pi_b(s)$, and r and s' are sampled from the true environment. We already know that given the Markov property, once we select an action a that r and s' in a tuple (s, a, r, s') represent true samples from the underlying Markov environment. The challenge then becomes to sample an action a from $\pi_b(s)$ using the actions sampled by the sampling distribution $\pi_e(s)$ for the current state s. To do this, a rejection sampling algorithm[9] samples a uniform number u between 0 and 1, and accepts a sample (s, a, r, s') from D if $u < \frac{\pi_b(a|s)}{M\pi_e(a|s)}$, where $\pi_b(a|s)$ is the probability under the candidate distribution of sampling action a for state s, $\pi_e(a|s)$ is the corresponding quantity for the sampling distribution, and M is an upper bound on their ratio, $M \geq \max_a \frac{\pi_b(a|s)}{\pi_e(a|s)}$. M is computed by iterating over actions[10] (line 8). It is well known that samples rejection sampling accepts represent true samples from the desired distribution, here π_b [56].

Slightly surprisingly, even if $\mathbb{A}$ always outputs the sampling policy π_e, we do not always consume all samples (in other words PSRS is not *self-idempotent*), unless the original ordering of the streams is preserved (see appendix B). Still, in the face of stochasticity PSRS can be significantly more data-efficient than the Queue-based evaluator.

Theorem 4.5.1. *Assume the environment is an MDP with state space $\mathcal{S}$, π_e is known, and*

[9]One might wonder if we could reduce variance by using an importance sampling instead of rejection sampling approach here. Although in theory possible, one has to keep track of all the different states of the algorithm with and without each datapoint accepted, which is computationally intractable.

[10]This approach is efficient in the since that it takes time linear in $|\mathcal{A}|$, however in very large action spaces this might be too expensive. In certain situations it may be possible to analytically derive a bound on the ratio to avoid this computation.

Algorithm 2 Per-State Rejection Sampling Evaluator

1: Input: Dataset D, RL Algorithm $\mathbb{A}$, Start state s_I, π_e
2: Output: Output: $R^{\mathbb{A}}$ s.t. $R^{\mathbb{A}}(i)$ is sum of rewards in ep. i
3: $Q[s] = Queue(RandomOrder((s_i, a_i, r, s') \in D \text{ s.t. } s_i = s)), \forall s \in \mathcal{S}$
4: **for** $i = 1$ to ∞ **do**
5: $s = s_I$,$t = 0, r_i = 0$
6: Let π_b be $\mathbb{A}$'s initial policy
7: **while** $\neg(t > 0$ and $s_t == s_I)$ **do**
8: $M = \max_a \frac{\pi_b(a|s)}{\pi_e(a|s)}$
9: $(a, r, s') =$ Q[s].pop()
10: **if** Q[s] is empty **then return** $R^{\mathbb{A}}$
11: Sample $u \sim Uniform(0, 1)$
12: **if** $u > \frac{\pi_b(a|s)}{M\pi_e(a|s)}$ **then**
13: Reject sample, go to line 9
14: Update $\mathbb{A}$ with (s, a, r, s'), yields new policy π_b
15: $r_i = r_i + \gamma^t r, s = s', t = t + 1$
16: $R^{\mathbb{A}}(i) = r_i$

for all a, $\pi_e(a) > 0$ if $\pi_b(a) > 0$. Then if the evaluator produces a (s, a, r, s') tuple, the distribution of (s, a, r, s') tuple returned by PSRS (and subsequent internal randomness χ) given any history of interactions H_T is identical to the true distribution the agent would have encountered if was run online. Precisely, $P_{\mathcal{R}}(s, a, r, s', \chi|H_T, g_T) = P_{\mathcal{E}}(s, a, r, s', \chi|H_T)$, which gives us that $P_{\mathcal{R}}(H_{T+1}|H_T, g_T) = P_{\mathcal{E}}(H_{T+1}|H_T)$.

Proof Sketch. The proof follows fairly directly from the fact that given finite dataset, rejection sampling returns samples from the correct distribution (Lemma B.3.1 in appendix B).

4.6 Per-Episode Rejection Sampling

The previous methods assumed the environment is a MDP with a known state space. We now consider the more general setting where the environment consists of a (possibly high dimensional, continuous) observation space $\mathcal{O}$, and a discrete action space $\mathcal{A}$. The dynamics of the environment can depend on the full history of prior observations, actions, and rewards, $h_t = o_0, \ldots, o_t, a_0, \ldots, a_{t-1}, r_0, \ldots, r_{t-1}$. Multiple existing models, such as POMDPs and PSRs, can be represented in this setting. We would like to build an evaluator that is *representation-agnostic*, i.e. does not require Markov assumptions, and whose sample-efficiency does not depend on the size of the observation space.

We introduce the Per-Episode Rejection Sampler (PERS) evaluator (Algorithm 3) that evaluates RL algorithms in these more generic environments. In this setting we assume that the dataset D consists of a stream of episodes, where each episode e represents an ordered trajectory of actions, rewards and observations, $(o_0, a_0, r_0, o_1, a_1, r_1, \ldots, r_{l(e)})$ obtained by executing the sampling distribution π_e for $l(e) - 1$ time steps in the environment. We assume that π_e may also be a function of the full history h_t in this episode up to the current time point. For simplicity of notation, instead of keeping track of multiple policies π_b, we simply write π_b (which could implicitly depend on χ).

PERS operates similarly to PSRS, but performs rejection sampling at the episode level. This involves computing the ratio of $\frac{\Pi_{t=0}^{l(e)} \pi_b(a_t|h_t)}{M \Pi_{t=0}^{l(e)} \pi_e(a_t|h_t)}$, and accepting or rejecting the episode according to whether a random variable sampled from the uniform distribution is lower than the computed ratio. As M is a constant that represents the maximum possible ratio between the candidate and sampling *episode* probabilities, it can be computationally involved to compute M exactly. Due to space limitations, we present approaches for computing M in appendix B. Note that since the probability of accepting an episode is based only on a ratio of action probabilities, one major benefit to PERS is that its sample-efficiency does not depend on the size of the observation space. However, it does depend strongly on the episode length, as we will see in our experiments.

Although PERS works on an episode-level, to handle algorithms that update after every timestep, it updates $\mathbb{A}$ throughput the episode and "rolls back" the state of the algorithm if the episode is rejected (see Algorithm 3).

Unlike PSRS, PERS is self-idempotent, meaning if $\mathbb{A}$ always outputs π_e we accept all data. This follows since if $\pi_e(a_t|h_t) = \pi_b(a_t|h_t)$, $M = 1$ and $\frac{\Pi_{t=0}^{l(e)}\pi_b(a_t|h_t)}{M\Pi_{t=0}^{l(e)}\pi_e(a_t|h_t)} = 1$.

Theorem 4.6.1. *Assuming π_e is known, and $\pi_b(e) > 0 \rightarrow \pi_e(e) > 0$ for all possible episodes e and all π_b, and PERS outputs an episode e, then the distribution of e (and subsequent internal randomness χ) given any history of episodic interactions H_T using PERS is identical to the true distribution the agent would have encountered if it was run online. That is, $P_{\mathcal{E}}(e, \chi|H_T) = P_{\mathcal{R}}(e, \chi|H_T, g_T)$, which gives us that $P_{\mathcal{R}}(H_{T+1}|H_T, g_T) = P_{\mathcal{E}}(H_{T+1}|H_T)$.*

Proof Sketch. The proof follows fairly directly from the fact that given finite dataset, rejection sampling returns samples from the correct distribution (Lemma B.3.1 in appendix B.

4.7 Unbiasedness Guarantees in the Per-Episode case

Our previous guarantees stated that if we return a sample, it is from the true distribution given the history. Although this is fairly strong, it does not ensure $R^A(i)$ is an unbiased estimate of the reward obtained by A in episode i. The difficulty is that across multiple runs of evaluation, the evaluator may terminate after different numbers of episodes. The probability of termination depends on a host of factors (how random the policy is, which state we are in, etc.). This can result in a bias, as certain situations may be more likely to reach a given length than others.

For example, consider running the queue-based approach on a 3-state MDP: s_I is the initial state, if we take action a_0 we transition to state s_1, if we take action a_1 we transition to s_2. The episode always ends after timestep 2. Imagine the sampling policy chose a_1 99% of the time, but our algorithm chose a_1 50% of the time. If we run the queue approach many times in this setting, runs where the algorithm chose a_1 will be much more likely to reach timestep 2 than those where it chose a_0, since s_2 is likely to have many more samples than

s_1. This can result in a bias: if the agent receives a higher reward for ending in s_2 compared to s_1, the average reward it receives at timestep 2 will be overestimated.

One approach proposed by past work [48] is to assume T (the maximum timestep/episode count for which we report estimates) is small enough such that over multiple runs of evaluation we usually terminate after T; however it can be difficult to fully bound the remaining bias. Eliminating this bias for the state-based methods is difficult, since the the agent is much more likely to terminate if it transitions to a sparsely-visited state, and so the probability of terminating is hard to compute as it depends on the unknown transition probabilities.

However, modifying PERS to use a fixed M throughout its operation allows us to show that if PERS outputs an estimate, that estimate is unbiased (Theorem 4.7.1). In practice one will likely have to overestimate this M, for example by bounding $p(x)$ by 1 (or $(1-\epsilon)$ for epsilon-greedy) and calculating the minimum $q(x)$.

Theorem 4.7.1. *If M is held fixed throughout the operation of PERS, π_e is known, and $\pi_b(e) > 0 \rightarrow \pi_e(e) > 0$ for all possible episodes e and all π_b, then if PERS outputs an estimate of some function $f(H_T)$ at episode T, that estimate is an unbiased estimator of $f(H_T)$ at episode T, in other words, $\mathbb{E}_{\mathcal{R}}[f(H_T)|g_T, \ldots, g_1] = \sum_{H_T} f(H_T) P_{\mathcal{E}}(H_T) = \mathbb{E}_{\mathcal{E}}[f(H_T)]$. For example, if $f(H_T) = R^{\mathbb{A}}(T)$, the estimate is an unbiased estimator of $R^{\mathbb{A}}(T)$ given $g_T, \ldots, g_1$.*

Proof Sketch. We first show that if M is fixed, the probability that each episode is accepted is constant $(1/M)$. This allows us to show that whether we continue or not (g_T) is conditionally independent of H_{T-1}. This lets us remove the conditioning on H_{T-1} in Theorem 4.6.1 to give us that $P_{\mathcal{R}}(H_T|g_T, \ldots, g_1) = P_{\mathcal{E}}(H_T)$, meaning the distribution over histories after T accepted episodes is correct, from which conditional unbiasedness is easily shown.

Although useful, this guarantee has the downside that the estimate is still conditional on the fact that our approach does not terminate. Theorem 4.7.2 shows that it is possible to use a further modification of Fixed-M PERS based on importance weighting to always issue

unbiased estimates for N total episodes. For a discussion of the empirical downsides to this approach, see appendix B.

Theorem 4.7.2. *Assuming for each T, $R^{\mathbb{A}}(T)$ is divided by by $\phi = 1 - Binomial(N, 1/M).cdf(k-1)$, and after terminating at timestep k we output 0 as estimates of reward for episodes $k+1, \ldots, N$, and M is held fixed throughout the operation of PERS, and π_e is known, and $\pi_b(e) > 0 \rightarrow \pi_e(e) > 0$ for all possible episodes e and all π_b, then the estimate of reward output at each episode $T = 1 \ldots N$ is an unbiased estimator of $R^{\mathbb{A}}(T)$.*

Proof Sketch. Outputting an estimate of the reward at an episode T by either dividing the observed reward by the probability of reaching T (aka $P(g_T, ..., g_1)$), for a run of the evaluator that reaches at least T episodes, or else outputting a 0 if the evaluation has terminated, is an importance weighting technique that ensures the expectation is correct.

4.8 Experiments

Any RL algorithm could potentially be run with these evaluators. Here, we show results evaluating Posterior Sampling Reinforcement Learning (PSRL) (Osband et al. [114], Strens [137]), which has shown good empirical and theoretical performance in the finite horizon case. The standard version of PSRL creates one deterministic policy each episode based on a single posterior sample; however, we can sample the posterior multiple times to create multiple policies and randomly choose between them at each step, which allows us to test our evaluators with more or less revealed randomness.

Comparison to a model-based approach We first compare PSRS to a model-based approach on SixArms [135], a small MDP environment. Our goal is to evaluate the cumulative reward of PSRL run with 10 posterior samples, given a dataset of 100 samples collected using a uniform sampling policy. The model-based approach uses the dataset to build an MLE MDP model. Mean squared error was computed against the average of 1000 runs against the true environment. For details see appendix B. In Figure 4.4 we see that the model-based approach starts fairly accurate but quickly begins returning very poor es-

Figure 4.4: SRS tends to be much more accurate than a model-based approach.

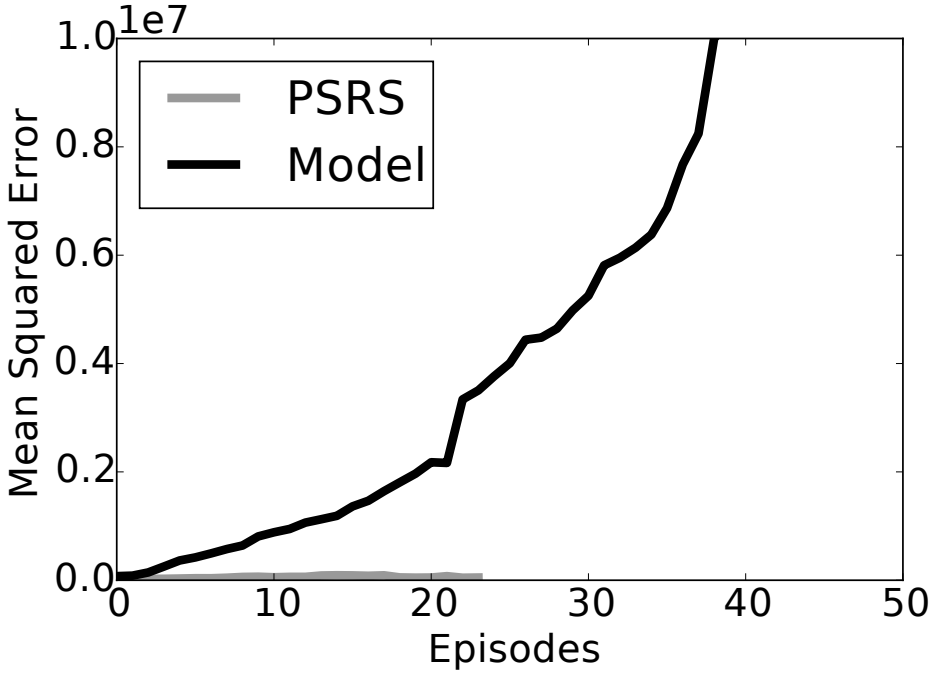


Figure 4.5: Comparing on Treefrog Treasure with 3 timesteps and 1 PSRL posterior sample.

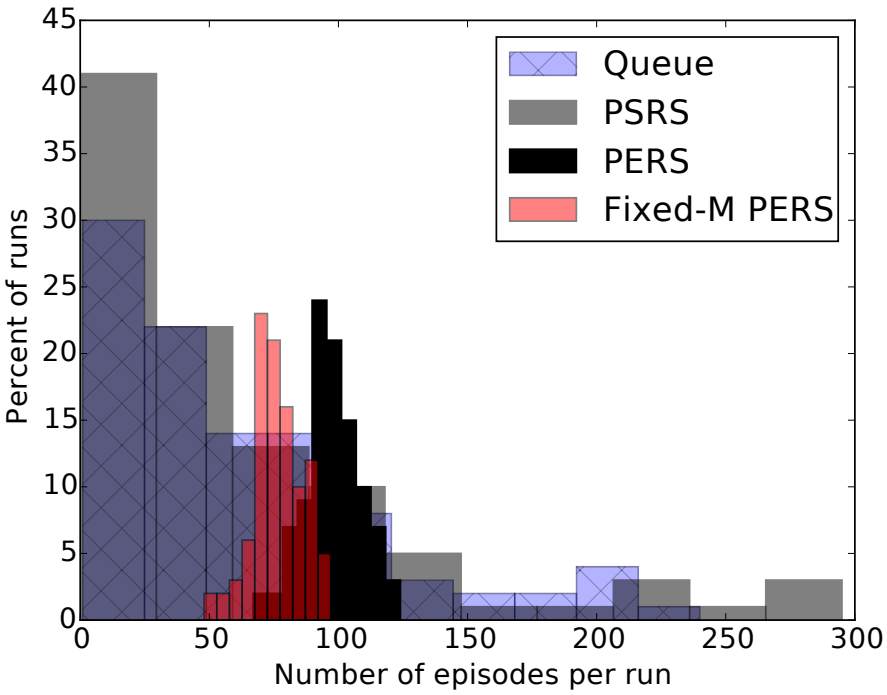
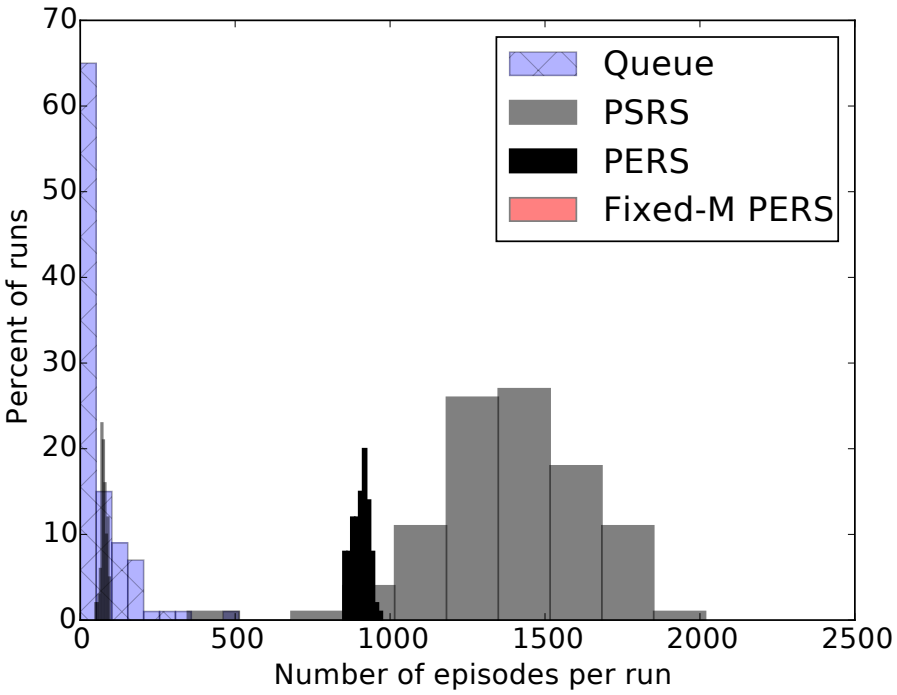


Figure 4.6: Comparing on Treefrog with 3 timesteps and 10 PSRL posterior samples.



timates. In this setting, the estimates it returned indicated that PSRL was learning much more quickly than it would in reality. In contrast, our PSRS approach returns much more accurate estimates and ceases evaluation instead of issuing poor estimates.

Length Results All three of our estimators produce samples from the correct distribution at every step. However, they may provide different length trajectories before termination. To understand the data-efficiency of each evaluator, we tested them on a real-world educational game dataset, as well as a small but well-known MDP example.

Treefrog Treasure is an educational fractions game (Figure 4.2). The player controls a frog to navigate levels and jump through numberlines. We have 11 actions which control parameters of the numberlines. Our reward is based on whether students learn (based on pretest-to-postest improvement) and whether they remain engaged (measured by whether the student quit before the posttest). We used a state space consisting of the history of actions and whether or not the student took more than 4 tries to pass a numberline (note that this space grows exponentially with the horizon). We varied the considered horizon between 3 and 4 in our experiments. We collected a dataset of 11,550 players collected from a child-focused educational website, collected using a semi-uniform sampling policy. More complete descriptions of the game, experimental methodology, method of calculating M, and details of PSRL can be found in appendix B.

Figures 4.5 and 4.6 show results on Treefrog Treasure, with histograms over 100 complete runs of each evaluator. The graphs show how many episodes the estimator could evaluate the RL algorithm for, with more being better. PERS does slightly better in a short-horizon deterministic setting (Figure 4.5). Increasing the posterior samples greatly improves performance of rejection sampling methods (Figure 4.6).

We also examined an increased horizon of 4 (graphs provided in appendix B). Given deterministic policies on this larger state space, all three methods are more or less indistinguishable; however, revealing more randomness causes PERS to overtake PSRS (mean 260.54 vs. 173.52). As an extreme case, we also tried a random policy: this large amount of revealed randomness benefits the rejection sampling methods, especially PERS, which evaluates for

much longer than the other approaches. PERS outperforms PSRS here because there are small differences between the random candidate policy and the semi-random sampling policy, and thus if PSRS enters a state with little data it is likely to terminate.

The fixed-M PERS method does much worse than the standard version, typically barely accepting any episodes, with notable exceptions when the horizon is short (Figure 4.5). Since it does not adjust M it cannot take advantage of revealed randomness (Figure 4.6). However, we still feel that this approach can be useful when one desires truly unbiased estimates, and when the horizon is short. Finally, we also note that PERS tends to have the lowest variance, which makes it an attractive approach since to reduce bias one needs to have a high percentage of runs terminating after the desired length.

The state space used in Treefrog Treasure grows exponentially with the horizon. To examine a contrasting case with a small state space (6 states), but a long horizon (20), we also test our approaches in Riverswim [114], a standard toy MDP environment. The results can be found in appendix B, but in general we found that PERS and its variants suffer greatly from the long horizon, while Queue and PSRS do much better, with PSRS doing particularly well if randomness is revealed.

Our conclusion is that the PERS does quite well, especially if randomness is revealed and the horizon is short. It appears there is little reason to choose Queue over PSRS, except if the sampling distribution is unknown. This is surprising because it conflicts with the results we showed in section 4.2. In that section we found a queue-based approach to be more efficient than rejection sampling in a non-contextual bandit setting, since data remained in the queues for future use instead of being rejected. The key difference between the settings is that in bandits there is only one state, so we do not encounter the problem that we happen to land on an undersampled state, hit an empty queue by chance, and have to terminate the whole evaluation procedure. If the candidate policy behaves randomly at unvisited states, as is the case with 10-sample PSRL, PSRS can mitigate this problem by recognizing the similarity between sampling and candidate distributions to accept the samples at that state, therefore being much less likely to terminate evaluation when a sparsely-visited state is encountered.

4.9 Related work

Work in reinforcement learning has typically focused on evaluating fixed policies using importance sampling techniques [119]. Importance sampling is widely-used in off-policy learning, as an objective function when using policy gradient methods [90, 118] or as a way to re-weight samples in off-policy TD-learning methods [100, 139, 99]. Additionally, this approach enabled us to evaluate learned policies on a complex real-world setting in chapter 3.[11] However, this work focuses on evaluating fixed policies, we are not aware of work specifically focusing on the problem of evaluating how an RL algorithm would learn online, which involves feeding the algorithm new training samples as well as evaluating its current performance. It is worth noting that any of the above-mentioned off-policy learning algorithms could be evaluated using our methods.

Our methods do bear a relationship to off-policy learning work which has evaluated policies by synthesizing artificial trajectories [54, 55]. Unlike our work, this approach focuses only on evaluating fixed policies. It also assumes a degree of Lipschitz continuity in some continuous space, which introduces bias. There are some connections: our queue-based estimator could be viewed as related to their work, but focused on evaluating learning algorithms in the discrete MDP policy case.

One area of related work is in the area of (possibly contextual) multi-armed bandits, in which the corresponding problem is termed "nonstationary policy evaluation". Past work has showed evaluation methods that are guaranteed to be unbiased [92], or have low bias [47, 48], but only assuming an infinite data stream. Other work has focused on evaluators that perform well empirically but lack this unbiasedness [111]. This is similar to the guarantees we showed in the non-contextual bandit setting (section 4.2), that issued feedback comes from the true distribution even with finite data. However, in addition to focusing on the more general setting of reinforcement learning, we also show stronger guarantees of unbiasedness

[11] Indeed follow up work by Thomas et al. [146] has also used offline policy evaluation to cope with other real-world domains.

even given a finite dataset.

4.10 Conclusion

In this chapter we presented three novel approaches for evaluating how RL algorithms perform online: the most important differences are summarized in Table 4.1. All methods have guarantees that, given some history, if a sample is output it comes from the true distribution. Further, we developed a variant of PERS with even stronger guarantees of unbiasedness. Empirically, there are a variety of tradeoffs to navigate between the methods, based on horizon, revealed randomness in the candidate algorithm, and state space size. We anticipate these approaches will find wide use when one wishes to compare different reinforcement learning algorithms on a real-world problem before deployment. Further, we are excited at the possibility of using these approaches to create real-world testbeds for reinforcement learning problems, perhaps even leading to RL competitions similar to those which related contextual bandit evaluation work [92] enabled in that setting [34]. Future theoretical work includes analyzing the sample complexity of our approaches and deriving tight deviation bounds on the returned estimates. Another interesting direction is developing more accurate estimators, e.g. by using doubly-robust estimation techniques [47].

Algorithm 3 Per-Episode Rejection Sampling Evaluator

Input: Dataset of episodes D, RL Algorithm $\mathbb{A}$, π_e
Output: Output: $R^{\mathbb{A}}$ s.t. $R^{\mathbb{A}}(i)$ is sum of rewards in ep. i
Randomly shuffle D
Store present state $\mathcal{A}$ of algorithm $\mathbb{A}$
$M = calculateEpisodeM(\mathcal{A}, \pi_e)$ (see appendix B)
$i = 1$, Let π_b be $\mathbb{A}$'s initial policy
for $e \in D$ **do**
    $p = 1.0, h = [], t = 0, r_i = 0$
    for $(o, a, r) \in e$ **do**
        $h \to (h, o)$
        $p = p\frac{\pi_b(a|h)}{\pi_e(a|h)}$
        Update $\mathbb{A}$ with (o, a, r), output new policy π_b
        $h \to (h, a, r)$, $r_i = r_i + \gamma^t r$
    Sample $u \sim Uniform(0, 1)$
    if $u > \frac{p}{M}$ **then**
        Roll back algorithm: $\mathbb{A} = \mathcal{A}$
    else
        Store present state $\mathcal{A}$ of algorithm $\mathbb{A}$
        $M = calculateEpisodeM(\mathcal{A}, \pi_e)$
        $R^{\mathbb{A}}(i) = r_i, i = i + 1$
return $R^{\mathbb{A}}$

Chapter 5

EFFICIENT EXPLORATION IN THE PRESENCE OF DELAY

This chapter is based on work the author published at AAAI 2015.

5.1 Introduction

A key part of AI systems is the ability to make decisions under uncertainty. Multi-armed bandits [148] are a common framework for deciding between a discrete number of choices with uncertain payoffs. For example, we might wish to select the best ad to show on a webpage to maximize revenue, or we may have several different educational strategies and we want to select the one that maximizes learning. However, we do not want to decouple exploration and exploitation: we want to use the feedback we get from pulling one arm to inform the next pull.

The standard bandit problem has been well studied, resulting in algorithms with near-optimal performance both in theory [11] and empirically [152]. However, most real-world problems do not match the bandit framework exactly. One difference is delay: if we present a problem to a student, other students who we need to interact with may arrive before the original student has finished. Also, when we have distributed systems with thousands of incoming users, real-time policy updates may be infeasible; instead, one typically launches a fixed (possibly stochastic) policy for a batch of users, updating it only after a period of time. These related problems, delayed feedback and batch updating, pose challenges for typical bandit algorithms theoretically [42] and empirically [29]. Although stochastic algorithms perform best empirically in the face of delay [29], deterministic algorithms have the best theoretical guarantees [75].

Second, we may know heuristic algorithms that do not possess good theoretical guar-

antees, but tend to perform well. For example, encouraging an algorithm to be more exploitative often results in good short-term performance [29], but may cause it to eventually settle on a sub-optimal arm. Or, we can make a structural assumption that similar arms share information which enables good performance if this assumption is true [125], but if false results in poor performance. Ideally, we could incorporate a heuristic to help improve performance while retaining strong theoretical guarantees.

Third, we may have a prior data set consisting of information from a subset of arms, such as a website which decides to add new ads to help improve revenue. We want to make use of the old ad data without being overly biased by it, since the data can be drawn from an arbitrary arm distribution and thus possibly hurt performance. Ideally, we want a method that guarantees good black-box algorithms will perform well (or even better) when leveraging a prior dataset.

In this chapter, we present solutions to these three problems, all of which are based on the queue method. The queue method was inspired by the recent work of Joulani et al. [75], and is conceptually simple: instead of allowing our black-box bandit algorithm to actually pull from arm i in the true environment, we draw a reward from a queue associated with i. This allows us to place arbitrary samples in the queues while ensuring that the black-box algorithm behaves as it would if it was actually pulling from the true environment. In this work, we show how this method can be used to produce good solutions to three problems. First, we present Stochastic Delayed Bandits (SDB), an algorithm that improves over QPM-D by taking better advantage of stochastic algorithms in the face of delay. Next, we show how one can use SDB to take advantage of heuristics while retaining strong theoretical guarantees. Finally, we show how SDB can be robust to prior data. In addition to theoretical analysis, we demonstrate good empirical performance on both synthetic simulations and real world data drawn from thousands of students playing an educational game.

5.2 Background

We consider stochastic multi-armed bandit problems, in which there are a set of N arms labeled as integers $i \in \{1, \ldots, N\}$, each of which has an unknown reward distribution with mean μ_i. In the standard online (i.e. non-delayed) problem formulation, at each timestep $t \in \{1, \ldots, T\}$ the agent pulls arm $i \in \{1, \ldots, N\}$ and receives a reward r_t drawn iid from arm i's reward distribution. The goal of the agent is to maximize its expected reward, or equivalently minimize its expected cumulative regret $\mathbb{E}R(T) = \sum_{i=1}^{N} \Delta_i L_{i,T}$, where $\Delta_i = (\max_j \mu_j) - \mu_i$ and $L_{i,T}$ is the number of times the algorithm has played arm i at time T.

We additionally consider versions of the problem with some *delay*: the reward for a pull at time t becomes available to the algorithm only at some time $t' > t$. We assume there is some unknown, arbitrary process which generates delay times[1], but similar to Joulani et al. [75] we assume the delay is bounded by some unknown maximum τ_{max}.

We want to draw a distinction between *online updating* and *batch updating* in the presence of delay. In the *online updating* case, the algorithm can explicitly control which arm it pulls at every time t, even if it receives no feedback at time t. In the *batch updating* case, the algorithm must specify a single distribution over arm pulls to be run for an entire batch of pulls, and can only update it once the batch is complete. Batch updating makes more sense for distributed systems where deploying new policies is challenging to perform online (but can be done every hour or night). To simplify the analysis we assume that we observe the rewards of all arms pulled during the batch once the batch completes; thus the maximum delay, τ_{max}, is equal to the maximum batch size[2]. We again assume this quantity is unknown.

5.3 Stochastic Delayed Bandits

One central question underlies our four problems: how can we give arbitrary data to an arbitrary bandit algorithm BASE such that BASE behaves exactly as it would if it were run

[1] i.e., the delay could be chosen adversarially depending on the current action as well as on the history of past pulls and rewards.

[2] Lemma C.1.1 in appendix C addresses the case where this assumption is relaxed.

online? Such a procedure would allow us to feed delayed samples, prior datasets, and samples collected by a heuristic to Base without impacting its behavior, thus retaining theoretical guarantees. To solve this problem efficiently, we must heavily rely on the stochastic assumption: each reward from arm i is drawn iid. As observed by Joulani et al. [75], this assumption allows us to put collected rewards into queues for each arm. When Base selects an arm I we simply draw an item from the queue I, allowing Base to behave as it would if it pulled arm I in the true environment. This method defines approaches we call queue-based bandit algorithms (see Algorithm 4).

However, the more challenging question is how to sample to put items in the queues. If we want to guarantee good regret with respect to the Base algorithm, this sampling cannot be truly arbitrary; instead, the behavior of Base must influence the sampling distribution. In the online case, the obvious solution is to add a sample to a queue as soon as Base wishes to pull an arm with an empty queue, but it is more challenging to determine what to do when rewards are returned with some delay. Joulani et al. [75] addressed this problem with their QPM-D meta-algorithm, which continues to put items in the requested empty queue until they receive at least one reward for that arm (namely Algorithm 4 with getSamplingDist being $q_I = 1; q_{i \neq I} = 0$). They proved that QPM-D achieves the best-known bound on stochastic bandit regret under delay.

Although QPM-D has a good regret bound, as we will see in our experiments, its empirical performance is somewhat limited. This is mostly due to the fact that it is a deterministic algorithm, while algorithms producing stochastic arm policies have been shown to perform much better in practice when delay is present [29], since they make better use of the time between observations. For example, consider a batch update setting with a very long initial batch: a stochastic policy can get information about all arms, while a deterministic policy uses the entire batch to learn about just one arm. And, since the sampling procedure is to some extent decoupled from the behavior of Base, one would like to incorporate heuristics to better guide the sampling procedure while retaining a strong regret bound.

We present a novel meta-algorithm, Stochastic Delayed Bandits (SDB) (Algorithm 5),

which takes better advantage of stochasticity and heuristics to improve performance in the presence of delay. The basic idea is that in addition to a (possibly stochastic) bandit algorithm BASE, we also take as input an arbitrary (possibly stochastic) algorithm HEURISTIC. SDB tries to follow the probability distribution specified by the heuristic algorithm, but with the constraint that the queues do not get too large. Specifically, we keep an estimate B of τ_{max}, and try to prevent any queue from being larger than B. If a queue for arm i approaches size B, we only allow the algorithm to assign a small amount of probability to arm i, and assign zero probability if it is over B. However, if all queues are small compared to B we allow HEURISTIC to control the sampling distribution. Intuitively, this approach keeps the queues small; this is desirable because if the queues grow too large there could be a large gap between our current performance and that of BASE run online. We prove a regret bound on SDB within a constant factor of that of QPM-D, and in our experiments we will see that it performs much better in practice.

Theorem B.1. *For algorithm 4 with any choice of procedure* GETSAMPLINGDIST *and any online bandit algorithm* BASE, $\mathbb{E}R_T \leq \mathbb{E}R_T^{\text{BASE}} + \sum_{i=1}^{N} \Delta_i \mathbb{E}S_{i,T}$ *where* $S_{i,T}$ *is the number of elements pulled for arm* i *by time* T*, but not yet shown to* BASE.

The proof of Theorem B.1 is provided in appendix C. The proof sketch is that for each item, it has either been assigned to queue or it has been consumed by BASE. We know the regret of BASE on the samples it pulls from the queues is upper bounded by $\mathbb{E}R_T^{\text{BASE}}$, so combining that with the regret of the items in each queue gives us the stated regret.

Theorem B.2. *For SDB,* $\mathbb{E}R_T \leq \mathbb{E}R_T^{\text{BASE}} + N\tau_{max}$ *in the online updating setting, and* $\mathbb{E}R_T \leq \mathbb{E}R_T^{\text{BASE}} + 2N\tau_{max}$ *in the batch updating setting.*

The proof of Theorem B.2 is in appendix C. The proof proceeds by showing that the size of any queue cannot exceed τ_{max} given online updating, and $2\tau_{max}$ given batch updating . In the online update case the proof is simple because we always assign 0 probability to arms with more than B elements, and $B \leq \tau_{max}$ always. In the batch update case the proof

Algorithm 4 General Queue-Based Bandit Algorithm

Let $\alpha \in (0, 1]$ be a user-defined mixing weight
Create an empty FIFO queue $Q[i]$ for $i \in \{1, \ldots, N\}$
Initialize queue size counters $S_i = 0$ for $i \in \{1, \ldots, N\}$
Initialize $B = 1$; Initialize list L as an empty list
Let p be BASE's first arm-pull prob. distribution
Let h be HEURISTIC's first arm-pull prob. distribution
Draw $I \sim p$
for $t = 1$ to ∞ **do**
    while $Q[I]$ is not empty **do**
        Remove reward r from $Q[I]$, decrement S_I
        Update BASE with r and I
        Get BASE arm distribution p; Draw $I \sim p$
    $q =$ getSamplingDist(...)
    Sample from environment with distribution q
      (once if online updates or else for one batch)
    Increment S_i by the # of times arm i was sampled
    for each sample that has arrived **do**
        Observe reward r_i and add to $Q[i]$.
        Add the delay (number of timesteps since
          selected) of this sample to L
        Update HEURISTIC with reward r_i
        Get next HEURISTIC arm distribution h
    Set B to the maximum delay in L.

Algorithm 5 Stochastic Delayed Bandits (SDB)

uses getSamplingDistSDB in line 13 of Algorithm 4

1: **procedure** getSamplingDistSDB(h, p, I, S, B, α)
2: $\mathcal{A} = \{1, \ldots, N\}$ ▷ *$\mathcal{A}$ are the arms we're willing to alter*
3: $q = h$ ▷ *q will be a modification of h that avoids sampling from full queues.*
4: $q = (1 - \alpha)q + \alpha p$ ▷ *We start by mixing in a small amount of p to so that if the queues h wants to sample are full we are close to p.*
5: **for all** i **do**
6: $u_i = \max(\frac{B - S_i}{B}, 0)$ ▷ *u_i is the maximum allowable probability for q_i*
7: **while** $\exists i \in (\mathcal{A} - I)$ such that $q_i > u_i$ **do**
8: $d = q_i - u_i$ ▷ *d is the probability mass to redistribute to other arms*
9: $q_i = u_i; \mathcal{A} = \mathcal{A} - i$ ▷ *Set q_i to u_i and do not allow q_i to change further*
10: $sum = \sum_{j \in \mathcal{A}} q_j$
11: **for all** $j \in \mathcal{A}$ **do**
12: $q_j = q_j \times \frac{sum + d}{sum}$ ▷ *Redistribution of probability lost to unmodified arms*
13: **return** q
14: **end procedure**

proceeds similarly, except it can take up to one batch to notice that the queue has gotten too large and zero out the probabilities, so we incur an extra additive term of τ_{max}.

Heuristics When choosing a sampling distribution, SDB starts with a weighted combination of the distribution produced by the HEURISTIC and BASE algorithms, where $\alpha \in (0, 1]$ of the probability mass is put on BASE.[3] It is important to observe that SDB's regret bound (Theorem B.2) depends on neither α's value nor HEURISTIC's behavior.[4] This allows us to set α very low, thereby primarily trusting an arbitrary heuristic algorithm (at least initially), while achieving the best known theoretical guarantees in the online updating delay case, and within a constant factor in the batch updating case. So in the presence of delay, SDB allows arbitrary heuristics to help bandit algorithms get better reward in favorable cases, while avoiding poor worst-case behavior, as we shall see in the empirical results.

Incorporating Prior Data SDB can be used to handle access to prior data by initially placing the prior dataset in the queues and setting B to M, the maximum size of any queue. SDB ensures that the algorithm does not perform poorly when handed a (possibly adversarially) skewed distribution of arms, instead making sure that it remains close to its online performance. This is because if τ_{max} is taken to be the max of the true maximum delay and M, the same guarantees apply, since initially $S_i < \tau_{max}$ and $B < \tau_{max}$, therefore the induction in the proof of Theorem B.2 holds. In the experiments we will see that certain popular bandit algorithms can perform poorly when handed all prior data, motivating our approach.

5.4 Simulations

In this section we compare the simulated performance of our algorithms. To give a better picture of performance, simulations are averaged over many randomly-chosen but similar environments. The environments in Figures 5.1, 5.2, 5.4, and 5.5 consist of 10 Gaussian

[3] α cannot be 0, since empty queue $Q[I]$ might have zero probability, resulting in a division by 0 on line 12 of algorithm 5.

[4] This is because a bad heuristic will quickly fill up the queues for the bad arms, causing us to ignore it and instead follow the arms selected by BASE.

Figure 5.1: Comparing to QPM-D with Thompson Sampling as the black-box algorithm.

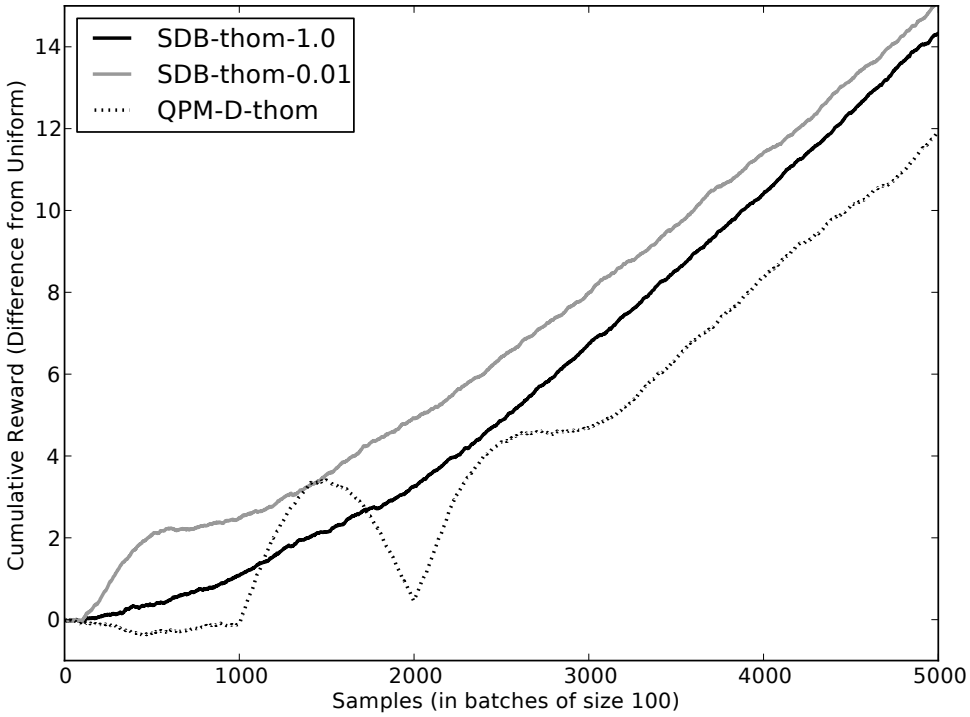


Figure 5.2: Comparing to heuristics with Thompson Sampling as the black-box algorithm.

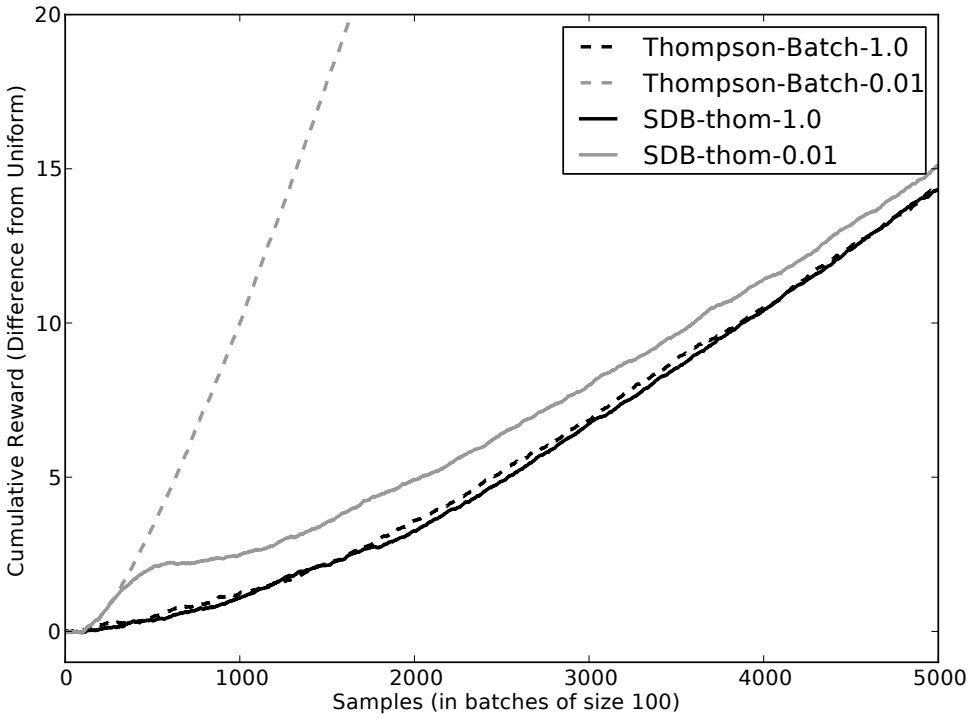


Figure 5.3: An example where the heuristic Thompson-Batch-0.01 performs worse.

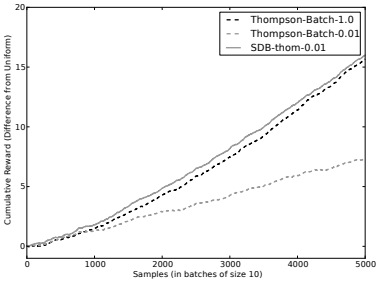


Figure 5.4: Comparing to QPM-D using UCB as the black-box algorithm.

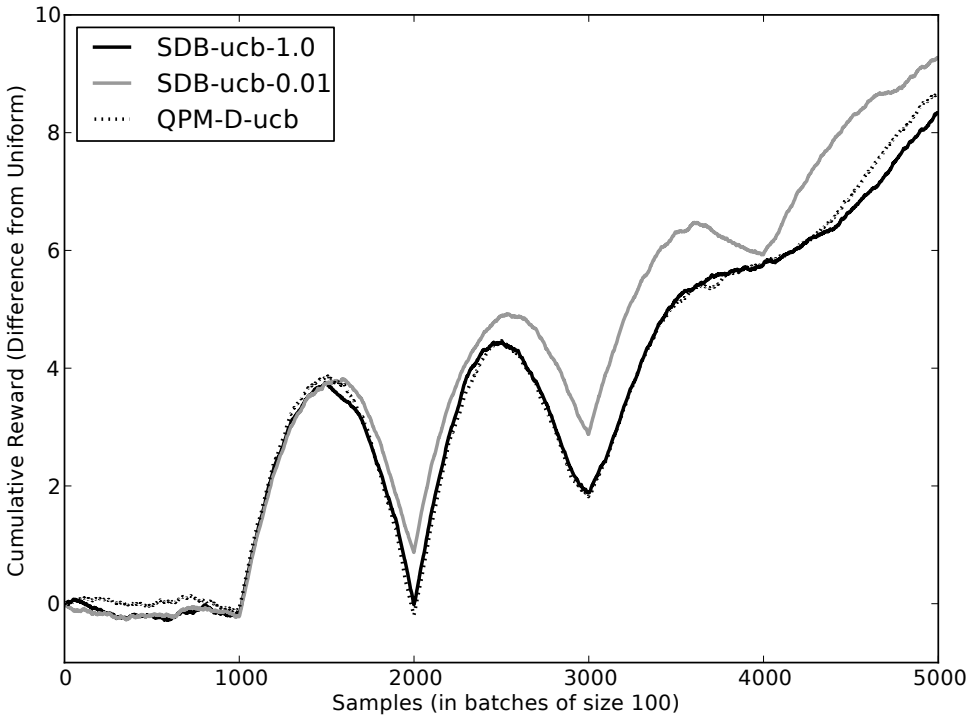


Figure 5.5: Two example algorithms that perform poorly when handed all samples but well inside SDB.

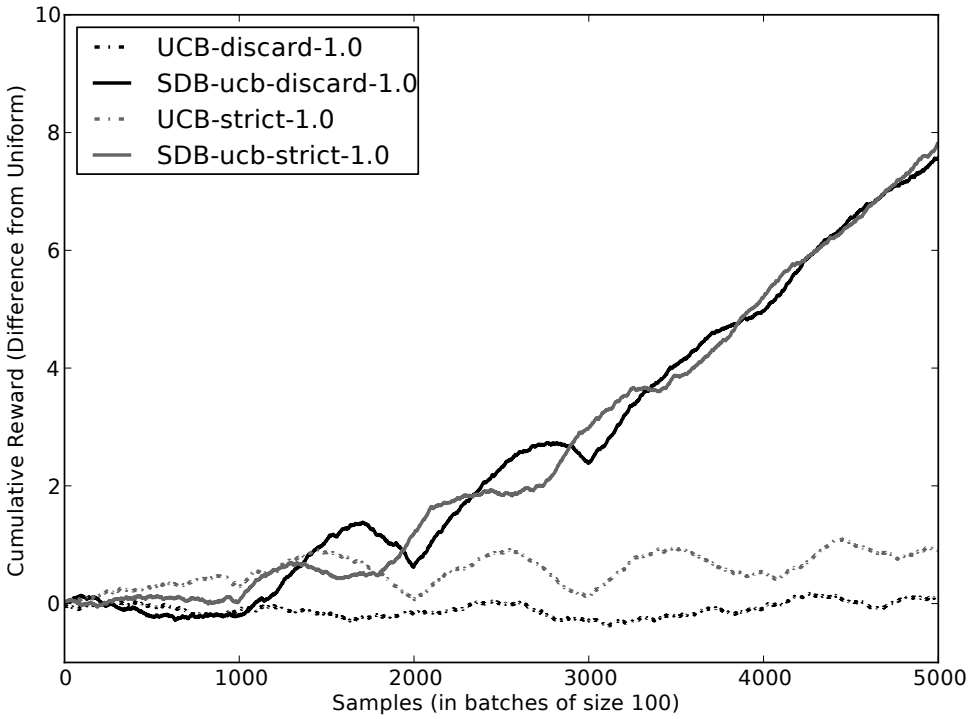


Figure 5.6: An example of a case where handing a prior dataset from a poor arm hurts Thompson-Batch but not SDB.

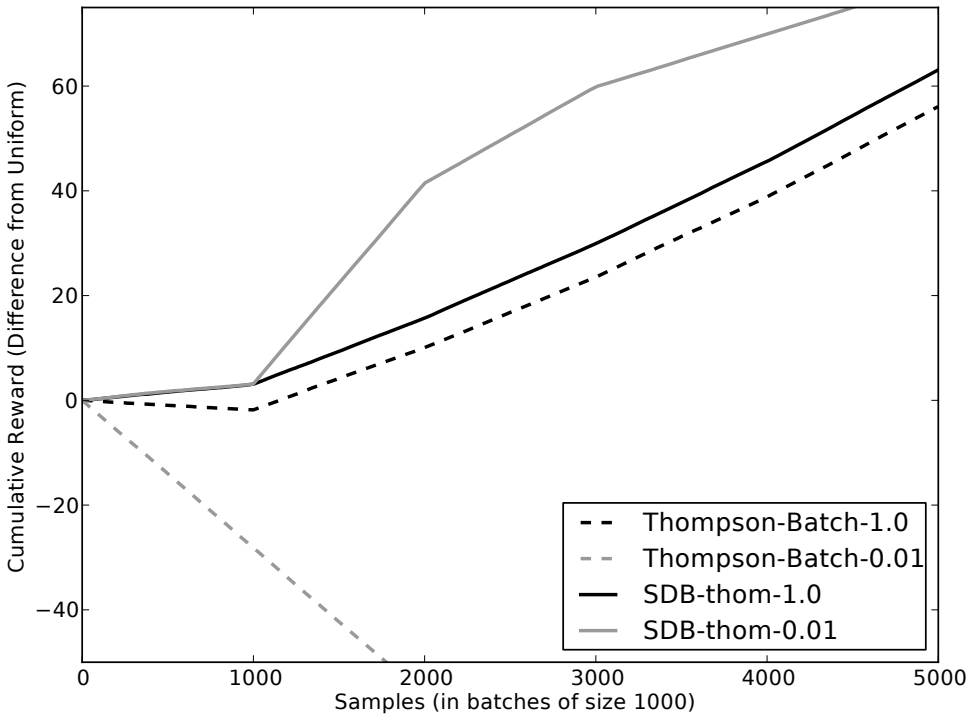


Figure 5.7: Simulation Results.

arms with means picked uniformly at random in the interval [0.475, 0.525] and variances picked in the interval [0, 0.05]. The environment in Figure 5.6 is also 10 Gaussian arms with means picked in the interval [0.9, 1.0] and variances in [0, 0.05]. In both cases the Gaussians were clamped to [0, 1] to make them fit the standard bandit formulation. Finally, Figure 5.3 has two Bernoulli arms, with means picked uniformly from [0.475, 0.525]. These environments were kept the same between candidate algorithms to reduce the variance. For each simulation, each candidate algorithm was run once on 1000 different environments, and subtracted from a uniform policy (to ensure an accurate baseline, the uniform policy was run 10 times in each of those 1000 environments).

Since SDB takes black-box algorithms as input, we have complete freedom in choosing bandit algorithms from the literature. We focus on Thompson Sampling [148], a Bayesian method that has gained enormous interest in the bandit community recently, since it typically outperforms most other algorithms in practice [29] while possessing optimal regret bounds [3] in the online case. Our rewards are non-binary and so we use the Gaussian-Gaussian variant presented in Agrawal et al. [3][5]. The distribution over arms is easy to sample from but challenging to compute exactly; thus we sample 100 times to construct an approximate distribution from which to sample. We also present some results using UCB [11], a popular bandit algorithm that also has optimal guarantees in the online case. The SDB parameter α which controls the weighting between Heuristic and Base is always set to 0.01 in our simulations and experiments.

Here we focus on the batch updating model of delay with batches of various sizes. For results in the online updating case (in which case our regret-bound exactly matches the best-known), see appendix C.

In the figures, SDB-thom-X refers to running SDB with Thompson-1.0 as the Base algorithm and Thompson-X as the Heuristic algorithm, and likewise for UCB.

Figure 5.1 shows a comparison of SDB to QPM-D using Thompson Sampling algorithms,

[5]Note that this does not mean we assume the arm reward distributions are Gaussian, Agrawal's method has optimal bounds given *any* distribution.

in environments with batches of size 100. The parameter in all of our graphs refers to variance scaling: 1.0 provides optimal online theoretical guarantees, while lower values cause Thompson Sampling to overexploit in the worst case. For SDB-thom-X, the BASE algorithm is always Thompson-1.0 to provide guarantees, while the HEURISTIC algorithm is Thompson-X. QPM-D also uses Thompson-1.0 as BASE. The results show that QPM-D does poorly initially, being equivalent to uniform for the first 1000 samples (since it devotes one batch to each of the 10 arms), and also performs poorly long-term. The strange shape of QPM-D is due to its determinism: when it samples a bad arm it must devote an entire batch to it, resulting in periods of poor performance. With an added heuristic, SDB uses it to substantially improve performance, especially early on, while retaining theoretical guarantees.

Figure 5.2 shows a comparison of SDB to heuristic approaches. Thompson-Batch-X refers to an algorithm which hands Thompson-X all samples after each batch instead of using queues (good regret bounds are not known for this approach). SDB-thom-1.0 and Thompson-Batch-1.0 perform similarly, which is encouraging since SDB has been shown to have strong theoretical guarantees while Thompson-Batch has not. SDB does look worse than Thompson-Batch-0.01 since it explores more to retain theoretical guarantees.

However, ignoring theoretical guarantees, why not just use a heuristic like the Thompson-Batch-0.01 algorithm, since it performs much better than the other algorithms in Figure 5.2? To explain why, see Figure 5.3. In this environment we see that Thompson-Batch-0.01 performs very poorly compared to Thompson-Batch-1.0, since the arm means are so hard to distinguish that it tends to exploit bad arms. However, SDB-0.01 performs as good or even a little better than Thompson-Batch-1.0, showing that it avoids being misled in environments where the heuristic is bad.

Figure 5.4 is similar to Figure 5.1 except we have used UCB instead of Thompson. Here we don't see a difference between SDB-ucb-1.0 and QPM-D, since there is no stochasticity in the black-box algorithms for SDB to exploit. However, SDB still sees improvement when heuristics are added, while retaining the theoretical guarantees.

Of course, UCB and Thompson are not the only reasonable bandit algorithms. In Figure

5.5 we have illustrated the benefit of SDB with two examples of bandit algorithms that perform very poorly if they are run with delay, but perform well under SDB. UCB-Strict acts similarly to UCB, but defaults to random performance if it observes a reward from an arm that does not have the highest upper confidence interval. UCB-Discard is also based on UCB but discards rewards it observes from arms that do not have the highest confidence interval (an alternative approach to ensuring batch and online performance are similar). As expected, handing all samples from a batch to these algorithms hurts performance, but running them inside SDB preserves good performance. Although many bandit algorithms do not exhibit this extreme behavior, these examples show that queues are needed to guarantee good performance for black-box algorithms.

Figure 5.6 shows an example of Thompson-Batch performing badly given prior data. In this case we supplied one batch (1000 samples) of prior data from the worst arm. Thompson-Batch puts slightly more weight on that arm initially, resulting in poor initial performance. The effect is exacerbated when decreasing the tuning parameter, as Thompson-Batch-0.01 focuses entirely on exploiting the poor arm, causing very poor performance. SDB, however, chooses not to sample that arm at all for the first batch since it already has enough data, resulting in better performance in this case. It also makes use of the heuristic to improve performance, avoiding the poor performance of the heuristic used alone.[6]

5.5 Experiments

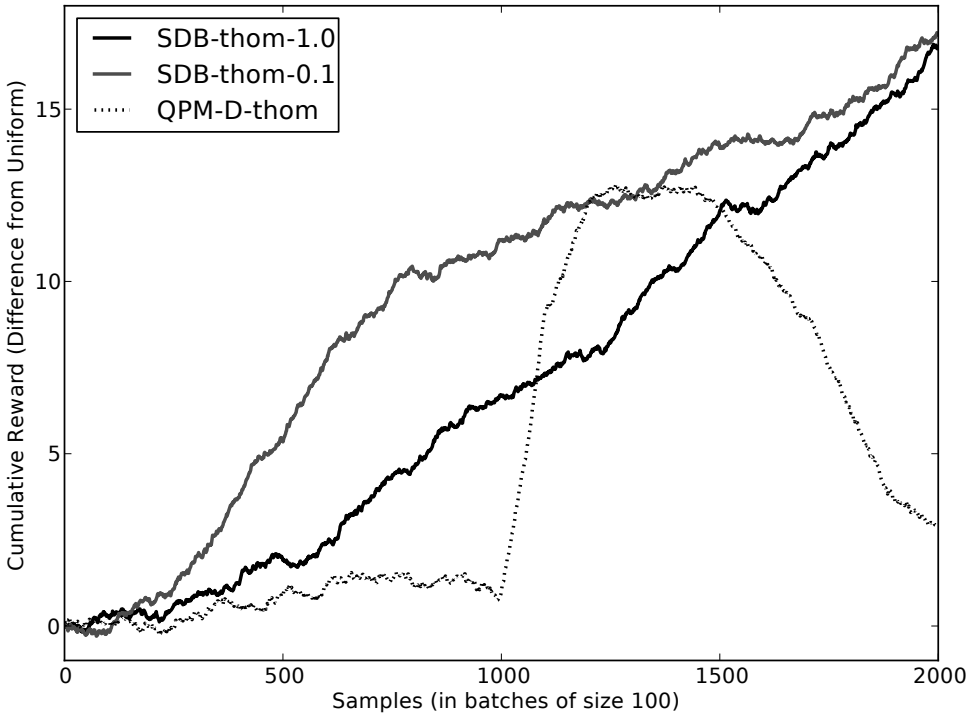
We now evaluate our algorithms on real world data drawn from Treefrog Treasure. The experimental setup is the same as in section 4.2.2.

To generate the cumulative reward graph we average each algorithm over 1000 samples from our SDB-based replayer (as in the simulations, Uniform received 10,000 samples). To ensure reliability, all results had at least 99.5% of trajectories reach the graphed length before

[6] Although SDB robustly handles a single batch of prior data, a very large prior dataset consisting of many batches might negatively impact both theoretical guarantees and performance. Addressing this issue is left for future work.

Figure 5.8: Treefrog Treasure: players guide a frog through a physics-based world, solving number line problems.

Figure 5.9: Results (using our queue-based estimator) on educational game data. SDB does better by leveraging the heuristic.



hitting an empty queue. We plot rewards in the original range, where 1 unit corresponds to 1 number line improvement on the posttest.

Figure 5.9 shows results on our data with batch size 100. We see that SDB exploits the heuristic to perform much better than simply using Thompson-1.0. The other algorithm which maintains strong theoretical guarantees, QPM-D, does much worse empirically. We are encouraged at the good early performance of SDB, which is critical in settings such as educational games where experiments may be terminated or players may leave if early performance is poor.

5.6 Related Work

Bandits with delay In this work we build off of Joulani's [75] approach to dealing with delay in stochastic multi-armed bandits, specifically their QPM-D algorithm. We improve over this work by (a) better exploiting stochastic algorithms to improve performance, as we show in our experiments, (b) explicitly considering the batch update setting, (c) incorporating an arbitrary heuristic to improve performance while retaining strong theoretical guarantees, (d) showing how queue-based methods can be used to incorporate arbitrary prior datasets and (e) showing how queue-based methods can be used to evaluate algorithms offline in an unbiased and highly data-efficient manner, without needing to know the original sampling distribution.

There have been prior approaches to handling delay in bandits prior to Joulani. Dudik et al. [49] assumed the delay was a fixed, known constant and online updating was allowed. Desautels et al. [42] analyzed the case of Gaussian Process Bandits, and developed an algorithm which relied on online updating with worse regret bounds than Joulani et al. [75]. Recent work in lock-in bandits [83] bears similarity to the batch update formulation but does not allow stochastic policies, and proves worse regret bounds than Joulani et al. [75].

Adding heuristics Although we are not aware of any work explicitly addressing balancing a heuristic and a principled algorithm in a bandit framework, this problem bears similarities to expert advice problems. While the traditional experts setting assumes full in-

formation (not bandit feedback) [96], the multi-armed bandits with expert advice problem is a better fit. However, as McMahan et al. [105] notes, the theoretical guarantees on solutions to this problem do not hold if the experts learn based on which arms we pull. McMahan explains that without restrictive assumptions on how the experts learn, achieving good regret in this case becomes an extremely challenging reinforcement learning problem. Our approach is considerably simpler, while retaining strong theoretical guarantees and achieving good empirical performance.

Evaluating bandits offline The standard approach to offline bandit evaluation is to build a simulator, either from scratch (e.g. [11]) or using collected data (e.g. section 4 of [29]). However, building a good simulator is challenging and comes without strong theoretical guarantees, sparking interest in data-driven estimators. Initial investigations ([87], [134]) focused on the problem of evaluating stationary policies, however we are interested in nonstationary policies that learn over time. Although there has been work in constructing biased evaluators of nonstationary policies that tend to perform well in practice ([111] and [47]), the state of the art in unbiased estimation (without some other good reward estimator) continues to be the rejection-sampling based replayer proposed by Li et al. in [92] and [47], which requires a known sampling distribution. We compare to this estimator and show that not only we can provide an unbiasedness guarantee without knowing the sampling distribution, but our evaluator is considerably more data-efficient.

5.7 Conclusion

In this chapter we identified the queue method, and showed how it can be used to solve three problems more effectively than prior solutions. Specifically, we presented Stochastic Delayed Bandits (SDB), an algorithm which leverages the distributions of black-box stochastic bandit algorithms to perform well in the face of delay. Although typically using an arbitrary heuristic worsens or eliminates theoretical guarantees, we showed how SDB can take advantage of a heuristic to improve performance while retaining a regret bound close to the best-known in the face of delay. We also showed how SDB is more robust to prior data than typical

approaches. These approaches have strong theoretical guarantees, and have good performance when evaluated both on synthetic simulations and on a real-world educational games dataset. Future work includes better methods for incorporating very large prior datasets, exploring how the queue method can be applied to more complex settings, and developing stochastic algorithms that match the best-known regret bound in the batch updating case.

Chapter 6

EFFICIENT EXPLORATION AND EFFECTIVE GENERALIZATION THROUGH BAYESIAN CLUSTERING

This chapter is based on work the author published at IJCAI 2016.

In the previous chapter, we studied efficient exploration in the bandit setting under delayed feedback. Here we look at how to learn quickly in the refeinforcement learning case, by carefully combining efficient exploration with efficient generalization.

6.1 Introduction: The Twin Problems of Exploration and Generalization

Developing agents that trade off exploration and exploitation while making decisions under uncertainty is a fundamental problem in AI, with applications in domains such as healthcare, robotics, and education. Typical solutions, based on optimism under uncertainty, treat under-sampled states and actions as high-value to encourage exploration. However, recent breakthroughs have come from observing that a non-optimistic Bayesian posterior sampling approach [148] can outperform optimistic methods empirically [29, 114], while achieving strong theoretical guarantees [3, 114].

Another fundamental problem is that of generalization. We typically know a ground state space, but learning over it directly can be slow due to data sparsity. In many cases, there are similarities between states, which can be exploited to speed learning. Much work seeks to find these similarities in order to aggregate the state space [104, 30, 95], and there is a rich literature on other feature-based generalization approaches such as deep neural networks [107].

These problems are usually solved separately. Either we explore/exploit without generalization (e.g. [114, 10]), or we learn to generalize while including some amount of random

exploration (e.g. [149, 104]). These areas are non-trivial to combine because acting with respect to a representation which is too compact may prevent us from collecting the kind of data necessary to refine it. To do both at once, we need some uncertainty over not just the transitions and rewards given the most likely representation, but also over the representation itself. Ideally, we could tackle exploration and generalization simultaneously in a way that guarantees asymptotically optimal performance, while also ensuring good performance given limited data. However, this is more challenging and has been less well-studied, especially if we desire powerful nonlinear generalization methods such as clustering.

A Bayesian approach to clustering state dynamics might be to use a prior that specifies states which are likely to share parameters, and sample from the resulting posterior to guide exploration. With limited data, this approach will prefer a smaller model and improve initial performance, but performance will continue to improve as more data leads us to consider dissimilar states to be distinct. However, sampling exactly from a clustering prior is intractable, as there are $O(n^n)$ interdependent clusterings to consider among n states. Therefore, past work has used Gibbs sampling, an MCMC technique, to sample approximately from a clustering prior [8]; unfortunately, these methods are computationally expensive and lack guarantees except in the limit of infinite runtime.

We propose Thompson Clustering for Reinforcement Learning (TCRL), a family of simple-to-understand Bayesian algorithms for reinforcement learning in discrete MDPs with a medium/small state space. TCRL carefully trades off exploration and exploitation using posterior sampling while simultaneously learning a clustering of the dynamics. Unlike MCMC approaches, both variants of TCRL are computationally efficient, run quickly in practice, and require no parameter tuning. TCRL-Theoretic achieves near-optimal Bayesian regret bounds, while improving empirically over a standard Bayesian exploration approach. And TCRL-Relaxed is guaranteed to converge to optimal behavior while empirically showing substantial improvement over state-of-the-art Bayesian state clustering algorithms across a variety of domains from the literature, even when no states are similar.

6.2 Setting and Background

We consider Markov decision problems, but depart slightly from the standard Markov Decision Process (MDP) formulation by defining the dynamics in terms of relative outcomes [89, 8]. Relative outcomes are useful in situations where a small number of events can summarize the dynamics, although they can encode any discrete MDP. Specifically, we assume a discrete state space $\mathcal{S}$ and an action set $\mathcal{A}$. In addition, one is given a set of relative outcomes $\mathcal{O}$ such that after taking an action $a \in \mathcal{A}$ from a state $s \in \mathcal{S}$ the agent observes an outcome $o \in \mathcal{O}$. The agent knows the reward function $R(s, a, o)$, which deterministically outputs a scalar value in $[r_{min}, r_{max}]$, and the transition function $T(s, a, o)$, which deterministically outputs a next state s'. However, it does not know the distribution over relative outcomes at each state. Experience comes in episodes of maximum length τ. The goal of the agent is to learn from experience to maximize the sum of rewards over time. For example, in an educational setting the state might be a partial history of problems and responses, an action might be to give a problem, and the outcomes indicate the student response, giving us reward and allowing us to determine how to update the state. In keeping with past posterior sampling work [114, 8], we focus on state spaces small enough to plan in exactly.

PSRL Posterior Sampling for Reinforcement Learning (PSRL) [114, 137] translates posterior sampling [148], a state-of-the-art exploration/exploitation method [29], to reinforcement learning over an MDP. Given a prior distribution over possible MDPs, PSRL samples an MDP from the posterior, solves the sampled MDP, and then runs that policy for an episode. Despite this very general formulation, the existing PSRL work in the non-factored MDP setting [114] learns the parameters of each state in $\mathcal{S}$ independently, due to the independent paramater priors used.

6.3 Related Work

Combining generalization with state-of-the-art exploration A small amount of past work has explored simultaneously generalizing and using state-of-the-art methods to con-

trol exploration in a way that provides guarantees. Typically such generalization has been guided by known properties of the environments, such as linear-quadratic systems [1] or factored MDPs with known structure [115]. We are not aware of another approach which combines powerful nonlinear generalization with state-of-the-art exploration in a way that is guaranteed to both run efficiently and converge to optimal behavior.

General State Aggregation Generalization in reinforcement learning is widely studied. Due to space, we discuss the most related work, state aggregation. Typically, these approaches focus on aggregating states instead of state-action pairs, aside from some recent work on aggregating state-action pairs given a fully-specified model [5], and work considering homomorphisms among actions when clustering at the state level [141]. State aggregation work often lacks formal guarantees [95, 149, 132]. Limited work has focused on careful exploration while clustering, either in deterministic systems [150], when selecting among a small number of models/aggregations [113] or when clustering is used solely to decrease computation [112].

Further, we focus on clustering dynamics, not states: clustered states share similar relative outcomes, but may have different values and transitions. Thus the focus here is not on speeding planning but on more efficient learning. Related work clustering outcome dynamics either assumes a known clustering [89, 25] or makes additional assumptions, e.g. that states can be clustered in a very small number of ways [43].

Bayesian State Aggregation iPOMDP [46] is a Bayesian method to learn in a POMDP environment while growing the state space. iPOMDP lacks guarantees when run for a finite time, is quite computationally expensive, and it is unclear how to leverage a known MDP state space in iPOMDP.

Bayesian RL Work in Bayesian reinforcement learning (e.g. [64, 155]) provides methods to optimally explore while learning an optimal policy. However, these approaches are typically computationally intractable, and are based on maximizing discounted returns across episodes which can lead to incomplete learning [125], in contrast to our approach which is guaranteed to converge to optimal behavior.

BOSS The most related work is Best of Sampled Set (BOSS) [8] which uses posterior sampling (with added optimism), and an (optional) clustering prior. Its theoretical guarantees are contingent upon drawing exact samples from the posterior, but empirically BOSS only draws approximate samples when using a clustering prior. BOSS clusters dynamics on a state-level, in contrast to our more flexible state-action clustering method. We experimentally compare to an enhanced version of their MCMC-based clustering method.

6.4 TCRL

Although PSRL (see Section 6.2) has good theoretical and empirical performance, Osband et al. [114] uses priors which treat the parameters of each state as independent (unless factored structure is known [115]). This leaves room for improvement by considering richer priors, such as assuming (some) nearby states share similar relative outcomes. For example, honking a horn on a robotic car likely has the same effect in most places, while the effects of turning are highly state-dependent. A good clustering of dynamics across state-action pairs would help us require less data about honking while preserving the distinction between turning in different locations.

A principled way to approach this problem is to formalize the intuition that many state-action pairs are similar as a clustering prior, and use it in the PSRL framework [114]. Ideally we could sample exactly from this clustering prior; however, this is known to be intractable. Past work [8] used MCMC approaches to draw an approximate sample given such a prior, but this approach is computationally expensive, sensitive to initialization, and lacks guarantees.

Here we present Thompson Clustering for Reinforcement Learning (TCRL), a family of approaches each of which leverage the structure of the state space[1] to efficiently cluster while simultaneously retaining good performance without a need for parameter tuning. The key idea of our TCRL approaches is that they prefer to cluster states that are nearby in the

[1]By structure, we refer to properties that can be deduced without any data, specifically the topology of the space as indicated by the transitions and rewards possible due to the relative outcomes, as well as the location of the start state.

original state space. Although it may be possible to construct examples where this causes TCRL to underperform, we believe this is a good fit for many real-world domains. For example, in e-commerce, a user's preferences are unlikely to drastically change after a single advertisement. Or in education, the state of a student is unlikely to change much after a single problem.

We introduce two specific algorithms which vary in the details of how this structure is used: TCRL-Relaxed and TCRL-Theoretic. However, both of these approaches cluster states separately for each action, ensuring a more flexible representation than an typical state-clustering approach.

6.4.1 Preliminaries: Choosing among a small number of clusterings

Let us first examine the simplified case of clustering the dynamics of two states, A and B, given some fixed action a. This will form a key building block for our approaches. One can define a prior probability $P(C)$, where C is the event that the two states are clustered (i.e. share identical dynamics given a). Given a set of data D (consisting of (s, a, o) tuples where $s = A$ or $s = B$), we then need to compute $P(D|C)$ and $P(D|\neg C)$. Once these are computed we can compute the posterior probability that these states are clustered[2] :

$$P(C|D) = \frac{P(D|C)P(C)}{P(D|C)P(C) + P(D|\neg C)P(\neg C)}. \tag{6.1}$$

Associated with the pair (s, a) is a continuous parameter vector $\vec{\theta}$, where the i^{th} component θ_i denotes the probability of generating the i^{th} relative outcome given s and a. Then $P(D|C)$ can be computed as $P(D|C) = \int P(D|\vec{\theta}, C)P(\vec{\theta}|C)d\vec{\theta}$ and similarly for $P(D|\neg C)$. Since $\vec{\theta}$ are the parameters of a categorical distribution over relative outcomes, the conjugate prior is a Dirichlet, and a closed-form solution to this integral for Dirichlet distributions is well-known. For N observations, the Dirichlet has parameters $\alpha_1, \ldots, \alpha_N$, and the integral

[2]Note that this is just an application of Bayes' Rule: We compute $P(D, C)$ and $P(D, \neg C)$ and then normalize. This approach bears a close connection to the Bayes Factor (and likelihood ratio) tests for comparing two competing hypotheses.

is $P(D|\alpha_1, \ldots \alpha_N) = \int P(D|\vec{\theta})P(\vec{\theta}|\alpha_1, \ldots \alpha_N)d\vec{\theta}$ which is equal to:

$$\frac{\Gamma(\sum_i \alpha_i)}{\Gamma(\sum_i n_i + \alpha_i)} \prod_{i=1}^{N} \frac{\Gamma(n_i + \alpha_i)}{\Gamma(\alpha_i)}, \tag{6.2}$$

where n_i is the number of occurrences of the i^{th} outcome in the dataset.

To compute $P(D|C)$, we first sum up the counts of the data over both A and B, that is let $n_i = n_{i,A} + n_{i,B}$, and then compute the probability of the data as in (6.2). To compute $P(D|\neg C)$, since states A and B are separate, their respective datasets D_A and D_B are conditionally independent given $\neg C$, so we can simply multiply the likelihoods. $P(D_A, D_B|\neg C) = P(D_A|\neg C)P(D_B|\neg C)$, so:

$$P(D_A, D_B|\neg C) = P(D_A|\vec{\alpha_A})P(D_B|\vec{\alpha_B}). \tag{6.3}$$

So, if we are given two states, we can sample from a simple clustering posterior exactly in constant time. Similarly, if we have a very small number of different clusterings we wish to choose between (e.g. all clustered, predefined half clustered, none clustered), we simply calculate the data likelihood for each cluster using equation (6.2) and multiply across clusters as in equation (6.3). However, so far we have not addressed how to scale up in order to perform clustering over a full state space. TCRL-Theoretic and TCRL-Relaxed offer two different solutions to this problem.

6.4.2 TCRL-Theoretic

There are $O(n^n)$ interdependent clusterings to consider among n states, so, if we wish to generate an exact sample from a posterior *efficiently*, we must reduce the clustering space. TCRL-Theoretic utilizes the structure of the state space to reduce the space of clusterings in a way that (as we will show experimentally) is sufficient to enable improvements in learning, while retaining near-optimal theoretical guarantees on the expected Bayesian regret. The general approach to reducing the space is to partition the states into independent groups, and then within each group propose a small number of clusterings.

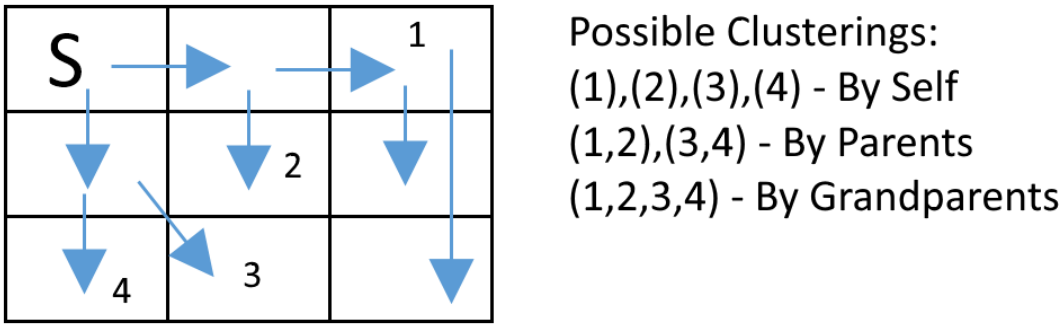


Figure 6.1: The balanced tree TCRL-Theoretic constructs in this gridworld if east children are processed before south children. S is the start state; the arrows go from parents to children in the tree. After constructing the tree, we cluster. Clustering options are shown for the 2nd layer of the tree.

To automatically construct this breakdown a priori, we rely on the structure of the MDP state space. Specifically, our algorithm creates a tree where the nodes are states and the root is the provided start state. The goal is to ensure the tree is fairly balanced, but at the same time ensure that states which are closely connected in the original MDP are closely connected in the tree. To achieve this we traverse the state-space starting from the start state, in a breadth-first fashion, where the traversal follows the transitions that are a priori possible due to the set of possible outcomes. However, since we wish the tree to be roughly balanced and binary, the procedure is complicated by the fact that each state may not have exactly two children. Therefore we use a subroutine getTwoChildren(p) which initiates another breadth-first traversal starting at state p looking for two unseen children to add as children of p. One complication here is that, since transitions may be one-directional, we may orphan nodes. To avoid this we first make sure to include likely orphans as children (immediate children of p without a link back to p). Any remaining orphans are added in a postprocessing step. See Figure 6.1 for an example, and algorithm 9 in section 6.8 for pseudocode detailing how the tree is constructed.

Given a tree, a natural way to reduce the number of clusterings is to cluster only within

each depth. However, further reduction is needed as each depth is still too large to allow us to consider all possible clusterings within it. Given that the tree structure tends to put nodes which are closer in the original state space graph closer in the tree (for example see Figure 6.1), it generally seems reasonable to say that states which have the same parent are more likely to cluster than nodes which only share the same grandparent, etc. So for each depth d we consider only $O(\log n)$ clusterings: clustered by depth d (unclustered), clustered by depth $d-1$ (states with same parents clustered together), clustered by depth $d-2$ (states with same grandparents clustered together),... clustered by depth 0 (everything clustered), which allows us to naturally interpolate between different levels of clustering at each depth. See Figure 6.1 for an example. For each depth d we use a prior on this clustering scheme which puts 0.5 probability on unclustering and $\frac{0.5}{d}$ on clustering by the other depths. The reason for putting more probability mass on the unclustered hypothesis is to reduce the risk of clustering too aggressively.

Next we calculate the probability of each of the clusterings at each depth using Bayes rule, as explained in Section 6.4.1. After building the tree and sampling a clustering, we sample the parameters of the MDP and solve it to produce the policy to run during the next episode. For details see Algorithm 6.

The runtime of the clustering step is $\mathrm{O}(\log^2 n)$, allowing the procedure to remain highly efficient and scalable. Note that a large part of the benefit is due to the independence among clustering decisions between depths: the total number of possible clusterings is $O((\log n)^{\log n})$, so this does not reduce to making the total number of clusterings very small, unlike some previous work [43, 153, 113].

Note that TCRL-Theoretic samples exactly from the posterior at each step. This allows us to bound the expected *regret* of TCRL-Theoretic. We define regret as in Osband et al. [114]: $R(T) = \sum_{e=1}^{\lceil T/\tau \rceil} V^* - V_{\pi_e}$, where V^* is the optimal expected per-episode reward, and V_{π_e} is the expected reward of running the policy generated at episode e, π_e, for one episode. Now we present the following theorem.

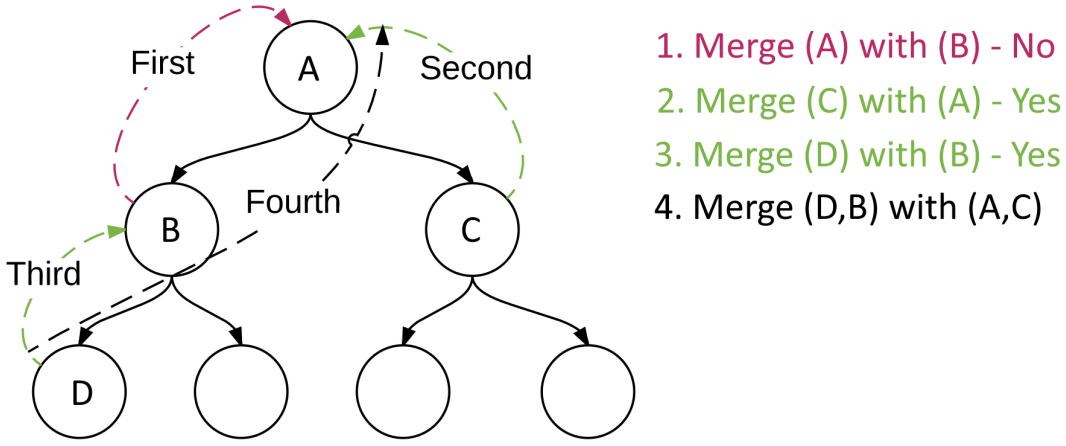


Figure 6.2: The first 4 clustering decisions that could be made by TCRL-Relaxed on an example DAG. In this run TCRL-Relaxed chose not to cluster first pair of states.

Theorem 6.4.1. *The Bayesian regret of TCRL-Theoretic is at most:*

$$O((r_{max} - r_{min})\tau|\mathcal{S}|\sqrt{|\mathcal{A}|T\log(|\mathcal{S}||\mathcal{A}|T)}).$$

Theorem 6.4.1 does not require a separate proof as it follows directly from the guarantees of PSRL [114], the only difference being scaling to $[r_{min}, r_{max}]$ due to the rewards not necessarily being in $[0, 1]$. Note that this is a Bayesian bound, which means that it applies assuming MDPs are truly drawn from the (in our case clustering) prior distribution; however, there is a close connection between frequentist and Bayesian regret [114].

6.4.3 TCRL-Relaxed

In order to sample exactly from the posterior (and thus retain the regret bounds of PSRL) while remaining efficient, TCRL-Theoretic restricts the set of clusterings severely, by only considering clusterings within each depth. A richer set of possible clusterings would likely improve performance if we could sample over them efficiently. TCRL-Relaxed is an approximate posterior sampling method that lacks regret guarantees, but it allows more powerful

clusterings, remains computationally efficient, and is guaranteed to converge to the optimal policy in the true (unclustered) MDP.

TCRL-Relaxed uses a semi-greedy agglomerative clustering approach. For each action it iterates through pairs of states, clustering them based on the procedure for clustering two states (Section 6.4.1). Note that, in contrast to TCRL-Theoretic, this is an approximation because the clustering decisions of future states are dependent on how we clustered past states, but we do not integrate over future clustering decisions when calculating the probability of clustering a pair.

In order to define the sequence of pairwise clustering decisions, we exploit structure in the state space to build a directed acyclic graph (DAG). To construct this DAG, we traverse the state space in a breadth-first fashion, ignoring states we have seen before, following edges based on the possible transitions (as defined by relative outcomes) from any given state. All parents of a state s which we reach before s are considered its parents in the DAG (and therefore, their ancestors are also the ancestors of s). Note that unlike TCRL-Theoretic, we do not attempt to balance the DAG. See algorithm 8 in section 6.8 for full details about how the DAG was constructed.

For each node, we walk through its ancestors in the DAG, testing whether the node (and all others clustered with it so far) can be added to the cluster of the next ancestor. See Figure 6.2 for an example, and Algorithm 7 for the pseudocode. Although clustering only with ancestors has its limitations (siblings cannot be directly clustered, although they can indirectly cluster through a common parent), ancestors are likely to have much more data than siblings as they are closer to the start state, so the benefit of clustering with them is larger. Further, clustering only with ancestors can often greatly increase computational efficiency (from $\mathrm{O}(n^2)$ to $\mathrm{O}(n \log n)$ if the state space is naturally balanced).

Theorem 6.4.2. *TCRL-Relaxed converges to optimal behavior in the limit of infinite experience.*

Proof. Recall we are in the finite-horizon setting, where experience comes in episodes of

maximum length τ. First, we show that if, under an optimal policy π^* in the true MDP, state-action pair (s,a) could be visited, it will be visited an infinite number of times by TCRL-Relaxed. Consider the decision of whether to cluster state A with a cluster C. Consider the case where C is composed of one or more states that share true parameters θ_C, but these are different from A's true parameters θ_A. Observe that when making this decision, the probability of choosing not to cluster is always non-zero and goes to one as more data is collected from A and C due to the consistency of the Bayesian hypothesis test. Since this probability is nonzero and approaches one for every A and C, we must sample a *valid* clustering (that is, one where no states with truly different parameters are clustered) an infinite number of times. As in PSRL, once we have a clustering we sample a policy according to the probability it is optimal given that clustering. Since, given a valid clustering, the probability of π^* being optimal is always non-zero and should converge to some non-zero value (1 if it is unique, $1/q$ if there are q optimal policies), then across an infinite number of rounds, π^* will be sampled an infinite number of times. So any (s,a) occurring in π^* will be sampled an infinite number of times.

Next we show that the distribution over outcomes at *any* (s,a) pair in the sampled MDP will either converge to the true distribution or be sampled a finite number of times. The only way that the outcome distribution of (s,a) does not converge to its true value in the limit of infinite samples is if it is clustered with at least one pair (s',a) with different parameters. However, if (s',a) is also sampled an infinite number of times, the Bayesian hypothesis test will uncluster the two states with probability approaching 1. And if (s',a) is sampled only a finite number of times, the samples will be overwhelmed by the samples from (s,a) and thus still converge to the true values.

At each episode we select one of a finite number of possible deterministic policies to deploy. So for a contradiction, assume that in the limit of infinite data, with probability $\epsilon > 0$ we choose a policy $\hat{\pi}$ with expected reward worse than π^*. In this case, we know that since $\hat{\pi}$ is sampled an infinite number of times, the outcome distributions for the (s,a) pairs that could be visited under $\hat{\pi}$ will converge to the true distributions. Therefore, the value of

$\hat{\pi}$ in the sampled MDP will converge to its true value. However, as we showed above, any (s,a) pairs that appear in π^* will also be sampled an infinite number of times, so the value of π^* will converge to its true value in the sampled MDP. Since the value of π^* is by definition greater than $\hat{\pi}$, and the planning step picks the optimal policy in the sampled MDP, we will converge to putting zero probability on $\hat{\pi}$, a contradiction. □

6.5 Simulation Results

Here we test performance on several MDP environments, averaging results over 100 runs unless otherwise indicated.

Baselines First, to evaluate the usefulness of clustering, we compare against a straightforward tabular approach which does not generalize between states: PSRL with independent parameter priors. Second, we compare TCRL to a powerful MCMC clustering method similar to that used in BOSS [8]. Specifically, we take the Chinese restaurant process (CRP) prior used as the BOSS cluster prior [8], and improve it by clustering states separately for each action, so that this approach, like TCRL, performs clustering of state-action pairs. Given this prior and an action, a typical Gibbs sampling approach is used following Asmuth et al. [8], where we repeatedly iterate through states, sampling from the conditional distribution over clusters for the current state given that the clustering of the other states is fixed. As with the TCRL approaches, after we sample a clustering we sample the parameters from each state to get an MDP, and then solve the MDP to obtain the policy to use for the next episode.

This baseline MCMC clustering method has several parameters; unfortunately, it is not always clear how to set them. First α, the CRP concentration parameter must be set. We choose the value of 0.5 recommended[3] by Asmuth et al. [8]. Second, since MCMC has no clear termination criterion, we must choose how long to run it. Since fast runtime is often very important, we control for time by, at each iteration, first running TCRL-Relaxed, and

[3]0.5 was recommended for Chain (similar to Riverswim). We found 0.5 to work better in MarbleMaze than the recommended value of 10.0, and to avoid tuning α, we fix $\alpha = 0.5$.

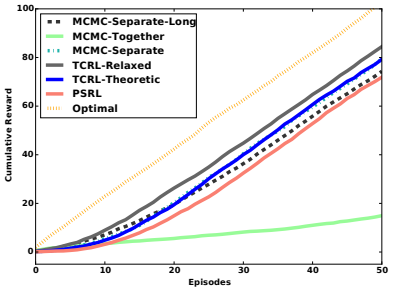


Figure 6.3: RiverSwim Results.

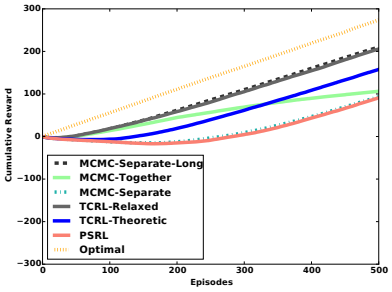


Figure 6.4: Marble Maze Results.

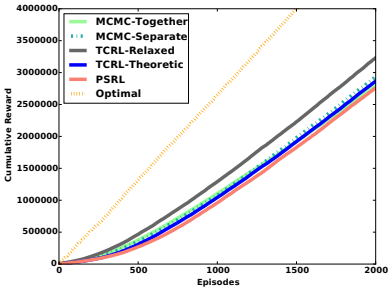


Figure 6.5: SixArms Results.

then running MCMC for the amount of time TCRL-Relaxed took. Note that controlling for time is always a somewhat inexact science (we are not claiming to have fully optimized either MCMC or TCRL methods). For environments similar to those studied by Asmuth et al. [8], we also compare running MCMC for the fixed number of iterations (i.e. sweeps across all variables) recommended[4] by Asmuth et al. [8]. Finally, one must set the initialization for MCMC. We compare two logical alternatives: Either all states are initialized in a single cluster (MCMC-Together), or all states are initialized in separate clusters (MCMC-Separate). We label the fixed iterations variant MCMC-Separate-Long.

Riverswim RiverSwim [136] is a 6-state chain MDP previously used to showcase PSRL [114]. For a diagram and complete description of the environment, see [114]. We used a horizon of 20 and defined 5 relative outcomes for moving left and right or staying with some reward. The results in this environment (Figure 6.3) show that MCMC-Together performs poorly, being unable to uncluster sufficiently given limited time. Interestingly, despite being given more runtime, MCMC-Separate-Long performs worse than MCMC-Separate, perhaps because the short runtime means MCMC-Separate is more likely to avoid over-clustering. TCRL-Theoretic matches the performance of the best MCMC approach, while

[4]We used 500 iterations in Riverswim (recommended for Chain, a similar environment), and 100 as recommended for Marble Maze.

Method	RiverSwim	MarbleMaze
MCMC-Separate-Long	290.38 ms	1253.47 ms
TCRL-Relaxed	0.47 ms	4.72 ms
TCRL-Theoretic	0.23 ms	1.28 ms

Table 6.1: Average time to sample a single clustering on RiverSwim and MarbleMaze

TCRL-Relaxed performs uniformly better than any MCMC approach. In Table 6.1, we found the TCRL approaches to be over 600x faster in this environment than a typical MCMC approach.

MarbleMaze MarbleMaze [8, 122] is a 36-state gridworld MDP previously used to showcase the benefit of MCMC clustering in BOSS [8]. For a diagram and complete description of the environment, see [8]. We used a horizon of 30 and a set of 5 outcomes denoting whether the agent moved in each cardinal direction or hit a wall (in keeping with past work [8], the coordinates of the goal and pits are assumed to be known). In Figure 6.4 we see that TCRL-Relaxed outperforms PSRL and the time-limited MCMC variants by a large margin, while MCMC-Separate-Long appears to slightly outperform it. However, in Table 6.1 we see in our experiment that variant took over 200x times longer than TCRL-Relaxed. By the end of 1000 episodes, TCRL-Theoretic also outperforms both time-limited MCMC variants. Interestingly, we see that MCMC-Together starts off much better than MCMC-Separate (and PSRL), but over time begins to flatten off, and by 500 episodes MCMC-Separate and PSRL are poised to overtake it. This is likely because aggressive clustering boosts early performance, but in order to reach optimal performance more and more distinctions are needed, which MCMC-Together is unable to make given limited time. On the other hand, MCMC-Separate does not appear to improve much over PSRL, likely because it has insufficient time to identify useful clusterings.

SixArms Six Arms [136] is a 7-state MDP arranged in a spoke pattern, where each of

the six outer states has very different dynamics/rewards, which presents a challenge for an approach based on clustering. For a diagram and environment description, see [136]. We used a horizon of 10 and 14 relative observations: moving to each outer state, staying with different rewards, and moving to the inner state. As expected, in Figure 6.5 we see that TCRL-Theoretic and the MCMC approaches struggle to improve over PSRL. However, TCRL-Relaxed shows a major improvement over PSRL and both time-limited MCMC variants. The fact that a clustering approach is able to improve over PSRL in the setting illustrates the potential benefit of state-action clustering: even though no states are similar in this domain, many (s,a) pairs are. We also observed that TCRL-Relaxed frequently clusters (s,a) pairs that have different dynamics if they are in low-value regions of the state space where we have less data and distinguishing dynamics is less important, allowing it to focus on improving performance in the higher-value areas.

200-state environment Next we examined a larger 200-state gridworld featuring one-dimensional walls [73], with an added start state in the bottom-right corner. For a diagram see [73]. The setup is similar to MarbleMaze, except that we chose the reward to be 100 for reaching the goal and -1 for each step taken, and since the problem was harder we used a longer horizon of 50 and averaged over 20 (instead of 100) runs. In Figure 6.6, the approaches which can aggressively cluster (TCRL-Relaxed, MCMC-Together, and TCRL-Theoretic) greatly outperform those that do little or no clustering (PSRL and MCMC-Separate), demonstrating the importance of generalization in larger environments. Here MCMC-Together performs similarly to TCRL-Relaxed, probably because most states share identical dynamics in this mostly empty gridworld and thus a highly nuanced clustering is not required.

6.6 Discussion

Overall, TCRL-Relaxed leverages the structure of the state space along with its pairwise Bayesian clustering to outperform PSRL and either choice of time-limited MCMC initialization, with the exception of the 200-state grid where it matched the best MCMC variant.

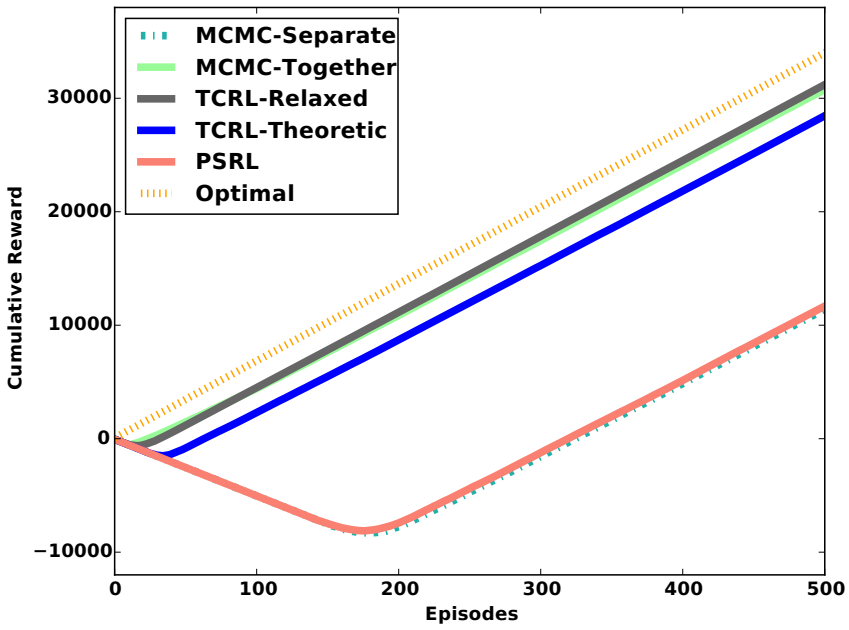


Figure 6.6: Results in the 200-state environment.

TCRL-Theoretic, although worse in performance than TCRL-Relaxed, always improves over PSRL, typically by a substantial amount, and has stronger guarantees. In contrast to TCRL which requires no parameter tuning, MCMC is extremely sensitive to initialization when time-limited. In the 200-state grid, initializing together does well and initializing separately does very poorly, but the opposite is true in RiverSwim. Using a large number of MCMC iterations (as specified by [8]) can improve performance, but in our experiments doing this takes 200x-600x longer than TCRL-Relaxed. Further, our MarbleMaze results show variants of MCMC can level off before reaching optimal performance, in contrast to TCRL. Finally, we find it encouraging that in SixArms, where we expected clustering to have little benefit, TCRL-Relaxed is able to improve.

6.7 Conclusion

In this chapter we presented TCRL-Theoretic and TCRL-Relaxed, two Bayesian approaches which allow us to generalize across states via clustering the dynamics while carefully controlling exploration. Unlike typical MCMC approaches, our techniques have no parameters to tune, and exploit the structure of the state space to remain computationally efficient. Further, TCRL-Relaxed is guaranteed to achieve optimal performance, while TCRL-Theoretic has stronger guarantees of achieving near-optimal Bayesian regret. We showed in simulation that our approaches improve over PSRL, with TCRL-Relaxed typically also outperforming both time-limited MCMC approaches by a substantial margin. One limitation we hope to address in future work is that TCRL struggles to scale to very large state spaces, in part due to the complexity of the required offline planning step. Approximate or online planning may be helpful here. Also, it would be interesting to investigate larger or continuous outcome spaces.

Algorithm 6 TCRL-Theoretic

Input: MDP with unknown dynamics, initial state I
tree = buildBalancedTree(I)
for $e = 1$ to ∞ **do**
    for $a \in \mathcal{A}$ **do**
        C_a = doCluster(tree)
        Sample parameters θ_{ac} for each cluster $c \in C_a$
    Create MDP M, using parameters θ_{ac}
    Solve M to get an optimal policy π
    Run π for an episode and update posterior
procedure doCluster(tree)
    clustersF = {}
    for depth = 1 to tree.maxDepth **do**
        probs = {}, clusters = {}
        depthNodes = tree.getStatesAtDepth(depth)
        for d = 0 to depth **do**
            prior = 0.5/depth
            if d = depth **then** prior = 0.5
            clusters[d]={}
            for s in depthNodes **do**
                ancestor = tree.getAncestorAtDepth(s, d)
                clusters[d][ancestor].add(s)
            probs[d] = prior $*$ $\prod_{a \in \text{clusters[d].keys()}}$ a.logLikelihood(data)
            (likelihood computed per (6.2) with $\forall i, \alpha_i = 1$)
        dF $\sim$ normalize(probs)
        clustersF.add(clusters[dF])
    return clustersF
end procedure

Algorithm 7 TCRL-Relaxed

Input: MDP with unknown dynamics, initial states I
for $e = 1$ to ∞ **do**
    for $a \in \mathcal{A}$ **do**
        C_a = doCluster(I)
        Sample parameters θ_{ac} for each cluster $c \in C_a$
    Create MDP M, using parameters θ_{ac}
    Solve M to get an optimal policy π
    Run π for an episode and update posterior
procedure doCluster(I)
    dag = BuildDAG(I), clusters = {}
    for s ∈ dag.TraverseBreadthFirst() **do**
        clusters[s] = [s], aClusters = []
        for anc ∈ dag.getAncestors(s) **do**
            aClusters.union(clusters[anc])
        for ac ∈ aClusters **do**
            if sampleMerge(ac, clusters[s]) **then**
                ac.add(clusters[s])
                for s' in clusters[s] **do**:
                    clusters[s'] = ac
    return clusters
end procedure
procedure sampleMerge(s1,s2)
    Compute posterior probability using data based on eqns (6.2) and (6.1), using a prior of 0.5 and $\forall i, \alpha_i = 1$
    Return true with posterior probability, false otherwise
end procedure

6.8 Detailed Algorithms

Algorithm 8 Subroutine for TCRL-Relaxed

```
procedure BUILDDAG(I)
    fringe = [I], seen = [], ancestors = {}, clusters = {}
    while fringe is not empty do
        s = fringe.pop()
        if s ∈ seen then continue
        seen.add(s);
        for child ∈ s.children() do▷ s.children() returns all children reachable after taking some action and observation.
            if child ∈ seen then continue
            ancChild = ancestors[s].add(s)
            if child ∈ fringe then
                ancestors[child].add(ancChild)
            else
                fringe.add(child)
                ancestors[child] = ancChild
end procedure
```

Algorithm 9 Subroutines for TCRL-Theoretic

```
procedure BUILDTREE(I)
    tree.setRoot(I), fringe = [I], seen = [], backptrs = []
    while fringe is not empty do
        s = fringe.removeFirst()
        if seen.contains(s) then continue else seen.add(s)
        children = GETATLEASTTWOCHILDREN(s, seen, fringe)
        tree.setChildren(s, children), fringe.add(children)
    while some state not in tree do
        for state in tree do
            if state.children() not all in tree then
                tree.UnionChildren(state, state.children())
    return tree
end procedure
```

Chapter 7

WHERE TO ADD ACTIONS IN HUMAN-IN-THE-LOOP REINFORCEMENT LEARNING

Sections 7.1-7.5 are based on work the author published at AAAI 2017.

7.1 Introduction: The Importance of Actively Incorporating Human Effort

Consider building a system to decide which hint to give a student playing an educational game, what to say to a user of a spoken dialog system, or what pictorial ad to show to generate the most revenue for a small business. In domains such as these, the space of all possible actions (pieces of text, soundwaves, or images) is far too large to explore from scratch without unreasonable quantities of data. A typical reinforcement learning (RL) approach is for human practitioners to first use their intuition and creativity to create a small set of discrete actions, and then use reinforcement learning techniques to learn to behave near-optimally within this more tractable space.

However, this initial action set is likely to be limited and highly imperfect, and so the performance of even the truly optimal policy with respect to the limited action space may be far below what is truly achievable. Because of this, we desire systems that add new actions to the set over time to come closer to optimal performance. Indeed, in practice it is common to try to improve performance further by creating new content, for example by writing new lines of dialog or designing new ads.

Unfortunately, automatic guidance on this process has been extremely limited, even in the simplified case of multi-armed bandits. It has typically been the task of domain experts to determine not just **what** new content to produce, but **when** and **where** to produce it.

In work not included in this thesis, we have investigated the problem of **when** to add

actions. This is a challenging problem, because if the expert produces new material too quickly, it might overwhelm the system, causing poor performance as the algorithm is unable to sufficiently explore all the arms. But if the expert does not add at all or adds to slowly, we risk underperforming due to missing potentially high-reward arms.

But, putting aside the question of **when** an algorithm should query an human for a new action, the task is still quite complex. In many cases, the expert needs context to compete their task. For example, in a dialogue system, asking the expert to write a new line of dialogue does not really make sense without providing information about what was said in the dialogue up to that point. In other words, we need to tell experts **where** to produce new action. Even in cases where the contextual information is not strictly necessary, asking the experts to tailor their action for a specific important situation has the potential to make the produced action much more useful.[1] In an RL framework, we can think of this problem as selecting the state where the next action should be added in order to maximize performance.

Related work in human-in-the-loop reinforcement learning has looked at where to query experts for demonstrations. A significant amount of work has focused on on querying the expert in a different setting where the agent's goal is merely to imitate [32, 77]. Related work by Clouse [36] studied at which state a reinforcement learning agent should query an imperfect expert for advice in an existing action set. However, this is a very different setting, as the assumption is that the human, though not perfect, can perform the task fairly well, and that the action space is sufficiently small to be amenable to autonomous exploration.

In this chapter, we propose Expected Local Improvement (ELI), as a heuristic to select the state where the next action should be added. ELI intelligently combines data gathered by a reinforcement learning algorithm, knowledge of the structure of the reinforcement learning problem (if present) and the behavior of the human experts it is interacting with to select states that will be most likely to boost performance. Although the ultimate goal is to run

[1]Experts might implicitly think of a situation when writing a new action. But selecting the best situation is a very difficult task, requiring understanding the domain, the algorithm, and drawing insights from a large dataset. Therefore, we seek to automate this process.

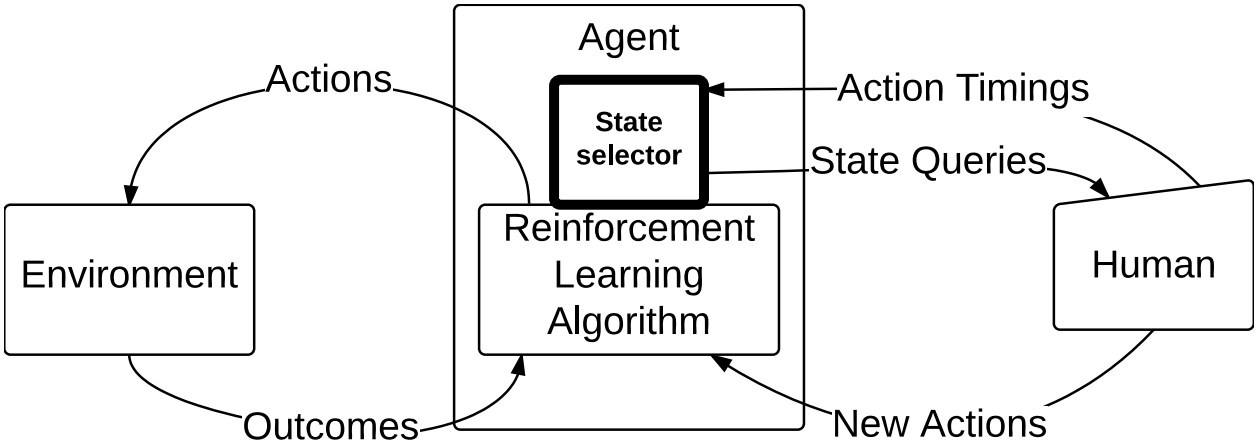


Figure 7.1: Proposed Human-in-the-Loop RL framework, in which a human provides new actions in response to state queries. Here we focus on the design of the state selector.

ELI with real humans on important real-world problems, similar to prior work [36, 4] we first evaluate algorithms in settings where both the RL environment and the "human" expert are simulated, as this allows for inexpensive and well-controlled comparisons. We find that ELI performs well across a variety of domains adapted from the literature, including a domain with over a million actions, a domain where the experts improve over time, and even a domain where the experts are quite poor.

Although this work takes the first step, there are many limitations. ELI is designed to work with a small, known state space, in order to enable richer adaptivity we need to investigate how to perform complex generalization in this new setting. We need ways to automatically help train experts to improve over time, and we need to better help experts visualize and understand the operation of the system. We need to give the experts sufficient freedom to be able to impact the game in a deep way, while keeping them constrained enough that they can settle on a good solution quickly. We need to understand how to direct experts in ways that play to their strengths while minimizing their weaknesses. We hope that this work helps open the door to investigating these fascinating directions.

7.2 Problem Setup

Although there has been much recent work focusing on the problem of effective generalization over a large, high dimensional state space in reinforcement learning, this task becomes much more complicated when the set of actions is not fixed but can be extended. As such, in this chapter we focus on the more traditional RL setting where a relatively small, discrete state space is known. Although we wish to consider Markov Decision Process (MDP) environments, a typical "tabula rasa" MDP setting does not fit many real-world scenarios where more structure is known.[2] Therefore, similar to previous chapters and past work [89, 8], we consider MDP domains where dynamics are specified in terms of relative outcomes.[3] Specifically, we assume a discrete state space $\mathcal{S}$ and a set of relative outcomes $\mathcal{O}$. We assume experience comes in episodes of maximum length τ. The agent starts with at least one action in $\mathcal{A}_{0s}$ for each s. At the start of the ℓ^{th} episode, generally $\mathcal{A}_{\ell,s} = \mathcal{A}_{\ell-1,s}$. However, if the human decides it is time to add a new action, the agent must pick a state s from $\mathcal{S}$, and the human proceeds to add an action[4] to $\mathcal{A}_{\ell,s}$. During the course of the ℓ^{th} episode, the agent is at a state $s \in \mathcal{S}$, takes an action $a \in \mathcal{A}_{s,\ell}$ and receives an outcome $o \in \mathcal{O}$. The agent knows the reward function $R(s, o)$ which deterministically outputs a scalar reward, and the transition function $T(s, o)$, which deterministically outputs a next state s'. However, it does not know the distribution over relative outcomes at each state and action, nor the process by which the human generates actions. The goal of the agent is to learn from experience to maximize the sum of rewards over time, both by picking good actions in the existing set during an episode and by selecting good states at which to add actions. In this chapter, we assume the former task is handled by a traditional reinforcement learning algorithm and focus on developing new algorithms for the latter problem. For convenience, let $S = |\mathcal{S}|$,

[2]For example, consider a gridworld where we know an agent can go in cardinal directions, and we know the location of the goal state but not where the walls are.

[3]Note, however, that in principle outcomes can encode any MDP with discrete rewards and states.

[4]We also allow humans to add extra actions at other states, e.g. if they feel the action they developed is appropriate elsewhere.

$A_\ell = \max_s |\mathcal{A}_{s,\ell}|$, and $O = |\mathcal{O}|$. An overview of the considered framework can be seen in Figure 7.1.

7.3 Related work

Active Imitation Learning One related setting is active imitation learning [62, 32, 76, 129, 77]. Here the agent's goal is to query an expert for demonstrations at states that will best help it improve its performance. However, in this setting the agent is not trying to optimize an external reward function, just imitate the expert, and therefore the challenges involved in combining autonomous reinforcement learning with human expertise do not arise. Additionally, state-of-the-art work in this area [77] assumes the action space is small and the experts are deterministic, making the techniques proposed ill-suited for our purpose.

Interactive/Human-in-the-Loop Reinforcement Learning There have been several studies of how humans should collaborate with reinforcement learning agents. Several interesting types of human feedback have been considered: Reward signals [147], policy quality information [61], entire trajectories [57], and more. We are only aware of very limited work (in the bandit setting) considering added actions [2, 159], which did not consider automated guidance on where to add actions.

Some work has focused on the related problem of deciding at which state to query expert for advice on the next action to take. For instance, Doshi-Velez et al. [45] studies this problem but assumes the expert is perfect and the reward signal is inaccurate. Most closely related to our work is Clouse [36], who proposes a heuristic method to determine when (at what state) an imperfect expert should be asked for advice in an online RL setting. The setting is different as Clouse deals with giving advice in an existing (small) action space, whereas we consider where one should extend the action space. In addition, the proposed heuristic asks for states at which the agent is uncertain, which does not make sense in our setting, since if we are uncertain about the values of the current actions at some state we should refrain from adding additional actions until we are more certain.

Large Action Spaces We propose leveraging human intuition to enable rapid learning

in vast action spaces. However, an alternative approach is to use function approximation to generalize over a large action space. Work in large discrete action spaces has typically assumed actions are featurized in a highly informative way [123, 50] and can still show poor performance when learning from scratch without prior knowledge. Past work in learning in large action continuous control problems [94] exploits knowledge of physical principles and large amounts of simulator data.

In the domain of conversational agents, related work has learned which actions to consider taking from a corpus of human conversations [91]. However, this work focuses on making use of a passively collected dataset of actions, we show in this chapter we can do better by intelligently directing human effort.[5]

7.4 Expected Local Improvement (ELI)

To ground our discussion, we assume that each state s is associated with an (unknown) distribution over actions with pdf $p_s(x)$, so that when we request an action at s the expert draws a (possibly continuous) action x sampled i.i.d. according to $p_s(x)$. [6] In some settings this stochastic assumption is reasonable, but in others it may not be, such as experts learning to improve the provided actions over time. We will empirically evaluate robustness to improving experts, as well as show a non-starving property (Lemma 7.4.1) under weaker assumptions.

Although our goal is to select states optimally under uncertainty, this is not a straightforward application of a standard exploration/exploitation approach (e.g. bandits), since:

- We wish to leverage the structure of the underlying RL problem to select states intelligently, instead of treating the problem in a tabula rasa fashion.

---

[5] An additional advantage to adding actions online is that information such as values of the current actions can be displayed to humans, to hopefully guide them to produce better actions.

[6] In some settings, such as text, the action space is finite or countably infinite, in which case we could use probabilities and sums instead of pdfs and integrals. However, here we solely consider the continuous case to simplify exposition.

- When a bandit algorithm encounters a positive reward, it is inclined to keep selecting the action that led to that reward. But here, a positive (high-value) action means just the opposite: we have found an action that performs well at this state so we should avoid selecting this state in favor of trying to improve performance elsewhere.

- Typical bandit algorithms learn the mean of the reward distribution, but here optimal behavior depends on higher moments of the action distribution. For example, imagine the action distributions at s_1 and s_2 are distributed normally (in terms of the action values) with identical means, but the distribution at s_1 has much higher variance than that at s_2. Then we should greatly prefer to select s_1 as we are much more likely to get a high-value action.

- We need to consider the uncertainty not just over the unknown distribution of new actions but also the unknown values of existing actions.

We desire methods that carefully direct human effort boost performance quickly, but not at the expense of long-term performance. Additionally, we want a heuristic that is scalable and easy to compute, without, for example, requiring explicitly planning over sequences of action additions.[7]

One heuristic that seems natural to consider is greedily adding the action that maximizes expected global improvement. However, here greediness becomes a significant detriment to long-term performance, as we are not incentivized to add actions that do not immediately improve performance but instead open up pathways to new states that have the potential to be high-value after adding more actions.

Perhaps surprisingly, a less global approach alleviates this issue. We propose maximizing the expected local improvement (ELI), that is, selecting the state s where adding the next action most improves the (optimal) value $V(s)$. Formally, we define the ELI of state s as

[7]One could consider casting the problem in a Bayesian RL-type framework (e.g. [155, 158]). However, the computational expense of this approach is prohibitive.

follows:

$$\int (V(s|x, \mathcal{A}_{s,\ell}) - V(s|\mathcal{A}_{s,\ell}))p_s(x)dx \tag{7.1}$$

where $V(s|x, \mathcal{A}_{s,\ell})$ is optimal value we would get at state s given our current set of actions and an additional action x.

This will result in a bottom up approach, where values of states where there is a high potential for immediate reward will be improved first and, once they are improved, we try to add actions to other states that transition to the high-value leaves. One potential downside of this approach is that we may prioritize adding actions at states that are (near) impossible to reach. However, in practice our approach for dealing with unknown dynamics alleviates this issue (see below).

Now, observe that for x such that $V(s|x, \mathcal{A}_{s,\ell}) = V(s|\mathcal{A}_{s,\ell})$ (i.e. actions which do not improve the value) the integral is zero so it suffices to take the integral over x such that $V(s|x, \mathcal{A}_{s,\ell}) > V(s|\mathcal{A}_{s,\ell})$:

$$\int (V(s|x, \mathcal{A}_{s,\ell}) - V(s|\mathcal{A}_{s,\ell}))p_s(x)dx \tag{7.2}$$

$$= \int_{x:V(s|x,\mathcal{A}_{s,\ell})>V(s|\mathcal{A}_{s,\ell})} (V(s|x, \mathcal{A}_{s,\ell}) - V(s|\mathcal{A}_{s,\ell}))p_s(x)dx \tag{7.3}$$

Next, consider estimating $V(s|x, \mathcal{A}_{s,\ell})$. An underestimate is dangerous here, as it may lead to ignoring (starving) states where there is a large potential for improvement. Therefore we follow the well-known principle of *optimism under uncertainty*, and overestimate the improvement by $V_{max}(s) = \max_x V(s|x, \mathcal{A}_{s,\ell})$, which is easy to compute with discrete outcomes as long as we have a way of estimating $V(s|\mathcal{A}_{s,\ell})$ for future states (see below). This gives us:

$$\int_{x:V(s|x,\mathcal{A}_{s,\ell})>V(s|\mathcal{A}_{s,\ell})} (V(s|x, \mathcal{A}_{s,\ell}) - V(s|\mathcal{A}_{s,\ell}))p_s(x)dx \tag{7.4}$$

$$\leq \int_{x:V(s|x,\mathcal{A}_{s,\ell})>V(s|\mathcal{A}_{s,\ell})} (V_{max}(s) - V(s|\mathcal{A}_{s,\ell}))p_s(x)dx \tag{7.5}$$

$$= (V_{max}(s) - V(s|\mathcal{A}_{s,\ell})) \int_{x:V(s|x,\mathcal{A}_{s,\ell})>V(s|\mathcal{A}_{s,\ell})} p_s(x)dx \tag{7.6}$$

$$= (V_{max}(s) - V(s|\mathcal{A}_{s,\ell}))P(V(s|x, \mathcal{A}_{s,\ell}) > V(s|\mathcal{A}_{s,\ell})) \tag{7.7}$$

However, calculating this requires estimating two unknown quantities: $P(V(s|x, \mathcal{A}_{s,\ell}) > V(s|\mathcal{A}_{s,\ell}))$ and $V(s|\mathcal{A}_{s,\ell})$. We discuss how this can be done below.

7.4.1 Estimating the value of existing actions

Let us first consider estimating $V(s|\mathcal{A}_{s,\ell})$, the value of a state given existing actions. Intuitively, if we don't know if the existing actions are high-value it seems wasteful to add another action there. Therefore, we propose using an optimistic estimate of the current value (and thus underestimate how much improvement is left). At first this seems contradictory, as we were trying to overestimate the improvement. However, we must consider the *source* of the uncertainty. If the source is a low number of added actions, using the principle of optimism under uncertainty we should overestimate the improvement of the next action, as adding an additional action will reduce uncertainty. But if the source is a low number of samples of existing actions, we should apply the principle of optimism under uncertainty again to the values of existing actions, as even though we cannot directly act there, when the RL algorithm does it will reduce our uncertainty. These two forms of optimism under uncertainty in ELI act synergistically by properly accounting for the source of the uncertainty and selecting states accordingly.

An additional benefit of overestimating $V(s|\mathcal{A}_{s,\ell})$ is that we will not select (nearly) impossible to reach states because we are extremely uncertain about their current values.

Fortuitously, the task of optimistically estimating the value of a transition/reward distribution given limited samples has been well-studied in the RL community as a way to drive exploration. These methods deal with optimistic long-term value in a simple dynamic programming fashion; calculating the optimistic $V(s,t)$ by treating the previously-calculated optimistic $V(s,t+1)$ as fixed and calculating an optimistic outcome distribution for each (s,a,t) tuple. The key difference among methods is in how the optimistic outcome distribution is calculated. We try a number of approaches for this component (see appendix D for details):

UCRL Following UCRL2 [71], this computes the most optimistic outcome distribution

subject to the constraint that the L1 norm varies from the MLE by at most $\sqrt{\frac{14\log(SA_\ell\tau\ell/\delta)}{\max(N,1)}}$ where ℓ is the number of episodes, N is the number of transition samples, and δ is a user-specified confidence parameter we set to 0.05 following Osband et al. [114].

MBIE Adapting a version of MBIE [135, 136] to our setting, this computes the most optimistic outcome distribution subject to the constraint that the L1 norm varies from the MLE by at most $\sqrt{\frac{2\log((2^O-2)/\delta)}{\max(N,1)}}$.

(Optimistic) Thompson Sampling This weakly optimistic approach [148, 103] draws a single sample from the posterior over outcome distributions under the constraint that the sample has greater value than the expected value under the posterior.

BOSS This approach [8] samples from the posterior over outcome distributions J times and returns the sample with maximum value. We set $J = 10$ based on the results in Asmuth et al. [8].

Of all the optimistic estimators, BOSS seems most attractive, since it combines prior information with a significant amount of optimism. Therefore for simplicity of exposition, we hereafter refer to ELI-BOSS as just ELI.

7.4.2 Estimating the probability of improvement

We now examine estimating $P(V(s|x, \mathcal{A}_{s,\ell}) > V(s|\mathcal{A}_{s,\ell}))$. A naïve approach is to replace this probability by 1 to trivially upper bound the integral, which intuitively corresponds to optimistically believing the next action will certainly result in a (maximal) improvement, avoiding i.i.d assumptions. The issue here becomes that the non-starving property is clearly violated, as we may simply sample the same state forever if no improvement is truly possible. A simple way to ensure this problem does not occur is simply to set a limit of L, such that we move on to another state if the selected state already has L actions. If all visited states have L actions we increment L. In general, it is unclear how to initialize L, so we set the minimum L of 1, allowing L to grow automatically to find the best value. We call this method **ELI-Limit**.

Intuitively, however, if we have selected a state numerous times already, it seems unlikely the next action will result in an improvement. To formalize this intuition, we define a binary variable X_m for each m such that $X_m = 1$ if the next action has Q-value greater than m and $X_m = 0$ otherwise. Now, since X_m is binary, we can learn it in a Bayesian fashion using a $Beta(\alpha_m + n_{1m}, \beta_m + n_{0m})$ posterior where α_m and n_{1m} represent the prior psuedocount and observed count of actions with Q-value greater than m, and likewise β_m and n_{0m} represent the prior psuedocount and observed count of actions with Q-value less than m. For the prior we let $\alpha_m = \beta_m = 1$ for all m, which corresponds to one pseudocount of maximum and minimum Q-value. Now in particular we are interested in the probability of further improvement, which corresponds to letting m be the maximum value over the existing actions ($m = \max_{a \in \mathcal{A}_{s,\ell}} Q(s,a)$). Therefore, after $|\mathcal{A}_{s,\ell}|$ added actions our posterior over the probability of improvement is $Beta(1, |\mathcal{A}_{s,\ell}| + 1)$ regardless of the Q-values.[8] Note that for a fixed m, the observed Q-values do matter, but since we keep changing m to be the maximum Q-value seen so far, the variable of concern X_m keeps changing, and so they do not.

Given this posterior distribution over $\theta = P(V(s|x, \mathcal{A}_{s,\ell}) > V(s|\mathcal{A}_{s,\ell}))$, a Bayesian approach is to integrate the objective function over all possible values of θ weighted by their posterior probability $P(\theta|\mathcal{A}_{s,\ell})$. Specifically, this changes equation (7.7) into:

$$\int (V_{max}(s) - \hat{V}(s|\mathcal{A}_{s,\ell}))\theta P(\theta|\mathcal{A}_{s,\ell}) d\theta \tag{7.8}$$

$$= (V_{max}(s) - \hat{V}(s|\mathcal{A}_{s,\ell})) \int \theta P(\theta|\mathcal{A}_{s,\ell}) d\theta \tag{7.9}$$

$$= (V_{max}(s) - \hat{V}(s|\mathcal{A}_{s,\ell})) E[\theta|\mathcal{A}_{s,\ell}] \tag{7.10}$$

where the last line comes simply from the definition of expectation. The expected value of θ under our $Beta(1, |\mathcal{A}_{s,\ell}| + 1)$ posterior derived above is simply $\frac{1}{|\mathcal{A}_{s,\ell}|+2}$. So the overall ELI

[8] Note that this is not correct in the edge case where we already have an action with maximum value, as the probability of improvement there should always be zero. However, in that case $(V_{max}(s) - \hat{V}(s|\mathcal{A}_{s,\ell}))$ should be zero resulting in zero score.

heuristic is:

$$= \frac{1}{|\mathcal{A}_{s,\ell}| + 2}(V_{max}(s) - \hat{V}(s|\mathcal{A}_{s,\ell})) \tag{7.11}$$

Since we are in a finite-horizon setting, values depend not just on s but on s, t. Our RL setting is undiscounted, so we simply sum up the scores for each s, t to generate an overall score for each s, and select the state with maximum score.

One property we desire is that ELI does not starve states as long as there is possible improvement remaining. This is a fairly weak property, as we could achieve it (at the cost of performance) by mixing in an ϵ-amount of uniformly random state selection. Intuitively, it seems as though ELI already has this property, as it does not starve states as long as there is still some difference between $\hat{V}(s)$ and $V_{max}(s)$. However, if the human adds rapidly adds large numbers of actions to all successor states s' of s and they are insufficiently explored, their values may go to $V_{max}(s')$ (because we take the max over a larger and larger set of random samples) and so (since $\hat{V}(s)$ is defined in terms of the $\hat{V}(s')$) it is possible that there becomes less and less difference between $\hat{V}(s)$ and $V_{max}(s)$ even if we do not add further actions at s. So showing ELI does not starve states as long as there is possible improvement is a bit nontrivial and requires additional assumptions. We do not wish to restrict the RL algorithm or the behavior of the human (w.r.t. action timings etc.), so we prove ELI is non-starving in a fairly general class of environments similar (just slightly more restricted) to previous posterior sampling work by Osband et al. [114].

Lemma 7.4.1. ***(Non-starving)*** *Consider ELI using a prior distribution f which consists of an independent Dirichlet prior on outcome distributions of $\alpha_i = c$ for some $c > 0$. Assume for a given $\epsilon > 0$, after N_ϵ actions are added to each state, additional actions improve the value of each state by at most ϵ. Let $\mathcal{C}$ be an arbitrary class of models with N_ϵ actions which has non-zero probability under our chosen prior. Assume that the true model M for the first N_ϵ actions is drawn from $\mathcal{C}$ according to $f(M|\mathcal{C})$.Finally, assume that for each s there exists $o_1, o_2 \in \mathcal{O}$ such that $T(s, o_1) = T(s, o_2); R(s, o_1) \neq R(s, o_2)$. Then, as the number of actions added by ELI goes to infinity, our ELI approach will eventually uncover actions at each state*

such that the optimal policy in the MDP with added actions is at least ϵ-optimal (with respect to the full set of actions).

Proof sketch (full proof in appendix D): The only way for us to starve a state is for its ELI score to go to zero. Since we only select it a finite number of times, the only way for this to occur is if $\hat{V}(s,t)$ approaches $V_{max}(s,t)$. But the cases such that this occurs have zero probability under the posterior.

7.5 Simulations

We show results using both simulated environments and simulated humans, similar to prior work [36, 4], as this allows us to compare algorithms in an inexpensive and well-controlled setting. Each state starts with a single action (similar to real-world settings where there is already a default strategy) and every 20 episodes a new action is added at the state the agent selects. We transform standard domains to this new setting by sampling a new (or initial) action uniformly at random from the ground space of actions, unless otherwise specified. Even with a fairly small space of ground actions, this is still a challenging problem, as (for example) the agent may unknowingly get the same suboptimal action repeatedly and need to select a state many times to get the optimal action.[9] All result graphs are averaged over 1000 runs.

ELI leaves open the choice of underlying RL algorithm. We use Posterior Sampling Reinforcement Learning (PSRL) [114, 137] due to its ability to explore efficiently. An additional advantage is that some variants of ELI can share the posterior used for PSRL.

We constrain all methods to only select states the agent has visited before.[10] We compare variations of ELI, **Random**, which selects a visited state uniformly at random, and **Frequency**, which selects states with probability proportional to the number of times they

[9]We may be able to prevent humans from duplicating an action, but there is still the possibility that humans generate an action with a very similar outcome distribution, which is roughly equivalent.

[10]This is partially motivated by the fact that in real-world domains it may be difficult to visualize unvisited states to humans.

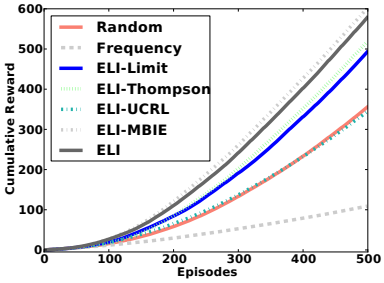


Figure 7.2: Riverswim results with a small outcome space.

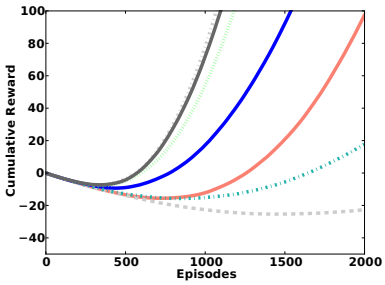


Figure 7.3: Marblemaze results with a small outcome space.

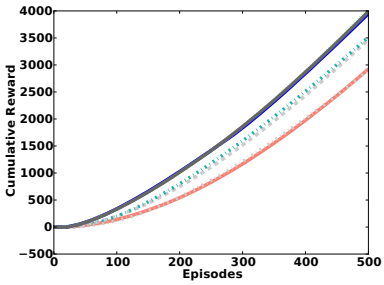


Figure 7.4: Results on the large-action task with random action generation.

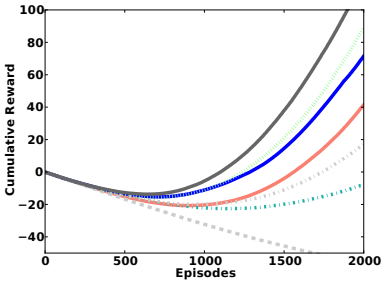


Figure 7.5: Marble Maze Results with a large outcome space.

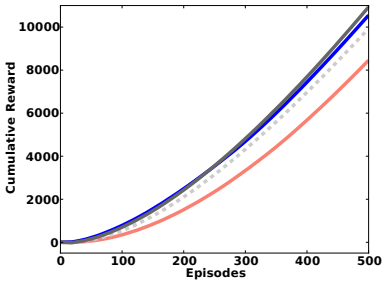


Figure 7.6: Results on the Large-Action task with improving action generation.

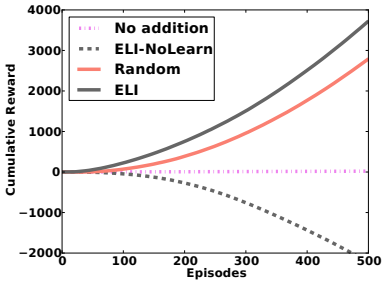


Figure 7.7: Results on the Large-Action task with poor action generation.

have been visited (i.e., explored) by the RL algorithm.

We examine performance on three environments adapted from the literature. **Riverswim** is a chain MDP with 6 states, 5 outcomes, and 2 ground actions per state that requires efficient exploration [114]. **Marblemaze** [8, 122] is a gridworld MDP with 36 states, 5 outcomes, and 4 ground actions per state that allows significant prior knowledge to be encoded in the outcomes framework. We use a modified version of the **Large Action Task** [123], an environment with 13 states, 273 outcomes, and 2^{20} ground actions per state, a ground action space far too large to explore directly. Details of these domains are in

appendix D.

Comparing algorithms The results in Riverswim (Figure 7.2) show that all variants of ELI except ELI-UCRL outperform the Random and Frequency baselines. In fact, we see that Frequency underperforms Random, because it focuses too heavily on high traffic states where no improvement is possible. ELI-UCRL likely underperforms due to its conservative confidence intervals, as it avoids selecting states unless they have been extremely well-sampled. We see ELI-Limit performs worse than our refined approach based on the Beta posterior. In this setting, however, the more strongly optimistic ELI-MBIE approach does seems to slightly outperform the proposed BOSS method, which in part indicates that our adapted MBIE confidence intervals are a better fit for our setting than the ones used in UCRL2.

In Marblemaze (Figure 7.3) we see similar trends to Riverswim, except that the boost in performance from Random to ELI is extremely large, and ELI also shows a large improvement over ELI-Limit. The performance improvements come from the fact that that ELI is able to leverage the small space of outcomes to focus on adding actions at states that are likely to be part of the optimal path to the goal.

In the Large Action Task (Figure 7.4) ELI again shows a strong improvement over random, and performs as well or better than the other ablated baselines. Here Frequency does better than Random, indicating that focusing on high-traffic states plays a much bigger role in proper selection than in Riverswim and Marblemaze, which makes sense as states in this environment are unlikely to come close to their maximal values and so re-sampling is usually valuable. ELI-MBIE, which performed very well in riverswim and marblemaze, performs quite poorly here. We suspect this has to do with the larger set of outcomes used in the large action task, which we study with our next experiment.

Informative vs Uninformative Outcomes Although in some scenarios a small and informative set of outcomes is known, in other settings we unfortunately do not know this additional structure. To test how performance changes in this setting we modified the marblemaze setup to have 96 outcomes, one for each possible reward and next state (see ap-

pendix D). In Figure 7.5 we see that although using uninformative outcomes did decrease the gap between ELI and Random compared to the informative outcomes case in Figure 7.3, ELI still performs quite well. These results confirm the poor performance of ELI-MBIE in large outcome spaces, which turns out to be due to the fact that the confidence interval used in MBIE causes it to become far too optimistic when the number of possible outcomes becomes large. The BOSS method does not suffer from this issue, though it is still optimistic enough to outperform ELI-Thompson here.

Different types of experts We now test a simulated expert who improves over time. When queried at a state the expert generates an action with equal (50%) or better (50%) immediate reward compared to the best action added at this state so far (details in appendix D). In Figure 7.6 we see that even though this setting is nonstationary and does not match the assumptions built when designing ELI, the ELI method still manages to perform well, outperforming both Random and Frequency baselines. Though the ELI-Limit method does not explicitly make these assumptions it does not manage to improve upon ELI here.

Often, human-in-the-loop RL approaches are thought to only be helpful if the human has enough expertise to perform the task reasonably well. To examine this we simulate a "poor" expert, where the probability of generating an action decreases linearly as the immediate reward of that action increases (see appendix D). For reference we display performance without action addition, and also ELI-NoLearn, which counterfactually measures how performance of a uniform random policy changes as ELI (fed with data from PSRL) adds actions. In Figure 7.7 we see that adding actions even at the states recommended by ELI simply worsens the quality of a random policy. Yet because the simulated human has a small probability of returning something high-value, the agent is able to leverage this to achieve excellent performance in this setting, especially when using ELI.

7.6 Real-World Experiment Setup

In previous sections we introduced the ELI algorithm and tested it on several simulated domains. While simulations can be illuminating, ELI is intended to work with real humans

in the loop. Therefore, in this section we study using real human experts to help a reinforcement learning system solve a real-world problem: improving the hints in Riddle Books, an educational game. Some prior work, in collaboration with Chen [31] looked at this problem, but in the restricted domain of getting crowdworkers to rewrite generic hints in a way that was more contextualized to the problem. However, in this section we explore using experts to navigate a broader space: adding arbitrary text-based hints to the game. The motivation is that a broader space should allow more human intuition and creativity to be brought to bear, and using experts instead of crowdworkers should result in higher-quality hints that can directly be shown to students. Indeed, we broaden the space even further by allowing experts to write interface "hint questions" that incentivize the students to engage with the hints on a deeper level.

We now proceed to describe the details of the (somewhat complex) experimental design.

Riddle Books To make the general problem of crowdsourcing educational content more concrete, we focus on improving hints for the educational math game Riddle Books (Figure Figure 7.8), developed by the Center for Game Science. Riddle Books teaches students how to reason conceptually about math word problems in an engaging way. Although the game eventually becomes quite complex, we focus our experiments on the initial addition and subtraction sections due to the steep dropoff common to online games. The game is based off of the Singapore Math curriculum, and requires students to build diagrams, called "models", of the conceptual relationship between items in the story. For example, if the question asks, "John has 2 cats and 1 dog. How many pets does he have total?", users would be required to drag out the important parts of the story and use them to build a model which shows a block with "2", a block with "1", and a brace labeled "total pets" spanning them. After doing this, students enter the second phase, where they are required to transform their diagram into an equation (e.g. in the previous example: 2+1 = total cats). At no time are students actually asked to perform the arithmetic, as the focus is on conceptual understanding of how to interpret word problems.

Riddle Books includes a hinting system to help students who are struggling with the

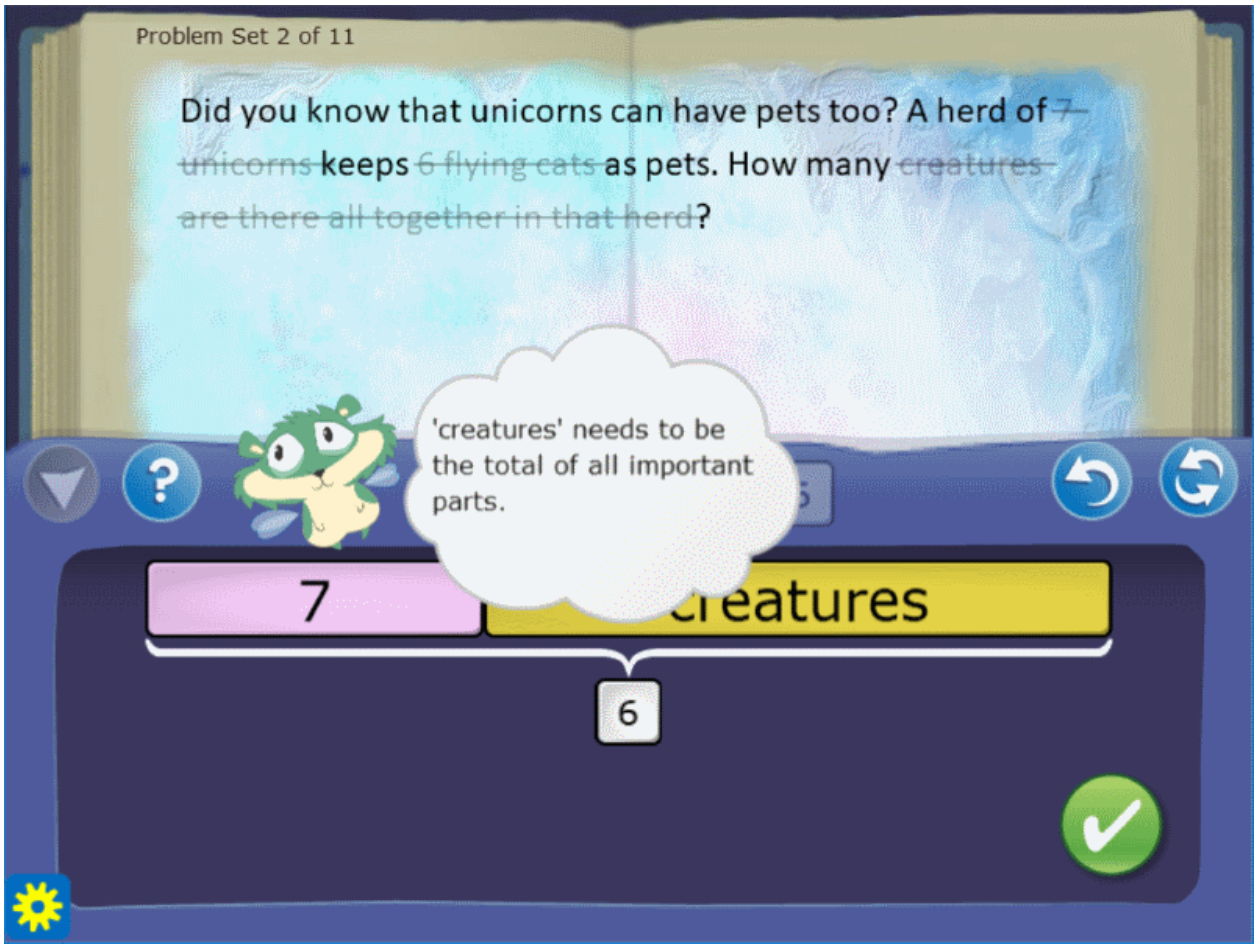


Figure 7.8: Riddle Books gameplay screen showing an addition problem and a default hint. To fix their model, the student should switch the positioning of the "6" and "creatures" elements.

levels. At any time, a student can press the hint button and receive a hint, shown in a text bubble next to an animated character. The game has a default hinting framework, which provides highly generic hints based on rules (e.g. if missing the "total" element in an addition problem, say, "The unknown part is missing? What is it?"). As the student presses the hint button multiple[11] times, the default hints go from more generic to more specific. We wish to improve these hints in three key ways: (1) By using educational experts instead of the original game developer, (2) by allowing them to refer to the near-exact situation the student is stuck in instead of using the generic rule-based framework, and (3) by broadening the space of hints to include interactive "hint questions" that incentivize deeper engagement. Because of the expense of using experts and the large number of situations students request help in, we wish to use ELI to efficiently direct expert effort in this setting, while the underlying reinforcement learning system determines which hints are most useful for which students.

Our goal is to increase student **mastery** of the topics taught in the game. In Riddle Books, there is a slightly different definition of mastery for each problem set, but in all cases once you have reached mastery you are immediately moves on to the next problem set.

- For the first problem set (addition), the game specifies that players must complete 2 out of the last three levels proficiently before reaching mastery. In this problem set, "proficiently" means that the player uses no hints and makes less than 2 mistakes.

- For the second problem set (addition/subtraction), players must complete one addition and one subtraction level proficiently before reaching mastery. In this problem set, "proficiently" means that the player uses no hints and makes no mistakes.

7.6.1 Hint Groups and Batches of Requests

In adapting ELI to this real-world use case, we encountered two new challenges which required modifying the ELI algorithm:

[11] maximum of three

Hint Groups ELI selects a single state at which to add hints. Yet in many real-world cases, states are defined in such a way that a an action which could be used at one state could be available at any state in the state's "group". In Riddle Books, to make the task straightforward for the experts, we ask experts to write hints based only on the current level and current student diagram.[12] Therefore, all other information we'd like to include in the state (such as proficiency history on past levels, or number of times the hint button has been pressed) is irrelevant for the purposes of determining whether the hint is valid there. So in this case, the state group would include all states with the same current level and student diagram, because actions (hint) tried in one state could just as easily be tried in another in the same group.

The question becomes, since ELI calculates a score for each state, how do we combine ELI scores from all states throughout a group? There are three natural options here: take the max, the sum, or the mean. Taking the max seems unattractive because it ignores the fact that in some groups you may be able to impact multiple very influential states. Taking the sum fixes this, but doesn't account for the fact that we may add this new action to a lot of states where it may not be needed, potentially harming performance through excess exploration. Therefore, we choose to average the ELI scores across the states in the state group.

Batches of Requests In the previous half of the chapter we considered only selecting a single state at which top add an action. Yet, it is often more efficient for people to complete a larger batch of tasks at their convenience rather than complete tasks one-at-a-time. This is a form of delay, and our results in chapter 5 indicate that it can be a major issue for optimism-driven algorithms such as UCB. Although ELI has an element of stochasticity due to the posterior sampling in BOSS [8], it is still the case that running ELI multiple times will typically produce the same top-scoring state. Although there are multiple ways to solve

[12]We could have designed this differently, but felt that even given additional context, experts were likely to focus their hint primarily on the level and current student model. Additionally, this approach allows a single expert hint to have a much larger impact as it can be applied in many states, thereby making better use of expert effort.

this problem, we employ the simple solution of simply choosing the top k scoring groups for a batch of size k. Therefore, the state groups chosen on each batch will all be distinct.

To make the comparison to the Random baseline fair, we also select k random state groups **without replacement**, so that both methods produce k unique queries.

7.6.2 Building the Overall Framework

Previously we showed a simplified diagram (Figure 7.1) to make it clear how state selection was key. The full diagram, showing all components that need to be instantiated to have a working Human-in-the-Loop RL system is shown in Figure 7.9. Below we describe how we implemented the various components for the Riddle Books hinting experiments:

State Selector ELI, as before.

Action Selector Given a state (or state group) that we wish to query about, we need to show some sort of action information to the experts to help them create a new action. At a minimum, we need to prevent workers from writing hints that are very similar to the existing hints.

Action Visualizer We need to visualize the existing actions to the experts. For purely textual hints, this is trivial: simply display the text associated with the hint. For simplicity, we use this style of visualization for the other types of hints as well, since encouraging variety in the text should naturally encourage different modalities as well.

State (Group) Visualizer We visualize the three parts of the state group:

- **The story** is displayed at the top with the relevant parts of the problem highlighted.
- **The student model(s)** are displayed in the middle using a javascript visualizer designed to closely match the visualizations displayed in game. Note that a challenge is that there may be many possible student models in this state group, so up to two[13] are selected using the state exemplar selector and displayed. A special case message is displayed when the player's board is empty.

[13]Only one is shown if only one model was was ever encountered in the data

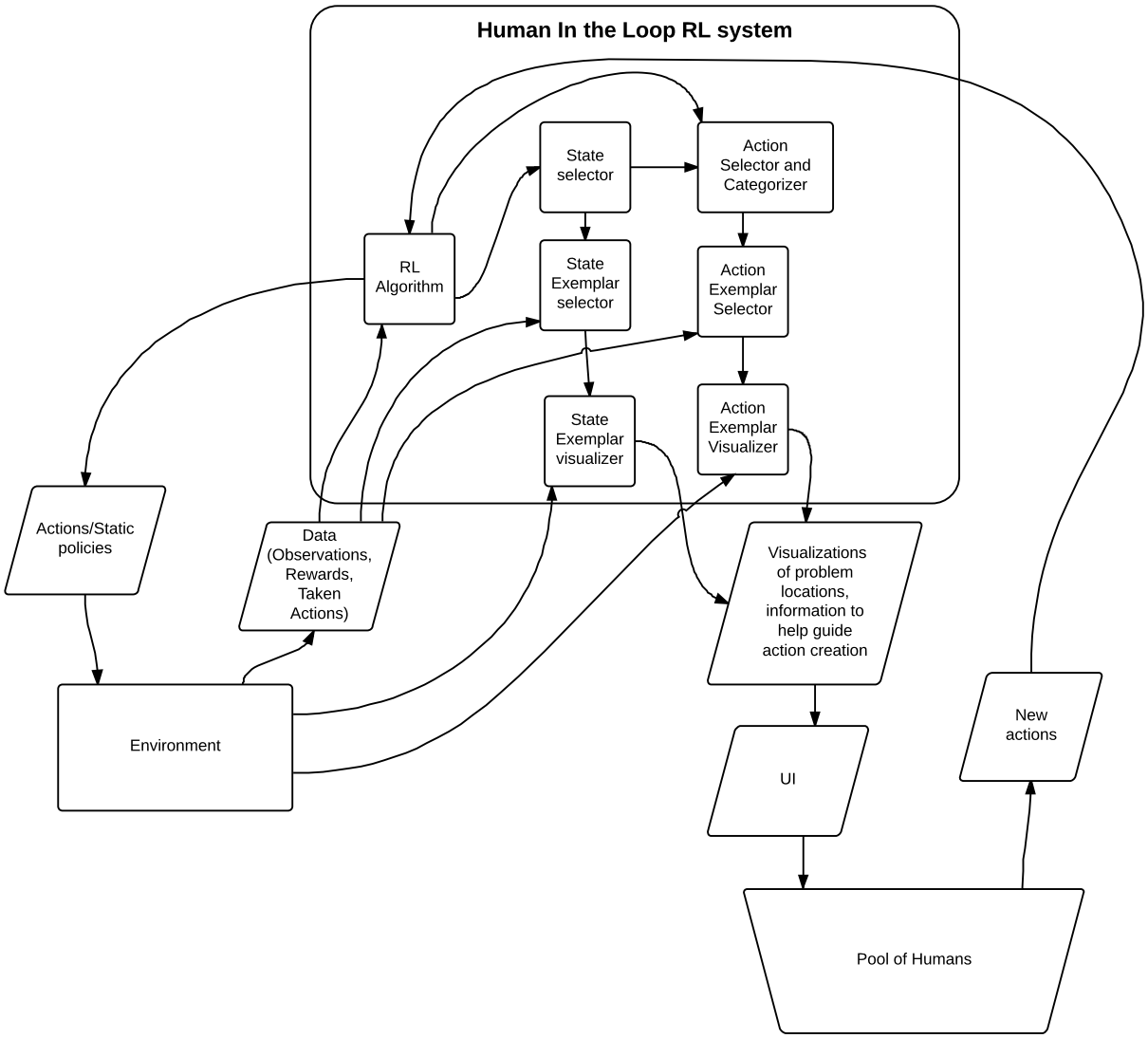


Figure 7.9: Full Human-in-the-Loop RL framework, with all key parts of the query generation pipeline expanded.

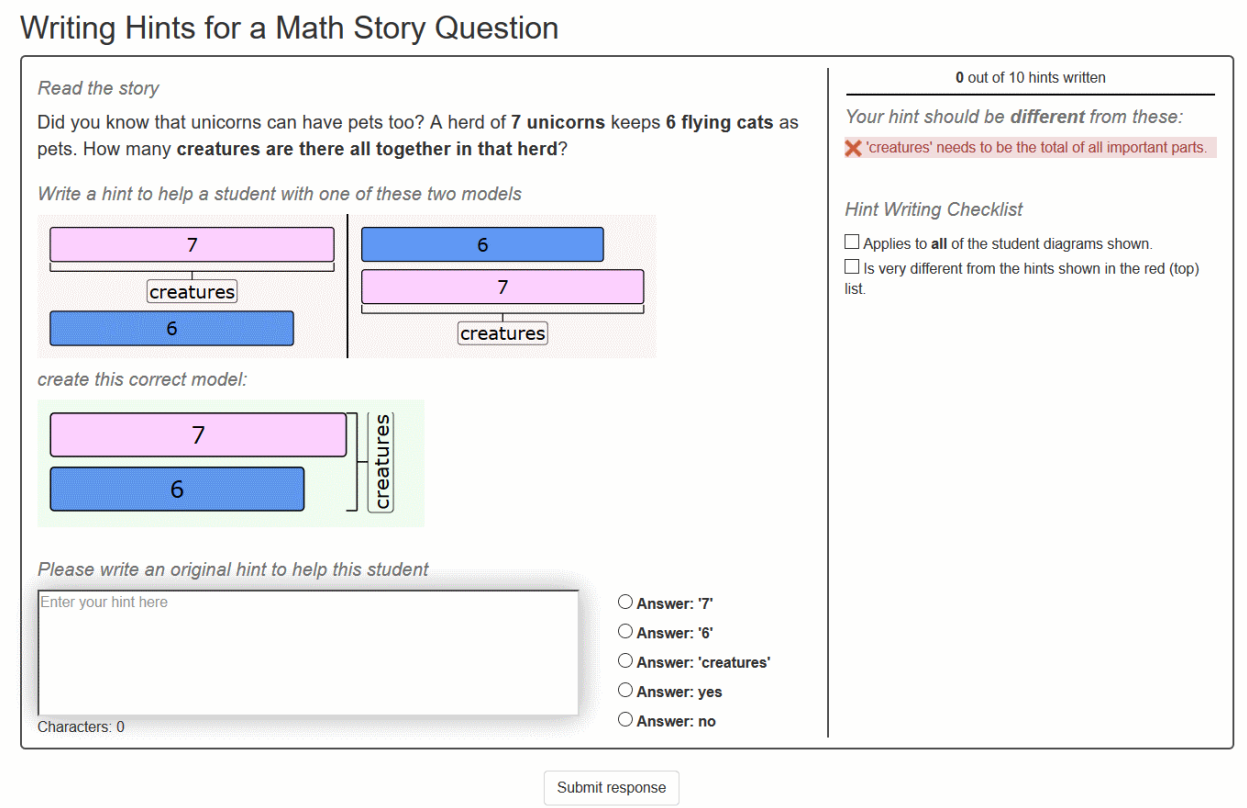


Figure 7.10: Hint-Writing User Interface used by the experts to add hints to the game.

- **A correct model** is displayed (using the same visualization code as the student models) at the bottom of the page. For most levels, there are several very different models that correctly describe the answer, all of which the game will accept. However, there is usually a single model that is the more-or-less intended solution. Given that the interface is already very complicated and we wanted to reduce user confusion, we display multiple correct models. The one exception is that, if the student appears to be working on a vertically-aligned addition solution, the visualizer will pick a vertically-aligned solution instead of the default horizontal one.

State Exemplar Selector We look through the data, and, of all examples that landed in this state group, we pick the 2 most frequent unique diagrams.

Action Exemplar Selector We look through the data, and, for all cases where the given (state group, action) was taken, we take the top 4. Note that this is non-trivial because the default hint action may produce slightly different hints depending on aspects of the context not accounted for by the state group (for example, the number of times the hint button has been pressed).

Expert User interface At the end of the day, the different visualizations are combined in a web-based user interface, shown in figure 7.10. The interface design is based off work in collaboration with Chen [31]. Because in early iterations the experts had trouble keeping track of hints, the interface keeps track of how many hints are completed to ensure the experts do the correct number. The UI randomizes and blinds the conditions shown to the experts.

7.6.3 Experiment Design

We wished to compare three conditions: Control (just the initial set of hints with none added), Random (state queries selected uniformly at random), and ELI (ELI-selected state queries). Upon starting the game, students are assigned uniformly at random to one of these three conditions.

Riddle Books was deployed to two popular educational games websites, Mathplayground.com and ABCya.com. Every night, the reinforcement learning system pulls the data for each condition and updates the policies and current set of prioritized requests for the next day. Periodically[14], educational experts were asked to write 10 new hints, 5 for the random condition and 5 for the ELI condition.

In the latest version of the experiment we used two individuals with significant expertise in education. One has a Masters degree in education and 6 years of teaching experience across all subjects at the high school level. The second individual holds a PhD in Mathematics Education and has two years of experience teaching high school math.

[14]Since ELI gives no guidance on *when* to add actions, we leave that design decision up to the experimenter and the schedules of the experts.

In order to emulate a case where we had already collected data from a standard deployment, data previously collected from 616 using all default hints was shared between all three methods [15].

The reinforcement learning problem is cast into an outcome MDP framework. It is very similar to the framework previously-defined in section 6.2, but with one important exception: instead of allowing arbitrary known functions T(s,a,o) and R(s,a,o), because the set of actions is unknown initially, we require these functions not to depend on the actions, that is, T(s,o) and R(s,o).

States The state includes:

- The progression index: A binary value which indicates whether it is in the first progression (addition only) or the second progression (alternating between addition and subtraction)

- The level index within the given progression: There are 6 levels in the first progression, and 8 in the second progression. The student moves on to the next progression as soon as they reach mastery. If they reach the end of the level set before reaching mastery, they are teleported back to the beginning.

- The classification of the student bar model: Individual bar models are too fined-grained to compose states, so we manually constructed groups of models, mostly based on invariance to vertical and horizontal inversion. This results in a total of 39 possible bar model types.

- The number of times the student has pressed the hint button in this bar model type – capped at 2 and reset when the student asks for a hint at a different model type. So there are three possibilities here - 0, 1, or greater than 1.

[15] Due to technical issues, 2/616 students has some missing transitions

- Information needed for mastery. In the first (addition) problem set, this is a tuple denoting which of the last two levels were completed proficiently. In the second (addition+subtraction) problem set, this is a tuple indicating whether an addition level has been completed proficiently and where a subtraction level has been completed proficiently.

There is also one special "bridge" state that connects the addition and subtraction components.

State Groups States are grouped by progression index, level index, and bar model classification. In other words, the number of times pressed and mastery information are ignored when determining state groups.

Outcomes Outcomes contain the number of levels completed, whether the individual quit or not, the next bar model classification, and (if levels were completed) the sequence of true/false level proficiency results.[16]

Actions The system initially starts with a single action (just give the default hint from the system). Over time, human experts write hints for certain states, which the system treats as additional actions. In other words, the action space consists of which hint to give (default or one of the expert-written hints).

In the latest version, the new hint is composed of a question with a multiple choice answer (Figure 7.11). There are two possible answer sets: yes/no, or asking the individuals to select one of the three labels in the problem. The student must press the correct answer before the hint disappears and they are allowed to return to solving the level. Forcing students to interact with the hint will hopefully make them actually try to read and understand it, thus increasing the effectiveness of the hints. Of course, writing hints in this manner requires expert effort, which is why we are using our system to direct expert expert to author them.

Rewards For part 1, the agent receives -1 if the player quits before reaching mastery. If the player reaches mastery, the agent receives +1 - 0.05*(# of levels it took to reach

[16]The transitions are coded such that any wraparound is always viewed as a "quit" from the perspective of the RL system, and the horizon resets as well.

Figure 7.11: Example of a yes/no Hint Question in Riddle Books. Students must select the correct answer before they can resume the game.

mastery).

For part 2, the agent receives 0 if the player quits before reaching mastery. If the player reaches mastery, the agent receives an additional +1 - 0.05*(# of levels it took to reach mastery).

Horizon The maximum horizon is at most 10 steps in part 1, and at most 20 steps total, though these limits are rarely reached.

Prior Our Dirichlet prior is designed to encourage two things: (1) sparsity, and (2) a disproportionate amount of prior probability mass is placed on reward-bearing observations. The first property is important because it allows us to learn more quickly in large outcome spaces, as we quickly concentrate probability mass around the outcomes we actually observe. The second property is important to give our methods a realistic sense of the reward distri-

bution at each state; since we have a large number of 0-reward outcomes[17], a naive approach would result in posterior samples typically yielding zero-reward. However, we want our prior to reflect our belief that despite the smaller number of outcomes, students are more likely to quit or reach mastery than they are to ask for a hint at another random state.[18]Specifically, if there are N total observations[19] we place $\frac{1}{N}$ on each of them, and then if there are N_R reward-bearing observations, we add $\frac{3}{N_R}$ to each of them.

7.6.4 Prior Versions

We gained a lot of insight into the experimental design and the tweaks needed to ELI through real-world iteration. In this section we describe each of the previous versions and the analysis that prompted the next iteration.

Version 1: Original. Major problem: ELI kept selecting the same group of states over and over, making its selections not very useful.

Version 2: Improve ELI to be aware of state groups. Major problem: ELI now started off well, but after a while began selecting states with very low traffic, and refused to ever re-select states. The issue was that, using the prior we selected, the "optimistic" sample generated was not actually very optimistic in terms of reward.

Version 3: Improved prior. Major problem: ELI now seems to be working very well. We see expert hints affecting a much higher percentage of people in the ELI condition compared to random, and ELI re-selects states when appropriate. But reward looked strange: ELI looked a little better than random, but about the same as control. This was very strange because random was directing new hints so poorly that almost no students were affected, so it and control should have been functionally equivalent. The only difference I was aware of was that the file size was much larger in the random and ELI condition than in the control

[17]students pressing the hint button at another location in the same progression

[18]Also, it is difficult for ELI's estimates to be at all "optimistic" if the posterior samples all have zero value.

[19]Note that the prior is also aware of "invalid" observations (for instance, completing 3 levels when there are only 2 remaining), and places 0 on them.

condition (due to the larger action space), which might have had an effect. But the bigger issue was that the difference was quite small, it seems like adding the expert hints was not having as large of an impact as we expected.

Version 4: Equalized/reduced file size, added ability to highlight elements of the UI (background, checkbox or one of three parts of the problem). Major problem: All approaches looked about the same. Further inspection revealed that hints with highlights seemed to have, on average, a small negative impact on immediate reward. This is likely due to the fact that although highlights may help students solve a single problem, they are unlikely to help long-term learning.

Version 5: Added Hint Questions to increase student interaction with hints. There are two types of questions: yes/no and whether the answer is one of three parts of the problem. Major problems: Reward looked promising, but hints were low-quality, especially in terms of grammar. A bug was discovered which resulted in the system thinking many students would be sent to the "bridge" state instead of terminating.

Version 6: Changed experts to more skilled individuals with more extensive teaching experience, and fixed bug. The is the current version.

7.7 Real-World Experimental Results

So far, in the latest version of the experiment 15 hints have been added to each condition, based on three days of hint writing spread roughly one week apart. In our experiments we filter only for students who pressed the hint button in an experimental level at least once, as the other students would not be impacted by improved hints. The current version of the experiment began on June 21st, 2017, and since data volume drops off during the summer we are getting roughly 15 players per condition per weekday who meet our criterion.

First, we investigate whether ELI was directing expert hints in a way that was actually impacting students. This is is an important and necessary step, as if the new content is not being shown to many students one would not expect to see any positive impact compared to the control condition. Additionally, change over time in the percent of hints shown can tell

us a large amount about whether the underlying reinforcement learning system is finding the new hints useful or not.

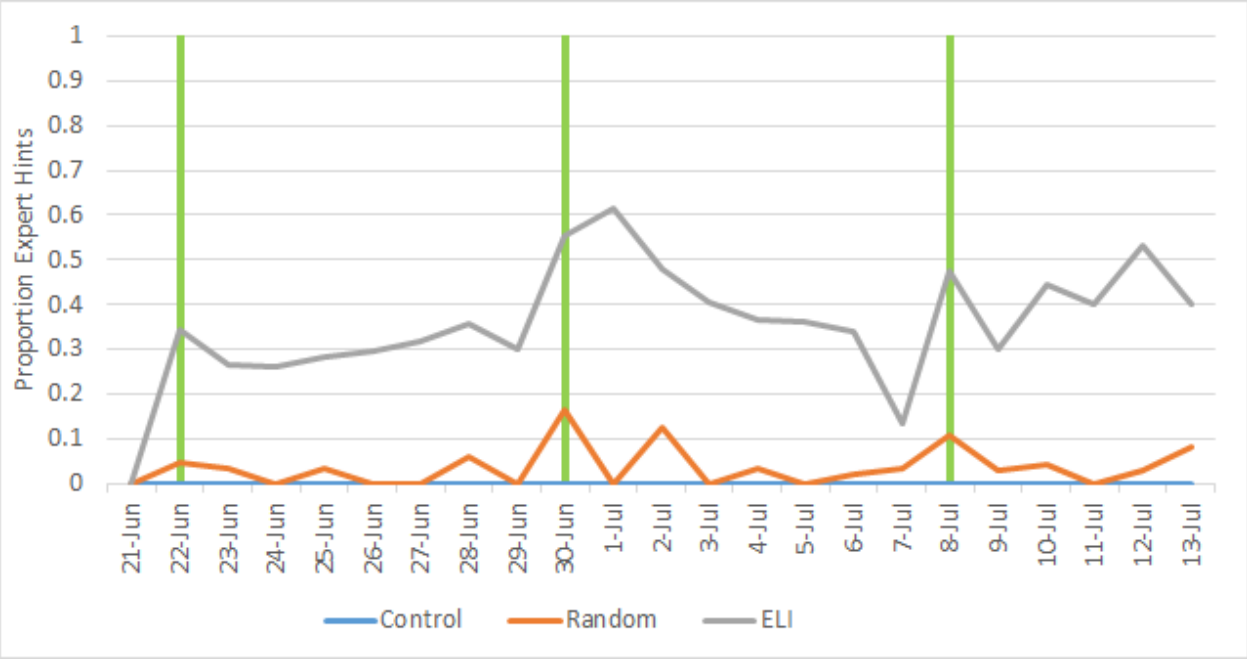


Figure 7.12: Proportion of expert hints actually shown to players across conditions. The green vertical lines mark the days that new hints were deployed into the game (i.e., they were written the day before).

In Figure 7.12 we see that the ELI condition shows many more expert hints to players than the version which randomly directs expert effort. Overall, 34% of the hints shown in the ELI condition were generated by experts, while expert hints accounted for only 4% of the hints in the Random condition. This implies that players are likely to see mainly default hints in both Random and Control conditions, which implex (since there are no other differences between conditions) we should on average lead to similar student outcomes across these conditions. Between the second and third addition in the ELI condition, we do see a bit of a "backoff" effect where the percent of expert hints peaks and then decreases, which suggests that in many states the expert hints might not have been useful (or at least

not show a clear benefit).

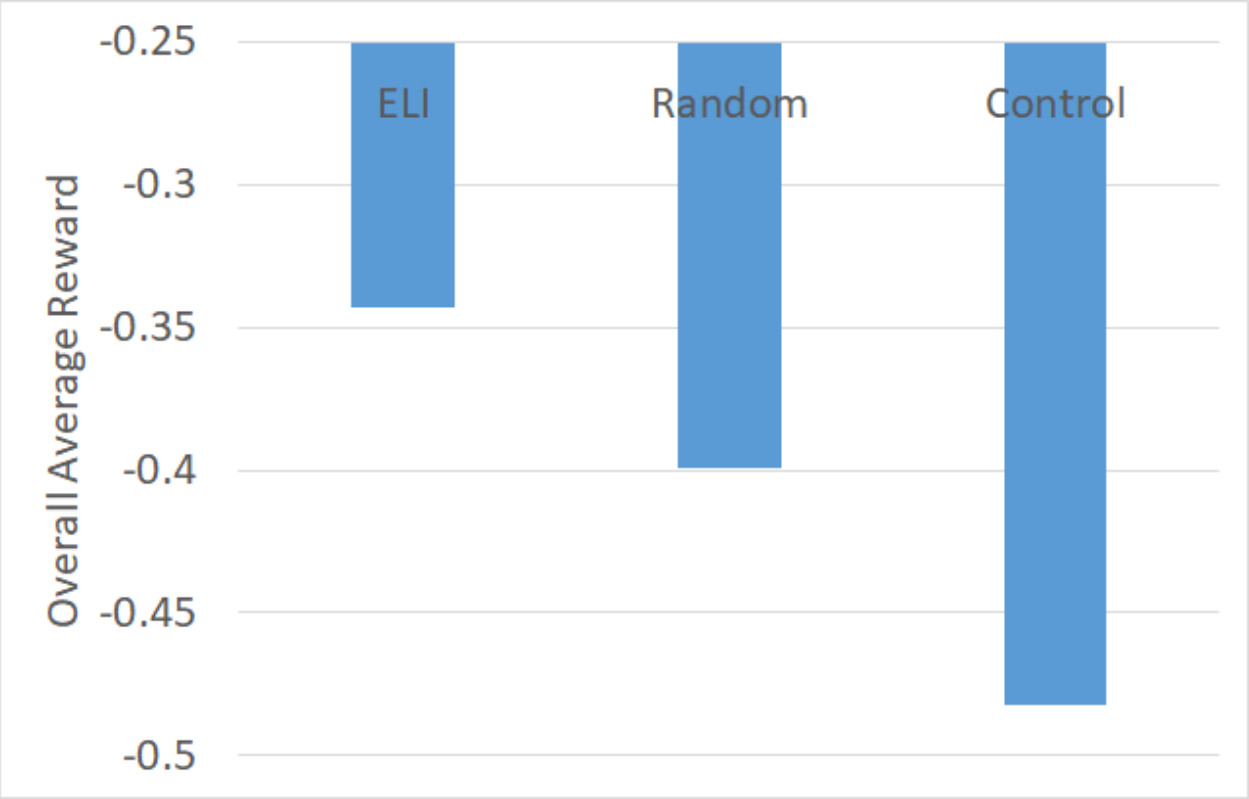


Figure 7.13: Overall cumulative reward compared across conditions.

Next we examine the reward across conditions. In Figure 7.13 the promising results indicate that, as expected, ELI seems better than random which seems better than control. The differences between conditions are relatively sizeable thus far, with ELI incurring 29% less negative reward than the Control condition.[20] However, there is a large amount of noise and a relatively small amount of data, so these results are not fully conclusive, and we plan to continue running the experiment for at least another week to better understand the true differences between conditions.

To dive a bit deeper into the reward, we can examine cumulative reward over time 7.14.

[20]To give some perspective, this could be achieved by making at least 15% more students reach mastery on the addition section.

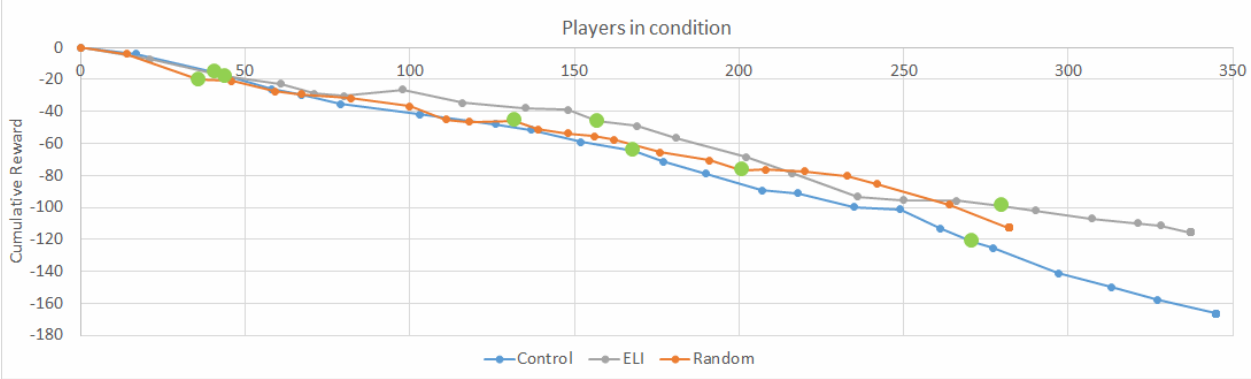


Figure 7.14: Cumulative reward across compared across conditions. The circular green markers show the points where new hints had just been added to the action space.

Here the large degree of noise becomes more apparent in the graph, which looks starkly less clean than the previous cumulative reward graphs we have shown which have been averaged over many runs. Overall, ELI does appear to be improving as we add more hints to the system, though again, running the experiment longer will help confirm this trend.

7.8 Conclusion and Future Work

In this chapter we proposed a new framework for human-in-the-loop reinforcement learning, where an automated agent leverages human intuition to learn effectively in vast action spaces. We identify a key challenge in this space, namely selecting a state at which to develop a new action, and present a new algorithm, Expected Local Improvement (ELI), which effectively addresses this challenge. We present a real-world educational game experiment which indicates the potential effectiveness of the approach at directing new content in useful places.

Freely exploring different modalities One difficulty is that we have had to manually define the modality in which hints are written. In the first version, it started as any piece of text, then became text+ highlight, then became text+multiple choice question. Ideally, we could have a system which allowed experts to use their intuition and creativity to more nimbly

navigate this space, without requiring the experimenter or game designer's involvement. Of course, navigating a richer space would certainly raise the bar on human contributions and require a longer time to develop new actions. Figuring our how to design this in a more accessible way, and training people to understand how to write richer (perhaps even Turing-complete) modifications to the game is a fascinating direction for future work.

Better Generalization Currently, we do not do any generalization, instead we learn about each (state, action) pair from scratch using PSRL. Using methods like TCRL here is complicated by the fact that the set of actions is quite diverse and unknown a prioi. Indeed, almost all generalization approaches in RL depend on the meaning of action "1" being somewhat similar across states, which is not the case when these are diverse hints written by people. Clearly, the first step is to generalize across state groups, which is fairly straightforward but could significantly speed learning. The next step would be to analyze the hints themselves to define a notion of similarity, for example using NLP techniques to get at lexical similarity. The difficulty here is that we don't have much data, and seemingly small changes might result in a big impact when shown to students. Involving humans in another part of the loop, determine which hints are similar, is a very interesting direction.

More Diversity in Hints One limitation of the current system is that, since it optimistically thinks the added hints are better, initially it rushes to try the newly-written hints in every applicable state. Yet, due to the fact that the hints apply after any number of hint button presses, this can be frustrating for students as they keep seeing the same hint over and over. One would hope that this would resolve itself in later batches, but this is not necessarily the case, as if hints performs poorly initially, they will be shown much less on future batches. Adding something to the state to prevent hints being shown multiple times seems an attractive option, but that is not a straightforward solution as it means that people are now having an (unintended) effect on the state space as well as the action space.

Better Optimistic estimates A key subproblem in ELI is finding high-quality optimistic estimates of reward. As the number of states to choose between increases, this problem becomes more and more vital. Although the BOSS approach worked well in small

state spaces, in larger spaces the small chance that all 10 samples are fairly low becomes a major downside to this approach, as we could end up selecting a state with little or no samples. Developing better techniques that are computationally efficient is an important future direction.

Better Expert Training Even for those with a background in education, this task is quite difficult, and what hints to write remains unclear. Indeed, throughout the course of the experiments I had several people ask me for more training. Although I tried to provide feedback, it's unclear that there's anyone who has sufficient expertise in this task to be able to train others. The major problem is that, using the current system, the experts are largely "blind" - very little feedback is given to them on how past hints have performed, and no guidance is given on what type of hint to write next. The only feedback provided in the current interface is a listing of the previous hints produced with an indication to write very different hints.

The most obvious next step is to include some positive examples that the experts should try to emulate. However, which hints to include in this set is unclear. It's also unclear if this will have a new positive impact, especially in domains such as ours which require creativity. One idea would be to use a contextual bandit to determine what hints should be shown in this set, to best train experts.

Another idea would be to have experts try to judge the quality of hints produced by other experts, as a form of training. Previous work [44] has shown that this type of training can be very effective in cases, especially in cases where such as ours where there is no authority that can produce the optimal response in every situation.

Of course, this is a very limited form of feedback, and there is an enormous amount of work to be done on helping experts better understand the student data gathered from the system's experience in order to better improve the system.

Chapter 8

CONCLUSION

8.1 Future Work

The future direction I am most excited about is continuing to develop and study a rich interaction paradigm between humans and reinforcement learning agents. Current work in this area is quite limited, often seeing humans solely as expert demonstrators or sources of reward. By developing algorithms that interact with humans in a complex and more nuanced manner, we can solve problems that are too difficult for humans or RL algorithms to solve on their own. There are fascinating research challenges here, both from an AI perspective, and from a Human-Computer Interaction (HCI) perspective. Some research questions I plan to investigate include:

What other aspects of an RL algorithm can humans help improve? So far, my work has examined interacting with humans to extend a discrete action space. Yet, humans can help in other ways: they can help us add observations, reduce dimensionality, improve reward, and even refine the algorithm itself. Leveraging all these aspects will allow us to move beyond simplistic models of human behavior to make full use of human capabilities.

How can we develop user interfaces that help humans improve over time? Humans are not static, but can learn to better interact with RL agents over time if given the right feedback. What information to show to help humans learn how to improve their interactions, and how to convey the agents partial understanding of the environment, are open questions deserving of further study from both AI and HCI perspectives.

How can we learn an effective state representation while simultaneously directing humans to extend the problem definition? Real word RL problems have an enormous state space, so to learn quickly it is vital we be able to generalize (or *abstract*) across states. However,

existing methods for state abstraction assume aspects of the problem definition (such as the action space) are fixed, indicating a need for new methods. This becomes particularly important when we wish to query humans at a particular state, as a human may have difficulty understanding the learned state abstraction.

8.2 Conclusion

In this thesis I have presented many separate projects, but all of these share a common goal: making reinforcement learning more practical in human-focused domains. To this end, in this thesis I examined three broad classes of problems that are particularly important in these domains: Offline evaluation, efficient exploration, and incorporating human experts in the loop. Although there are still numerous open problems and limitations, this thesis takes the first steps towards solving these key challenges.

As artificially intelligent agents become more pervasive and their capabilities grow, the question of how they should interact with humans can only become more important. There is much work to be done, but by continuing to explore these directions I hope to help bring about a bright future where humans and autonomous agents work together in harmony.

BIBLIOGRAPHY

[1] Yasin Abbasi-Yadkori and Csaba Szepesvári. Regret bounds for the adaptive control of linear quadratic systems. In *COLT*, pages 1–26, 2011.

[2] Jacob Abernethy, Kareem Amin, Michael Kearns, and Moez Draief. Large-scale bandit problems and KWIK learning. In *ICML*, pages 588–596, 2013.

[3] Shipra Agrawal and Navin Goyal. Further optimal regret bounds for Thompson sampling. In *AISTATS*, pages 99–107, 2013.

[4] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara Grosz. Interactive teaching strategies for agent training. In *IJCAI*. AAAI Press, 2016.

[5] Ankit Anand, Aditya Grover, Mausam Mausam, and Parag Singla. ASAP-UCT: abstraction of state-action pairs in UCT. In *IJCAI*, pages 1509–1515. AAAI Press, 2015.

[6] Erik Andersen, Yun-En Liu, Richard Snider, Roy Szeto, Seth Cooper, and Zoran Popović. On the harmfulness of secondary game objectives. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 30–37. ACM, 2011.

[7] Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

[8] John Asmuth, Lihong Li, Michael L Littman, Ali Nouri, and David Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *UAI*, pages 19–26. AUAI Press, 2009.

[9] Jean-Yves Audibert and Sébastien Bubeck. Minimax policies for adversarial and stochastic bandits. In *COLT*, pages 217–226, 2009.

[10] P Auer and R Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. *NIPS*, 19:49, 2007.

[11] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[12] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 322–331. IEEE, 1995.

[13] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The non-stochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[14] Peter Auer and Ronald Ortner. Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

[15] Benoit Baccot, Romulus Grigoras, and Vincent Charvillat. Reinforcement learning for online optimization of banner format and delivery. *Online Multimedia Advertising: Techniques and Technologies*, 2011.

[16] Tiffany Barnes and John Stamper. Toward automatic hint generation for logic proof tutoring using historical student data. In *ITS*. Springer, 2008.

[17] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(1):253–279, 2013.

[18] Yoshua Bengio and Paolo Frasconi. An input output HMM architecture. *NIPS*, 1994.

[19] Donald A Berry, Robert W Chen, Alan Zame, David C Heath, and Larry A Shepp. Bandit problems with infinitely many arms. *The Annals of Statistics*, pages 2103–2116, 1997.

[20] Thomas Bonald and Alexandre Proutiere. Two-target algorithms for infinite-armed bandits with bernoulli rewards. In *Advances in Neural Information Processing Systems*, pages 2184–2192, 2013.

[21] Byron Boots, Sajid Siddiqi, and Geoffrey Gordon. Closing the learning planning loop with predictive state representations. In *I. J. Robotic Research*, volume 30, pages 954–956, 2011.

[22] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1), 2000.

[23] BrainPOP. http://www.brainpop.com/.

[24] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[25] Emma Brunskill, Bethany R Leffler, Lihong Li, Michael L Littman, and Nicholas Roy. Provably efficient learning with typed parametric models. *JMLR*, 10:1955–1988, 2009.

[26] Emma Brunskill and Stuart Russell. RAPID: A reachable anytime planner for imprecisely-sensed domains. *UAI*, 2012.

[27] Alexandra Carpentier and Michal Valko. Simple regret for infinitely many armed bandits. *arXiv preprint arXiv:1505.04627*, 2015.

[28] Deepayan Chakrabarti, Ravi Kumar, Filip Radlinski, and Eli Upfal. Mortal multi-armed bandits. In *Advances in Neural Information Processing Systems*, pages 273–280, 2009.

[29] Olivier Chapelle and Lihong Li. An empirical evaluation of Thompson sampling. In *NIPS*, pages 2249–2257, 2011.

[30] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *IJCAI*, volume 91, pages 726–731, 1991.

[31] Yvonne Chen, Travis Mandel, Yun-En Liu, and Zoran Popović. Crowdsourcing accurate and creative word problems and hints. *AAAI HCOMP*, 2016.

[32] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1, 2009.

[33] Min Chi, Pamela Jordan, Kurt VanLehn, and Moses Hall. Reinforcement learning-based feature selection for developing pedagogically effective tutorial dialogue tactics. In *EDM*, 2008.

[34] Ku-Chun Chou and Hsuan-Tien Lin. Balancing between estimated reward and uncertainty during news article recommendation for icml 2012 exploration and exploitation challenge. In *ICML 2012 Workshop: Exploration and Exploitation*, volume 3, 2012.

[35] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI*, 1992.

[36] Jeffery A Clouse. An introspection approach to querying a trainer. Technical report, University of Massachusetts, 1996.

[37] Mark Cutler, Thomas J Walsh, and Jonathan P How. Reinforcement learning with multi-fidelity simulators. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3888–3895. IEEE, 2014.

[38] Yahel David and Nahum Shimkin. Infinitely many-armed bandits with unknown value distribution. In *Machine Learning and Knowledge Discovery in Databases*, pages 307–322. Springer, 2014.

[39] Yahel David and Nahum Shimkin. Pac algorithms for the infinitely-many armed problem with multiple pools. In *EWRL*. 2015.

[40] Yahel David and Nahum Shimkin. Refined algorithms for infinitely many-armed bandits with deterministic rewards. In *Machine Learning and Knowledge Discovery in Databases*, pages 464–479. Springer, 2015.

[41] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[42] Thomas Desautels, Andreas Krause, and Joel Burdick. Parallelizing exploration-exploitation tradeoffs with gaussian process bandit optimization. *arXiv preprint arXiv:1206.6402*, 2012.

[43] Carlos Diuk, Lihong Li, and Bethany R Leffler. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 249–256. ACM, 2009.

[44] Shayan Doroudi, Ece Kamar, Emma Brunskill, and Eric Horvitz. Toward a learning science for complex crowdsourcing tasks. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2623–2634. ACM, 2016.

[45] Finale Doshi, Joelle Pineau, and Nicholas Roy. Reinforcement learning with limited reinforcement: Using bayes risk for active learning in pomdps. In *Proceedings of the 25th international conference on Machine learning*, pages 256–263. ACM, 2008.

[46] Finale Doshi-Velez. The infinite partially observable Markov decision process. In *NIPS*, pages 477–485, 2009.

[47] Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. Sample-efficient nonstationary policy evaluation for contextual bandits. *arXiv preprint arXiv:1210.4862*, 2012.

[48] Miroslav Dudík, Dumitru Erhan, John Langford, Lihong Li, et al. Doubly robust policy evaluation and optimization. *Statistical Science*, 29(4):485–511, 2014.

[49] Miroslav Dudik, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. Efficient optimal learning for contextual bandits. *arXiv preprint arXiv:1106.2369*, 2011.

[50] Gabriel Dulac-Arnold, Richard Evans, Peter Sunehag, and Ben Coppin. Reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[51] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. In *Journal of Machine Learning Research*, pages 503–556, 2005.

[52] NA Fay and KD Glazebrook. On a no arrivals heuristic for single machine stochastic scheduling. *Operations research*, 40(1):168–177, 1992.

[53] Raphaël Féraud and Tanguy Urvoy. Exploration and exploitation of scratch games. *Machine Learning*, 92(2-3):377–401, 2013.

[54] Raphael Fonteneau, Susan Murphy, Louis Wehenkel, and Damien Ernst. Model-free monte carlo-like policy evaluation. In *Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, 2010.

[55] Raphael Fonteneau, Susan A Murphy, Louis Wehenkel, and Damien Ernst. Batch mode reinforcement learning based on the synthesis of artificial trajectories. *Annals of operations research*, 208(1):383–416, 2013.

[56] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*, volume 2. Taylor & Francis, 2014.

[57] Amit Gil, Helman Stern, and Yael Edan. A cognitive robot collaborative reinforcement learning algorithm. *World Academy of Science, Engineering and Technology*, 2009.

[58] John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 1989.

[59] Sergiu Goschin, Ari Weinstein, Michael L Littman, and Erick Chastain. Planning in reward-rich domains via pac bandits. In *EWRL*, pages 25–42, 2012.

[60] Kathrin Grave and Sven Behnke. Bayesian exploration and interactive demonstration in continuous state maxq-learning. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3323–3330. IEEE, 2014.

[61] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2625–2633, 2013.

[62] Daniel H Grollman and Odest Chadwicke Jenkins. Dogged learning for robots. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2483–2488. IEEE, 2007.

[63] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 2016.

[64] Arthur Guez, David Silver, and Peter Dayan. Scalable and efficient bayes-adaptive reinforcement learning based on Monte-Carlo tree search. *JAIR*, pages 841–883, 2013.

[65] Arthur Guez, Robert D Vincent, Massimo Avoli, and Joelle Pineau. Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *IAAI*, 2008.

[66] Hirotaka Hachiya, Takayuki Akiyama, Masashi Sugiayma, and Jan Peters. Adaptive importance sampling for value function approximation in off-policy reinforcement learning. *Neural Networks*, 22(10):1399–1410, 2009.

[67] Katherine A Heller and Zoubin Ghahramani. Bayesian hierarchical clustering. In *ICML*, pages 297–304. ACM, 2005.

[68] Eshcar Hillel, Zohar S Karnin, Tomer Koren, Ronny Lempel, and Oren Somekh. Distributed exploration in multi-armed bandits. In *NIPS*, pages 854–862, 2013.

[69] Marcus Hutter. Universal artificial intelligence: Sequential decisions based on algorithmic probability, 2005.

[70] Charles Lee Isbell, Michael Kearns, Satinder Singh, Christian R Shelton, Peter Stone, and Dave Kormann. Cobot in LambdaMOO: An adaptive social statistics agent. *AAMAS*, 2006.

[71] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *JMLR*, 11(Apr):1563–1600, 2010.

[72] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[73] Jeff Johns and Sridhar Mahadevan. Constructing basis functions from directed graphs for value function approximation. In *ICML*, pages 385–392. ACM, 2007.

[74] Nicholas K Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, pages 752–757. Citeseer, 2005.

[75] Pooria Joulani, Andras Gyorgy, and Csaba Szepesvari. Online learning under delayed feedback. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1453–1461, 2013.

[76] Kshitij Judah, Alan Fern, and Thomas Dietterich. Active imitation learning via state queries. In *Proceedings of the ICML Workshop on Combining Learning Strategies to Reduce Label Cost*. Citeseer, 2011.

[77] Kshitij Judah, Alan P Fern, Thomas G Dietterich, et al. Active lmitation learning: formal and practical reductions to iid learning. *JMLR*, 15(1):3925–3963, 2014.

[78] Varun Kanade, H Brendan McMahan, and Brent Bryan. Sleeping experts and bandits with stochastic action availability and adversarial rewards. In *International Conference on Artificial Intelligence and Statistics*, pages 272–279, 2009.

[79] Godfrey Keller and Alison Oldale. Branching bandits: a sequential search process with correlated pay-offs. *Journal of Economic Theory*, 113(2):302–315, 2003.

[80] Jyrki Kivinen and Manfred K Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[81] Robert Kleinberg, Alexandru Niculescu-Mizil, and Yogeshwer Sharma. Regret bounds for sleeping experts and bandits. *Machine learning*, 80(2-3):245–272, 2010.

[82] J Zico Kolter and Andrew Y Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 521–528. ACM, 2009.

[83] Junpei Komiyama, Issei Sato, and Hiroshi Nakagawa. Multi-armed bandit problem with lock-up periods. In *Asian Conference on Machine Learning*, pages 116–132, 2013.

[84] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *RSS*, 2008.

[85] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *JMLR*, 4:1107–1149, 2003.

[86] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[87] John Langford, Alexander Strehl, and Jennifer Wortman. Exploration scavenging. In *Proceedings of the 25th international conference on Machine learning*, pages 528–535. ACM, 2008.

[88] Tor Lattimore. The Pareto regret frontier for bandits. In *Advances in Neural Information Processing Systems*, pages 208–216, 2015.

[89] Bethany R Leffler, Michael L Littman, and Timothy Edmunds. Efficient reinforcement learning with relocatable action models. In *AAAI*, volume 7, pages 572–577, 2007.

[90] Sergey Levine and Vladlen Koltun. Guided policy search. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1–9, 2013.

[91] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. In *EMNLP*, 2016.

[92] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297–306. ACM, 2011.

[93] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for MDPs. In *ISAIM*, 2006.

[94] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[95] Stephen Lin and Robert Wright. Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning. In *SARA*, 2010.

[96] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.

[97] Michael L Littman, Anthony R Cassandra, and L Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning*, pages 362–370, 1995.

[98] Michael L Littman, Richard S Sutton, and Satinder P Singh. Predictive representations of state. In *NIPS*, 2001.

[99] Hamid Reza Maei and Richard S Sutton. GQ (λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, volume 1, pages 91–96, 2010.

[100] A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

[101] Shie Mannor, Duncan Simester, Peng Sun, and John N Tsitsiklis. Bias and variance in value function estimation. In *ICML*, 2004.

[102] Shie Mannor and John N Tsitsiklis. The sample complexity of exploration in the multi-armed bandit problem. *The Journal of Machine Learning Research*, 5:623–648, 2004.

[103] Benedict C May, Nathan Korda, Anthony Lee, and David S Leslie. Optimistic bayesian sampling in contextual-bandit problems. *JMLR*, 13(Jun):2069–2106, 2012.

[104] Andrew Kachites McCallum. Learning to use selective attention and short-term memory in sequential tasks. In *SAB*, volume 4, page 315. MIT Press, 1996.

[105] H Brendan McMahan and Matthew J Streeter. Tighter bounds for multi-armed bandits with expert advice. In *COLT*, 2009.

[106] Lilyana Mihalkova and Raymond J Mooney. Using active relocation to aid reinforcement learning. In *FLAIRS Conference*, pages 580–585, 2006.

[107] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[108] Peter Nash. *Optimal allocation of Resources Between Research Projects.* PhD thesis, University of Cambridge, 1973.

[109] Gergely Neu, András György, Csaba Szepesvári, and András Antos. Online Markov decision processes under bandit feedback. In *NIPS*, pages 1804–1812, 2010.

[110] Andrew Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. *Experimental Robotics IX*, pages 363–372, 2006.

[111] Olivier Nicol, Jérémie Mary, and Philippe Preux. Improving offline evaluation of contextual bandit algorithms via bootstrapping techniques. *arXiv preprint arXiv:1405.3536*, 2014.

[112] Ronald Ortner. Adaptive aggregation for reinforcement learning in average reward Markov decision processes. *Annals of Operations Research*, 208(1):321–336, 2013.

[113] Ronald Ortner, Odalric-Ambrym Maillard, and Daniil Ryabko. Selecting near-optimal approximate state representations in reinforcement learning. In *Algorithmic Learning Theory*, pages 140–154. Springer, 2014.

[114] Ian Osband, Dan Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. In *NIPS*, pages 3003–3011, 2013.

[115] Ian Osband and Benjamin Van Roy. Near-optimal reinforcement learning in factored MDPs. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *NIPS*, pages 604–612. Curran Associates, Inc., 2014.

[116] Ian Osband and Benjamin Van Roy. Bootstrapped Thompson sampling and deep exploration. *arXiv preprint arXiv:1507.00300*, 2015.

[117] National Mathematics Advisory Panel. *Foundations for success: The final report of the National Mathematics Advisory Panel*. US Department of Education, 2008.

[118] Leonid Peshkin and Christian R Shelton. Learning from scarce experience. *ICML*, 2002.

[119] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[120] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *ICRA 2008*, pages 2845–2851. IEEE, 2008.

[121] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *AISTATS*, 2011.

[122] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[123] Brian Sallans and Geoffrey E Hinton. Reinforcement learning with factored states and actions. *JMLR*, 5(Aug):1063–1088, 2004.

[124] J. Schatzmann, K. Weilhammer, M. Stuttle, and S. Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The Knowledge Engineering Review*, 21(2):97–126, 2006.

[125] Steven L Scott. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

[126] Yevgeny Seldin and Aleksandrs Slivkins. One practical algorithm for both stochastic and adversarial bandits. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1287–1295, 2014.

[127] Aaron P Shon, Deepak Verma, and Rajesh PN Rao. Active imitation learning. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 22, page 756. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[128] Susan M Shortreed, Eric Laber, Daniel J Lizotte, T Scott Stroup, Joelle Pineau, and Susan A Murphy. Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine learning*, 84(1):109–136, 2011.

[129] David Silver, J Andrew Bagnell, and Anthony Stentz. Active learning from demonstration for robust autonomous navigation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 200–207. IEEE, 2012.

[130] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[131] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *JAIR*, 16:105–133, 2002.

[132] Satinder P Singh, Tommi Jaakkola, and Michael I Jordan. Reinforcement learning with soft state aggregation. *NIPS*, pages 361–368, 1995.

[133] Edward Jay Sondik. The optimal control of partially observable markov processes., 1971.

[134] Alexander L Strehl, John Langford, Lihong Li, and Sham Kakade. Learning from logged implicit exploration data. In *NIPS*, volume 10, pages 2217–2225, 2010.

[135] Alexander L Strehl and Michael L Littman. An empirical evaluation of interval estimation for Markov decision processes. In *ICTAI 2004*, pages 128–135. IEEE, 2004.

[136] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[137] Malcolm Strens. A Bayesian framework for reinforcement learning. In *ICML*, pages 943–950, 2000.

[138] Kaushik Subramanian, Charles L Isbell Jr, and Andrea L Thomaz. Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 447–456. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[139] Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *arXiv preprint arXiv:1503.04269*, 2015.

[140] J. Tang and P. Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In *Neural Information Processing Systems*, 2011.

[141] Jonathan Taylor, Doina Precup, and Prakash Panagaden. Bounding performance loss in approximate MDP homomorphisms. In *NIPS*, pages 1649–1656, 2009.

[142] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Comm. of the ACM*, 38(3):58–68, 1995.

[143] Joel R Tetreault and Diane J Litman. Comparing the utility of state features in spoken dialogue using reinforcement learning. In *NAACL HLT*, 2006.

[144] J.R. Tetreault, D. Bohus, and D.J. Litman. Estimating the reliability of MDP policies: a confidence interval approach. In *NAACL HLT*, 2007.

[145] Olivier Teytaud, Sylvain Gelly, and Michele Sebag. Anytime many-armed bandits. In *CAP07*, 2007.

[146] Philip S Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence off-policy evaluation. *AAAI*, 2015.

[147] Andrea Lockerd Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *AAAI*, volume 6, pages 1000–1005, 2006.

[148] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, pages 285–294, 1933.

[149] Stephan Timmer and M Riedmiller. Abstract state spaces with history. In *NAFIPS*, pages 661–666. IEEE, 2006.

[150] Stephan Timmer and Martin Riedmiller. Safe Q-learning on complete history spaces. In *Machine Learning: ECML 2007*, pages 394–405. Springer, 2007.

[151] Benjamin Van Roy and Zheng Wen. Generalization and exploration via randomized value functions. *arXiv preprint arXiv:1402.0635*, 2015.

[152] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *Machine Learning: ECML 2005*, pages 437–448. Springer, 2005.

[153] Ngo Anh Vien, Wolfgang Ertel, Viet-Hung Dang, and TaeChoong Chung. Monte-Carlo tree search for Bayesian reinforcement learning. *Applied intelligence*, 39(2):345–353, 2013.

[154] Thomas J Walsh, István Szita, Carlos Diuk, and Michael L Littman. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 591–598. AUAI Press, 2009.

[155] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *ICML*, pages 956–963. ACM, 2005.

[156] Yizao Wang, Jean-Yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, pages 1729–1736, 2009.

[157] Tsachy Weissman, Erik Ordentlich, Gadiel Seroussi, Sergio Verdu, and Marcelo J Weinberger. Inequalities for the L1 deviation of the empirical distribution. *Hewlett-Packard Labs, Tech. Rep*, 2003.

[158] Peter Whittle. Arm-acquiring bandits. *The Annals of Probability*, 9(2):284–292, 1981.

[159] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z Gajos, Walter S Lasecki, and Neil Heffernan. Axis: Generating explanations at scale with learnersourcing and machine learning. In *Learning@ Scale*, pages 379–388. ACM, 2016.

[160] Shiqing Yu. On robust arm-acquiring bandit problems. Technical report, University of Michigan, 2014.

Appendix A

OFFLINE POLICY EVALUATION

A.1 ECR Consistency

In this appendix we examine one offline evaluation metric, ECR, and show that it is not statistically consistent and therefore not appropriate for selecting representations in high stakes domains. ECR was proposed by Tetreault et al. [143] in order to estimate the performance of a policy given a discrete MDP state space. The basic calculation is very simple: given an MDP with state space $\mathcal{S}$ and a probability p_s of starting in each $s \in \mathcal{S}$, one can compute ECR of a policy π as follows:

$$ECR(\pi) = \sum_{s \in \mathcal{S}} p_s V_\pi(s),$$

where the value function V_π is computed using the standard MDP policy evaluation approach and p_s is estimated from data. To compare two MDPs M_1 and M_2, one computes the optimal MDP policies π_1^* and π_2^* using the standard value iteration approach, and compares $ECR(\pi_1^*)$ and $ECR(\pi_2^*)$ to determine which state space is best for representing the problem. ECR has been used both in dialogue systems [143] and intelligent tutoring systems [33] to compare different MDP state spaces offline. Similar methods have been proposed by Mannor et al. [101] and Shortreed et al. has proposed a similar estimate for Fitted Q-Iteration [128].

While this ECR approach has led to empirically promising results, here we present a counterexample which shows that using ECR to compare across different state spaces can lead us to incorrectly estimate reward and select a poor state space, even given infinite data. The relevant MDPs are depicted in Figure A.1. Assume we have a true system with a horizon of 3, which gives reward according to the MDP M_T. Now imagine we want to consider the state space where states are defined by the last action taken (action 0, action 1,

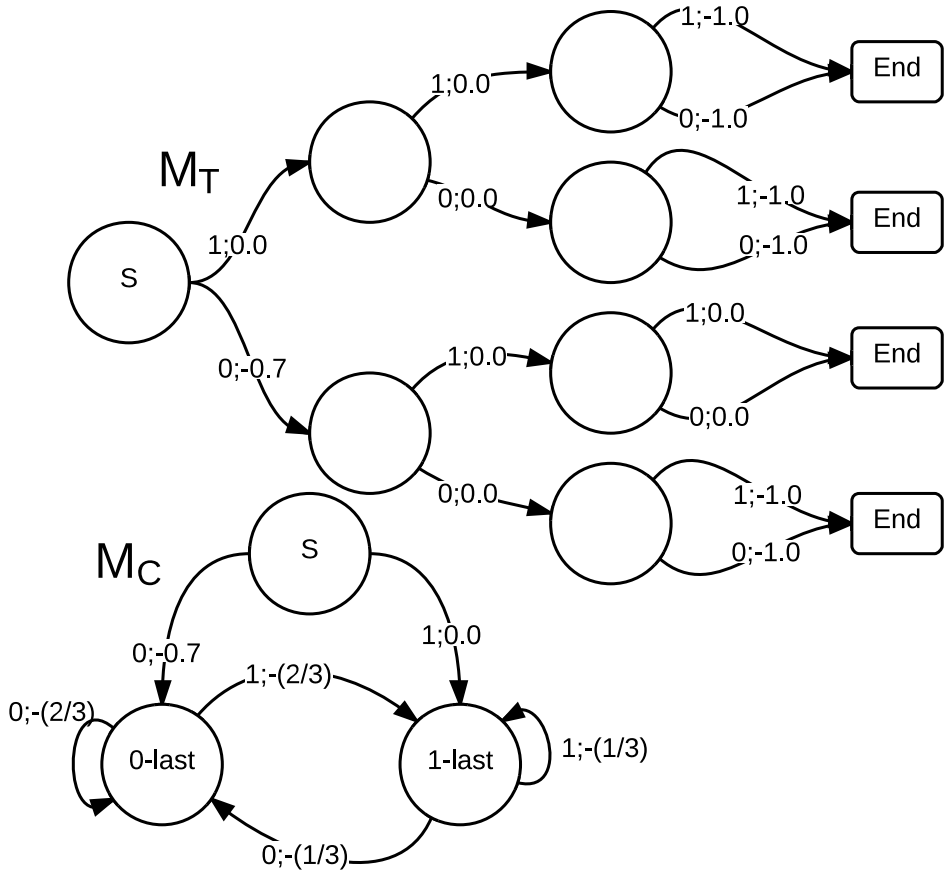


Figure A.1: A counterexample which shows ECR is not statistically consistent when used to select state spaces. The edges are labeled with $a; r$ labels indicating the action and reward. Both MDPs have a single start state S.

or nothing [the start state]). Assuming we had infinite data and thus could perfectly estimate all parameters of the MDP, this model would be M_C. Solving for the optimal policy π^*_C in M_C gives the sequence of actions $[1, 1, 1]$, and the ECR of this policy is $-\frac{2}{3}$. However, an optimal policy π^*_T in the true MDP M_T is $[0, 1, 0]$, and the ECR is -0.7. So, since the ECR of M_C is higher, we will choose M_C instead of the true model M_T, even given infinite data. Even more worrisome for high-stakes domains is that, having chosen M_C, we will have an estimate policy has value $-\frac{2}{3}$ when it actually has a score of -1, even with unlimited data,

meaning ECR is not statistically consistent.

The intuitive reason for this bias is that ECR assumes the state space is correct. If the state space in some way compacts the action history (meaning instances with different action histories can end up in the same state), then when computing the reward for a policy that reaches state s_i, reward will be averaged not just for episodes that followed the policy up to this point, but also others who reached s_i through a different set of actions.

Because of this lack of statistical consistency, we do not recommend ECR to compare representations in high stakes domains where accurate evaluation is of the highest importance.

Appendix B

ONLINE ALGORITHM EVALUATION

B.1 Bandit Algorithm Proof

Theorem C.1. *Queue Replay Evaluation estimates are unbiased conditioned on the fact that the algorithm produces a sequence of actions for which we issue estimates. Specifically,*

$$\mathbb{E}\left[r_t | s \in U_t\right] = \sum_{s'=\{i,\ldots,j\} \in U_t} p(s'|s' \in U_t, \theta)\mu_j$$

.

Proof. The proof builds on Lemma 4 in [75]. We can prove this by induction: At the initial step, the probability of the sequence and the expected value of the estimate at time 0 are trivially the same as in the real environment. Now, at time t, we know that the probability of seeing any past sequence of samples s'' is $p(s''|s'' \in U_{t-1}, \theta)$, and we also know all $t-1$ past estimates were issued with the correct prediction. So take any one of those sequences s'' (of length $t-1$). Now, the distribution over states of the algorithm given the past sequence of pulls is identical to the true system, since the state of the algorithm is defined by the past sequence of pulls and the rewards, and rewards were drawn iid from each arm. Therefore, $\hat{p}(j \text{ pulled}|s'') = p(j \text{ pulled}|s'')$, where $\hat{p}$ denotes the probability when running the queue-based evaluator. Therefore the probability of extending any $s'' \in U_{t-1}$ to any t-length sequence s' is the same as in the real environment, that is $\hat{p}(s'|s'' \in U_{t-1}) = p(s'|s'' \in U_{t-1})$. Therefore the marginal distribution $\hat{p}(s'|s'' \in U_{t-1}, s' \in U_t) = p(s'|s'' \in U_{t-1}, s' \in U_t)$. Now, given $s' \in U_t$, $s'' \in U_{t-1}$ since we must have not hit the end of the queue on step $t-1$ as well, so $\hat{p}(s'|s' \in U_t) = p(s'|s' \in U_t)$. Finally, if we pull arm j, the result is iid, so the expectation of r_t is μ_j. □

B.2 RL Algorithm Details

B.2.1 Queue Based RL Evaluation: Termination Upon Empty Queue

One question raised by the queue-based method is why do we terminate as soon as we hit an empty stream (meaning, run out of (r, s') tuples at the current (s, a))? Especially in an episodic setting, why not just throw away the episode, and start a new episode? The reason is that this would no longer draw samples from the true environment. To see why, imagine that after starting in s_I, half the time the agent goes to s_1, otherwise it goes to s_2. The candidate algorithm $\mathbb{A}$ picks action 1 in s_1, but the sampling policy avoids it 99% of the time. In s_2, both sampling and candidate approaches pick each action with equal probability. If we run a "restart if no data" approach, it is very unlikely to ever include a transition from s_I to s_1, since there aren't many episodes going through s_1 that picked the action we chose, causing us to quickly run out of data. So the algorithm will almost always be fed s_2 after s_I, leading to the incorrect distribution of states. Our approach does not have this problem, since in this case it will stop as soon as it hits a lack of data at s_2, leading to a balanced number of samples. However, a downside of this approach is that with very large spaces, we may not be able to produce long sequences of interactions, since in these scenarios we may very quickly run into an empty queue.

B.2.2 Calculating M for PERS

Let $\pi_b(e)$ be shorthand for $\Pi_{t=0}^{l(e)}\pi_b(a_t|h_t)$, and similarly for $\pi_e(e)$

In order to ensure that rejection sampling returns a sample from the candidate distribution, it is critical that M be set correctly. One way to understand the necessity of M is to consider that since the ratio $\frac{\pi_b(e)}{\pi_e(e)}$ can grow extremely large, we need an M such that $\frac{\pi_b(e)}{M\pi_e(e)}$ is a probability between 0 and 1. Therefore, $M \geq \max_e \frac{\pi_b(e)}{\pi_e(e)}$ Obviously, taking the maximum over all possible episodes is a bit worrisome from a computational standpoint, although in some domains it may be feasible. Alternatively, one can always use an overestimate. Some examples that may be useful are: $\max_T \frac{\max_{es.t.l(e)=T}\pi_b(e)}{\min_{es.st.l(e)=T}\pi_e(e)}$ or $\frac{\max_e \pi_b(e)}{\min_e \pi_e(e)}$ or even $\frac{1.0}{\min_e \pi_e(e)}$. How-

Algorithm 11 Efficient M calculation

Input: Candidate policy π_b, Exploration policy π_e, common state space $\mathcal{S}$,
Binary transition matrix T denoting whether a nonzero transition probability from s and s' is possible a priori, maximum horizon H.
Initialize $M_s = 1.0$ for all $s \in \mathcal{S}$
for t = 1 to H **do**
    for $s \in \mathcal{S}$ **do**
        $M'_s = max_a \frac{\pi_b(s,a)}{\pi_e(s,a)} max_{s'} T(s,a,s') M_{s'}$
    $M = M'$
return M_{s_I}

ever, one needs to be careful that the overestimate is not too extreme, or else rejection sampling will accept very few samples, since an overly large M lowers all probabilities (for example, doubling M means all probabilities will be normalized to [0,0.5]).

In certain cases, even if we are not willing to assume a state space for the purposes of evaluation, we know that π_b uses a discrete state space $\mathcal{S}_1$ and π_e uses a discrete state space $\mathcal{S}_2$. In this case we can formulate a common state space, either as a cross product of the two spaces or by observing that one is contained within the other, which allows us to avoid an exponential enumeration of histories as follows. Assume $\mathcal{A}$ does make use of additional internal randomness over the course of a single episode, and the horizon is upper bounded, and we have access to a binary transition matrix T denoting whether a nonzero transition probability from s and s' is possible a priori. Then the maximum from an episode starting at each state M_s, satisfies the following recursion:

$$M_{s,0} = 1.0$$

$$M_{s,t} = max_a \frac{\pi_b(s,a)}{\pi_e(s,a)} max_{s'} T(s,a,s') M_{s',t-1}$$

One can use a dynamic programming approach to efficiently solve this recurrence. See

algorithm 11.

B.2.3 PSRS: Self-Idempotent or Not?

If we randomize the data in our streams and use PSRS, scenarios such as the following can occur. Imagine we have three states, s_1, s_2, and s_3, and a candidate policy that is identical to the sampling policy, so that rejection sampling accepts samples with probability 1. In our initial dataset, assume we took 1 transition from s_1 to s_2, N transitions from s_2 to s_2, and one transition from s_2 to s_3. If we keep the order fixed, the trace will accept every transition in the order they occurred and thus behave exactly the same as the sampling policy, accepting all data. But, if we randomize the order, as will happen in general, we are very likely to spend less than N transitions in s_2 before we draw the tuple from $Q[s_2]$ that causes us to transition to s_3, leaving the remaining samples at s_2 uncollected.

B.3 Sample from True Distribution Guarantees

B.3.1 Properties of Rejection Sampling

Rejection sampling is guaranteed to return a sample from $p(x)$ given an infinite stream of samples from $q(x)$. However, in practice we only have a finite stream of size N and would like to know whether, conditioned on the fact that rejection sampling outputs an estimate before consuming the dataset, the accepted sample is distributed according to $p(x)$. Or formally,

Lemma B.3.1. *$P_R(x = a | r_1 \vee \cdots \vee r_N) = P_E(a)$, where P_R denotes the probability (or pdf) under rejection sampling, P_E denotes the probability under the candidate distribution, and r_i means all samples before the i^{th} are rejected, and x is accepted on the i^{th} sample.*

Proof. Proof by induction on N. The base case (N=0) is trivial because we never return an estimate given zero data. Assume $P_R(x = a | r_1 \vee \cdots \vee r_{N-1}) = P_E(x = a)$ and show for N.

$$P_R(x = a|r_1 \vee \ldots \vee r_N)$$
$$= P_R(r_N)P_R(x = a|r_N)$$
$$+ (1 - P_R(r_N))P_R(x = a|r_1 \vee \cdots \vee r_{N-1})$$

By the inductive hypothesis we have:

$$P_R(x = a|r_1 \vee \cdots \vee r_N)$$
$$= P_R(r_N)P_R(x = a|r_N) + (1 - P_R(r_N))P_E(x = a)$$

So it suffices to show $P_R(x = a|r_N) = P_E(x = a)$.

$$P_R(x = a|r_N) = \frac{P_R(x = a, r_N)}{P_R(r_N)}$$

$$P_R(x = a|r_N) = \frac{P_R(x = a, r_N)}{\sum_b P_R(x = b, r_N)}$$

Since we perform rejection sampling:

$$= \frac{q(a)\frac{P_E(a)}{Mq(a)}}{\sum_b q(b)\frac{P_E(b)}{Mq(b)}}$$

where q is the sampling distribution.

$$= \frac{\frac{P_E(a)}{M}}{\sum_b \frac{P_E(b)}{M}}$$

$$= \frac{\frac{P_E(a)}{M}}{\frac{1}{M}}$$

$$= \frac{MP_E(a)}{M}$$

$$= P_E(a)$$

□

B.3.2 A note on the base case

Note that our guarantees say that, if our evaluators do not terminate, our the produced pair of tuple (or episode) and χ comes from the correct distribution given some history. However, these guarantees does not explicitly address how the initial history H_0 is chosen. The following lemma (with trivial proof) addresses this issue explicitly:

Lemma B.3.2. *Under evaluators Queue, PSRS, and PERS, $P_{\mathcal{R}}(H_0) = P_{\mathcal{E}}(H_0)$.*

Proof. For all of our evaluators, the initial state is correctly initialized to s_I, and the initial χ is drawn correctly according to $\mathbb{A}$, so the initial history H_0 (consisting of initial state and initial χ) is drawn from the correct distribution. □

B.3.3 Queue-based evaluator guarantees

Theorem C.1. *Assuming the environment is an MDP with state space $\mathcal{S}$ and the randomness involved in drawing from π_b is treated as internal to $\mathbb{A}$, given any history of interactions H_T, if the queue-based evaluator produces a (s, a, r, s') tuple, the distribution of this tuple and subsequent internal randomness χ under the queue-based evaluator is identical to the true distribution the agent would have encountered if it was run online. That is, $P_{\mathcal{R}}(s, a, r, s', \chi | H_T, g_T) = P_{\mathcal{E}}(s, a, r, s', \chi | H_T)$, which gives us that $P_{\mathcal{R}}(H_{T+1} | H_T, g_T) = P_{\mathcal{E}}(H_{T+1} | H_T)$.*

Proof. Recall that for the queue-based evaluator we treat the randomness involved in drawing an action from the distribution π_b as part of the χ stored in the history. Therefore, given H_T (which includes χ), $\mathcal{A}$ deterministically selects a_T given H_T. Given an MDP and the history of interactions H_T at timestep t, and assuming for convenience $s_{-1} = s_I$,

$P_{\mathcal{E}}(s, a, r, s'|H_T) = \mathbb{I}(s = s'_{t-1}, a = a_t)P_{\mathcal{E}}(r, s'|s, a)$, where the conditional independences follow from the fact that in an MDP, the distribution of (r, s') only depends on s, a. Under our evaluator, $P_{\mathcal{R}}(s, a, r, s'|H_T, g_T) = \mathbb{I}(s = s'_{t-1}, a = a_T)P_{\mathcal{R}}(Q[s, a].pop() = (r, s'))$, since the state is properly translated from one step to the next, and the action a is fixed to a_T. So we just need to show $P_{\mathcal{R}}(Q[s, a].pop() = (r, s')) = P_{\mathcal{E}}(r, s'|s, a)$. But since the (r, s') tuple at the front of our $Q[s, a]$ was drawn from the true distribution[1] given s, a, but independent of the samples in our history, it follows immediately that $P_{\mathcal{R}}(Q[s, a].pop() = (r, s')) = P_{\mathcal{E}}(r, s'|s, a)$, and thus $P_{\mathcal{R}}(s, a, r, s'|H_T, g_T) = P_{\mathcal{E}}(s, a, r, s'|H_T)$. Given an (s, a, r, s'), the algorithm's internal randomness χ is drawn from the correct distribution and thus $P_{\mathcal{R}}(s, a, r, s', \chi|H_T, g_T) = P_{\mathcal{E}}(s, a, r, s', \chi|H_T)$. □

B.3.4 Per-state RS evaluator Guarantees

Theorem 4.5.1. *Assume the environment is an MDP with state space $\mathcal{S}$, π_e is known, and and for all a, $\pi_e(a) > 0$ if $\pi_b(a) > 0$. Then if the evaluator produces a (s, a, r, s') tuple, the distribution of (s, a, r, s') tuple returned by PSRS (and subsequent internal randomness χ) given any history of interactions H_T is identical to the true distribution the agent would have encountered if was run online. Precisely, in the case that we accept a tuple, $P_{\mathcal{R}}(s, a, r, s', \chi|H_T) = P_{\mathcal{E}}(s, a, r, s', \chi|H_T)$, which gives us that $P_{\mathcal{R}}(H_{T+1}|H_T) = P_{\mathcal{E}}(H_{T+1}|H_T)$.*

Proof. Note that the candidate distribution π_b is deterministically output given H_T, since H_T includes any internal randomness χ by definition.

Given some H_T at timestep T, and assuming for convenience s_{-1} is a fixed start state, we know $s = s'_{t-1}$. So, we accept each (a, r, s') tuple with probability:

$$\frac{\pi_b(a|s)}{M * \pi_e(a|s)}$$

[1]For further discussion of this property of queues see Joulani, Gyorgy, and Szepesvari [75]

where $M = \max_a \frac{\pi_b(a|s)}{\pi_e(a|s)}$ always. Equivalently, we can say we accept each tuple with probability:

$$\frac{\pi_b(a|s)P_{\mathcal{E}}(r, s'|s, a)}{M * \pi_e(a|s)P_{\mathcal{E}}(r, s'|s, a)}$$

Let $P_{explore}$ denote the probability in the true environment under the sampling policy. Then we have the probability of accepting this tuple is:

$$\frac{P_{\mathcal{E}}(a, r, s'|s, H_T)}{M * P_{explore}(a, r, s'|s)}$$

where the conditioning on H_T is introduced because the specific π_b chosen depends on H_T. We can write $M = \max_a \frac{\pi_b(a|s)}{\pi_e(a|s)} = \max_{a,r,s'} \frac{\pi_b(a|s)}{\pi_e(a|s)} = \max_{a,r,s'} \frac{\pi_b(a|s)P_{\mathcal{E}}(r,s'|s,a)}{\pi_e(a|s)P_{\mathcal{E}}(r,s'|s,a)} = \max_{a,r,s'} \frac{P_{\mathcal{E}}(a,r,s'|s,H_T)}{P_{explore}(a,r,s'|s)}$. The (a, r, s') tuples in each PSRS stream Q[s] are drawn according to $P_{explore}(a, r, s'|s)$, since the actions in each stream are drawn according to $\pi_e(a|s)$ and r and s' are then drawn according to $P_{\mathcal{E}}(r, s'|s, a)$ by the Markov assumption. Therefore, since we draw (a, r, s') according to $P_{explore}(a, r, s'|s)$ but we wish to draw from $P_{\mathcal{E}}(a, r, s'|s, H_T)$, this is a straightforward application of rejection sampling. Since $M \geq \max_{a,r,s'} \frac{P_{\mathcal{E}}(a,r,s'|s,H_T)}{P_{explore}(a,r,s'|s)}$, rejection sampling guarantees that a returned sample is sampled according to $P_{\mathcal{E}}(a, r, s'|s, H_T)$, even if conditioned on only having a finite dataset (Lemma B.3.1). In other words, $P_{\mathcal{R}}(a, r, s'|s, H_T, g_T) = P_{\mathcal{E}}(a, r, s'|s, H_T)$. Since in both cases s is deterministically extracted from the last tuple of H_T, this implies $P_{\mathcal{R}}(s, a, r, s'|H_T, g_T) = P_{\mathcal{E}}(s, a, r, s'|H_T)$

Given an (s, a, r, s'), the algorithm's internal randomness χ is drawn from the correct distribution and thus $P_{\mathcal{R}}(s, a, r, s', \chi|H_T, g_T) = P_{\mathcal{E}}(s, a, r, s', \chi|H_T)$. □

B.3.5 Per-Episode Rejection Sampling Guarantees

Theorem 4.6.1. *Assuming π_e is known, and $\pi_b(e) > 0 \rightarrow \pi_e(e) > 0$ for all possible episodes e and all π_b, and PERS outputs an episode e, then the distribution of e (and subsequent internal randomness χ) given any history of episodic interactions H_T using PERS is identical to the true distribution the agent would have encountered if it was run online. That is,*

$P_{\mathcal{E}}(e, \chi|H_T) = P_{\mathcal{R}}(e, \chi|H_T, g_T)$, *which gives us that* $P_{\mathcal{R}}(H_{T+1}|H_T, g_T) = P_{\mathcal{E}}(H_{T+1}|H_T)$.

Proof. An episode e consists of some sequence of actions, observations and rewards $o_0, a_0, r_0, o_1, a_1, r_1, \ldots$. Let χ_t denote the internal randomness generated by algorithm $\mathbb{A}$ after receiving r_t and o_{t+1}, so that $\chi = \chi_0, \ldots, \chi_{l(e)}$. Let h_t denote the within-episode history at time t, namely $h_t = o_0, \ldots, o_t, a_0, \ldots, a_{t-1}, r_0, \ldots, r_{t-1}, \chi_0, \ldots, \chi_{t-1}$. Given H_T, for notational convenience we assume the algorithm $\mathbb{A}$ outputs a single policy π_b which maps h_t (recall this includes any internal randomness) to action probabilities, that is $\mathbb{A}$ chooses actions according to $\pi_b(a_t|h_t)$. The sampling policy likewise chooses actions according to $\pi_e(a_t|h_t)$ (however the χ component of h_t is not used by the sampling policy). The environment generates observations and rewards from some unknown distribution $P_{\mathcal{E}}$ given the past history, in other words according to $P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t)$.[2] We accept episodes with probability:

$$\frac{\prod_{t=0}^{l(e)} \pi_b(a_t|h_t)}{M \prod_{t=0}^{l(e)} \pi_b(a_t|h_t)}$$

where $M \geq \frac{\prod_{t=0}^{l(e)} \pi_b(a_t|h_t)}{\prod_{t=0}^{l(e)} \pi_e(a_t|h_t)}$ always. This can also be written as:

$$\frac{\prod_{t=0}^{l(e)} \pi_b(a_t|h_t) P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t) P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)}{M \prod_{t=0}^{l(e)} \pi_e(a_t|h_t) P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t) P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)} \tag{B.1}$$

Let $P_{explore}(e)$ denote the probability of episode e under the exploration policy π_e in the true environment. Then we have the probability of accepting this episode is:

$$\frac{P_{\mathcal{E}}(e, \chi|H_T)}{M * P_{explore}(e) \prod_{t=0}^{l(e)} P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)} \tag{B.2}$$

where the conditioning on H_T in the numerator is introduced because how $\mathbb{A}$ updates π_b depends on H_T. We can write $M \geq \max_e \frac{\prod_{t=0}^{l(e)} \pi_b(a_t|h_t) P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t) P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)}{\prod_{t=0}^{l(e)} \pi_e(a_t|h_t) P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t) P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)} = \max_e \frac{P_{\mathcal{E}}(e, \chi|H_T)}{P_{explore}(e) \prod_{t=0}^{l(e)} P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)}$. The episodes e in our dataset are drawn according to $P_{explore}(a, r, s'|s)$, since the action at each timestep is drawn according to $\pi_e(a_t|h_t)$, and then

[2] We assume whether the episode continues or not is an observation, i.e. is it also drawn from an unknown distribution given the history.

the next reward and observation are drawn according to $P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t)$ by the episodic assumption. And during the operation of PERS $\mathbb{A}$ draws the internal randomness χ according to the correct distribution at each step, $P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)$. Since we draw e, χ according to $P_{explore}(e)\prod_{t=0}^{l(e)} P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)$, but wish to draw from $P_{\mathcal{E}}(e, \chi)$, this is a straightforward application of rejection sampling. Since $M \geq \max_e \frac{P_{\mathcal{E}}(e,\chi|H_T)}{P_{explore}(e)\prod_{t=0}^{l(e)} P_E(\chi_t|h_t,r_t,o_{t+1},H_T)}$, rejection sampling guarantees that any returned sample is sampled according to $P_{\mathcal{E}}(e, \chi|H_T)$, even if conditioned on only having a finite dataset (Lemma B.3.1). So $P_{\mathcal{R}}(e, \chi|H_T, g_T) = P_{\mathcal{E}}(e, \chi|H_T)$.

□

B.4 Empirical Performance of Importance-Weighted Fixed-M PERS

Despite the stronger guarantees, one should be cautious about interpreting the empirical results generated by the variant proposed in Theorem 4.7.2. Although the expectation is correct, for a single run of the algorithm, the estimates will tend to rise above their true value due to the importance weights (leading one to believe the algorithm is learning more than it truly is) before abruptly dropping to zero. Averaging together multiple unbiased estimates is more likely to give a better picture of behavior.

B.5 Unbiasedness Proofs

Theorem 4.7.1. *If M is held fixed throughout the operation of the per-episode rejection sampling replayer, π_e is known, and $\pi_b(e) > 0 \rightarrow \pi_e(e) > 0$ for all possible episodes e and all π_b, then if the evaluator outputs an estimate of some function $f(H_T)$ at episode T, that estimate is an unbiased estimator of $f(H_T)$ at episode T, in other words, $\mathbb{E}_{\mathcal{R}}[f(H_T)|g_T, \ldots, g_1] = \sum_{H_T} f(H_T)P_{\mathcal{E}}(H_T) = \mathbb{E}_{\mathcal{E}}[f(H_T)]$. For example, if $f(H_T) = R^{\mathbb{A}}(T)$, the estimate is an unbiased estimator of $R^{\mathbb{A}}(T)$ given $g_T, \ldots, g_1$.*

Proof Sketch. We first show that if M is fixed, the probability that each episode is accepted is constant $(1/M)$. This allows us to show that whether we continue or not (g_T)

is conditionally independent of H_{T-1}. This lets us remove the conditioning on H_{T-1} in Theorem 4.6.1 to give us that $P_{\mathcal{R}}(H_T|g_T) = P_{\mathcal{E}}(H_T)$, meaning the distribution over histories after T accepted episodes is correct, from which conditional unbiasedness is easily shown.

Proof. First, we will calculate the probability (over randomization in the dataset, algorithm, and evaluator) that we accept the i^{th} episode in our dataset, e_i, given some H_T (over some sequence $\mathbb{S}$ of acceptances/rejections in the first $i-1$ episodes).

$$P(\text{Accept } e_i|H_T, \mathbb{S}) = \sum_e P(e_i = e)P(e \text{ is accepted } |H_T, \mathbb{S}) \tag{B.3}$$

If we let $q(e)$ refer to the probability of the episode under the sampling distribution,

$$P(\text{Accept } e_i|H_T, \mathbb{S}) = \sum_e q(e)P(e \text{ is accepted } |H_T, \mathbb{S}) \tag{B.4}$$

Recall that as part of Theorem 4.6.1 (equation (B.1)) we showed the per-episode rejection replayer accepted an episode with probability

$$\frac{\prod_{t=0}^{l(e)} \pi_b(a_t|h_t)P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t)P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)}{M \prod_{t=0}^{l(e)} \pi_e(a_t|h_t)P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t)P_E(\chi_t|h_t, r_t, o_{t+1}, H_T)}$$

$$= \frac{\prod_{t=0}^{l(e)} \pi_b(a_t|h_t)P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t)}{M \prod_{t=0}^{l(e)} \pi_e(a_t|h_t)P_{\mathcal{E}}(r_t, o_{t+1}|a_t, h_t)}$$

$$= \frac{P_{\mathcal{E}}(e|H_T)}{Mq(e)}$$

where $P_{\mathcal{E}}(e|H_T)$ refers to the probability of episode e under the $\mathbb{A}$ given H_T, and M denotes the normalizer, which is constant in this variant of the algorithm. So we have:

$$P(\text{Accept } e_i|H_T, \mathbb{S}) = \sum_e P(e \text{ sampled } |H_T, \mathbb{S})\frac{P_{\mathcal{E}}(e|H_T)}{Mq(e)} \tag{B.5}$$

$$P(\text{Accept } e_i|H_T, \mathbb{S}) = \sum_e q(e)\frac{P_{\mathcal{E}}(e|H_T)}{Mq(e)} \tag{B.6}$$

$$P(\text{Accept } e_i|H_T, \mathbb{S}) = \frac{1}{M}\sum_e P_{\mathcal{E}}(e|H_T) \tag{B.7}$$

And since $\sum_e P_{\mathcal{E}}(e|H_T) = 1$,

$$P(\text{Accept } e_i|H_T, \mathbb{S}) = \frac{1}{M} \tag{B.8}$$

$P(\text{Accept } e_i|H_T, \mathbb{S})$ is a constant it is independent of H_T and $\mathbb{S}$, so:

$$P(\text{Accept } e_i|H_T, \mathbb{S}) = P(\text{Accept } e_i) = \frac{1}{M} \tag{B.9}$$

So we accept the i^{th} episode with probability $1/M$, where M is a constant, and therefore independent of $\mathbb{S}$ and H_T. We will now proceed to show that since this is a constant, it does not cause any histories to be more likely than others, from which the unbiased property will follow.

Next, we will prove by induction that $P_{\mathcal{R}}(H_T|g_T, \ldots, g_0) = P_{\mathcal{E}}(H_T)$. Recall that H_T denotes a trajectory of T episodic interactions, g_T denotes the event that we continue[3] (i.e. do not terminate evaluation) from time $T-1$ to time T, $P_{\mathcal{R}}(x)$ denotes the probability of x under the replayer, and $P_{\mathcal{E}}(x)$ denotes the probability of x in the true (online) environment under the candidate policy. The base case is trivial (Since g_0 always occurs, $P_{\mathcal{R}}(H_0|g_0) = P_{\mathcal{R}}(H_0)$, and by Lemma B.3.2, $P_{\mathcal{R}}(H_0) = P_{\mathcal{E}}(H_0)$). We now assume this holds for $T-1$ and we will show it holds for T.

$$\begin{aligned} &P_{\mathcal{R}}(H_T|g_T, \ldots, g_0) \\ &\quad = \sum_{H_{T-1}} P_{\mathcal{R}}(H_T|H_{T-1}, g_T, \ldots, g_0) P_{\mathcal{R}}(H_{T-1}|g_T, \ldots, g_0) \end{aligned} \tag{B.10}$$

Our next step is to show that $g_T \perp H_{T-1}|g_{T-1}$, so that we can turn the right term of equation (B.10) into the inductive hypothesis ($\perp$ denotes independance). Let i be the

[3]Since we always generate the initial history H_0, for the purposes of induction we here condition on g_0 even though that is not strictly necessary since it always occurs.

number of episodes consumed after accepting the $(T-1)^{th}$ episode, so that there are exactly $N-i$ episodes remaining after accepting the $(T-1)^{th}$. Since, as we showed in equation (B.9), $P(\text{Accept } e_i) = \frac{1}{M}$, given g_{T-1} and $N-i$ episodes remaining, g_T is drawn from a $Bernoulli(1-(1-\frac{1}{M})^{N-i})$ distribution. So, since M and N are both constants, $g_T \perp H_{T-1}|g_{T-1}, i$. So we can write:

$$P(g_T, H_{T-1}|g_{T-1}) = \sum_i P(g_T, H_{T-1}|g_{T-1}, i)P(i|g_{T-1}) \tag{B.11}$$

$$= \sum_i P(g_T|g_{T-1}, i)P(H_{T-1}|g_{T-1}, i)P(i|g_{T-1}) \tag{B.12}$$

So we need to show $H_{T-1} \perp i|g_{T-1}$.

$$P(H_{T-1}, i|g_{T-1}) = \frac{1}{P(g_{T-1})}P(H_{T-1}, g_{T-1}, i) \tag{B.13}$$

Now, we know the i^{th} episode must have been accepted, but the $(T-2)^{th}$ acceptances could have occurred anywhere in the $(i-1)^{th}$ steps. The probability of the j^{th} e, χ pair in the history, $H_{T-1}[j]$ being accepted on the next episode given some sequence of previously accepted/rejected episodes $\mathbb{S}$ is

$$\begin{aligned} &P(H_{T-1}[j] \text{ produced}|H_{j-1}, \mathbb{S}) \\ &= P(H_{T-1}[j] \text{ sampled}|H_{j-1}, \mathbb{S})P(H_{T-1}[j] \text{ accepted}|H_{j-1}, \mathbb{S}) \\ &= q(H_{T-1}[j]|H_{j-1})\frac{P_{\mathcal{E}}(H_{T-1}[j]|H_{j-1})}{Mq(H_{T-1}[j]|H_{j-1})}, \end{aligned} \tag{B.14}$$

where $q(H_{T-1}[j]|H_{j-1})$ denotes the probabilities of producing the j^{th} e, χ tuple in the history given H_{j-1} and sampling actions according to π_e, and $P(H_{T-1}[j]$ accepted) is derived as per equation (B.2) in Theorem 4.6.1. Equation (B.14) can be simplified:

$$P(H_{T-1}[j] \text{ produced}|H_{j-1}, \mathbb{S}) = \frac{P_{\mathcal{E}}(H_{T-1}[j]|H_{j-1})}{M} \tag{B.15}$$

Note that this depends on H_{j-1} but is independent of $\mathbb{S}$. The probability of rejection given $\mathbb{S}, H_{j-1}$ is a constant $1 - \frac{1}{M}$ (see equation (B.9)), so returning to equation B.13 we have:

$$P(H_{T-1}, i|g_{T-1}) \tag{B.16}$$

$$= \frac{1}{P(g_{T-1})} \sum_{\mathbb{S} \text{ s.t. } \mathbb{S} \text{ compat } (i, T-1)} P(\mathbb{S}, H_{T-1}) \tag{B.17}$$

where ($\mathbb{S}$ compat $(i, T-1)$), means $|\mathbb{S}| = i$, there are $T-1$ acceptances in $\mathbb{S}$, and $\mathbb{S}[i-1]$ is not a rejection. Now, computing $P(\mathbb{S}, H_{T-1})$ would consist of multiplying quantities like P(reject episode k | $\mathbb{S}$ up to $k-1$)
$* P$(accept $H_{T-1}[3]$ on episode k+1 | H_2, $\mathbb{S}$ up to k)$* \ldots$. Note, however, that the probability of rejecting an episode is independent of the particular past sequence of acceptances and rejections (see equation (B.9)), and so is the probability of accepting the next item in the history (see equation (B.15)). Therefore, for every $\mathbb{S}$, $P(\mathbb{S}, H_{T-1})$ is the multiplication of the probability of the initial history $P(H_0)$ with $i - T + 1$ rejection probabilities together with the probabilities of accepting the $T-1$ items in the history. There are $\binom{i-1}{T-2}$ possible different sequences $\mathbb{S}$ such that ($\mathbb{S}$ compat $(i, T-1)$), since the last element is always accepted. Therefore:

$$\begin{aligned} & P(H_{T-1}, i|g_{T-1}) \\ &= \frac{P(H_0)}{P(g_{T-1})} \binom{i-1}{T-2} (1 - \frac{1}{M})^{i-T+1} \prod_{j=1}^{T-1} \frac{P_{\mathcal{E}}(H_{T-1}[j]|H_{j-1})}{M} \end{aligned} \tag{B.18}$$

since the probability of rejecting the k^{th} episode is $(1 - \frac{1}{M})$ from (B.9), and from (B.15) the probability of accepting the j^{th} element in the history on the k^{th} episode is $\frac{P_{\mathcal{E}}(H_{T-1}[j]|H_{j-1})}{M}$.

By Lemma B.3.2, $P_{\mathcal{R}}(H_0) = P_{\mathcal{E}}(H_0)$, so:

$$\begin{aligned} & P(H_{T-1}, i|g_{T-1}) \\ =& \frac{P_{\mathcal{E}}(H_0)}{P(g_{T-1})}\binom{i-1}{T-2}(1-\frac{1}{M})^{i-T+1}\prod_{j=1}^{T-1}\frac{P_{\mathcal{E}}(H_{T-1}[j]|H_{j-1})}{M} \end{aligned} \tag{B.19}$$

$$\begin{aligned} =&(P_{\mathcal{E}}(H_0)\prod_{j=1}^{T-1}P_{\mathcal{E}}(H_{T-1}[j]|H_{j-1}))\frac{1}{P(g_{T-1})} \\ &* \binom{i-1}{T-2}(1-\frac{1}{M})^{i-T+1}\prod_{j=1}^{T-1}\frac{1}{M} \end{aligned} \tag{B.20}$$

Where (B.20) follows by simply reordering the multiplication. Now by definition, $P_{\mathcal{E}}(H_{T-1}) = P_{\mathcal{E}}(H_0)\prod_{j=1}^{T-1}P_{\mathcal{E}}(H_{T-1}[j]|H_{j-1})$, so we have:

$$\begin{aligned} & P(H_{T-1}, i|g_{T-1}) \\ =& P_{\mathcal{E}}(H_{T-1})\frac{1}{P(g_{T-1})}\binom{i-1}{T-2}(1-\frac{1}{M})^{i-T+1}\prod_{j=1}^{T-1}\frac{1}{M} \end{aligned} \tag{B.21}$$

Now by induction, $P_{\mathcal{E}}(H_{T-1}) = P_{\mathcal{R}}(H_{T-1}|g_{T-1})$, so:

$$\begin{aligned} & P(H_{T-1}, i|g_{T-1}) \\ =& P(H_{T-1}|g_{T-1})\frac{1}{P(g_{T-1})}\binom{i-1}{T-2}(1-\frac{1}{M})^{i-T+1}\prod_{j=1}^{T-1}\frac{1}{M} \end{aligned} \tag{B.22}$$

Note that $\binom{i-1}{T-2}(1-\frac{1}{M})^{i-T+1}\prod_{j=1}^{T-1}\frac{1}{M}$ is just the probability of consuming i elements and accepting exactly $T-1$ of them (namely the binomial formula, where the probability of success (acceptance) is $\frac{1}{M}$), so:

$$P(H_{T-1}, i|g_{T-1}) = P(H_{T-1}|g_{T-1})\frac{P(i, g_{T-1})}{P(g_{T-1})} \tag{B.23}$$

By the definition of conditional probability:

$$P(H_{T-1}, i|g_{T-1}) = P(H_{T-1}|g_{T-1})P(i|g_{T-1}) \tag{B.24}$$

So since $P(H_{T-1}, i|g_{T-1}) = P(H_{T-1}|g_{T-1})P(i|g_{T-1})$, $H_{T-1} \perp i|g_{T-1}$. Now returning to equation (B.12) we have:

$$\begin{aligned} & P(g_T, H_{T-1}|g_{T-1}) \\ =& \sum_i P(g_T|g_{T-1}, i)P(H_{T-1}|g_{T-1}, i)P(i|g_{T-1}) \end{aligned} \tag{B.25}$$

Since $H_{T-1} \perp i|g_{T-1}$:

$$P(g_T, H_{T-1}|g_{T-1}) \tag{B.26}$$

$$= \sum_i P(g_T|g_{T-1}, i)P(H_{T-1}|g_{T-1})P(i|g_{T-1}) \tag{B.27}$$

$$= P(H_{T-1}|g_{T-1}) \sum_i P(g_T|g_{T-1}, i)P(i|g_{T-1}) \tag{B.28}$$

$$= P(H_{T-1}|g_{T-1}) \sum_i \frac{P(g_T, g_{T-1}, i)}{P(i, g_{T-1})} \frac{P(i, g_{T-1})}{P(g_{T-1})} \tag{B.29}$$

$$= P(H_{T-1}|g_{T-1}) \frac{\sum_i P(g_T, g_{T-1}, i)}{P(g_{T-1})} \tag{B.30}$$

$$= P(H_{T-1}|g_{T-1}) \frac{P(g_T, g_{T-1})}{P(g_{T-1})} \tag{B.31}$$

$$= P(H_{T-1}|g_{T-1})P(g_T|g_{T-1}) \tag{B.32}$$

Where equation (B.31) follows since i is marginalized out, and (B.32) follows by the definition of conditional probability. So equation (B.32) tells us that $g_T \perp H_{T-1}|g_{T-1}$.

Given this conditional independence, $P_{\mathcal{R}}(H_{T-1}|g_T, g_{T-1}, \ldots, g_1) = P_{\mathcal{R}}(H_{T-1}|g_{T-1}, \ldots, g_0)$, and $P_{\mathcal{R}}(H_{T-1}|g_{T-1}, \ldots, g_0) = P_{\mathcal{E}}(H_{T-1})$ by induction. So $P_{\mathcal{R}}(H_{T-1}|g_T, \ldots, g_0) = P_{\mathcal{E}}(H_{T-1})$. Therefore, picking up from (B.10), we have:

$$\begin{aligned} & P_{\mathcal{R}}(H_T|g_T, \ldots, g_0) \\ &= \sum_{H_{T-1}} P_{\mathcal{R}}(H_T|H_{T-1}, g_T, \ldots, g_0)P_{\mathcal{R}}(H_{T-1}|g_T, \ldots, g_0) \end{aligned} \tag{B.33}$$

$$= \sum_{H_{T-1}} P_{\mathcal{R}}(H_T|H_{T-1}, g_T, \ldots, g_0)P_{\mathcal{E}}(H_{T-1}) \tag{B.34}$$

Now observe that, as we showed in Theorem 4.6.1, given that we update,

the update is drawn from the true distribution given the current history, namely[4] $P_{\mathcal{R}}(H_T|H_{T-1}, g_T, \ldots, g_0) = P_{\mathcal{E}}(H_T|H_{T-1})$. Plugging this in we have:

$$P_{\mathcal{R}}(H_T|g_T, \ldots, g_0) = \sum_{H_{T-1}} P_{\mathcal{E}}(H_T|H_{T-1}) P_{\mathcal{E}}(H_{T-1}) \tag{B.35}$$

$$P_{\mathcal{R}}(H_T|g_T, \ldots, g_0) = \sum_{H_{T-1}} P_{\mathcal{E}}(H_T, H_{T-1}) \tag{B.36}$$

Marginalizing out H_{T-1}:

$$P_{\mathcal{R}}(H_T|g_T, \ldots, g_0) = P_{\mathcal{E}}(H_T) \tag{B.37}$$

Which completes the induction. All that remains is to show that the expectation is correct. For any function f over histories of interactions (such as cumulative reward, final reward, etc.),

$$\mathbb{E}_{\mathcal{R}}[f(H_T)|g_T, \ldots, g_1] = \sum_{H_T} f(H_T) P_{\mathcal{R}}(H_T|g_T, \ldots, g_1) \tag{B.38}$$

Since g_0 always occurs we can freely condition on it:

$$\mathbb{E}_{\mathcal{R}}[f(H_T)|g_T, \ldots, g_1] = \sum_{H_T} f(H_T) P_{\mathcal{R}}(H_T|g_T, \ldots, g_0) \tag{B.39}$$

By (B.37):

$$\mathbb{E}_{\mathcal{R}}[f(H_T)|g_T, \ldots, g_1] = \sum_{H_T} f(H_T) P_{\mathcal{E}}(H_T) \tag{B.40}$$

$$= \mathbb{E}_{\mathcal{E}}[f(H_T)] \tag{B.41}$$

Therefore the estimate is unbiased. □

[4]Recall that g_T implies $g_{T-1}, \ldots, g_0$.

Theorem 4.7.2. *Assuming for each T, $R^{\mathbb{A}}(T)$ is divided by by $\phi = 1 - Binomial(N, 1/M).cdf(T-1)$, and after terminating after k episodes are produced we output 0 as estimates of reward for episodes $k+1, \ldots, N$, and M is held fixed throughout the operation of the per-episode rejection sampling replayer, and π_e is known, and $\pi_b(e) > 0 \rightarrow \pi_e(e) > 0$ for all possible episodes e and all π_b, then the estimate of reward output at each episode $T = 1 \ldots N$ is an unbiased estimator of $R^{\mathbb{A}}(T)$.*

Proof. The expectation of reward at episode T can be written as:

$$\begin{aligned}\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)] =& P(g_T, ..., g_1)\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)|g_T, ..., g_1] \\ &+ P(\neg(g_T, ..., g_1))\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)|\neg(g_T, ..., g_1)]\end{aligned} \tag{B.42}$$

where, as above, $g_T, ..., g_1$ denotes the probability of not terminating before episode T. If we do not reach epsiode T (the second case), modified PERS outputs 0, so:

$$\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)] = P(g_T, ..., g_1)\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)|g_T, ..., g_1] \tag{B.43}$$

Now, in the remaining case (that we reach T), we divide the original value by ϕ. In theorem 4.7.1 we showed that the expectation of the the unweighted estimates conditioned on reaching T was unbiased, giving us:

$$\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)] = P(g_T, ..., g_1)\frac{\mathbb{E}_{\mathcal{E}}[R^{\mathbb{A}}(T)]}{\phi} \tag{B.44}$$

Now, since the probability of accepting each episode i is $1/M$ (see Theorem 4.7.1, equation (B.9)) and there are N total episodes, the probability that we reach episode T (aka $P(g_T, ..., g_1)$) is $P(Binomial(N, 1/M) \geq T) = 1 - Binomial(N, 1/M).cdf(T-1) = \phi$. So:

$$\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)] = \phi\frac{\mathbb{E}_{\mathcal{E}}[R^{\mathbb{A}}(T)]}{\phi} \tag{B.45}$$

$$\mathbb{E}_{\mathcal{R}}[R^{\mathbb{A}}(T)] = \mathbb{E}_{\mathcal{E}}[R^{\mathbb{A}}(T)] \tag{B.46}$$

□

B.6 Experiments

PSRL

PSRL leaves open the choice of prior. Additionally, in many cases, one has additional knowledge that can help speed learning (for example one may know there is only a certain set of discrete reward possibilities, or one may know certain transitions are impossible). A convenient way to incorporate this kind of knowledge, and also simplify the choice of prior, is the relative outcomes framework [8]. In this setting, at each state action pair, one of k discrete outcomes occurs. Given a state, action and outcome, it is possible to deterministically extract the reward and next state. In theory any MDP can be encoded in this framework, but it gives us the ability to limit the space of possibilities to speed learning. Given a set of k discrete outcomes, we must clear a categorical distribution (probability) vector over outcomes. Then we can use the conjugate prior, a k-dimensional Dirichlet. Since we did not have any additional information the prior was flat ($\alpha_i = 1$ for all outcomes i).

Details of comparison to model-based

The comparison shown in Figure 4.4 was done in the SixArms environment [135], an environment with six actions and seven states. For a complete description and diagram of the environment see Strehl et al. 2004 [135]. We defined 14 relative outcomes for use with PSRL, one for remaining put in a state, six for moving to each outer state, six for staying in the same state and receiving one of the non-zero rewards, and one for moving to the start state. We treated this a finite horizon problem, where episodes could be at most length 10. The algorithm evaluated was PSRL [114], with 10 posterior samples after each episode.

The model-based approach works by building the MLE MDP model from data, and sampling from it to generate experience. Specifically, in the relative outcome setting it need only estimate the probability of getting any outcome given a state an action, as well as the

distribution over initial outcomes. These were estimated in the MLE sense, so for example if we had only 1 observed outcome of o at (s,a), 100% probability was put on o at (s,a). For state-action pairs with no data, a uniform distribution over outcomes was used.

To carry out the evaluation, we first calculated the "true" cumulative reward by averaging 1000 runs of PSRL against the true environment. Then, for each evaluator (PSRS and model-based) we sampled 100 different datasets of 100 episodes from SixArms using a uniform sampling policy. We then ran 10 complete runs of the evaluator on that dataset, each of which returned estimates of cumulative reward. For PSRS, we only reported estimates up to the minimum evaluation length across those ten runs[5]. The squared error was computed between the mean of the 10 runs and the true cumulative reward curve. Finally, to compute the MSE, the average over the 100 runs was taken (ignoring cases where no estimate was returned).

Treefrog Treasure Experiment Setup

Treefrog Treasure is an educational fractions game (Figure 4.2). The player controls a frog to navigate levels and jump through numberlines. Our ***action set*** is defined as follows: After giving an in-game pretest we give a first set of lines, then we want to either increase the difficulty level of one parameter (e.g. removing tick marks), stay, or go back, which we encoded as 11 actions. Our ***reward*** is terminal and ranges from -2 to 4. It measures engagement (measured by whether the student quit before the posttest) and learning (measured by pretest-to-posttest improvement). After each step we additionally receive a binary observation informing us whether the student took more than 4 attempts on the last pair of lines. The relative outcomes defined for PSRL included either terminating with some reward (-2 - 4) or continuing with one of the two observations. We used a **state space** consisting of the history of actions and the last observation. The true horizon is 5, but for reasons of data sparsity (induced by a large action space and thus large state space) we varied the horizon

[5]This was following the approach proposed in section 4.2 to mitigate the bias introduced by having a wide variance in evaluation lengths, a problem we discuss in section 4.7.

between 3 and 4 in our experiments.

Our dataset of 11,550 players was collected from BrainPOP.com, an educational website focused on school-aged children. The sampling policy used was semi-uniform and changed from day-to-day. We logged the probabilities for use with the rejection-sampling evaluator, modifying the rejection sampling approaches slightly to recalculate M at every step based on the changing sampling policy.

For PERS we use[6] Algorithm 11 to calculate M, updating it based both on the change in the algorithm and the change in the sampling policy. We also tried a variant which fixed M so as to achieve the unbiasedness guarantees in Theorem 4.7.1. To calculate this M we upper bounded the probability of an episode under our candidate distribution by 1, and calculated the minimum probability our sampling distribution could place on any episode.[7]

Further Treefrog Treasure Results

We also examined an increased horizon of 4. Recall that in Treefrog treasure, more timesteps means a larger state space. Given deterministic policies on this larger state space, all three methods are more or less indistinguishable (Figure B.1) ; however, revealing more randomness causes PERS to overtake PSRS (mean 260.54 vs. 173.52), Figure B.2). As an extreme case, we also tried a random policy in Figure B.3: this large amount of revealed randomness benefits the rejection sampling methods, especially PERS, who evaluates for much longer than the other approaches. PERS outperforms PSRS here because there are small differences between the random candidate policy and the semi-random sampling policy, and thus if PSRS enters a state with little data it is likely to terminate.

Figure B.1: Comparing with 4 timesteps and 1 PSRL posterior sample.

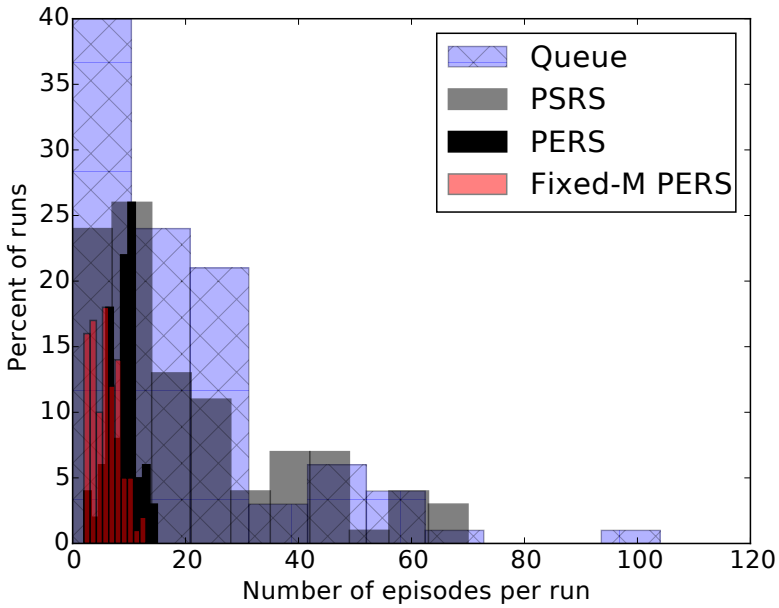


Figure B.2: Comparing with 4 timesteps and 10 PSRL posterior samples.

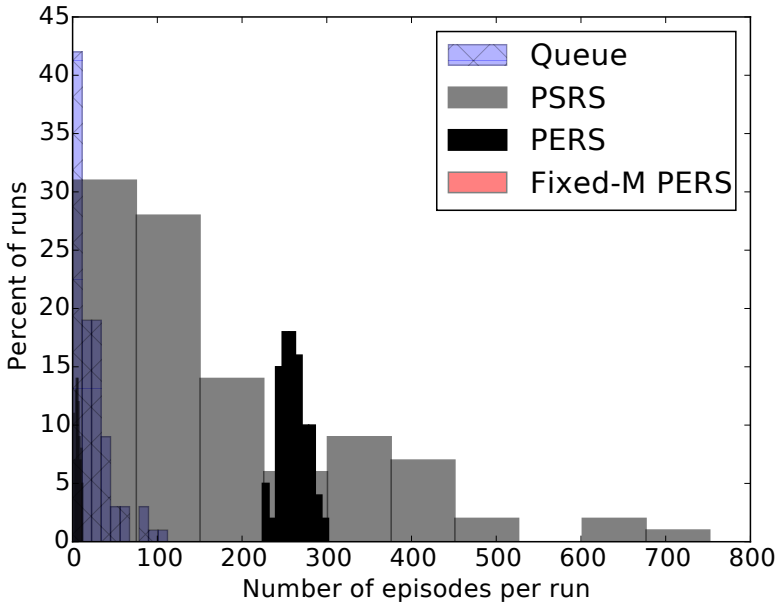


Figure B.3: Comparing a revealed, uniformly-random policy with 4 timesteps.

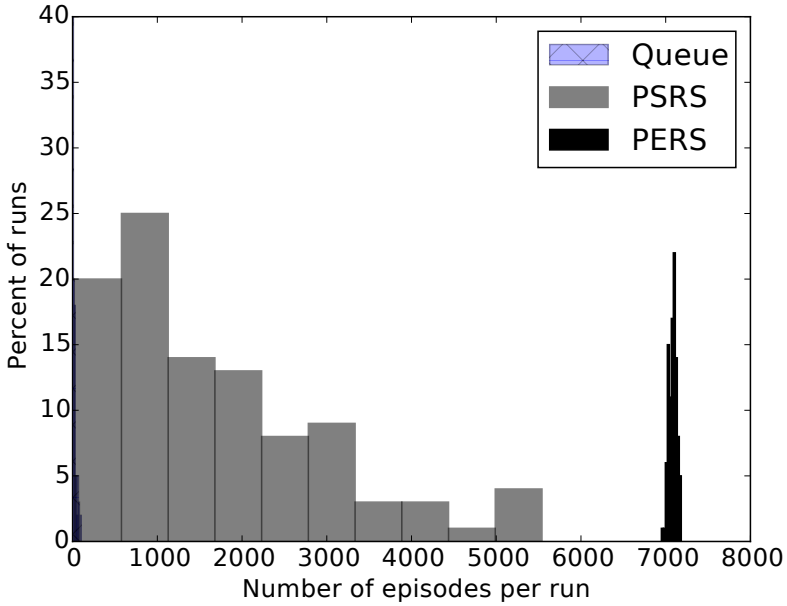


Figure B.4: Comparing on RiverSwim with 10 PSRL posterior samples.

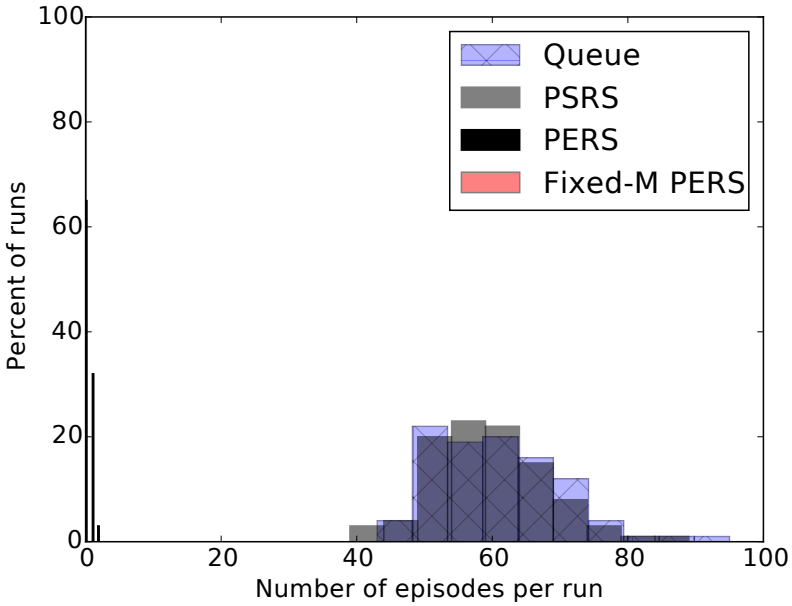


Figure B.5: Comparing on RiverSwim with a uniformly-random policy which does not reveal its randomness. Per-episode rejection sampling is invisible because it always accepts 0 samples.

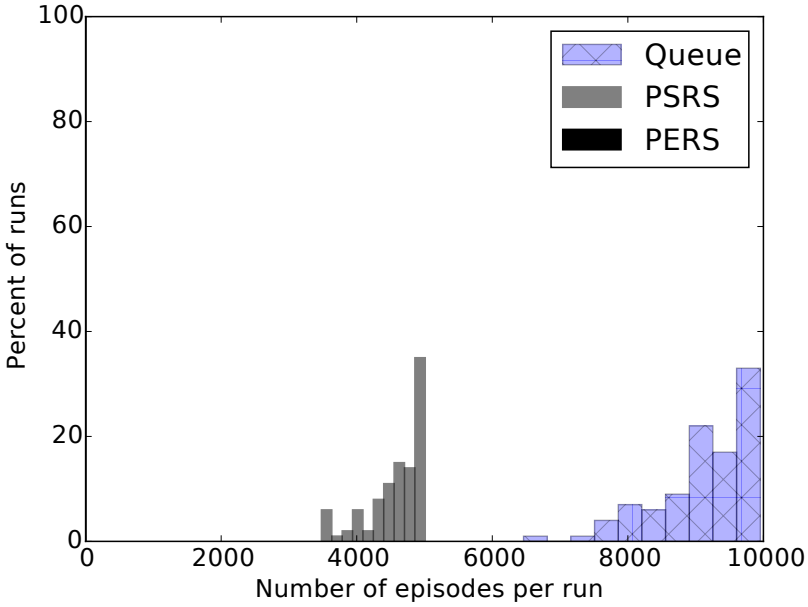
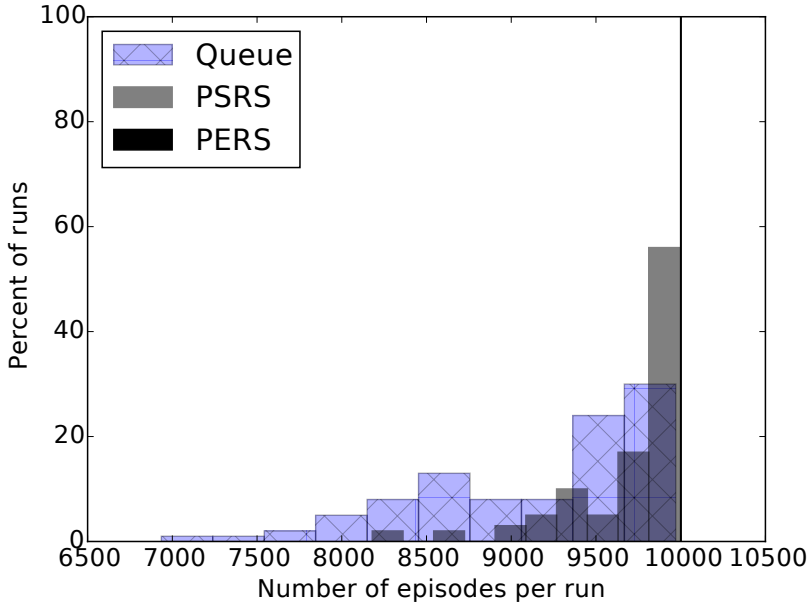


Figure B.6: Comparing on RiverSwim with a uniformly-random policy which reveals its randomness. PERS is the fine line at 10,000, since it always accepts all data.



Riverswim

RiverSwim is a 6-state MDP where one can either stay to the left for small reward, or venture nosily to the right for higher reward. For a complete description of the environment and a diagram see Osband et al. 2013 [114]. In this domain we defined a space of 5 relative outcomes: {*MovedRight, MovedLeft, Staywith0reward, Staywith5/1000reward,Staywith1reward*}. To generate histograms over 100 sizes, for each evaluator we sampled 10 datasets of 10000 episodes from this MDP, using a uniform sampling policy, and used each dataset for 10 runs of evaluation.

The results with 10-sample PSRL are shown in figure B.4. The per-episode rejection sampler was only able to accept a few episodes due to their length, while the two state-based evaluators leveraged the known state space to accept a substantially larger number of episodes. To more clearly understand sensitivity to revealed randomness, we tested two versions of a random policy: one which revealed its randomness and one which did not. In the case where the randomness was hidden (Figure B.5), we see that the per-episode rejection sampling approach does so poorly it never accepts any episodes and thus is hard to see on the graph, the per-state rejection sampling has fairly mediocre performance, and the queue-based method does the best. This is because the per-state rejection sampler, by treating the data as a stream, discards a lot of data for arms it did not choose to pull, unlike the queue-based approach which only discards data when it is consumed by the algorithm. We suspect the effect is particularly visible here for two reasons: (a) There are a small number of states, making it more similar to the bandit case (section 4.2) where Queue does well and (b) the policy is not explicitly driving exploration towards poorly-visited states, which often causes both state-based methods to fail fairly quickly. When the randomness is revealed (Figure B.6), the two rejection sampling approaches improve greatly in performance. One

[6]With straightforward extensions to handle policies that depend on both s and t.

[7]This is probably fairly close to the minimum fixed M, since PSRL randomly generates distributions by sampling N times from the posterior, and we have to consider the worst case over that randomization, which always has a small probability of placing probability 1 on the minimum-probability episode under the sampling distribution.

can observe how the per-episode version is self-idempotent (since the sampling policy was the same as the candidate policy, all 10,000 episodes were accepted) while the per-state is not.

Appendix C

EXPLORATION UNDER DELAY

C.1 Stochastic Delayed Bandits

Theorem B.1. *For algorithm 4 with any choice of procedure* getSamplingDist *and any online bandit algorithm* Base*,*

$$\mathbb{E}R_T \leq \mathbb{E}R_T^{\text{Base}} + \sum_{i=1}^{N} \Delta_i \mathbb{E}S_{i,T} \tag{C.1}$$

where $S_{i,T}$ is the number of elements pulled for arm i by time T, but not yet shown to Base*.*

Proof. Let T' be the number of times we have updated Base. Note that since the samples fed to Base were drawn iid according to the arm distributions and given on the arms it requested, the regret on these samples is $R_{T'}$. Clearly $T' \leq T$ since for every sample we give Base, we must have correspondingly taken a step in the true environment[1]. So the expected regret of only those steps in which we update Base can be upper bounded by

$$\mathbb{E}R_T^{\text{Base}}. \tag{C.2}$$

Now we must consider those timesteps for which we requested a sample but have not given it to Base by time T. The number of such samples from arm i is exactly $S_{i,T}$. So the regret on these timesteps is equal to

$$\sum_{i=1}^{N} \sum_{j=1}^{S_{i,T}} \rho_{i,j} \tag{C.3}$$

[1] We also make the trivial assumption that the provided regret bound monotonically increases with T.

Where $\rho_{i,j}$ denotes the regret of the j^{th} sample not passed to BASE for arm i. Taking the expectation gives us:

$$\sum_{i=1}^{N} \mathbb{E} \sum_{j=1}^{S_{i,T}} \rho_{i,j} \tag{C.4}$$

By Wald's equation this is:

$$= \sum_{i=1}^{N} \mathbb{E} S_{i,T} \mathbb{E} \rho_{i,j} \tag{C.5}$$

$$= \sum_{i=1}^{N} \mathbb{E} S_{i,T} \Delta_i \tag{C.6}$$

Combining this with equation (C.2) gives us the required bound. □

Theorem B.2. *For SDB,*

$$\mathbb{E} R_T \leq \mathbb{E} R_T^{\text{BASE}} + N \tau_{max} \tag{C.7}$$

in the online updating setting, and

$$\mathbb{E} R_T \leq \mathbb{E} R_T^{\text{BASE}} + 2N \tau_{max} \tag{C.8}$$

in the batch updating setting.[2]

Proof. First note that on all timesteps $B_t \leq \tau_{max}$, since $B_1 = 1$ and $\tau_{max} \geq 1$ and for all $t > 1$, B_t is equal to the empirical max delay.

We will now show that in the online updating case, $S_{i,t}$ when running SDB never exceeds τ_{max}. Clearly $S_{i,0} = 0$ and so the bound holds in the base case. Assume $S_{i,t} \leq \tau_{max}$ at t, and we will prove it for $t+1$. Let I_t denote the value of variable I in the SDB algorithm at time

[2] One small technical comment: Note that $\tau_{max} = 1$ refers to the case of no delay, since the arm pulled at time t is observed at time $t+1$. It appears that Joulani et. al. 2013 used $\tau_{max} = 0$ to denote no delay, so there is an off-by-one difference in τ_{max} between their paper and this thesis. This difference simply results in the SDB bound $\mathbb{E} R_T^{\text{BASE}} + \sum_i \Delta_i * c\tau_{max}$ becoming $\mathbb{E} R_T^{\text{BASE}} + \sum_i \Delta_i * c(\tau_{max} + 1)$ if we use the definition of Joulani et al., so in the worst case the online updating regret bound of SDB differs by at most N from that of QPM-D. This very small difference in the bounds could be eliminated by simply modifying the update of B in SDB to calculate maximum delay minus one instead of maximum delay, with a negligible effect on the performance of the algorithm. The only remaining issue would be that the $+N$ term would remain in the no-delay case since B is initialized to 1, however, the regret bounds would be the exactly same in the case of nonzero online updating delay.

t. For arms $j \neq I_t$,the algorithm sets $p_{jt} = 0$ if $S_{jt} \geq B_t$, and since $B_t \leq \tau_{max}$ we cannot add samples such that $S_{j,t+1} > \tau_{max}$. So only $S_{I_t,t+1}$ can be greater than τ_{max}. However, we know Q_{I_t} must be empty, or variable I could not be set to point to it. But if we have more than τ_{max} samples assigned to I_t and not yet observed, one must have been assigned more than τ_{max} steps ago, meaning the delay must be greater than τ_{max}, a contradiction. So $S_{I_t,t+1}$ also cannot be greater than τ_{max}.

So we have shown that $S_{i,t} \leq \tau_{max}$ for all i and t in the online updating case. Plugging into Theorem B.1 and observing that $\Delta_i \leq 1$ gives us the stated bound.

Now we will show $S_{i,t} \leq 2\tau_{max}$ in the batch update case. Note that in this case we refer to timesteps from the perspective of batches, where after every batch the timestep increments. Let B_t be the value of SDB variable B at time t.

Assume $S_{i,t} \leq 2\tau_{max}$ at t, and we will prove it for $t + 1$.

For arm I_t, Q_{I_t} must be empty before the batch begins, so $S_{I_t,t} = 0$ since we have assumed all rewards from each batch are observed upon its completion. Since the batch size cannot be larger than τ_{max} it follows immediately that $S_{I_t,t+1} \leq \tau_{max} \leq 2\tau_{max}$. We now consider arms $i \neq I_t$.

In the case where $S_{i,t} \geq B_t$ and $i \neq I_t$, SDB sets $p_{i,t} = 0$, so $S_{i,t+1} = S_{i,t} \leq 2\tau_{max}$.

If $i \neq I_t$ and $S_{i,t} < B_t$, we know $B_t < \tau_{max}$, so $S_{i,t} < \tau_{max}$. So since the batch size cannot be larger than τ_{max}, $S_{i,t+1} \leq S_{i,t} + \tau_{max} < 2\tau_{max}$.

So we have shown that $S_{i,t} \leq 2\tau_{max}$ for all i and t in the batch updating case. Plugging into Theorem B.1 and observing that $\Delta_i \leq 1$ gives us the stated bound. □

Lemma C.1.1. *In the mixed case where we update in batches, but not all elements are guaranteed to return before the end of the batch,*

$$\mathbb{E}R_T \leq \mathbb{E}R_T^{\text{BASE}} + N(\tau_{max} + 2\beta_{max}) \tag{C.9}$$

for SDB, where β_{max} is the maximum size of a batch.

Proof. Observe that in this setting, while τ_{max} is the maximum delay of a sample before it

returns, the maximum delay before we can process it is $\beta_{max} + \tau_{max}$, since there are at most β_{max} steps between when a sample returns and when we can process it.

Let t refer to the timesteps on which updates occur. Let I_t denote the value of variable I in the SDB algorithm at time t, and similarly for B.

Clearly $S_{i,0} = 0$ and so the bound holds in the base case. Assume $S_{i,t} \leq 2\beta_{max} + \tau_{max}$ at time t, and we will prove it for $t + 1$.

In the case $i = I_t$, we know that Q_{I_t} is empty at time t. Therefore, $S_{I_t,t} \leq \tau_{max}$ since if we have more than τ_{max} samples assigned to I_t and not yet observed, one must have been assigned more than τ_{max} steps ago, meaning the delay must be greater than τ_{max}, a contradiction. The most we can put in before the next update is β_{max}, so $S_{I_t,t+1} \leq \beta_{max} + \tau_{max} \leq 2\beta_{max} + \tau_{max}$.

In the case where $i \neq I_t$ and $S_{i,t} \geq B_t$, SDB sets $p_{i,t} = 0$, so $S_{i,t+1} = S_{i,t} \leq 2\beta_{max} + \tau_{max}$.

If $i \neq I_t$ and $S_{i,t} < B_t$, we know $B_t < \tau_{max} + \beta_{max}$, so $S_{i,t} < \tau_{max} + \beta_{max}$. So since the batch size cannot be larger than β_{max}, $S_{i,t+1} \leq S_{i,t} + \beta_{max} < \tau_{max} + 2\beta_{max}$.

So we have shown that $S_{i,t} \leq \tau_{max} + 2\beta_{max}$ for all i and t. Plugging into Theorem B.1 and observing that $\Delta_i \leq 1$ gives us the stated bound. □

C.1.1 Comparison to prior work

Joulani et al. [75] proposed a closely related algorithm QPM-D. If we extend their online updating analysis to the batch updating cases, their algorithm has a regret bound with a an additive term of $N\tau_{max}$ in the online updating and batch updating settings and $N(\tau_{max} + \beta_{max})$ in the mixed setting. Hence we see that SDB matches the bound of QPM-D in the online updating setting, but the additive term worsens by at most a factor of two when updates come in batches. Creating an algorithm that retains the bound of QPM-D in the batch case but also retains most of the empirical benefit of SDB is an important direction for future work.

C.2 Delay with Online Updating results

We here discuss the supplemental online updating results, which complement the batch updating results presented in the main text.

The online updating case is relevant in many situations, and the comparison between SDB and QPM-D is a bit fairer because both possess the same theoretical guarantees in this case. Below we repeat the same experiments as in the batch case, the only difference being that each algorithm can update its distribution after each step instead of in batches.[3]

Figures C.1-C.6 show the results running SDB in the online updating case in a variety of simulations (see section 5.4 for explanations of the environments used). Note that SDB-thom-X refers to running SDB with Thompson-1.0 as the BASE algorithm and Thompson-X as the HEURISTIC algorithm, and likewise for UCB.

Figure C.7 gives the results on actual data using our unbiased queue-based estimator. We see much the same results as we did in the batch updating setting, showing good empirical performance in a variety of scenarios. One notable difference is that in the UCB figures (C.4 and C.5) we see that SDB has essentially "smoothed" out performance, eliminating the the troughs of very poor performance. The reason for this is that in the online case, if the heuristic chooses an arm to pull, SDB will initially allow it to put full probability mass on that arm, but then smoothly decrease the amount of probability it is allowed place on that arm, shift the remainder to the arm(s) recommended by BASE. Hence instead of just pulling one arm per batch, in an online updating setting SDB can spread mass between multiple arms, even given deterministic black-box algorithms.

[3]The distribution of Uniform, QPM-D, UCB, UCB-Strict, and UCB-Discard does not change if new data is not observed, so their performance should be the same as in the batch updating setting. Therefore, we did not re-run those algorithms.

Figure C.1: Comparing to QPM-D with Thompson Sampling as the black-box algorithm.

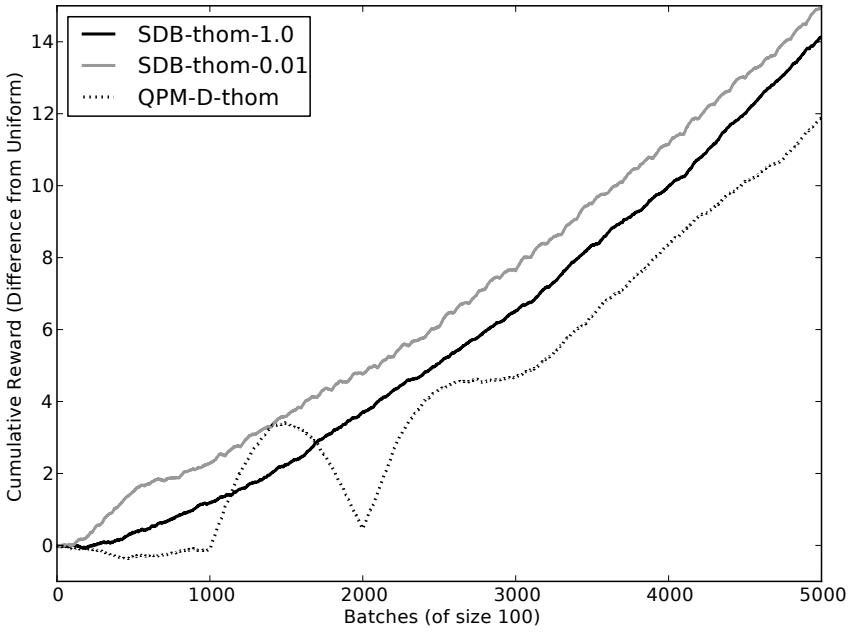


Figure C.2: Comparing to heuristics with Thompson Sampling as the black-box algorithm.

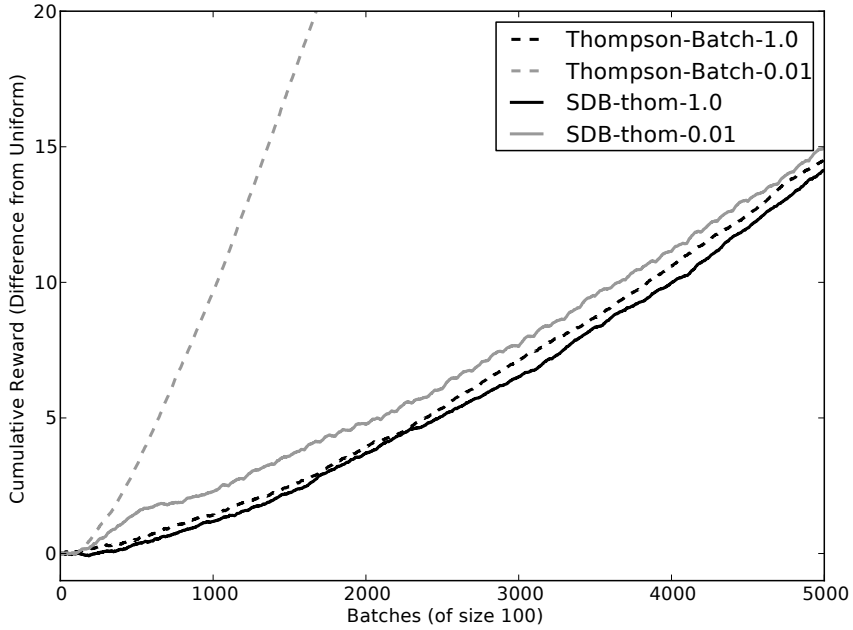


Figure C.3: An example where the heuristic Thompson-Batch-0.01 performs worse.

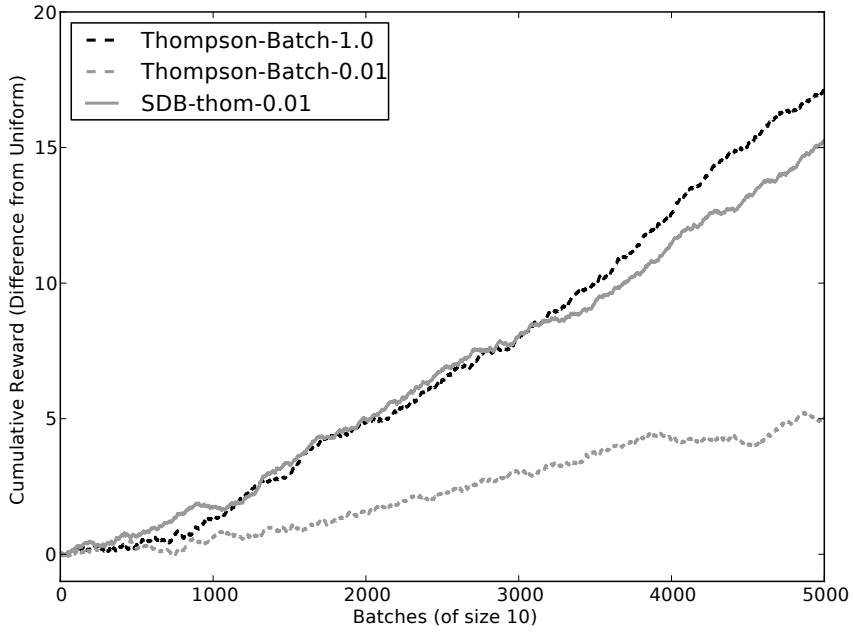


Figure C.4: Comparing to QPM-D using UCB as the black-box algorithm.

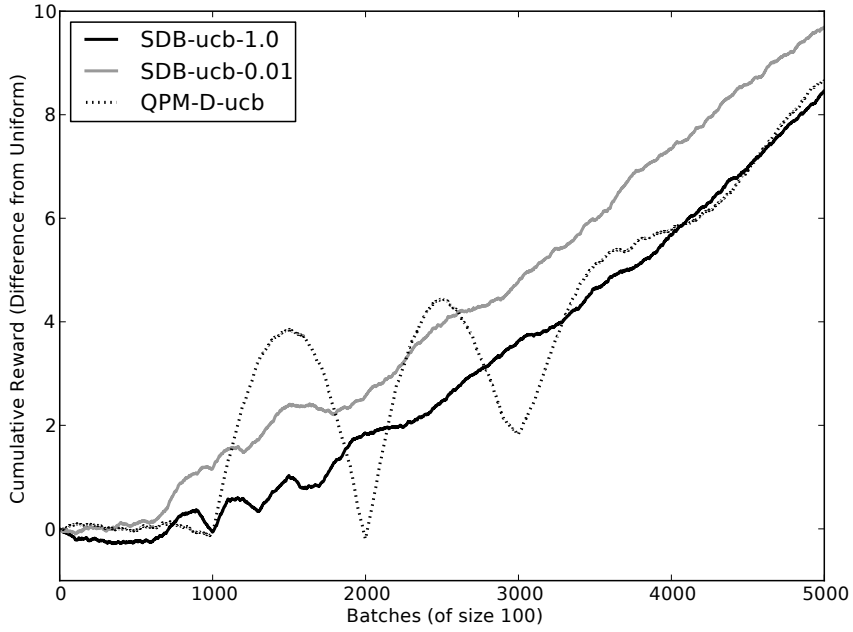


Figure C.5: Two example algorithms that perform poorly when handed all samples but well inside SDB.

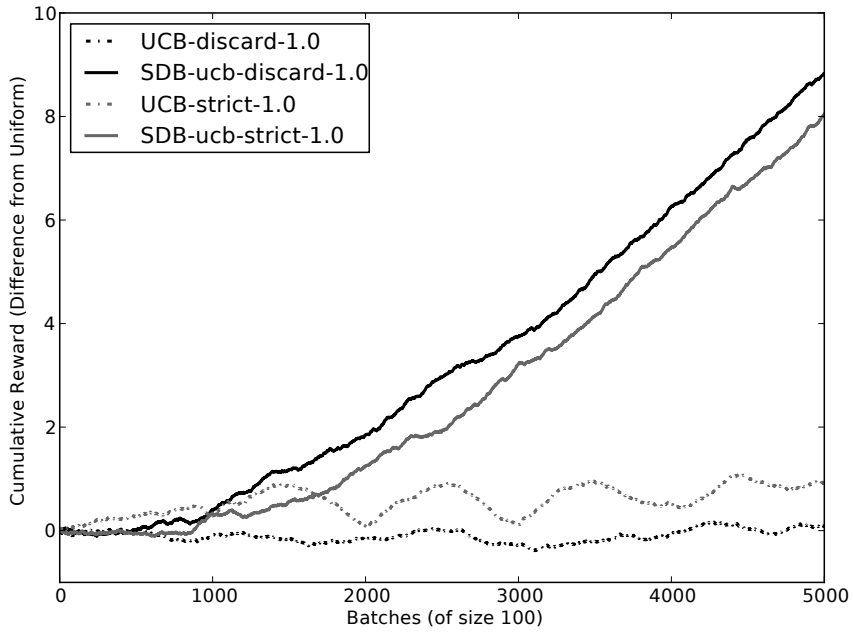
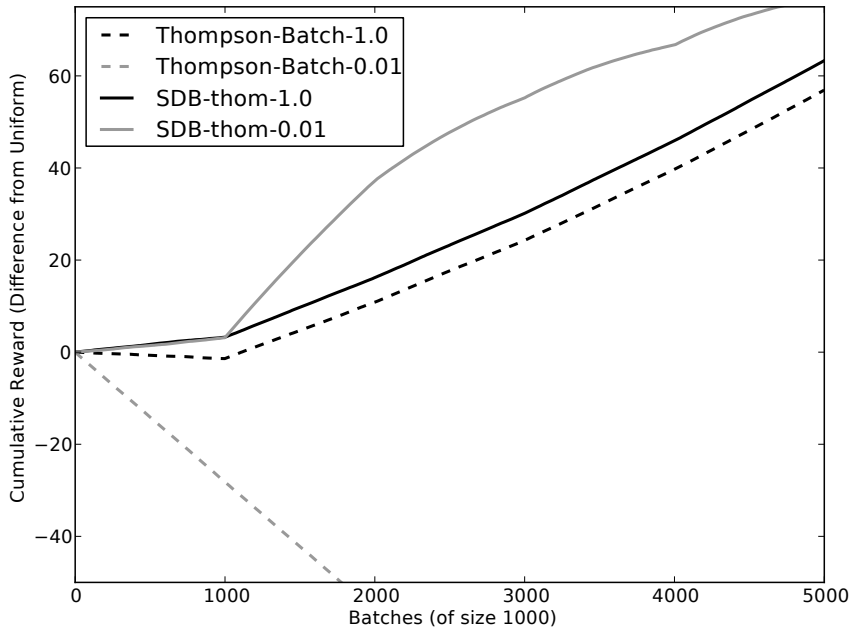


Figure C.6: An example of a case where handing a prior dataset from a poor arm hurts Thompson-Batch but not SDB.



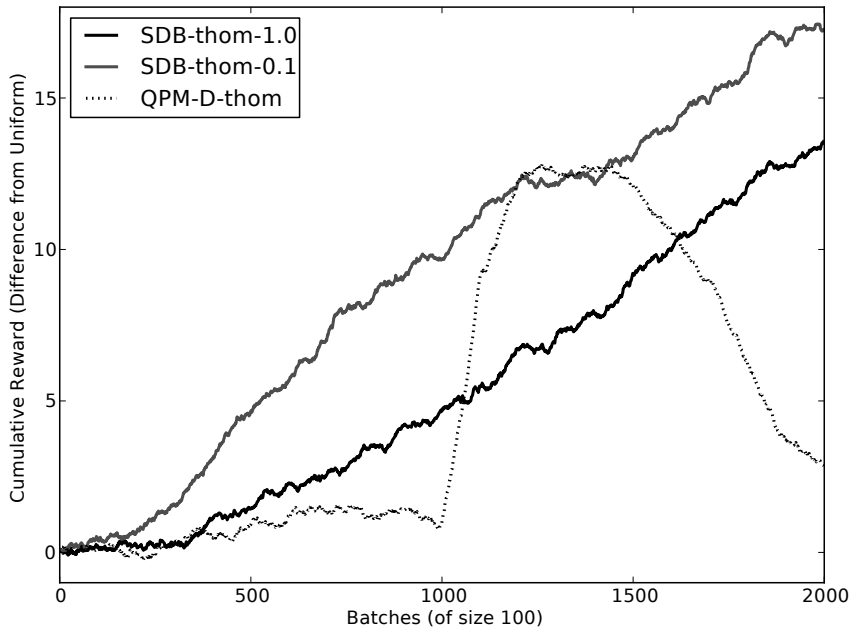


Figure C.7: Results (using our queue-based estimator) on educational game data. SDB sees major improvements by leveraging the heuristic.

Appendix D

WHERE TO ADD ACTIONS

D.1 Details of Optimistic Estimation Methods

UCRL This strongly optimistic approach is based on UCRL2 [71], which defines a confidence set over each transition/reward distribution, and takes the maximum valid distribution in the confidence set when planning. Specifically, in a finite-horizon setting it allows the L1 norm of the transition distribution to deviate from the MLE by at most $\sqrt{\frac{14\log(SA_\ell\tau\ell/\delta)}{\max(N,1)}}$, where ℓ is the number of episodes, N is the number of transition samples, and δ is a user-specified confidence parameter. UCRL's bound incorporates global uncertainty to ensure that the true MDP is within the confidence set with high probability [10]. We use this bound to quantify the uncertainty over our outcome distribution, setting $\delta = 0.05$ as in Osband et al. [114].[1]. The advantage of UCRL2's constraint on the L1 is that it is easy to calculate the most optimistic distribution that obeys the constraint, as explained in Strehl and Littman [135]. Since we are in a finite horizon setting, we calculate the most optimistic distribution for each (s, t) pair.

MBIE Model-based Interval Estimation (MBIE) [135, 136] is a very similar idea to UCRL, but simply bounds the local L1 divergence at each state with probability $1 - \delta$. The original MBIE algorithm [135] used a bound on their transition dynamics of $\sqrt{\frac{2(|S|-1)\log(N+1)-\log(\delta)}{N}}$. However, similar to later versions [136], we use the bound shown by Weissman et al. [157], namely that for a discrete distribution with O outcomes, the probability of the error in L1 divergence being ϵ or greater after N samples is at most $(2^O - 2)e^{-N\epsilon^2/2}$,

[1]Note that since it is not immediately clear how to translate the UCRL analysis to an MDP specified in terms of outcomes, we simply use apply UCRL's bound on the transition distribution to our outcome distribution. Also, even though it seems as though $\delta = 1/\ell$ is needed to induce sublinear expected regret for UCRL, we follow Osband et al. [114] for the parameter values.

to derive the refined bound of $\sqrt{\frac{2\log((2^O-2)/\delta)}{N}}$. The advantage of this bound is that it decreases faster with the number of samples N and matches our outcomes setting. Again we let $\delta = 0.05$ corresponding to 95% confidence, and use $\max(N, 1)$ to deal with the 0-sample case.

(Optimistic) Thompson Sampling Thompson sampling [148], also known as posterior sampling, has been shown to be one of the best performing algorithms empirically at handling the exploration/exploitation tradeoff [29, 114]. Unfortunately, it is not optimistic. Although this tends to be a strength in a explore/exploit setting where it tends to boost exploitation, here it is a major weakness as if the sample happens to be pessimistic we may wastefully add an action at a state where an existing action may already be good. One previously proposed way of partially alleviating this problem is Optimistic Bayesian Sampling, [103] which rejects samples with values lower than the mean of the distribution.[2] The mean of a Dirichlet with parameters $\alpha_1, \ldots \alpha_O$ is a vector X with components $\frac{\alpha_i}{\sum_j \alpha_j}$, so we reject sampled distributions with values of less than $V(X)$. The choice how to set the prior (Dirichlet parameters) is up to the user, however if a Bayesian algorithm such as PSRL [114] is used, the user will likely have to supply a prior distribution in any case.

BOSS An alternate approach to inject optimism into a posterior sample is simply to sample J times and take the sample with maximum value. This is the key idea behind the Best of Sampled Set (BOSS) algorithm [8], which showed this approach can have attractive theoretical properties in a traditional explore/exploit setting. Unfortunately, a significant downside of BOSS is that it is unclear how to set J. Asmuth et al. [8] found a value of $J = 10$ to work best empirically so we use that value without further tuning.

[2]OBS as proposed by May et al. [103] takes the max of the sample and the mean, we instead resample if the current sample is lower than the mean to further increase optimism. Additionally, since we have Dirichlet distributions we must be careful in distinguishing the mean of the posterior distribution with the expected final value. However, since the values of each outcome are considered to be a constant and not dependent on the outcome distribution, it suffices to dot product the mean of the Dirichlet with the vector of values.

D.2 Proof of Lemma 7.4.1

Lemma 7.4.1 (Non-starving) *Consider ELI using a prior distribution f which consists of an independent Dirichlet prior on outcome distributions of $\alpha_i = c$ for some $c > 0$. Assume for a given $\epsilon > 0$, after N_ϵ actions are added to each state, additional actions improve the value of each state by at most ϵ. Let $\mathcal{C}$ be an arbitrary class of models with N_ϵ actions which has non-zero probability under our chosen prior. Assume that the true model M for the first N_ϵ actions is drawn from $\mathcal{C}$ according to $f(M|\mathcal{C})$.[3] Finally, assume[4] that for each s there exists $o_1, o_2 \in \mathcal{O}$ such that $T(s, o_1) = T(s, o_2); R(s, o_1) \neq R(s, o_2)$. Then, as the number of actions added by ELI goes to infinity, our ELI approach will eventually uncover actions at each state such that the optimal policy in the MDP with added actions is at least ϵ-optimal (with respect to the full set of actions).*

Proof. For a contradiction, assume there is some non-vanishing probability that the optimal policy in the MDP with added action is less than ϵ-optimal. For this to be the case, clearly there must be a state s and timestep t such that $|A_{s,\ell}| < N_\epsilon$ as $\ell \to \infty$ and $V(s,t)$ is not epsilon-optimal, but if we added additional actions to that s, $V(s,t)$ in the MDP with added actions would be epsilon-optimal. Since the number of total actions added by ELI goes to infinity, there must be at least one other state $s' \neq s$ such that the number of added actions added by ELI at s' go to infinity. Since our ELI method selects the state with the highest ELI score, there must be an infinite number of rounds where the ELI score of s' is greater than that of s. Now, since the ELI score for s' contains a factor of $\frac{1}{|\mathcal{A}_{s',\ell}|+2}$ the score will go to zero as the number of actions at state s' goes to infinity. Therefore, the only way for us to add a finite number of actions to s is for the ELI score of state s to go to zero as well. Since we sum over timesteps and all ELI scores are nonnegative, this means the score of s

[3]This assumption on the relationship of the true model to the prior is fairly weak, and similar to that used in the analysis of PSRL [114].

[4]This assumption is due to the considered outcome setting, in fact in discrete MDPs Osband et al. [114] proposes priors which treat rewards and transitions independently which would imply a stronger assumption.

at time t must go to zero as well.

Now, we know the ELI score for state s at time t is $\frac{1}{|\mathcal{A}_{s,\ell}|+2}(V_{max}(s,t) - \hat{V}(s,t|\mathcal{A}_{s,\ell}))$, or, since we know $|A_{s,\ell}| < N_\epsilon$, at least $\frac{1}{N_\epsilon+2}(V_{max}(s,t) - \hat{V}(s,t|\mathcal{A}_{s,\ell}))$. Clearly, the only way for this to go to zero is for $\hat{V}(s,t|\mathcal{A}_{s,\ell})$ to go to $V_{max}(s,t)$, which means for some $a_s \in \mathcal{A}_{s,\ell}$ $\hat{Q}(s,a_s,t)$ converges to $V_{max}(s,t)$.

By assumption, there exists $o_1, o_2 \in \mathcal{O}$ such that $T(s,o_1) = T(s,o_2); R(s,o_1) \neq R(s,o_2)$. Without loss of generality, label o_1 and o_2 such that $R(s,o_1) > R(s,o_2)$. Clearly, in order for $\hat{Q}(s,a_s,t)$ to approach $V_{max}(s,t)$ the estimated $\hat{P}(o_2|s,a_s)$ must go to zero as ℓ increases, or we could have increased $\hat{Q}(s,a_s,t)$ by shifting the probability mass to $\hat{P}(o_1|s,a_s)$. If we sample (s,a_s) a finite number of times, the BOSS sampling procedure will sample $\hat{P}(o_2|s,a_s) > 0$ with some positive and non-vanishing probability. If we sample $\hat{P}(o_2|s,a_s)$ an infinite number of times, it will converge to its true value. The probability of any outcome distribution placing zero probability on any outcome under the posterior MDP distribution $f(M)$ is zero, and thus likewise under $f(M|\mathcal{C})$. Since the distribution over outcomes at (s,a_s) was sampled according to $f(M|\mathcal{C})$, $P(o_2|s,a_s) > 0$ with probability 1 in the true (sampled) model. So $\hat{Q}(s,a_s,t)$ does not converge to $V_{max}(s,t)$. □

D.3 Simulation Domains

Riverswim [136] is a chain MDP with 6 states and 2 ground actions per state that requires efficient exploration [114]. For a diagram and complete description of the environment, see Osband et al. [114]. Similar to chapter 6 we used a horizon of 20 and use 5 relative outcomes for moving left and right or staying with some reward. We used a flat ($\alpha_i = 1$) Dirichlet prior over outcome distributions for PSRL.

Marblemaze [8, 122] is a gridworld MDP with 36-states and 4 ground actions per state that allows a significant amount of prior knowledge to be encoded in the outcomes framework [8]. For a diagram and complete description of the environment, see Asmuth et al. [8]. As in chapter 6, we used a horizon of 30 and a set of 5 outcomes denoting whether the agent moved in each cardinal direction or hit a wall (in keeping with past work, the coordinates of

the goal and pits are assumed to be known). We also set the reward for falling in a pit to be -0.03. As in riverswim, we used a flat ($\alpha_i = 1$) Dirichlet prior over outcome distributions for PSRL.

We also try a variant of Marblemaze with an uninformative outcome space. Here, the outcome space consists of every possible combination of the 3 possible reward values and the 31 possible valid next states (plus the terminal signal for falling into one of the four pits or reaching the goal). To allow PSRL to learn faster in this large outcome space, we used a Dirichlet prior of ($\alpha_i = \frac{1}{O}$) over outcome distributions, which encourages sparsity.

The Large Action Task was introduced by Sallans et al. [123] as a testbed for algorithms that cope with an action space too large to explore directly. In this setting, states and actions are described by vectors of K bits.[5] On each run the environment is initialized by picking 13 bit vectors uniformly at random to represent the states of the problem. Then each of these states is associated with another bit vector (again chosen uniformly at random) to represent the optimal action. At the start of an episode a fresh bit vector is selected uniformly at random, and the the agent starts the episode at the closest (in hamming distance) state to that vector.[6] The agent picks a bit vector as an action and receives a reward equal to the number of bits that matched between the chosen action and the optimal action minus $K/2$.[7] In Sallans et al. the environment was closer to a contextual bandit setup, where after each timestep a new key state was drawn independent of the action taken. To make the problem more interesting from a reinforcement learning perspective, we deterministically transition by XORing the chosen action and current state vector and finding the closest (in hamming distance) next key state. In our experiments we choose $K = 20$, so there are only 13 states

[5]Sallans et al. considered cases where the number of bits for states and actions differed, but we keep them the same for simplicity.

[6]For implementational reasons we do not actually sample a fresh vector every time, instead we sample 100 vectors when the environment is created to construct an approximate distribution over key states and sample from that at the start of each episode.

[7]Sallans et al. did not subtract off $K/2$ but we do so to get a better sense of the difference among algorithms in terms of cumulative reward, as this linear transform does not meaningfully change the problem but simply normalizes rewards so that random achieves zero reward.

but 2^{20} ground actions. We use a horizon of 5, and the outcome space is uninformative with one outcome for each key state and reward, so 273 total outcomes. Due to the large outcome space we use a sparse prior, similar to large outcomes version of marblemaze ($\alpha_i = \frac{1}{O}$).

Improving experts in the large action task behave as follows. The initial action for s is drawn by first uniformly sampling a hamming distance, and then uniformly sampling a vector with that hamming distance from the optimal action for s. Subsequent actions are generating by either (with 50% probability) generating another random vector with the same hamming distance as the closest currently available action for s, or (with the remaining 50% probability) choosing an improved hamming distance uniformly at random and then choosing a vector with that hamming distance at random.

Poor experts in the large action task behave as follows. The initial action is generated uniformly at random, and subsequent actions are generated with probability proportional to their hamming distance to the optimal action, so poor actions are much more likely to be generated.