POLICY-BASED EXPLORATION FOR EFFICIENT REINFORCEMENT LEARNING

A Dissertation
Presented to
The Academic Faculty

By

Kaushik Subramanian

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

May 2020

POLICY-BASED EXPLORATION FOR EFFICIENT REINFORCEMENT LEARNING

Approved by:

Dr. Charles Isbell, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Andrea Thomaz, Co-Advisor
Department of Electrical and Computer Engineering
University of Texas at Austin

Dr. Mark Riedl
School of Computer Science
Georgia Institute of Technology

Dr. Thad Starner
School of Computer Science
Georgia Institute of Technology

Dr. Peter Stone
Department of Computer Science
University of Texas at Austin

Date Approved: April 24, 2020

The laws of probability, so true in general, so fallacious in particular.

Edward Gibbon

This work is dedicated to my parents, my sister and my loving partner. They have been by my side through trying times and continue to be a never-ending source of inspiration and encouragement.

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Charles Isbell. He has been an insightful, patient and strong mentor over the course of my degree. Charles makes things look very easy when they in reality aren't, from solving challenging research questions to communicating science to large audiences and writing with clarity. Charles has helped me in more ways than I can count and continues to do so. As Prof. Michael Littman once told me "Charles is larger than life". I agree with this assessment wholeheartedly.

I would like to thank my co-advisor Prof. Andrea Thomaz. Andrea taught me how to focus on the work, how to approach research and how to be efficient at it. I do not believe I have completely mastered the skills but I am significantly better now than when I started. Andrea guided me through my research explorations and exploits by providing an supportive and engaging learning environment.

To my committee members Prof. Mark Riedl, Prof. Thad Starner and Prof. Peter Stone. Their feedback on my research topic, experiments and dissertation were important to help me focus and improve my work.

My journey towards completing this document and the doctoral degree started with Prof. Michael Littman years ago in Rutgers University. He was the first person to introduce me to the idea of reinforcement learning and since then has served as a guide to me. He has always been ready to get on a call and talk when I have needed it. Michael I will remember to pay it forward.

I would like to thank members of the pfunk research group who have become dear friends. To Ashley Edwards, Himanshu Sahni, Pushkar Kolhe, Yannick Schroecker who have spent a significant amount of time discussing research ideas with me. I can hardly forget our Friday "lab meetings" with free lunch (sponsored by Charles), assisting and grading CS4641. I will cherish the memories of staying late in the conference room completing corrections.

To Jonathan Scholz, Luis Carlos Cobo Rus, Arya Irani, Peng Zang, Martin Levihn who helped me with my struggles during my Ph.D. with suggestions, ideas and distractions. I never did have a chance to say thank you, but I would like to now. Jon, I remain the immovable object and I imagine you continue to be the unstoppable force.

To members of the simL research group, Baris, Maya, Crystal, Shane. They helped me during my initial years at Georgia Tech in the robotics program and I learned a great deal from working with them on Simon and PR2 robots.

I would like to thank the friends I made outside of the university, my roommates and friends in the community where I lived. Thank you for ensuring that I had life outside of the university. I am ever grateful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This work is focused on the design and analysis of novel methods for exploration of a reinforcement learning agent. We introduce a policy-based approach that learns to explore meaningful aspects of decision-making problems autonomously and using human assistance. The thesis we seek to demonstrate is that, **policy-guided exploration for reinforcement learning agents leads to faster convergence to the optimal policy than automatic value-based and state-of-the-art learning from demonstration methods and is robust to noisy human signals**. This line of research raises questions about how to efficiently explore the search space of a problem and how to balance the exploration-exploitation trade-off inherent to reinforcement learning agents. To support the claim and address these challenges, the main contributions of document are summarized below:

- **Agent-guided Exploration from Human Demonstration** - We learn a policy useful for exploration from human demonstrations using supervised learning. The agent uses statistical properties of regression algorithms for reinforcement learning to communicate its model uncertainty. This allows the human to provide samples useful from the agent's perspective. The learned exploration policies leads to faster convergence to the solution than learning from optimal demonstration and model-free exploration strategies. We show the effectiveness of this approach on two game domains with high-dimensional continuous states, extended goal horizons and sparse rewards.

- **Autonomous Agent-guided Exploration** - We build on the work on learning exploration policies using human demonstrations to show how we can learn such policies autonomously. The agent, guided by the statistical measures, solves for a policy that helps the agent explore. Using the learned policy for exploration helps the agent obtain the optimal policy efficiently and with improved sample complexity over existing approaches. These are demonstrated in a classical control problem and a high-

dimensional game domain.

- **Policy Shaping with Humans** - This work presents a probabilistic approach to combining human signals with a reinforcement learning model. We model human feedback as a policy signal and when utilized with Bayesian RL for exploration, show that we can solve the decision-making problem with fewer parameters than state of the art methods and are robust to noisy human input.

- **Exploration in Monte Carlo Tree Search using Action Abstractions** - In this work, we provide an alternate interpretation to exploratory human demonstrations. We show how human demonstrations, when instantiated as temporal action abstractions, can be used to overcome the difficulties of Monte Carlo reinforcement learning methods.

CHAPTER 1
INTRODUCTION

We are currently living in a world where technology forms an integral part of people's lives. We interact with them in various forms, digital as well as physical systems. They are designed to improve the quality of our lives in different ways from transportation, security to health care, education and communication. Of particular interest in this document is the topic of automation. Over the years we have designed systems to automate several problems which otherwise would require enormous amount of human-hours to parse through, for example in the construction of cars, printing of newspapers. The automation is focused on removing human involvement and making the process fast, accurate and more efficient overall.

Automation is helpful for tasks that involve hard manual labor and more recently can also be used to solve complex problems, specifically problems that involve reasoning and intelligence. Designing and analyzing systems that can automatically solve such problems has been the interest of many industrialists, scientists, researchers and philosophers. It falls broadly under the topic of Artificial Intelligence (AI), i.e. designing an artificial system with abilities to reason and solve problems. The topic of AI refers to all aspects of intelligence, namely reasoning, knowledge representation, learning, planning, creativity, perception, social interactions, language, motion, etc.

In this work we particularly focus on the topic of learning or machine learning. It is the task of providing machines the capacity to learn and adapt to solve problems. Machine learning has been an active area of research for several years and forms an essential part of present-day technological systems, for example face recognition in photos, converting speech to text, autonomous driving and numerous other examples.

1.1 Machine Learning

Machine learning is a data-oriented approach to problem-solving. For a given problem, data is collected and mathematical models are built from the data to recognize predictive patterns that help in solving the underlying problem. Machine learning falls broadly in three categories: supervised learning, unsupervised learning and reinforcement learning. In the most common setting, supervised learning, data is collected in the form of attributes which describe the problem under different conditions accompanied with a label that we are interested in predicting. Data of this form is fed into a learning algorithm which generates a model capable to making the same predictions as those made in the data it was trained on as well as data it did not get to learn from. In this approach, the algorithm learns a general model that can make useful predictions. An example is to predict the objects in an image from a set of known objects. Unsupervised learning refers to the case where the data used to learn from is not accompanied with a label (is unsupervised). In this case, the learner is tasked with recognizing patterns without being explicitly being provided the labels it is expected to be able to reproduce. Problems involving grouping or clustering largely fall in this category where the data is not assigned a explicit label. Lastly reinforcement learning is a formulation useful to solve sequential decision-making problems. These are problems that require algorithms to consider the temporal effects of decisions being made over long periods of time. In this dissertation, we focus on solving decision-making problems using reinforcement learning.

1.2 Deep Learning

Deep learning [1] is an approach to machine learning which focuses on building multi-layered encoders of data to facilitate automatic abstraction and better generalization. This approach has been popularized recently using neural networks by designing network architectures with multiple (hidden) layers between the input data and the output predictions.

The advantages offered in this approach are that the algorithm learns to automatically transform the data into representations that are better suited to make the required predictions, specifically when dealing with non-linear relationships between the input data and predictions. Deep Learning has had a wide variety of success with image data [2], text [3], audio [4], robotics [5] and a host of other areas including sequential decision-making problems.

1.3 Reinforcement Learning

Reinforcement Learning (RL) [6] is the field of research focused on solving sequential decision-making tasks modeled as Markov Decision Processes. Researchers have shown RL to be successful at solving a variety of problems like games (Backgammon [7], Go [8], Atari [9], StarCraft [10]), robot tasks (soccer [11], helicopter control [12]) and system operations (inventory management [13]).

The reinforcement learning method is similar in essence to training a pet using positive reinforcement. Every good action taken is rewarded and over time that behavior is reinforced and generalized across different situations. In many ways, this is how we train computer agents using reinforcement learning. A task is defined by a reward function which specifies what is good and bad. Using such a the task description, a mapping of situations to actions is learned that informs the learner of the actions to take such that the rewards accrued over time are maximized. More recently, reinforcement learning methods have gained a lot of attention due to the success of combining RL and deep learning as deep reinforcement learning. The algorithms based on this approach make use of the generalization and abstraction properties of deep learning to solve complex non-linear optimization problems for RL.

1.4 Research Challenges

One of the challenges often faced when applying reinforcement learning methods to complex problems is that of sample complexity. The amount of data required to train a good

model requires exploring a potentially large high-dimensional non-linear state space which in the worst case can be prohibitively expensive and in the best case handled with large compute and smart algorithms over a long period of training time. In the RL literature this search is related to balancing the exploration-exploitation trade off - a central problem in Reinforcement Learning and the main topic of this dissertation.

The exploration-exploitation trade off here refers to the problem of deciding what to explore in the search space of the domain and when to exploit the knowledge gained from what has been searched. In the most common use-case, the reason this problem arises is that RL approaches rely on obtaining samples useful for learning the underlying structure without always using smart methods to explore the state space. Traditional methods either use a fixed (uniformly random) policy or value-based metrics [14, 15] that in some cases can result in redundant and/or unsafe exploration. More recently, researchers have designed several measures and heuristics to help tackle this challenge including visitation counts [16], sampling [17], random exploration [18], intrinsic motivation [19, 20] and model-based approaches [21, 22].

A combination of smart exploration along with careful design of the learning algorithm and its parameter can be effective solution to reinforcement Learning problems. In this work, we tackle the challenge of smart exploration in RL, by using human interaction and autonomous methods.

1.5 Hypothesis and Goals

The hypothesis we make in this dissertation focuses on proposing alternate approaches to exploration. We hypothesize that an approach focused on learning to explore as policies directly helps overcome some of the computational challenges related to exploration. Additionally we hypothesize that such an approach when used in the interactive setting with information from people can be robust to noisy information from humans.

We present policy-based methods that serve to

1. Bias an RL agent's exploration to cover the search space of the domain efficiently

2. Balance the exploration-exploitation trade-off for an RL agent learning from human signals

In designing an exploration policy for sequential decision-making problems, it is important to help the agent reach parts of the search space that are necessary to model in order to solve the problem. These include guiding the agent towards noisy, stochastic regions of the problem as well as regions of high reward.

To facilitate exploration in these parts of the domain, we design autonomous and interactive methods. The autonomous method poses the optimization function as a linear regression problem and use relevant measures [23] to help identify *influential* regions. Such an approach has the advantage of learning to explore the domain from the agent's perspective while taking into account the underlying representation, the RL algorithm as well as the problem definition. We follow up with using information humans (both machine learning experts and non-experts) to help the agent explore. Specifically we utilize their knowledge of the rules of the domain and the optimal behavior and to acquire information that would lead the agent towards *influential* regions of the search space. The goal is to show how policy-based approaches autonomous and interactive, are able to solve long horizon problems using exploratory demonstrations while outperforming traditional exploration and interactive learning methods.

1.6 Contributions

The thesis statement for the work is: **Policy-guided exploration for Reinforcement Learning agents leads to faster convergence to the optimal policy than automatic Reinforcement Learning and state-of-the-art Learning from Demonstration methods and is robust to noisy human signals**.

First, we address the concern of biasing exploration for RL agents towards efficient

learning. We present a policy-based approach called Exploration from Demonstration (EfD) that learns a stationary exploration policy using human demonstrations. We show how using such a policy for exploration provides convergence speed-ups. We then improve EfD, using concepts of active learning, to make it sample efficient. We use the inductive bias of RL algorithms to provide feedback to the user about the algorithm's uncertainty. Using this approach, agents are more likely to acquire samples that are useful from the algorithm's perspective.

We follow up on this work by relaxing the requirement of human/oracle information from the EfD algorithm and present an approach that can autonomously learn an exploration policy. The policy is constantly updated based on the progress of the learner and is able to guide the agent towards influential regions of the state space. We also show how this approach can be used in deep reinforcement learning algorithms to solve Atari games in a straightforward manner.

We then tackle the problem of balancing the exploration-exploitation trade-off in RL. Bayesian RL algorithms have been used to address this problem and have had some measure of success on simple domains. We present a probabilistic method called Policy Shaping that combines human evaluations with Bayesian Q-learning. We show how this approach is robust to noisy, sub-optimal human signals, learns when and where to explore and provides performance speedups. Unfortunately due to the nature of Bayesian Q-learning, Policy Shaping is limited to small domains.

Finally we present an approach that makes use of some of the inherent structure in the exploratory human demonstrations to assist Monte Carlo Tree Search (MCTS) algorithms in exploration. We show how the demonstrations can be interpreted as temporal action abstractions (specifically options and constraints) and show how when combined with MCTS can be used to overcome the algorithm's limitations and efficiently solve large-scale problems. Overall we show how using policy-based methods to bias exploration provides performance speed-ups, is sample efficient and outperforms value-based methods.

We implement our methods on popular arcade games and control problems and highlight the performance improvements that can be achieved using our approach. We show how this work on policy-based exploration autonomously and using humans to help agents efficiently explore sequential decision-making tasks is an important and necessary step in applying reinforcement learning to complex problems.

1.7 Structure

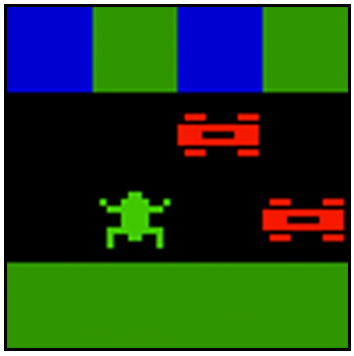
This dissertation is organized as follows: Chapter 2 provides details on relevant background for this dissertation, detailing concepts in reinforcement learning and related research works. Chapter 3 introduces a policy-based approach to exploration and how it can be combined with human information. Chapter 4 directly builds on Chapter 3 showing how we can automate exploration without the need for human help. Chapter 5 focuses on tackling the problem of exploration and exploitation by using a Bayesian approach to RL in combination with human binary critique. Finally we show how general policy-based approaches can scale to solve large problems with long horizons and sparse rewards by using them to overcome the limitations of monte carlo tree search.

1.8 Domains

Here we provide descriptions of the domains that are used for experiments in this dissertation. These include popular arcade games as well as classical control problems.

Gridworld This is a basic domain popularly used in reinforcement learning papers as an instructional domain to help elucidate key ideas where relevant. It is a typically setup as a grid of squares either as a square or a rectangle. There are four actions that the agent can take to access the four cardinal directions. The effects of the actions may be deterministic or stochastic The reward function includes a step cost for every step taken a goal reward fr landing in the goal cell which is predefined. Transitions leading into the bounding walls

(a) PacMan



(b) Frogger

Figure 1.1: Sample maps for PacMan and Frogger game domains showing the agent and non-playable characters.

keeps the agent's state unchanged. An episode starts with the agent randomly placed in the grid and stops when the agent reaches the goal.

Pac-Man consists of a 2-D grid with food, walls, ghosts, and the Pac-Man avatar (see Figure 1.1a). The goal is to eat all the food pellets while avoiding moving ghosts (+500). Points are also awarded for each food pellet (+10). Points are taken away as time passes (-1) and for losing the game (-500). The action set consisted of the four primary cartesian directions. The state representation included Pac-Man's position, the position and orientation of the ghost and the presence of food pellets. There exist PacMan maps that have long horizons to complete the problem, making it a suitable testbed for our models. [1]

Frogger consists of a 2-D map with moving cars, water hazards, and the Frogger avatar (see Figure 1.1b). The goal is to navigate from the bottom to the top of the grid while avoiding the cars and water pits (shown as dark squares in the top row). Each car drives one space per time step. The car placement and direction of motion is randomly determined at the start and does not change within an episode. As a car disappears off the end of the map it reemerges at the beginning of the road and continues to move in the same direction. The cars moved only in one direction, and they started out in random positions on the road.

[1]The version of PacMan we used is an open-source implementation available online at http://www-inst.eecs.berkeley.edu/ cs188/pacman/pacman.html

Figure 1.2: A snapshot of the Cart Pole control problem.

Each lane was limited to one car. The action set consisted of the four primary cartesian directions and a stay-in-place action. Within an episode, the transitions are deterministic. The reward function is $+1000$ for reaching the goal, -100 for dying (directly hitting a car, crossing over a car, falling into water) and 0 everywhere else. An episode starts with the agent in a random position in the bottom row and stops when the agent dies or reaches the goal. Frogger lends itself to scaling to multiple sizes, making it useful to test how algorithms scale. Further domain details relevant to the experiment will be described in the relevant chapters.

Cart Pole The goal of this domain is to balance a pole on top of a movable cart for as long as possible. The domain presents a challenge because the agent is required to learn a policy that will keep it balanced forever. The reward function, even with large amounts of discounting must continue to contribute towards learning such a policy. The agent has 3 actions in the form of forces that can be applied to the cart, [-10N 0N 10N]. The forces are noisy - a random number is uniformly sampled from [-2 2] and added to the force. The reward function is 1 for balancing the pole and a 0 penalty for failing to do so. We limited the required balancing time to 1000 steps. The raw state space is 4D and consists of the position and velocity of the cart along with the angle and angular velocity of the pole.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter we provide a more formal introduction to reinforcement learning and the algorithms used to solve decision-making problems. We provide details on the existing work in research topics relevant to this dissertation, namely - exploration in RL and interactive learning systems.

2.1 Reinforcement Learning

Reinforcement Learning (RL) defines a class of algorithms for solving problems modeled as a Markov Decision Process (MDP). An MDP is specified by the tuple $M = \langle S, A, T, R, \gamma \rangle$, which defines the set of possible world states, S, the set of actions available to the agent in each state, A, the transition function $T : S \times A \rightarrow \Pr[S]$, a reward function $R : S \times A \rightarrow \mathbb{R}$, and a discount factor $0 \leq \gamma \leq 1$. The goal of a reinforcement learning algorithm is to identify a deterministic, $\pi : S \mapsto A$ or stochastic policy, $\pi : S \mapsto Pr[A]$, which maximizes the expected reward from the environment.

For a given MDP, the value function $V^{\pi}(s)$ represents the expected long-term reward (utility) of being in state s and following policy π thereafter. It is defined as

$$V^{\pi}(s) = \sum_{a} \pi(s,a)[R(s,a) + \gamma \sum_{s'} \mathcal{T}(s,a,s')V^{\pi}(s')]$$

We also define the Q-function or action-value function $Q^{\pi}(s,a)$ as the expected long-term reward of taking action a in state s, transitioning to s', and following policy π thereafter. Mathematically, the Q-function is computed as

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'} \mathcal{T}(s,a,s') \sum_{a'} \pi(s',a')Q^{\pi}(s',a')$$

The optimal state-value function is $V^*(s) = \max_\pi V^\pi(s)$, the optimal action-value function is $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, and the solution to an MDP is any optimal policy π^*, which maximizes the value function for every state.

Several algorithms have been designed to solve reinforcement learning algorithms - dynamic programming, temporal difference and Monte Carlo methods. Dynamic programming methods are guaranteed to find the optimal solution assuming they have access to the MDP, specifically the transition and reward function. Temporal difference learning algorithms solve the MDP by interacting with the environment and making incremental updates to the solution. They are guaranteed to converge under general stochastic approximation conditions. Monte Carlo methods are suited to handle large problems by solving for the optimal policy given the current state. This approach is independent of the size of the state space. In this work we focus on temporal difference learning and Monte Carlo methods.

2.2 Temporal Difference Learning

Temporal difference learning defines a class of algorithms that learn the optimal value function online by directly interacting with the MDP. The algorithms do not assume direct access to the transition and reward function and as such only have access to samples from them. A set of transitions made by the RL agent from a starting state to any terminal state comprises a single episode. Every transition made by an RL agent is comprised of $\langle s, a, s', r \rangle$. The agent takes action a in state s, receives reward r and transitions to state s'. Each such transition is a sample of the transition and reward function of the underlying MDP. Q-learning [] is a temporal difference algorithm and perhaps one of the most commonly used methods in RL. Q-learning attempts to find the optimal Q-value function, $Q^*(s,a)$, by using a incremental recursive approach as shown here,

$$Q(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)) \tag{2.1}$$

Algorithm 1 Q-learning

repeat(for each episode):
  Initialize s
  repeat(for each step of episode):
    Choose a (ϵ-greedy action selection)
    Take action a, observe $R(s, a)$, s'
    Update $Q(s, a)$
    $s \leftarrow s'$
  until s is terminal
until end of learning

Here $\alpha \mapsto [0, 1]$ is the learning rate. The error term (inside the parenthesis) is called the TD-error or temporal difference error and represents the current error estimate of learning algorithm. The Q-learning algorithm is described below.

The agent selects actions in every state and updates the Q-function. Here the Q-function is stored in a tabular form of size $|S| \times |A|$ The algorithm is guaranteed to converge for a finite state space under stochastic approximation conditions and assuming every state-pair is visited an infinite number of times. Empirically the algorithm arrives at the optimal policy in finite time.

2.2.1 Exploration-Exploitation

An important step in temporal-difference learning algorithms like Q-learning is in how actions are selected at every step. This aspect is the main focus of this research. The goal of the RL agent is to find the optimal value function by visiting states and trying different actions. Searching for the RL solution creates a dilemma for an RL agent. At every decision step, the agent has to decide where to search or whether it has completed the search. The former is referred to as exploration while the latter is called exploitation. Thus the agent has to decide whether it has all the information to make the optimal decision or explore more in search of the optimal solution. The dilemma created here is called the *exploration-exploitation* dilemma and is a crucial aspect of every RL algorithm. The ideal RL algorithm balances this trade-off at every step and will arrive at the solution taking the

optimal number of steps.

There are several approaches in the existing literature that describes heuristics useful to tackle this dilemma. We provide a review of these approaches below. For the tabular case, a brute force approach of trying every action in every state an infinite number of times will guarantee the learning algorithm converges. In this work we tackle the problem of exploration - guiding the agent to different parts of the MDP which might lead to the optimal solution. We also tackle the role of human input in balancing the exploration-exploitation trade-off.

At this point, we would like to note that Q-learning is an *off-policy* algorithm. The algorithm makes recursive updates assuming that when the agent reaches the next state s', the optimal action will be chosen. However this is not always the case as the agent can explore and choose an action that could be suboptimal in s'. This is important aspect of Q-learning as it allows the agent to take random actions with the goal of exploring while always learning about the optimal value function.

2.2.2 Function Approximation

When dealing with large and/or continuous domains, it is often intractable to maintain a tabular representation of the Q-function. In such cases we use Q-learning with linear function approximation [24]. The Q-function is represented as a linear function of state-action features, $Q(s,a) = \phi(s,a)^T\theta$. Here $\phi(s)$ provide a feature-based representation of a state in an MDP. We can obtain $\phi(s,a)$ by duplicating the features $\phi(s)$ for all actions and only activate the ones for the action under consideration. For example if $|A| = 4$, $\phi(s,3) = [\mathbf{0}, \mathbf{0}, \phi(s), \mathbf{0}]^T$. For every transition, the Q-learning algorithm updates its weight vector, θ using first-order gradient methods in the following manner:

$$\begin{aligned} \delta &= R(s,a) + \gamma \max_{a' \in A} [\phi(s',a')^T\theta] - \phi(s,a)^T\theta \\ \theta &= \theta + \alpha\, \delta\, \phi(s,a) \end{aligned} \tag{2.2}$$

The error, δ is the Temporal Difference (TD) error (also loosely referred to as Bellman error). The algorithm, while not guaranteed to converge in the general case, is found to perform well in practice.

2.3 Monte Carlo Methods

Monte Carlo methods are a general approach to MDP planning that use online Monte-Carlo simulation to estimate Q-values. Monte Carlo Tree Search (MCTS) is one such algorithm. The basic observation behind MCTS algorithms is that for MDPs with $\gamma < 1$, there is an effective horizon H beyond which rewards do not significantly affect the optimal policy for the agent's *current state*. This places a theoretical (though perhaps still intractable) bound on the number of steps that must be considered to accurately estimate the Q-values of the current state.

MCTS algorithms perform a forward search from the current state, selecting and branching on actions and possible transitions from $P(s'|s,a)$, out to some depth d. From this search, we can estimate the d-horizon Q-values:

$$Q^d(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{d-1}(s',a') \tag{2.3}$$

where $Q^1(s,a) = R(s,a)$.

Note that Equation 2.3 requires iterating over both the set of actions and possible transitions in the MDP. The number of possible transitions defined by $T(s,a,s')$ is $|S|$, the total number of states; however, Kearns et al. 2002 showed that it is possible to obtain ϵ-optimal Q-value estimates for the current state from a set of sampled transitions, and that the number of samples C per state was independent of $|S|$.

Unfortunately, MCTS remains exponential in the *depth* of the tree. The sample complexity of uninformed MCTS is then $O(|A| * C)^H$ [25], corresponding to a depth-H tree. To address the exponential blow-up in H, practical MCTS implementations must typically

truncate the tree expansion at some depth or time threshold, and approximate the values of the leaf nodes by evaluating a fixed (possibly random) "roll-out" policy.

Monte Carlo methods have desirable properties when tackling large problems, however as explained they have a search problem that grows exponentially. In such a case, developing smart ways to truncate and explore the tree is useful and even necessary for MCTS to have a powerful impact on real problems. In this work, we will focus on what kind of biases are useful for exploration in MCTS and how we can instantiate and utilize them.

2.4 Exploration in Reinforcement Learning

Exploration in RL is commonly achieved using two methods: ϵ-greedy and softmax action selection. In ϵ-greedy action selection, exploration is performed by uniformly sampling a random action with probability ϵ and the current best action (greedy action selection) with probability $1 - \epsilon$. The choice of ϵ is left to the designer. In some cases, a decay schedule is used where the value of ϵ is decayed over time as the agent gains more domain experience. Softmax action selection takes a more informed approach to exploration. Instead of sampling a random action, the actions are weighted according to their respective Q-value estimates and sampled from the resulting distribution. The most common implementations use a Boltzmann distribution in the following manner: $\pi^B = \frac{e^{Q(s,a)/\tau}}{\sum_{a=1}^{|A|} Q(s,a)/\tau}$ where τ is a positive temperature parameter. A higher temperature value results in a more uniform distribution while a lower value results in greedy action selection. Besides using these standard approaches, there are several automated ways of exploring in RL.

Rmax [26] was an automatic approach introduced to perform exploration. It stems from the idea of optimism in the face of uncertainty. The agent is motivated to maintain visitation rates to states and attempt all actions that weren't tried before. The approach performs well in practice, however R-max scales exponentially in the number of state variables which makes it intractable for sufficiently large problems. There are several value-based methods like UCB [14], its variants [65, 27], Bayesian approaches [28] studied in the con-

text of multi-arm bandits, that perform effective exploration by maintaining statistics about changes in the value function and the number of times state-action pairs have been visited. While successful in smaller domains, these approaches run into sample complexity issues when dealing with high-dimensional long horizon domains we aim to solve. Model-based methods [15, 29, 30, 31] have had success in several domains, specifically in robotics where the dynamics of the robot and the environment they are acting in are either already known or a part of learning algorithm. These methods however are sensitive to any inherent noise and stochasticity in the model and can overfit to the errors in an imperfect model.

A smart exploration method was proposed by Gehring and Precup [32] which uses the residual (TD error) as a reward for a Q-function, whose implied policy is then used for exploration. The intuition behind this approach being that state-action pairs that have high residuals should be visited and tried more often. This method relies on stable estimates of the residual. We adapt a version of this approach in our experiments. Using an internal reward function [33] is an interesting approach to exploration in RL. In this method a user is required to design a reward function for skill learning which is often non-trivial. While the authors provide empirical results on small-sized domains, the idea presented is promising and warrants further exploration. An approach relevant to work presented in this dissertation is that of Active RL [34] where the authors model the problem as a POMDP. They model the sensitivities of the policy to the unknown transition and reward function and build exploration strategies focusing on these aspects of the problem. This method relies on using Newton's method to solve the problem until convergence (which cannot be guaranteed) and they need to solve the MDP numerous times to test the sensitivity. The work authored by Akiyama et al. [35] is similar to out work, where they leverage concepts of least squares approaches to guide exploration in policy iteration methods. The main difference is that the approach they design requires the problems to have certain properties with respect to the reward distribution and as such it is not directly comparable.

Related to the work presented in this dissertation is the notion of intrinsic motivation

[36]. The idea is for the learning agent to direct its sources of reward and optimize an intrinsic function. There are several measures that can be used to define such a intrinsic function ranging from uncertainty [37], curiosity [38], information gain and empowerment [20]. In each case, the agent uses a specific criteria to optimize and intrinsically motivate it. In addition to this, the agent uses an external sources of reward, often times related to the problem definition as extrinsic motivation. Breaking down the sources of reward in this way allows the agent to focus and optimize them separately. This approach has had a lot of success in skill learning [39, 40, 41] for reinforcement learning agents. The survey paper on intrinsic motivation [42] provides of greater detail on the topic.

2.4.1 Deep Reinforcement Learning

Deep reinforcement learning has seen a lot of recent success in solving complex game domains and robot related problems. Deep Q-learning [43], one of the initial approaches in this space, focused on using deep neural networks for value function approximation combined with experience replay and reward clipping to provide human-level performance in Atari games. Algorithms like DQN utilized traditional approaches to exploration (ϵ-greedy, softmax, etc.) and did not primarily focus on dealing with related challenges. Since this result, a variety of deep RL methods have been developed utilizing the actor-critic architecture.

Trust-region policy optimization [44] and proximal policy optimization [45] are examples of this. The approach uses the KullbackLeibler (KL) divergence criteria to ensure monotonic policy improvement by using a local approximation of the expected reward. The use of trust regions can potentially help exploration, though it focuses on stable generalization and policy improvement. Deep deterministic policy gradients [46] is an off-policy algorithm designed for continuous control with deep RL. Exploration in this case is achieved by adding noise sampled from a predefined noise process to the actions. There is flexibility here in being able to add noise from intelligent sources that have knowledge of the domain.

This approach has been found to work well in practice. Soft actor critic algorithm [47] takes a principled approach to combining generalization and exploration for deep reinforcement learning. The algorithm modifies the reward function by taking in to account the entropy in the critic and actor update steps. The approach makes use of a scaling parameter which controls the balance between exploration and exploitation. Experimentally this parameter has been found to sensitive and hard to tune for different domains.

With improvements in algorithms, several approaches have been developed that focus primarily on exploration for deep RL. These include Bootstrapped DQN [48] which relies on starting with an initial random policy, samples a value function from its posterior and uses bootstrapping to approximate the true value function. Depending on the prior chosen, the algorithm helps the agent explore diverse aspects of the state and action space and experiments reveal that the extent of exploration depends a great deal on the prior chosen to compute the value function. Random Network Distillation [18] computes exploration bonuses from the predictions error of a random neural network. The approach shows that a relatively simplistic approach can help the agent explore novel parts of the state space. The motivation here is that the errors will be low on states that have been visited while higher on novel states. Another similar approach is count-based exploration [16] where hashmaps are maintained for all visited states and actions along with the number of times they have been visited. This is then used as a bonus to drive the agent to reach novel aspects of the domain. Several of these approaches drive exploration by the notion of novelty. The intrinsic curiosity [49] approach promotes exploration via curiosity. It formulates curiosity as the ability to predict the outcomes of its actions in a learned embedding of the world. The errors in its prediction are used as reward to guide the agent towards exploring these regions of the state-action space.

Some of the most successful methods in deep RL recently include AlphaGo [50], MuZero [8] where monte carlo tree search methods are used to plan online and explore either by random action selection or by using heuristics (like UCB [14]). These methods

are strong function approximators and have done well given sufficient training time and data.

2.5 Hierarchical Reinforcement Learning

Action abstractions like Options [51] were introduced in the hierarchical reinforcement learning literature as a principled approach to learning from temporally extended actions. They instantiate policies which represent different sub-tasks for a problem and use them to accelerate planning. Constraints introduced more recently [52] instantiate policies that capture negative outcomes in a domain, by looking over multiple timesteps, and use that information to guide action selection for the agent. Guliz and Feigh [53] were able to show that humans solving problems (specifically game domains) by using these action abstractions. These approaches have been used to solve problems independently [54] and together [55].

There have been other methods introduced in the literature that have approached the problem of combining different forms of action abstraction. One approach is the Concurrent Actions Model [56, 57] which formally describes a framework where an agent plans over concurrent temporally extended actions. These actions have different kinds of termination schemes which are similar to abstractions used in our approach. In their work, they highlight that the bottleneck for their approach is an efficient way of searching through the space of multi-actions that can be run in parallel. Our PGSS algorithm aims to solve exactly that problem. [58] constructs different types of skills and uses Q-learning to learn domain specific skill combinations. We note that in their work it is not clear how non-terminating skills can be utilized in the Q-learning framework. Overall while our approach has the same motivation as theirs, the intelligent use of the action abstractions in MCTS helps to overcome the exploration complexities of Q-learning. Recent work closely related to ideas presented in this dissertation [59] use options as action abstractions in MCTS to solve partially observable MDPs. While their method does not utilize constraints as action

abstractions, they show advantages of temporal actions for MCTS planning.

Taking advantage of the action abstractions as domain knowledge includes the cost of defining them for the problems we would like to solve. There are several methods that aim to solve the problem of instantiating options [60, 61, 62, 63, 64] and constraints [52] either automatically or using human input. We add that devising automated ways of instantiating these abstractions is not the main focus of our work. We will show in our experiments that our incorporation of domain knowledge in into MCTS achieves compelling gains over complex problems that potentially offset the initial computations spent in instantiation.

2.6 Monte Carlo Methods

Algorithms such as Upper-Confidence Trees (UCT) [65] and Forward Search Sparse Sampling (FSSS) [66] attempt to generate the bias by using relative Q-values. The intuition is that if we had high confidence in $Q(s, a_1) > Q(s, a_2)$ for two actions a_1, a_2, obtaining further samples from s, a_2 would be wasteful: they cannot change the Q-value of s. From this we see that the best case search policy is in fact the optimal policy π^*, as it wastes no samples on sub-optimal trajectories. The search policy can have a significant role in the complexity of MCTS: with an optimal policy, the required number of samples for an accurate Q-estimate is closer to $C * H$ than $(|A| * C)^H$.

The state of the art techniques in MCTS [67] include SS [25], UCT [65], its variants and FSSS [66]. They provide different ways of performing action selection in MCTS. The respective exploration strategies depend on good Q-value estimates which are often time-consuming and hard to obtain. The results also depend heavily on the choice of parameters used for learning (number of sampled trajectories and branching depth). We seek to circumvent this problem by directly incorporating domain knowledge in the form of action abstractions to bias action selection. Recently MCTS has been combined with deep learning methods [68] to facilitate function approximation in large game domains [69, 50]. While these methods are able to leverage the generalization capabilities of deep networks to

generate state-of-the-art performance, they require large amounts of training data to learn the parameters of deep neural network models. We argue for the incorporation of domain knowledge to help exploration in MCTS without significant computational costs. We note that there has been prior work on using MCTS with expert knowledge [70]. This approach focuses on using human knowledge of the boardgame Go to define a comprehensive set of rules that help in directing exploration of the tree. While the idea behind this approach is similar to ours, their implementation encodes expert knowledge as part of the computations that measure value confidence bounds and requires careful tuning of several coefficients which sometimes result in conflicting learning objectives.

2.7 Interactive Machine Learning

There is a wide variety of work in the field of Interactive Machine Learning, namely Learning by Demonstration [71], Imitation Learning [72], Policy Shaping [73] and TAMER [74]. These approaches aim to learn the optimal policy from human critique or demonstrations. A key feature of Reinforcement Learning is the use of a reward signal. The reward signal can be modified to suit the addition of a new information source (this is known as *reward shaping* [75]). This is the most common way human feedback has been applied to RL [76, 77, 78, 79, 80]. However, several difficulties arise when integrating human feedback signals that may be infrequent, or occasionally inconsistent with the optimal policy–violating the necessary and sufficient condition that a shaping function be potential-based [75]. Another difficulty is the ambiguity of translating a statement like "yes, that's right" or "no, that's wrong" into a reward. Manual processing of the data can yield *ad hoc* approximations for specific domains. Researchers have also extended reward shaping to account for idiosyncrasies in human input. For example, adding a drift parameter to account for the human tendency to give less feedback over time [76, 81].

Advancements in recent work sidestep some of these issues by showing human feedback can instead be used as policy feedback. For example, Thomaz and Breazeal [82]

added an *UNDO* function to the negative feedback signal, which forced an agent to backtrack to the previous state after its value update. Work by Knox and Stone [74, 83] has shown that a general improvement to learning from human feedback is possible if it is used to directly modify the action selection mechanism of the Reinforcement Learning algorithm. Although both approaches use human feedback to modify an agent's exploration policy, they still treat human feedback as either a reward or an estimate of the extended utility of taking an action. In our work, we assume human feedback is not an evaluative reward, but is a label on the optimality of actions. Thus the human's feedback is making a direct statement about the policy itself, rather than influencing the policy through a reward. Our work similarly focuses on people's knowledge of the policy where we want to allow people to simply critique the agent's behavior ("that was right/wrong"). Related work in the area of transfer learning [84, 85], where an agent learns with "advice" on how it should behave. This advice is provided as first order logic rules and is also provided offline, rather than interactively during learning. Our approach only requires very high-level feedback (right/wrong) and is provided interactively.

Active Reward Learning [86] have been used to learn a reward function from human feedback and use that in an RL algorithm. They use the human to provide input on task executions - a score to the execution - that they then smooth using Gaussian Processes and Bayesian Optimization. Reward function design in general is known to be a hard problem as there are always possibilities of loops in the learned policy. It is not clear how the Active Reward Learning approach overcomes this problem. A paper similar in theme to the work presented is one on active imitation learning by state queries [87]. The authors present an approach where the human interacts with the agent by giving a optimal action in a specific state or by saying that the state is bad. The query-states are chosen by a query-by-committee approach based on Bayesian Active Learning. In their approach, they assume the learner has access to a simulator of the MDP and also do not explicitly handle the case where humans provide a bad state response - they simply memorize those states.

In other works, rather than have the human input be a reward shaping input, the human provides demonstrations of the optimal policy. Several papers have shown how the policy information in human demonstrations can be used for inverse optimal control [88, 89], for teaching [90], to seed an agent's exploration [91, 92], and in some cases be used DAgger [93] is a no-regret online learning approach used for supervised learning. The method learns from training data and then executes the learned model. For every mistake made, the human demonstrator provides more examples in that space. These examples are appended to the training set and the learner is retrained. While it is not strictly an RL approach, it is mainly providing examples useful for a supervised learner. We note here that in many cases, methods in the literature make assumptions on optimality of human information used to assist machine learning.

Preference based reinforcement learning [94] relies on the idea of using preferences from domain experts to design the goals of a problem and solve the problem by satisfying as many of the preferences as possible. Researchers have shown that using this formulation helps overcome reward shaping problems by allowing algorithms to use non-numeric rewards as well as reducing the dependence on an machine learning expert to carefully design the reward function. However there are several open questions with this approach regarding exploration in large domains, satisfying multiple objectives and scalability to large domains with deep networks. This remains an active area of research.

CHAPTER 3

AGENT-GUIDED EXPLORATION FROM DEMONSTRATION

In this chapter, we demonstrate how to explore the search space of a problem using human help. We design an exploration policy using human demonstrations and use the learned policy to guide the RL agent. An issue that often comes up in such approaches is that the policies learned in this manner can be ineffective for the agent as the demonstrations are not necessarily helping the agent's model. To address this concern, we draw inspiration from concepts of active learning and design a agent-guided approach to obtaining demonstrations. We use statistical properties of least squares methods to query the agent's learned model and communicate its uncertainty to the human. The human can now provide data to the agent in parts of the search space that are useful from the agent's perspective.

Expected Contributions - We present a sample efficient method towards human-guided exploration for an RL agent. We show how our method can be used to efficiently solve problems with high-dimensional state variables, long horizons and sparse reward functions.

3.1 Approach

In this section we describe an interactive approach to exploration in reinforcement learning using statistical properties relevant to the underlying learning algorithm. We outline properties of exploration policies, statistical measures useful for this purpose and how these measures can be used to solicit interaction that drives the agent's exploration.

3.1.1 Exploration Policies

For sequential decision-making problems, exploration policies are used to guide the agent to different parts of the search space so as to obtain good estimates of the value function while covering as much of the state-action space as possible. There are a number of factors

affecting this exploration such as the dynamics of the domain, the sparsity of rewards, the size of the state-action space and the problem horizon (steps to goal). In addition to these properties, a subtle but important aspect of exploration that is implicit in existing methods is that exploration policies are not strictly stationary. As the agent gathers more information about the world, the exploration policy changes to accommodate the learned model.

Traditional methods, as explained earlier, explore using a variety of methods ranging from a uniformly random policy to value-based heuristics. These methods are prone to redundant exploration and (in some cases) expensive sample requirements. The idea of inefficient exploration also applies to methods involving human interaction, as human data is often limited to specific regions of the search space while relying on the learning algorithm to generalize effectively. To account for the characteristics of exploration policies while overcoming the limitations of existing methods, we present a policy-based approach to exploration. We solicit demonstrations based on the agent's uncertainty about its model to guide the agent to cover the search space more efficiently. For a given MDP, model uncertainty arises from a combination of stochastic elements in the domain (transition and reward function) and insufficiently explored states and actions. Keeping this in mind, we investigate properties of relevant RL algorithms and select measures that serve to characterize the model uncertainty with the goal of designing effective exploration policies.

In our work we use Q-learning with linear function approximation which uses gradient methods to perform optimization. The loss function used is akin to minimizing the squared loss of the Bellman error [95]. Using this information, we use statistical properties of analogous methods that have been well-studied, like linear regression or least squares, to understand the impact of each data point or observation on the learning agent's model.

3.1.2 Statistical Measures

In order to understand the agent's model uncertainty, we review statistical analysis of linear regression methods [23] to find measures useful for our purposes. In linear regression prob-

lems, input observations can be scored by their *influence* to measure the effect they have on the learned model. A high influence score points towards observations that merit further investigation. Influence is computed as a combination of two measures: Leverage and Discrepancy. Leverage is a measure of how far a specific observation is from the convex hull of known observations. It helps recognize outliers and can also be considered a measure of novelty. Discrepancy is related to how much an observation contributes towards model error. It is computed for each data point and captures the goodness of fit for the model being trained by introspecting the model error if that datapoint were to be removed from the dataset. From the perspective of exploration in RL, these measures help to identify novel parts of the state-action space (using Leverage) and how much the observations already experienced contribute to model error (using Discrepancy). A key insight into using these type of measures for RL is that the data the agent trains on does not contain any outliers as every observation made by the agent, by interacting with the domain, is relevant to solving the MDP. This indicates that there is essentially no data to be discarded. We hypothesize that an RL agent that actively explores the observations that have high leverage and high discrepancy, i.e. overall high influence, will lead to more efficient exploration to solve the MDP.

In order to utilize these statistical measures we explicitly set up the problem as a linear set of equations that correspond to the standard form, $X\beta = y$. An RL agent is solving the MDP to optimize the function, $Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$ where $Q(s,a) = \phi(s,a)^T\theta$. It is straightforward to see that the left-hand side of the optimization function, $Q(s,a)$ or $\phi(s,a)^T\theta$ takes the place of $X\beta$ and the right-hand side forms y. The input data for our approach comes from transitions of the RL agent as it is attempting to solve the problem. The state-action features, $\phi(s,a)$ observed by the agent during transitions are used to populate the rows of the data matrix, X ($n \times k$ for n observations and k features). Given this formulation we define the statistical measures useful for exploration.

Leverage

Leverage is a measure useful to determine how well the state-action space has been covered as it detects outliers in the data. Given independent variables, X, we compute leverage, h using the hat matrix, H as follows [23]:

$$H = X(X^T X)^{-1} X^T \tag{3.1}$$

The hat matrix maps the vector of dependent variables (y) to the vector of fitted values, $\hat{y} = Hy$. The diagonal elements of the hat matrix, h_{ii} are the leverages, which describe the influence each dependent variable value has on the fitted value for observation i. Leverage values are in the range $[0, 1]$. A high value indicates that the observation is an outlier and vice versa. A fixed threshold parameter is used to detect the presence of outliers. We use 0.5 as the cut-off to indicate if an observation is an outlier. For RL problems, a high leverage indicates that the respective state-action pair is an outlier, i.e. is novel and has not been visited often.

Typically RL algorithms require large amounts of data to solve the MDP which makes it infeasible to store all the transitions in a batch. In addition to that, computing leverage can pose computational issues as it requires taking the inverse of a matrix of size $k \times k$ (for k features) which can be very large for high-dimensional problems. To address the memory and computational concerns, we use the Sherman-Morrison formula to incrementally compute the inverse of $X^T X$. The formula is stated as follows:

$$(A + x^T x)^{-1} = A^{-1} - \frac{A^{-1} x^T x\, A^{-1}}{1 + x A^{-1} x^T} \tag{3.2}$$

where A^{-1} is initially set to $\frac{1}{\delta} I$ (identity matrix of size $k \times k$) and δ is a small positive number, say $1e^{-4}$. Using this computation, the leverage of an instance x of data matrix X can be computed as $x A^{-1} x^T$.

Discrepancy

This measure captures the observations that the learning algorithm is unable to model thus leading to large errors. Discrepancy is computed using the externally studentized residual [23]. These residuals are obtained by computing the residual for an observation and dividing it by the standard error (or standard deviation). This is done to reduce the effect of the variance in the errors and allow residuals to be compared. An externally studentized residual is one that computes the residual by taking into account the difference in the learned model with and without the observation in question. For observation i, the discrepancy, t_i can computed as:

$$\begin{aligned} t_i &= \frac{e_i}{\sqrt{MSE_{(i)}(1-h_{ii})}} \\ MSE_{(i)} &= \frac{(n-p)MSE - \frac{e_i^2}{(1-h_{ii})}}{n-p-1} \end{aligned} \tag{3.3}$$

Here MSE represents the mean-squared error, n is the number of samples, p the number of independent variables, h_{ii} is the leverage for observation i and e_i is the TD error for sample i. $MSE_{(i)}$ is the mean squared error for the model based on all observations excluding sample i. We note that MSE is typically computed using batch data which is computationally infeasible to store in large scale RL domains. It does not lend itself to incremental computations due to the $\max$ operator in the RL optimization function (when computing MSE and e_i). We circumvent this problem by storing a batch data matrix, update it with new observations using a (FIFO) sliding window and compute the required parameters [96] online. In the analysis of linear systems, when the absolute value of the externally studentized residual, $|t_i|$ is greater than 2, the corresponding observation is considered an outlier that needs further investigation. Henceforth we use the term discrepancy to represent the externally studentized residual.

Note that the observations that are marked as high leverage and/or high discrepancy continue to be visited until the stateful hat matrix A and/or the trained model no longer recognize these observations as influential.

Using these measures the RL agent can identify observations in the MDP that require further exploration. We now describe how we use them to solicit demonstrations for exploration.

3.1.3 Demonstration Query

For every transition made by the RL agent, it computes the leverage and discrepancy and compares it to the respective thresholds. If either threshold is exceeded, we identify the corresponding observation as influential. To learn more about that observation and reduce its influence, we learn a policy using guidance from a person or a simulated oracle that drives the agent towards these observations. The goal of the human/oracle is to prescribe the shortest path for the flagged observation. We consider an automatic approach of computing the path in the next chapter.

Consider the state associated with an influential observation the agent transitioned into as s^+. To encourage exploration to s^+, it is important to bridge the gap between regions of low influence to those of high influence. Intuitively this can be explained by the idea that state-action pairs that have low influence are likely to have been frequently visited and sufficiently explored. Therefore designing a policy from parts of the state-action space that the agent knows well and visits often to those with high influence is most likely to encourage exploration to and around s^+. As explained before, leverage provides a way to identify data points that have been visited often. To acquire the necessary low influence data point that is to be connected to s^+, we review every observation i, in the current episode and compute the corresponding leverage: $h_i = \phi(s_i, a_i)A^{-1}\phi(s_i, a_i)^T$. We then compute the mean leverage, μ_h from these observations and find the state, s_i that corresponds to the data point with the closest leverage,

$$\underset{i}{\operatorname{argmin}} |h_i - \mu_h| \tag{3.4}$$

Once the low and high influence observations have been identified, we collect exploratory demonstrations either from a person or a simulated oracle using a Graphical User Interface (GUI) for the domain. When the algorithm queries for demonstrations, the GUI highlights the states that need to be connected by demonstrations. Using the GUI, the user can a) provide demonstration(s), b) choose to ignore the query and c) stop interacting with the algorithm altogether. The simulated oracle provides demonstrations by following the shortest distance path between the queried states. There are no inherent assumptions made about quality or quantity of demonstrations. The only requirement is for the user to be knowledgeable about the MDP dynamics to help the agent navigate in the domain. For every demonstration provided, we learn an oracle exploration policy π^O using standard supervised learning algorithms and sample an action from this policy when the agent decides to explore.

We note here that when soliciting demonstrations, the final state in the demonstration may be different from the query state requested by the agent. This is likely to be observed in domains with stochastic elements and/or non-playable characters. While there is no straightforward way to ensure a certain state is visited in an MDP, our experimental results show that the policy learned using our approach is effective at driving the agent towards influential parts of the MDP as it continues to actively request user demonstrations.

3.1.4 Action Selection

The oracle exploration policy, π^O defines a policy that when followed is likely to guide the agent from regions of low influence to those of high influence. However using such a policy alone to explore in and of itself can be insufficient for the purposes of RL where the goal is to arrive at the optimal policy as soon as possible. Additionally the nature of exploration is non-stationary and as such if there are limited demonstrations, the agent is less likely to explore the set of influential regions in the MDP. To account for these properties, we design

our exploration policy as follows:

$$\pi^E \propto (\pi^O + \pi^L) \cdot \pi^B \tag{3.5}$$

where π^O is the oracle exploration policy, π^L is the leverage value $\forall a \in A$ for state s and π^B represents the Boltzmann exploration policy. We note that the leverage values lie in the range $[0, 1]$ and for our purposes can be used as probabilities. A leverage value closer to 1 will have the effect of sampling the corresponding action more often. Intuitively π^E represents the exploration policy that chooses between the oracle demonstration or the leverage values, weighted by the softmax Q-values of the actions in the state. This exploration policy allows the agent to reach regions of high influence using human demonstrations or select actions with high leverage while actively seeking the goal. We use π^E in an ϵ-greedy fashion to facilitate exploration in our approach.

3.1.5 Exploration from Demonstration

We now outline our approach with all the pieces defined using Algorithm Block 2. The objective of Exploration from Demonstration (EfD) is to learn the optimal policy using RL while ensuring the agent actively explores regions of the state-action space that have a potentially large influence on the learned model.

EfD as described in Algorithm Block 2 has a tendency to query the user demonstrations repeatedly as high influence regions are often in close proximity to others. This results in the leverage and discrepancy thresholds being crossed very often within the same episode. In order to make EfD more user-friendly we include a predefined fixed time period, T_s where the agent executes self-play without any user queries. During self-play Q-learning and leverage parameters (θ & A^{-1}) continue to be updated. We note that EfD does not modify any theoretical guarantees of the methods used as Q-learning is an off-policy algorithm. This completes the description of our approach.

Algorithm 2 Exploration from Demonstration (EfD)

repeat(for each episode):
  Initialize s
  repeat(for each step of episode):
    Compute π^E (Eqn. 3.5)
    Choose a (ϵ-greedy action selection using π^E)
    Take action a, observe r, s'
    Store transitions $\langle s, a, r, s' \rangle$
    Update θ and A^{-1} (Eqn. 2.2 & 4.4)
    Compute leverage and discrepancy (Eqn. 3.1 & 4.5)
    if high influence at s **then**
      $s^+ \leftarrow s$
      Compute starting state, s_i (Eqn. 3.4)
      Query demonstrations from s_i to s^+
      Update θ and A^{-1}
      Self-play for T_s (includes parameter updates)
    end if
    $s \leftarrow s'$
  until s is terminal
  Decay ϵ
until end of learning

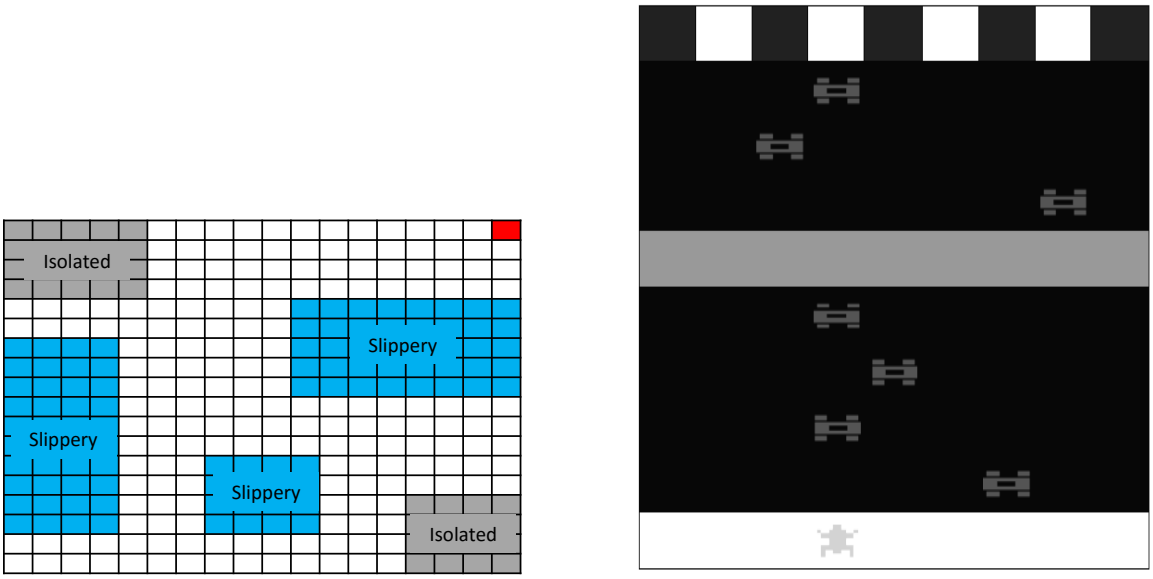
3.2 Experimental Setup

To validate the performance of EfD we conduct experiments on a gridworld and popular arcade game domain and compare our method to several baselines. In this section we describe the domains used in our experiments and the relevant baselines.

3.2.1 Domains

We use two domains to empirically highlight the performance and properties of EfD. We represent these domains as MDPs in the following manner:

Gridworld. This domain is designed by adapting the specifications outlined in [32]. In addition to the introduction provided in Chapter 1 5.4, we describe the specific experimental setup here. We implement an 18×18 discrete grid (Figure 3.1a) with four deterministic actions that move the agent up, down, left and right. The goal is to reach the top right corner



(a) Gridworld (b) Frogger1x

Figure 3.1: A snapshot of the two domains used in our experiments. The first is a Gridworld with regions of interest and the second is the Frogger domain of size 1x.

of the grid. For every step taken, the agent accrues a step cost of -1 and a reward of 0 at the goal state. The blue shaded regions represent slippery squares. If the agent transitions out of a slippery square (both into an unshaded square or another slippery square), the reward is uniformly distributed in the interval $[-12, +10]$. The gray shaded regions represent isolated squares. Any transition that leads the agent into this region from an empty square has a 0.1 probability of success. Once inside, movements within the isolated region as well as those leading out are not restricted. Transitions leading into the bounding walls keeps the agent's state unchanged. An episode starts with the agent randomly placed in the grid and stops when the agent reaches the goal. We used identity features to represent the state space.

Frogger. We utilize the game of Frogger (Section 5.4 in our experiments. The state space of the domain consists of the agent's position along with the position and direction of travel of cars in the grid and represented using binary features. The domain can be made more complex, i.e. have a longer solution horizon, by increasing the number of intermediate rows between the start and goal positions. We use this property to show how our method

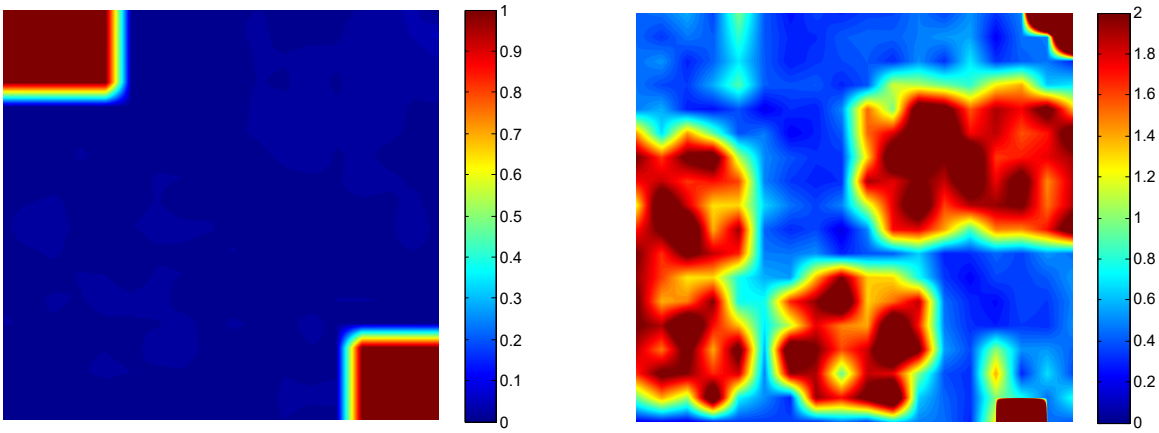
scales with the size of the domain. We refer to the domain configuration in Figure 3.1b as Frogger1x and use Frogger2x to indicate doubling the number of rows used in Frogger1x and Frogger4x to indicate a quadruple version of the same.

3.2.2 Baselines

We implement five baselines in our experiments and compare their performance to EfD. Uniform random exploration and softmax exploration comprise two of the baselines. The remaining three are defined as follows:

Learning from Demonstration + RL. In this method we acquire demonstrations of optimal behavior from people or a simulated oracle. These demonstrations are used to learn a policy using supervised learning methods (in this case logistic regression). We use this policy as the seed policy to initiate RL. We execute the algorithm on the domain and report the results.

Exploration by TD error. This approach draws from insights highlighted in this work [32] and learns an exploration policy based on TD error. In our implementation we use the current estimate of TD error (the absolute value) as the reward for a given state-action pair and learn a Q-function using this information. A policy is extracted from the Q-function using softmax action selection. The Q-function learned in this process plays the role of driving the agent towards parts of the state-action space that have high TD error in order to gather more information in those regions. We note that this approach is not strictly consistent with standard MDP assumptions as the reward function is non-stationary (TD error is constantly changing). While we counteract this effect to a certain degree by using a small learning rate and a decaying exploration parameter, our experiments show that performance was not greatly affected by this characteristic.



(a) Leverage heat map

(b) Discrepancy heat map

Figure 3.2: Leverage and discrepancy heat maps generated from random exploration of the gridworld domain.

Exploration by Leverage. We derive an exploration policy by computing the leverage on data consisting of visited state-action pairs. For any given state, we compute the leverage for all actions, normalize the results and use it as a distribution from which we sample exploratory actions. As explained earlier leverage captures outliers in the data, which in this case would represent actions, for a given state, that have not been tried often. This way by sampling from normalized leverage values for all actions in a state, the agent is more likely to sample new actions. This baseline is useful to signify the importance of exploratory demonstrations.

3.3 Experiments and Results

We implement EfD for the chosen domains and highlight the results achieved along with several tests that provide insight into EfD's performance under different experimental conditions.

3.3.1 Using Leverage and Discrepancy for Exploration

In this experiment we use the gridworld (Figure 3.1a) to show the utility of leverage and discrepancy as useful measures to guide exploration. The gridworld is suitable for this purpose due to several design choices made. Firstly the isolated (gray) regions in the grid represent parts of the state-space that are hard to reach and thus unlikely to be explored by the agent. Secondly the non-deterministic reward function (represented by slippery blue patches on the grid) pose some difficulty to the learning algorithm in accurately modeling the underlying value-function. We conduct two experiments to show the individual contribution of the chosen exploration measures.

To highlight the utility of leverage, we perform a random walk in the gridworld for 50 episodes. In each episode, the agent starts in a random position and moves randomly until the goal is reached. For every step taken, features of visited state-action pairs are used to form the data matrix X. Using X, we compute the hat matrix H (see Equation 3.1) and use that to derive the leverage $h(s, a)$ for every state-action pair.

In Figure 3.2a, we plot a heat map using $\max_a h(s, a)\ \forall s \in S$. The heat map shows the correspondence between the regions of high leverage ($h \geq 0.5$) and hard to explore regions of the gridworld (isolated gray regions). Leverage, as explained earlier, is used to identify outliers in the data. In this case the heat map presents the gray region as outliers which necessitates the need for further exploration. Analogously this effect can be seen in other more complex high-dimensional domains where it is hard to entirely cover the state-action space. Leverage, as shown here, captures regions that the learning agent is unable to reach often.

While leverage captures how often state-action pairs have been visited, it does not capture details about the transition function, reward function and RL algorithm's learned model. Here we show how discrepancy is useful for this task. We perform Q-learning with linear function approximation on the gridworld domain for 50 episodes. We set $\gamma = 0.99$, $\alpha = 0.1$ and $\epsilon = 1.0$. We store all the transitions $\langle s, a, r, s' \rangle$ from the sampled episodes

and compute the mean squared error using the learned weight vector θ (refer Equation 4.5). The mean squared error is used to derive the discrepancy $t(s, a)$ for every state-action pair.

In Figure 3.2b, we plot a heat map using $\max_a |t(s, a)| \ \forall s \in S$. The heat map shows the correspondence between the regions of high discrepancy ($|t| > 2$) and the slippery regions in the gridworld. Intuitively this is to be expected as the TD error in these areas is likely to have large magnitude and high variance and that warrants further investigation. We note that the bright red patch in the top right corner of the heat map signifies the high residual obtained at the goal and its adjoining states. Using this heat map as a threshold for exploration would draw the agent towards the slippery patches and the goal until the learning algorithm captures the underlying model and the residual decreases.

The experiment serves to highlight the roles played by leverage and discrepancy and how they guide the agent's exploration. Leverage guides the agent towards state-action pairs that have not been visited often during learning and the discrepancy guides the agent towards regions of the domain which the learning algorithm has difficulty modeling the value-function.

3.3.2 EfD for Frogger

In this experiment we instantiate the EfD algorithm in the Frogger domain for different sizes of the problem, Frogger 1x (10 rows), Frogger 2x (18 rows) and Frogger 4x (34 rows). We perform Q-learning with linear function approximation with $\gamma = 0.99$, $\alpha = 0.0006$ and $\epsilon_{start} = 0.8$. We use the following decay schedule for the exploration parameter: $\epsilon = \frac{\epsilon_{start} \times N_0}{(N_0 + Ep\#)}$ where N_0 is the decay rate and $Ep\#$ is the current episode number. We set N_0 as 1500, 2500 and 5000 respectively for the three versions of Frogger. The threshold parameters for EfD were fixed with the leverage threshold set at 0.5 and discrepancy threshold at 2. The self-play time period for EfD was set to $T_s = 500$, $T_s = 2000$ and $T_s = 5000$ steps respectively for the three versions of Frogger. The temperature for softmax Boltzmann exploration was set to 50. We acquired demonstrations from two users

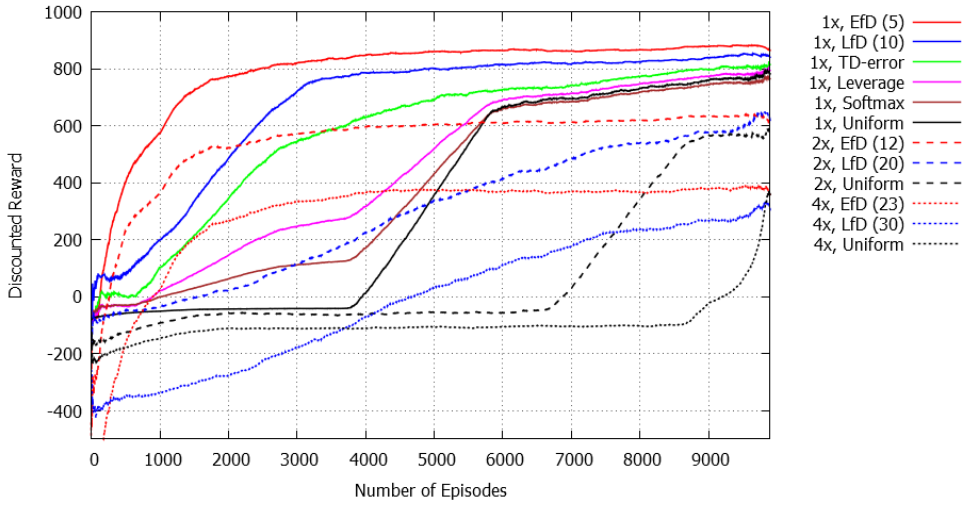
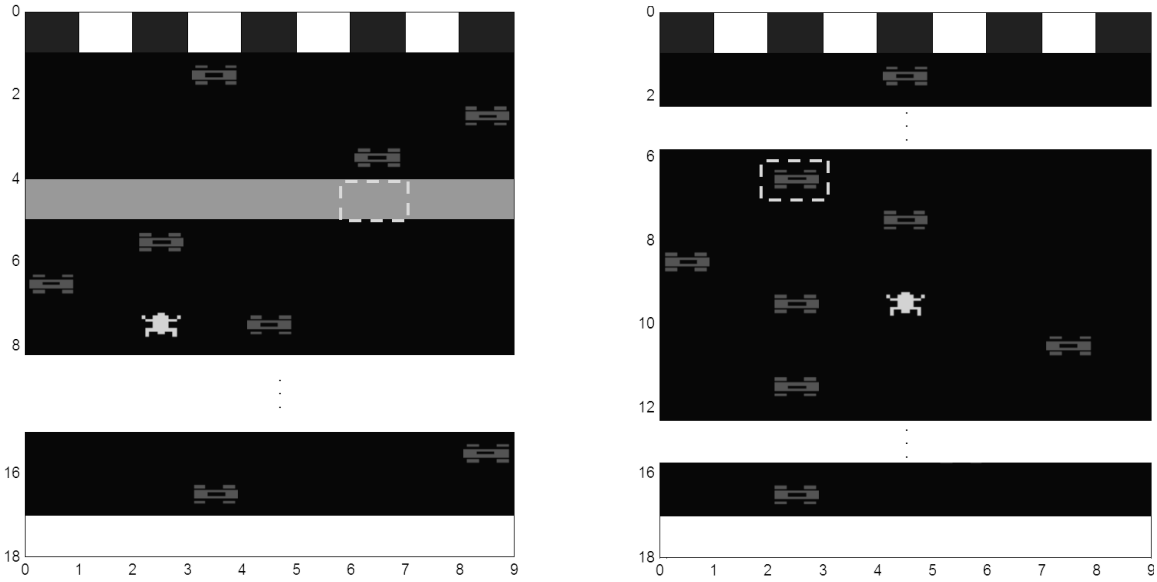


Figure 3.3: Performance of EfD and baselines on the Frogger game domain of varying sizes (1x, 2x and 4x) averaged over 10 trials. The numbers in parenthesis are the average number of input user demonstrations.

who were both familiar with the dynamics of the game. We received anywhere from 5 to 30 demonstrations depending on the size of the domain that was being tested. Demonstration time in total was no more than 10 to 15 mins. The human policy was learned using logistic regression with a learning rate of 0.01. We plot the results of this experiment along with comparative baselines in Figure 3.3. We see that EfD converges to the optimal policy faster than the baselines using a small number of demonstrations. To ease readability we plot only a subset of the baseline methods in Figure 3.3 as the performance of the baselines (TD-error, Leverage and Softmax) were consistent across the three sizes. The performance is further improved over the baselines as the size of the domain is increased. This is explained by highlighting how EfD queries and utilizes user demonstrations. In the initial stages of learning, the user is queried with demonstrations leading to states in the rows closer to the bottom row. As the agent gains experience, demonstrations are requested for states further up. In this process, the agent incrementally explores the rows until it finally reaches the goal in the top row. Such an incremental learning approach makes it easier for agent to reach the goal as well as easier for the user to provide demonstrations. This also explains why LfD (+ RL) does not perform as well as EfD for larger grids. The size of



(a) Agent queries the user for demonstrations to the highlighted position based on the leverage threshold.

(b) Agent queries the user for demonstrations to the highlighted position based on the discrepancy threshold.

Figure 3.4: Examples of the types of states queried by the agent during EfD when applied to Frogger2x (18 rows).

the domain limits the search space covered by the human demonstrations as well as prohibits optimal demonstrations from start to the goal. EfD performs better by using the RL algorithm's inductive bias as well as the underlying representation to acquire incremental demonstrations that are most useful to the agent. These results were consistent across both users. The TD-error and leverage baseline methods while more informed do not perform as well due to their redundant exploration. An interesting observation of EfD from our experiments is that, by using thresholds for the statistical measures, with sufficient experience the agent automatically ceases to request demonstrations. In which case, we observe that the agent has enough information to model the Q-function and solve the MDP.

3.3.3 Types of States Queried

Here we take a closer look at the types of states queried by the agent during EfD. We present results from the Frogger2x domain which consists of 18 rows from start to goal. Figure 3.4a is an example of an agent query, based on the leverage measure, where a demonstration is

required between the frog near row 8 to the highlighted grid position near row 4. An observation that exceeds the leverage threshold indicates that the agent has not visited that state-action pair often and therefore requires input demonstrations. Such a query points towards how EfD gathers information about the state-action - decomposing the domain in smaller regions. Demonstrations are requested from known regions to unknown regions and often they are in close proximity to each other. Providing a demonstration for such a query would be easier than providing optimal demonstrations from start to end in this domain. Figure 3.4b is an example of an agent query based on the discrepancy threshold. We note that the highlighted position is around a car near row 6 which is a terminal state with reward -100. The discrepancy here exceeded the threshold as the agent's current model was unable to make an accurate prediction of the Q-value of an action in this state and thus requested a demonstration.

We would also like to highlight a few uncommon queries that provide interesting insights into the method. In some cases the agent requests demonstrations from a state where the frog is closer to the goal to states where the frog is further away. From the perspective of solving the MDP, using such a policy would encode suboptimal information, however for policy-based exploration, it only serves to get a better estimate of the Q-function. Note that exploration is carried out by combining the human policy with softmax policy (Section 3.1.4) which therefore ensures the agent select actions that are more likely to lead it towards the goal. For some queries we observe that the agent's position on the grid remains the same for both start and final states, while the position of cars is different. With respect to EfD, these are different states and therefore it is a valid demonstration query. This shows how EfD makes demonstration queries by taking into account the underlying representation.

3.3.4 Effect of Input Demonstrations and Threshold Parameters

In this experiment, we test the sensitivity of EfD to the quality of demonstrations used to learn the exploration policy. While we do not place any assumptions on the quality of demonstrations, we analyze the degrees to which performance is affected as the quality of demonstration is varied. We use the simulated oracle for this experiment under different demonstration noise conditions: Oracle0.1, Oracle0.3, Oracle0.5. An oracle with noise 0.1 (Oracle0.1) will provide the required demonstration 90% of the time and 10% of the time, take random actions. The results of this experiment are summarized in Table 3.1. As

	Frogger1x (5)	Frogger2x (12)	Frogger4x (23)
Oracle0.0	**2560 $\pm$ 150**	**3194 $\pm$ 230**	**4430 $\pm$ 410**
Oracle0.1	2752 $\pm$ 321	3470 $\pm$ 527	4893 $\pm$ 564
Oracle0.3	2648 $\pm$ 469	3304 $\pm$ 699	5218 $\pm$ 866
Oracle0.5	5102 $\pm$ 932	6329 $\pm$ 875	7688 $\pm$ 1043
ϵ-greedy	6570 $\pm$ 120	8555 $\pm$ 212	10000 $\pm$ 405

Table 3.1: EfD performance as a function of the quality of input demonstrations from a simulated oracle in the Frogger domain. The values represent the number of episodes taken by each method to converge to the optimal policy. The numbers in parenthesis are the number of input demonstrations the simulated oracles are limited to. We include results from the ϵ-greedy baseline for comparison. The results are averaged over 10 trials.

evidenced by the table, the performance of EfD varies based on the quality of input demonstrations. Relative to Oracle0.0 (optimal oracle demonstrations), Oracle0.1 and Oracle0.3 achieve similar performance. This is explained by the fact that for most query demonstrations there exist multiple ways to reach the desired state and neither path is any more optimal than the other from the perspective of exploration. By introducing noise in the oracle demonstrations, they can potentially explore more states than Oracle0.0 which can include both low and high influence observations. However this has the effect of increased variance in performance for noisy simulated oracles. Oracle0.5 (with 50% random action selection) has large variance in its performance but still outperforms ϵ-greedy uniform random exploration.

In our experiments, we set the leverage threshold at 0.5 and the discrepancy threshold

at 2. Changes to these parameters directly affect the number of demonstrations queried by the agent which affects the amount of exploration carried out by the agent. Higher values results in fewer demonstration queries and as a result most of the exploration is carried out by the agent autonomously. On the other hand lower thresholds result in frequent queries which has the effect of learning an exploration policy close to a uniform policy. In general, from tests in our domain, we find that setting leverage threshold to 0.5 and discrepancy threshold anywhere in the range $[2, 6]$ provides the best results.

3.4 Summary and Discussion

Here we highlight the benefits of specifically learning an exploration policy for RL in the context of EfD. When faced with large domains with sparse rewards and long horizons, a policy-based approach is less vulnerable to the large sample requirements of value-based methods as the information acquired from a single demonstration allows the agent to extends its range of exploration over multiple timesteps. Additionally such a method does not concern itself solely with reward information. The statistical measures used in EfD (leverage and discrepancy) focus on different aspects of the MDP which allow the algorithm to function well across a wider class of problems. This is in contrast to value-based methods which rely on large samples of reward information to estimate the uncertainty in the value function often made complicated in sparse reward and long horizon domains. Also EfD does not require optimal demonstrations to learn but instead demonstrations that serve to connect two regions of the agent's choice. As these demonstrations are used for exploration, they can be potentially noisy (which may in some cases help the agent).

In this chapter we presented a model-free policy-based approach called Exploration from Demonstration (EfD) that performs interactive exploration for RL algorithms. Our method adapts statistical measures of linear regression to capture aspects of an MDP that are important to explore and model in order to learn the optimal Q-function. We highlight the properties of these measures in an instructional gridworld MDP and empirically

test our approach on a popular arcade game under different experimental conditions. We show how EfD scales to larger problems and outperforms baselines using only exploratory demonstrations while placing very few requirements on the quality and quantity of input data. Our method is particularly suited to problems which have a long horizon and sparse rewards as well as those domains where optimal demonstrations are hard to acquire. In the future we would like to extend EfD to learn a model of the MDP, thus allowing the algorithm to request examples from arbitrary states rather that waiting to transition to those areas. Another interesting avenue for future work is the idea of extending EfD to work with more data efficient RL algorithms like Least Squares Policy Iteration [97].

CHAPTER 4

AUTONOMOUS AGENT-GUIDED EXPLORATION

In the previous chapter we show how human demonstrations can be used to guide exploration. The demonstrations serve to help connect regions of the state-action space based on statistical properties of the underlying learning algorithm (linear regression in this case). As such the demonstrations drive the agent from well-modeled areas of the MDP to explore other areas of interest that are influential. We show the EfD approach is successful at solving domains with sparse rewards, long horizons and does not require optimal demonstrations from people.

While EfD has several desirable properties, we make several algorithmic assumptions that limit its applicability across a general class of MDPs. We primarily assume that the availability of external input in the form of sample demonstrations. When available, this can be very helpful for learning however it places a strong requirement on applicability of the algorithm for different problems. Additionally it requires the human interacting be familiar with the dynamics of the domain and be able to interpret the visual representation to provide demonstrations. The algorithm is also not ideally suited to handle highly stochastic domains and domains with a large number of actions due to additional computational costs.

To overcome some of these obstacles, we design an *autonomous* policy-based approach to explore using the same statistical measures described in the earlier chapter. We begin by providing a high-level perspective of the approach, followed by a detailed description of the algorithm, theoretical properties and experiments on a classical control problem and game domains while comparing it to existing autonomous exploration strategies.

4.1 Approach

To facilitate automatic exploration, it is important to guide the agent towards aspects of the MDP useful to learn the optimal policy. We seek to learn an exploration policy that captures information that the statistical measures help the agent in identifying. To achieve this autonomously, we use the computed measures as a source of reward in an auxiliary MDP that when solved provides a policy that facilitates exploration.

In this section we describe the design of the exploration MDP, associated reward functions and how the policies learned can be used for exploration in RL algorithms.

4.1.1 Autonomous Exploration

In the previous chapter we highlighted the choice of statistical measures and how they were used as criteria to solicit human demonstrations. The demonstrations guide the agent towards parts of the MDP that are a) hard to reach (leverage) and b) hard to model (discrepancy). To extend the work to a larger class of problems, it is desirable to learn these policies autonomously without human assistance.

To that end we setup the problem of learning exploration policies as solving a auxiliary MDP, $M^{exp} = \langle S, A, \mathcal{T}, R^{exp}, \gamma \rangle$. The tuple follows the original MDP M, with a modified reward function R^{exp}. The reward function is defined based on the statistical criteria which we use to learn an exploration policy by solving the MDP, M^{exp}.

This setup is designed to motivate the agent to learn a policy that encourages it to reach states with high values for leverage and discrepancy. As a consequence, more data is gathered in these areas of interest which in turn helps lower the computed statistical measures over time and helps the agent move on to other areas. Overall the agent would be solving for both the optimal policy for the original MDP, M while also solving for an exploration strategy towards areas of interest using M^{exp}.

To solve the MDPs, we make use of off-policy RL algorithms like Q-learning. To

facilitate online learning of the exploration policy, it requires a) careful reward function design and b) online computation of the statistical measures. In the following sections, we describe online computations of the criteria, the exploration reward functions and finally show how we can use them to setup and solve the MDP.

4.1.2 Reward Functions for Exploration

To learn the exploration policy associated with leverage and discrepancy, we design an artificial reward function for each metric. We setup the rewards to be a function of the computed leverage and discrepancy values. We bound the range of these metrics and rescale them to be used as a reward function. Assuming the desired reward function falls in the range $[R^{exp}_{min}, R^{exp}_{max}]$, for the given metric, we can compute the reward function as

$$R^{exp} = R^{exp}_{min} + \frac{R^{exp}_{max} - R^{exp}_{min}}{I_{max} - I_{min}} \tag{4.1}$$

Here $[I_{min}, I_{max}]$ represent the range for the chosen metric of influence (leverage or discrepancy). Leverage has a bounded range $[0, 1]$. Numbers closer to 1 indicate that the datapoints are likely to be outliers and as such would want these datapoints to have a higher reward to encourage the agent explore them. Discrepancy has a lower bound of 0 but no upper bound. From linear regression literature [23], values greater than 3 are typically considered influential and worth exploring. To provide a broad range for the reward function, we set the range for this to be $[0, 9]$. By scaling the influence metrics to act as a reward function, we can design it without the need for thresholds (required in the EfD work). The scaling has the effect of rewarding states and actions that have high leverage or discrepancy.

We can now compute the reward function for leverage, R^{exp}_L and discrepancy, R^{exp}_D using the above formulas and provided ranges. This provides with the necessary reward

function for our exploration MDP, M^{exp}:

$$R^{exp} = R_L^{exp} + R_D^{exp} \tag{4.2}$$

We set up a reward function, one for each metric, and use them to create an exploration MDP. Solving this MDP provides a policy that will guide the agent towards points of influence (taking into account both metrics). We note that this reward function is non-stationary as the metrics calculated change over time. This aspect is crucial to exploration as exploring the global search space can be non-stationary as the agent gains more experience. Solving the MDP under these conditions is not guaranteed to converge, though we will show that policies learned help the agent explore.

4.1.3 Computing Statistical Measures Online

To compute the reward function, it is necessary to be able to compute leverage and discrepancy online with every transition. Typically these measures are computed over a batch of data stored in memory. This approach is challenging when working with complex domains due to potential memory constraints and inherent computational complexity. Here we show how we can compute leverage and discrepancy incrementally to be used to define the necessary reward components.

Leverage

Leverage, as explained in the previous chapter, is a measure useful to determine how well the state-action space has been covered as it detects outliers in the data. For a given state-action pair, the leverage is computed as,

$$h_i = \phi(s_i, a_i) A^{-1} \phi(s_i, a_i)^T \tag{4.3}$$

To facilitate autonomous learning, we would need to setup incremental computation of

the inverse of matrix A. This can set up using the Sherman-Morrison formula:

$$(A + x^T x)^{-1} = A^{-1} - \frac{A^{-1}x^T x\, A^{-1}}{1 + xA^{-1}x^T} \tag{4.4}$$

With every transition, we update the inverse of the matrix using this formulation. The variable x here represents a sample from the transitions: state-action features.

Discrepancy

In the EfD work, we stored transitions and computed the discrepancy on a moving window of data. This is memory intensive in complex domains which often require a large number of samples to be able to solve for the model. To compute the required reward function, we provide a formulation where the discrepancy can be computed online without storing data as transitions in memory. Here is the formula to compute the externally studentized residual or discrepancy:

$$t_i = \frac{e_i}{\sqrt{MSE(1 - h_{ii})}} \tag{4.5}$$

While h_{ii} or the leverage is available from A matrix, computing the mean squared error (MSE) in this case poses some constraints when working without a batch of data in memory. To compute this metric online, we have deal with errors in the value function V, which makes the equation we are solving for,

$$\begin{aligned} \phi(s)\theta_V &= r + \gamma\phi(s')\theta_V \\ (\phi(s) - \gamma\phi(s'))\theta_V &= r \end{aligned} \tag{4.6}$$

With basic matrix algebra by multiplying both sides by $(\phi(s) - \gamma\phi(s'))^T$ we can com-

pute a analytical solution to the value function.

$$
\begin{aligned}
(\phi(s) - \gamma\phi(s'))^T(\phi(s) - \gamma\phi(s'))\theta_V &= (\phi(s) - \gamma\phi(s'))^T r \\
\theta_V &= [(\phi(s) - \gamma\phi(s'))^T(\phi(s) - \gamma\phi(s'))]^-1(\phi(s) - \gamma\phi(s'))^T r
\end{aligned}
\tag{4.7}
$$

Here solving for θ_V can be achieved by using a least squares temporal difference learning approach (LSTD) [98]. This is important also when computing the studentized residual. We note the LSTD learning for the value function is an on-policy method. To learn this incrementally, as before, we use the Sherman Morrison formula to compute the require matrix inverse. While this gives us the least squares weights, in order to compute the discrepancy we need to compute the mean squared error over all the data points. We now provide a mathematical approach to computing the MSE incrementally. The MSE we are computing:

$$
\sum_t [r_t - (\phi(s_t) - \gamma\phi(s'_t))\theta_{V_t}]^2 \tag{4.8}
$$

We can expand the equation by taking the square and separating the individual components of the resulting equation. This allows us to incrementally compute the sum over the experienced transition samples as follows:

$$
\begin{aligned}
D &= D + \phi(s)\phi(s)^T + \gamma^2\phi(s')\phi(s')^T - 2\gamma\phi(s)\phi(s')^T \\
F &= F + 2r(\gamma\phi(s')^T - \phi(s)^T) \\
G &= G + r^2
\end{aligned}
\tag{4.9}
$$

The above equations (with $D =$, $F = 0$ and G = 0 to begin with) allow us to compute the MSE for least squares regression in a online manner. An important aspect that facilitates this is the on-policy aspect of the value function, ie. it does not concern itself with the actions directly. Using these updates, we can compute the mean squared error in the

following manner:

$$\begin{aligned} SSE &= \sum(diag(D(\theta_V \theta_V^T))) + F\theta_V + G \\ MSE &= \frac{SSE}{(n-p-1)} \end{aligned} \tag{4.10}$$

With every transition taken by the agent, we can compute the learned weights, compute the MSE and the discrepancy online without the need to store data as transitions.

4.1.4 Learning the Exploration Policy

With the reward functions defined and an online approach to computing the required metrics, we can use them to solve for the exploration policies. Any choice of RL algorithm can used here to solve the exploration MDP. We use Q-learning with function approximation which performs regression using gradient descent. The update rules follow the standard implementation [6].

Sampling Policy . With every sample we update the Q-function for exploration and are in the process learning $Q^{exp}(s,a) = \theta^{exp}\phi(s,a)$. We can select action with maximum value as our choice, $\text{argmax}\, Q^{exp}(s,a)$. This selection can be used as part of an ϵ-greedy strategy where the action for exploration can be selected from the policy implied by $Q^{exp}(s,a)$ instead of uniform random selection. We note that the selection of the exploratory action in this case is deterministic and not probabilistic.

4.1.5 Automatic Policy Exploration

We now outline the overall approach putting together the individual pieces described earlier.

The automatic exploration strategy improves upon the work in the earlier chapter by removing the thresholds for leverage and discrepancy. The main inputs to the algorithm as the ϵ parameter and the associate decay schedule if any. We will show in the experiments

Algorithm 3 Automatic Policy Exploration

repeat(for each episode):
  Initialize s
  repeat(for each step of episode):
    Select action a according to Q^{exp}
    Take action a, observe r, s'
    Update θ_Q, MSE, A^{-1},
    Compute leverage and discrepancy
    Compute the reward function for leverage and discrepancy
    Update θ^{exp}
    $s \leftarrow s'$
  until s is terminal
until end of learning

that results are stable to the choice of the reward function ranges for the statistical metrics.

4.1.6 Properties

Here we provide some insight into the theoretical properties of this approach.

Convergence Any off-policy reinforcement learning algorithms are suitable with our approach and as such the convergence properties of these algorithms remain unaltered.

Parameter Optimization Our approach does not involve parameters introduced in the EfD work, namely the removal of thresholds for the statistic measures, the need for mixing time and we will show the reward parameters are domain agnostic. As the action selection is deterministic, we remove the need for conversion of Q-values to action polices using softmax action selection. This removes the need for the boltzmann parameter. It does include the ϵ parameter to give the algorithm a stochastic chance to explore.

Start-state Distribution . This approach to exploration is agnostic to the state state distribution. It uses the statistical criteria to guide exploration and the start state distribution only control how often and how fast state-actions pairs are visited.

Complexity . The algorithm has a computational complexity $O(n^2)$ where n is dimensions of feature space. This is due to the matrix operations to compute the least squares solution as well as the incremental computation of the mean squared error. Additionally as we not storing any samples, the memory complexity is in the worst case $O(n^2)$ as we storing a matrix.

4.2 Experiments

In this section we instantiate the algorithm in different problems to evaluate its performance. We utilize gridworlds, a classical control problem and game domains to test the algorithm's performance in continuous environments, stochastic conditions and domains with long horizons and sparse rewards.

4.2.1 Baselines

Count-based Exploration [16]. In this approach, counts are maintained directly for the tabular case and using hash maps for the continuous case to keep track of state visitations and use them as intrinsic rewards to bias action selection for exploration.

Continuous Texplore [99]. This represents a model-based approach to learning to explore for continuous domains. The approach builds multiple regression trees to learn to predict the next state and uses directed planning to solve for the optimal policy.

We also compare our approach to traditional forms of exploration: softmax action selection, optimism in the face of uncertainty and random exploration.

4.2.2 Instructional MDP - Gridworld

The gridworld used in this experiment is of size 3×13. The agent has to navigate from the state labeled 'S' to the red square labeled 'G'. The agent has four directional actions with

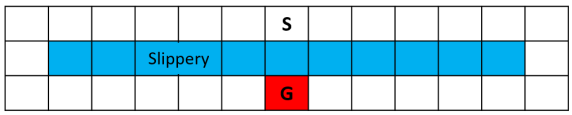


Figure 4.1: Instructional Gridworld domain with start state, goal state and blue regions of interest

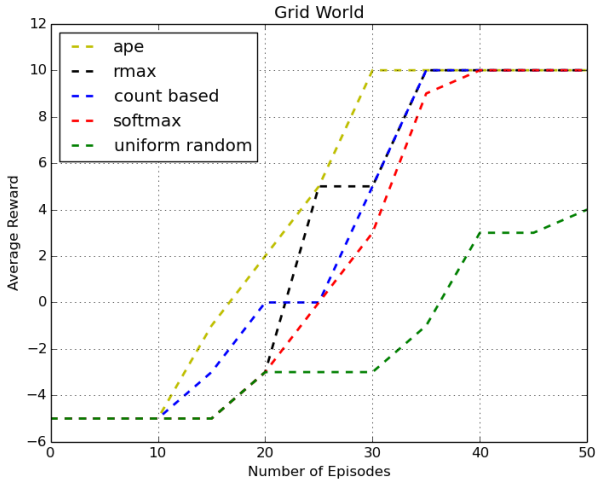


Figure 4.2: Learning results for the Gridworld domain averaged over 10 trials

deterministic transitions. The blue squares represent puddles in the grid which penalize the agent. The reward at the goal state is 10, -5 for stepping into the blue square and a step cost of 0. In this domain the agent has to learn to reach the red square by avoiding the puddle squares.

We compare APE Q-learning with the baseline algorithms. The results shown in Figure 4.2 show improved performance over random exploration for the algorithms that attempt to explore in a more meaningful manner. APE, Count-based and RMax approaches do particularly well as they explore optimistically to find the path around the slippery region to reach the goal on the other side. It is non-trivial for a uniform random approach to chance upon the optimal path without sufficient number of samples.

4.2.3 Classic Control - CartPole

In the cart pole domain 5.4, the goal is keep the pole balanced with discrete forces applied to cart. To represent the state space for learning, we made use of Fourier basis functions [100] of order 3. This resulted in a 256-dimensional state space of real numbers in the range $[-1$ $1]$. This domain is useful to test exploration algorithms in continuous state space domains and to see how well they capture the underlying structure and value function.

We compare the approaches mentioned earlier with the APE algorithm and results are shown in 4.3. We see that APE and the count-based method have slow starts as their criteria encourages them to explore all parts of the search space to better understand their effects on the model. Specifically the discrepancy measure leads the APE agent to explore falling over repeatedly until it is able to model that outcome well.

Alternatively Texplore focuses its efforts in parts of the problem that are more relevant to optimal policy and as such has a better start. The differences between the methods appear closer towards convergence where APE converges to better solution earlier using a more complete understanding of the underlying structure while Texplore is subject to the noise captured in its model predictions. APE is able to achieve competitive performance without the need for learning a model or keep track of visitations which in different domains can lead to redundant and unsafe exploration.

4.2.4 Game Domain - Frogger

In this experiment, we recreate the experiments in work on EfD and show that we can learn the optimal policy using APE without human help. The results are shown in Figure 4.4. They compare the performance of APE with other relevant baselines. We show how APE achieves the optimal policy faster than the baselines but slower than EfD. This is understandable as domain knowledge in the form of human demonstrations was useful to learn the respective exploration policies.

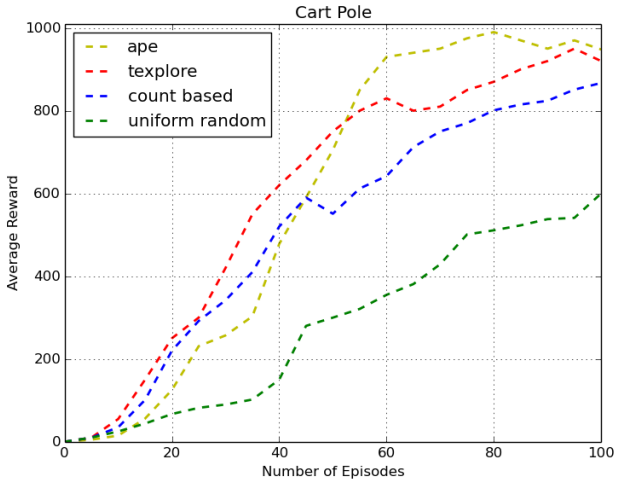


Figure 4.3: CartPole training results averaged over 10 trials

4.3 Discussion and Conclusion

In this work, we present automated approach to explore using the statistical measures described in the previous work. The agent learns two exploration policies that guides it towards novel states as well as states that are hard to model.

An immediate assumption the algorithm makes is the value function is linear in the features used to represent the domain. While this is applicable in several domain, it is not always trivial to instantiate such features for all domains. To extend this approach to deep reinforcement learning algorithms, a trivial solution is to perform the required computations using the weights of last layer of network, assuming they represent a fully connected layer with linear activation (which is the case most often). In the future we are exploring more principled ways of allowing APE to work with deep reinforcement learning algorithms.

We would also like to add that APE does not directly tackle the exploration-exploitation trade-off. The algorithm explores the MDP and as it has satisfied its criteria for exploration, begins to exploit. In the strictest sense, it does not balance the trade-off though it does automate the transition between the two based on the samples experienced by the agent.

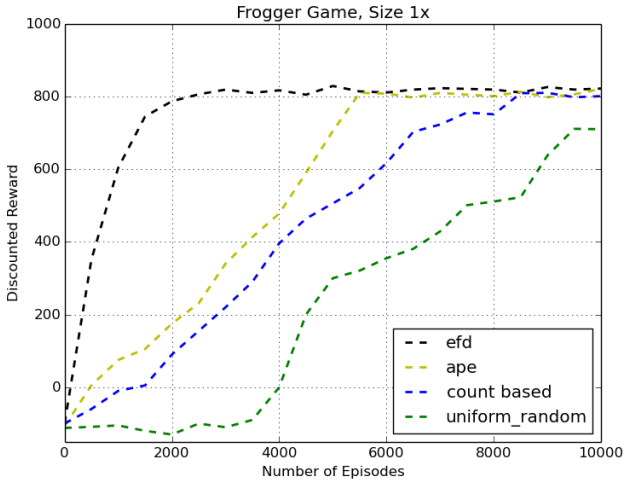


Figure 4.4: Learning results from Frogger from using autonomous exploration strategies with EfD learning from human assistance as a reference.

Additionally as a result of switching between exploration and exploitation, the algorithm supports non-stationary environments as well which is where several algorithms have difficulty solving. The natural switch allows APE to detect changes in the environment due to reward function or transition function and the statistical measures are able to capture the change and proceed the explore them.

We are interested in scaling APE to work with continuous actions. In the current formulation, it is not clear how to compute leverage and residual in domains with continuous state and actions.

CHAPTER 5
HUMAN-GUIDED EXPLORATION USING POLICY SHAPING

A long term goal of Interactive Reinforcement Learning is to incorporate non-expert human feedback to solve complex tasks. Some state-of-the-art methods have approached this problem by mapping human information to rewards and long term expected utilities of actions and iterating over them to compute better control policies. In this work we argue for an alternate, more effective characterization of human feedback: Policy Shaping. We introduce **Advise**, a Bayesian approach that attempts to maximize the information gained from human feedback by utilizing it as direct policy labels. We compare **Advise** to state-of-the-art approaches and show that it can outperform them and is robust to infrequent and inconsistent human feedback.

5.1 Introduction

A long–term goal of machine learning is to create systems that can be interactively trained or guided by non-expert end-users. This chapter focuses specifically on integrating human feedback with Reinforcement Learning. One way to address this problem is to treat human feedback as a shaping reward [76, 77, 78, 79, 80]. Yet, recent papers have observed that a more effective use of human feedback is as direct information about policies [82, 74]. Most techniques for learning from human feedback still, however, convert feedback signals into a reward or an expected action utility. We introduce *Policy Shaping*, which formalizes the *meaning* of human feedback as policy feedback, and demonstrates how to use it directly as policy advice. We also introduce **Advise**, an algorithm for estimating a human's Bayes optimal feedback policy and a technique for combining this with the policy formed from the agent's direct experience in the environment (Bayesian Q-Learning).

We validate our approach using a series of experiments. These experiments use a sim-

ulated human teacher and allow us to systematically test performance under a variety of conditions of infrequent and inconsistent feedback. The results demonstrate two advantages of **Advise**: 1) it is able to outperform state of the art techniques for integrating human feedback with Reinforcement Learning; and 2) by formalizing human feedback, we avoid ad hoc parameter settings and are robust to infrequent and inconsistent feedback.

5.2 Reinforcement Learning

We use an implementation of the Bayesian Q-learning (BQL) Reinforcement Learning algorithm [28], which is based on Watkins' Q-learning [101]. Q-learning is one way to find an optimal policy from the environment reward signal. The policy for the whole state space is iteratively refined by dynamically updating a table of Q-values. A specific Q-value, $Q[s, a]$, represents a point estimate of the long-term expected discounted reward for taking action a in state s.

Rather than keep a point estimate of the long-term discounted reward for each state-action pair, Bayesian Q-learning maintains parameters that specify a normal distribution with unknown mean and precision for each Q-value. This representation has the advantage that it approximates the agent's uncertainty in the optimality of each action, which makes the problem of optimizing the exploration/exploitation trade-off straightforward. Because the Normal-Gamma (NG) distribution is the conjugate prior for the normal distribution, the mean and the precision are estimated using a NG distribution with hyperparameters $\langle \mu_0^{s,a}$, $\lambda^{s,a}$, $\alpha^{s,a}$, $\beta^{s,a} \rangle$. These values are updated each time an agent performs an action a in state s, accumulates reward r, and transitions to a new state s'. Details on how these parameters are updated can be found in [28]. Because BQL is known to under-explore, $\beta^{s,a}$ is updated as shown in [102] using an additional parameter θ.

The NG distribution for each Q-value can be used to estimate the probability that each action $a \in A_s$ in a state s is optimal, which defines a policy, π_R, used for action selection. The optimal action can be estimated by sampling each $\hat{Q}(s, a)$ and taking the argmax. A

large number of samples can be used to approximate the probability an action is optimal by simply counting the number of times an action has the highest Q-value [28].

5.3 Policy Shaping

In this section, we formulate human feedback as policy advice, and derive a Bayes optimal algorithm for converting that feedback into a policy. We also describe how to combine the feedback policy with the policy of an underlying Reinforcement Learning algorithm. We call our approach **Advise**.

5.3.1 Model Parameters

We assume a scenario where the agent has access to communication from a human during its learning process. In addition to receiving environmental reward, the agent may receive a "right"/"wrong" label after performing an action. In related work, these labels are converted into shaping rewards (e.g., "right" becomes $+1$ and "wrong" -1), which are then used to modify Q-values, or to bias action selection. In contrast, we use this label directly to infer what the human believes is the optimal policy in the labeled state.

Using feedback in this way is not a trivial matter of pruning actions from the search tree. Feedback can be both inconsistent with the optimal policy and sparsely provided. Here, we assume a human providing feedback knows the right answer, but noise in the feedback channel introduces inconsistencies between what the human intends to communicate and what the agent observes. Thus, feedback is consistent, $\mathcal{C}$, with the optimal policy with probability $0 < \mathcal{C} < 1$.[1]

We also assume that a human watching an agent learn may not provide feedback after every single action, thus the likelihood, $\mathcal{L}$, of receiving feedback has probability $0 < \mathcal{L} < 1$. In the event feedback is received, it is interpreted as a comment on the optimality of the action just performed. The issue of credit assignment that naturally arises with learning

[1] Note that the consistency of feedback is not the same as the human's or the agent's confidence the feedback is correct.

from real human feedback is left for future work (see [83] for an implementation of credit assignment in a different framework for learning from human feedback).

5.3.2 Estimating a Policy from Feedback

It is possible that the human may know any number of different optimal actions in a state, the probability an action, a, in a particular state, s, is optimal is independent of what labels were provided to the other actions. Subsequently, the probability s, a is optimal can be computed using only the "right" and "wrong" labels associated with it. We define $\Delta_{s,a}$ to be the difference between the number of "right" and "wrong" labels. The probability s, a is optimal can be obtained using the binomial distribution as:

$$\frac{\mathcal{C}^{\Delta_{s,a}}}{\mathcal{C}^{\Delta_{s,a}} + (1-\mathcal{C})^{\Delta_{s,a}}}\,, \tag{5.1}$$

Although many different actions may be optimal in a given state, we will assume for this work that the human knows only one optimal action, which is the one they intend to communicate. In that case, an action, a, is optimal in state s if no other action is optimal (i.e., whether it is optimal now also depends on the labels to the other actions in the state). More formally:

$$\mathcal{C}^{\Delta_{s,a}}(1-\mathcal{C})^{\sum_{j\neq a}\Delta_{s,j}} \tag{5.2}$$

We take Equation 5.2 to be the probability of performing s, a according to the feedback policy, π_F (i.e., the value of $\pi_F(s, a)$). This is the Bayes optimal feedback policy given the "right" and "wrong" labels seen, the value for $\mathcal{C}$, and that only one action is optimal per state. This is obtained by application of Bayes' rule in conjunction with the binomial distribution and enforcing independence conditions arising from our assumption that there is only one optimal action. A detailed derivation of the above results is available in the Appendix Section A.1 and A.2.

5.3.3 Reconciling Policy Information from Multiple Sources

Because the use of **Advise** assumes an underlying Reinforcement Learning algorithm will also be used (e.g., here we use BQL), the policies derived from multiple information sources must be reconciled. Although there is a chance, $\mathcal{C}$, that a human could make a mistake when s/he does provide feedback, given sufficient time, with the likelihood of feedback, $\mathcal{L} > 0.0$ and the consistency of feedback $\mathcal{C} \neq 0.5$, the total amount of information received from the human should be enough for the the agent to choose the optimal policy with probability 1.0. Of course, an agent will also be learning on its own at the same time and therefore may converge to its own optimal policy much sooner than it learns the human's policy. Before an agent is completely confident in either policy, however, it has to determine what action to perform using the policy information each provides.

We combine the policies from multiple information sources by multiplying them together: $\pi \propto \pi_R \times \pi_F$. Multiplying distributions together is the Bayes optimal method for combining probabilities from (conditionally) independent sources [103], and has been used to solve other machine learning problems as well (e.g., [104]). This is one of the primary advantages of working directly in policy space to combine information from multiple sources. Note that BQL can only approximately estimate the uncertainty that each action is optimal from the environment reward signal. Rather than use a different combination method to compensate for the fact that BQL converges too quickly, we introduced the exploration tuning parameter, θ, from [102], that can be manually tuned until BQL performs close to optimal.

5.4 Experimental Setup

We evaluate our approach using two game domains, Pac-Man and Frogger (see Fig. 5.1). These domains present popular games which would be familiar to most people as well as well as an understanding of how to play the game well. A more detailed description of the

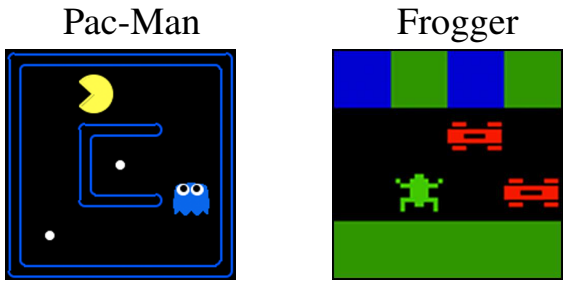


Figure 5.1: A snapshot of each domain used for the experiments. Pac-Man consisted of a 5x5 grid world with the yellow Pac-Man avatar, two white food pellets, and a blue ghost. Frogger consisted of a 4x4 grid world with the green Frogger avatar, two red cars, and two blue water hazards.

domains is available in Chapter 1 .

5.4.1 Constructing an Oracle

We used a simulated oracle in the place of human feedback, because this allows us to systematically vary the parameters of feedback likelihood, $\mathcal{L}$, and consistency, $\mathcal{C}$ and test different learning settings in which human feedback is less than ideal. The oracle was created manually by a human before the experiments by hand labeling the optimal actions in each state. For states with multiple optimal actions, a small negative reward (-10) was added to the environment reward signal of the extra optimal state-action pairs to preserve the assumption that only one action be optimal in each state.

5.5 Experiments

5.5.1 A Comparison to the State of the Art

In this evaluation we compare Policy Shaping with **Advise** to the more traditional Reward Shaping, as well as recent Interactive Reinforcement Learning techniques. Knox and Stone [74, 83] tried eight different strategies for combining feedback with an environmental reward signal and they found that two strategies, *Action Biasing* and *Control Sharing*, consistently produced the best results. These methods use human feedback rewards to modify

the policy, rather than shape the MDP reward function to learn an alternate utility function. These strategies directly influence policy action selection and are the closest to our proposed method. As will be seen, **Advise** has similar performance to these state of the art methods, but is more robust to a noisy signal from the human and other parameter changes.

Action Biasing uses human feedback to bias the action selection mechanism of the underlying RL algorithm. Positive and negative feedback is declared a reward r_h, and $-r_h$, respectively. A table of values, $H[s, a]$ stores the feedback signal for s, a. The modified action selection mechanism is given as $\text{argmax}_a \hat{Q}(s, a) + \text{B}[s, a] * H[s, a]$, where $\hat{Q}(s, a)$ is an estimate of the long-term expected discounted reward for s, a from BQL, and $\text{B}[s, a]$ controls the influence of feedback on learning. The value of $\text{B}[s, a]$ is incremented by a constant b when feedback is received for s, a, and is decayed by a constant d at all other time steps.

Control Sharing modifies the action selection mechanism directly with the addition of a transition between 1) the action that gains an agent the maximum known reward according to feedback, and 2) the policy produced using the original action selection method. The transition is defined as the probability $P(a = \text{argmax}_a H[s, a]) = min(\text{B}[s, a], 1.0)$. An agent transfers control to a feedback policy as feedback is received, and begins to switch control to the underlying RL algorithm as $\text{B}[s, a]$ decays. Although feedback is initially interpreted as a reward, Control Sharing does not use that information, and thus is unaffected if the value of r_h is changed.

Reward Shaping, the traditional approach to learning from feedback, works by modifying the MDP reward. Feedback is first converted into a reward, r_h, or $-r_h$. The modified MDP reward function is $R'(s, a) \leftarrow R(s, a) + \text{B}[s, a] * H[s, a]$. The values to $\text{B}[s, a]$ and $H[s, a]$ are updated as above.

The parameters to each method were manually tuned before the experiments to maximize learning performance. We initialized the BQL hyperparameters to $\langle \mu_0^{s,a} = 0, \lambda^{s,a} = 0.01, \alpha^{s,a} = 1000, \beta^{s,a} = 0.0000 \rangle$, which resulted in random initial Q-values. We set the

	Ideal Case ($\mathcal{L}=1.0, \mathcal{C}=1.0$)		Reduced Consistency ($\mathcal{L}=0.1, \mathcal{C}=1.0$)		Reduced Frequency ($\mathcal{L}=1.0, \mathcal{C}=0.55$)		Moderate Case ($\mathcal{L}=0.5, \mathcal{C}=0.8$)	
	Pac-Man	Frogger	Pac-Man	Frogger	Pac-Man	Frogger	Pac-Man	Frogger
BQL + Action Biasing	0.58 ± 0.02	0.16 ± 0.05	-0.33 ± 0.17	0.05 ± 0.06	0.16 ± 0.04	0.04 ± 0.06	$\mathbf{0.25 \pm 0.04}$	0.09 ± 0.06
BQL + Control Sharing	0.34 ± 0.03	0.07 ± 0.06	-2.87 ± 0.12	-0.32 ± 0.13	0.01 ± 0.12	0.02 ± 0.07	-0.18 ± 0.19	0.01 ± 0.07
BQL + Reward Shaping	0.54 ± 0.02	0.11 ± 0.07	-0.47 ± 0.30	0 ± 0.08	0.14 ± 0.04	0.03 ± 0.07	0.17 ± 0.12	0.05 ± 0.07
BQL + **Advise**	$\mathbf{0.77 \pm 0.02}$	$\mathbf{0.45 \pm 0.04}$	$\mathbf{-0.01 \pm 0.11}$	0.02 ± 0.07	$\mathbf{0.21 \pm 0.05}$	$\mathbf{0.16 \pm 0.06}$	0.13 ± 0.08	$\mathbf{0.22 \pm 0.06}$

Table 5.1: Comparing the learning rates of BQL + **Advise** to BQL + Action Biasing, BQL + Control Sharing, and BQL + Reward Shaping for four different combinations of feedback likelihood, $\mathcal{L}$, and consistency, $\mathcal{C}$, across two domains. Each entry represents the average and standard deviation of the cumulative reward in 300 episodes, expressed as the percent of the maximum possible cumulative reward for the domain with respect to the BQL baseline. Negative values indicate performance worse than the baseline. Bold values indicate the best performance for that case.

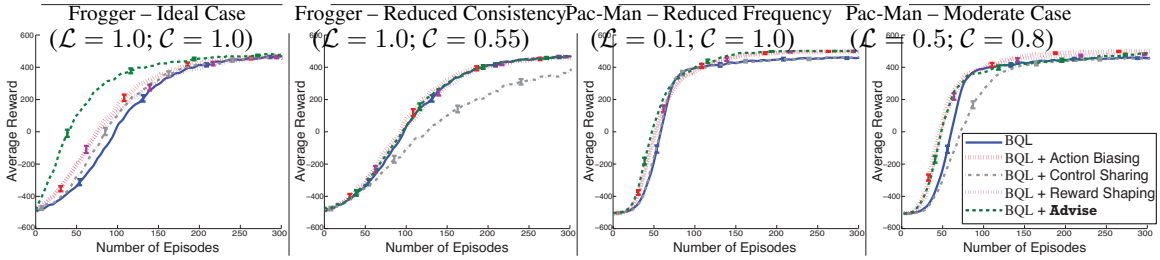


Figure 5.2: Learning curves for each method in four different cases. Each line is the average with standard error bars of 500 separate runs to a duration of 300 episodes. The Bayesian Q-learning baseline (blue) is shown for reference.

BQL exploration parameter $\theta = 0.5$ for Pac-Man and $\theta = 0.0001$ for Frogger. We used a discount factor of $\gamma = 0.99$. Action Biasing, Control Sharing, and Reward Shaping used a feedback influence of $b = 1$ and a decay factor of $d = 0.001$. We set $r_h = 100$ for Action Biasing in both domains. For Reward Shaping we set $r_h = 100$ in Pac-Man and $r_h = 1$ in Frogger [2]

We compared the methods using four different combinations of feedback likelihood, $\mathcal{L}$, and consistency, $\mathcal{C}$, in Pac-Man and Frogger, for a total of eight experiments. Table 5.1 summarizes the quantitative results. Fig. 5.2 shows the learning curve for four cases.

In the ideal case of frequent and correct feedback ($\mathcal{L} = 1.0$; $\mathcal{C} = 1.0$), we see in Fig.

[2] We used the conversion $r_h = 1$, 10, 100, or 1000 that maximized MDP reward in the ideal case to also evaluate the three cases of non-ideal feedback.

5.2 that **Advise** does much better than the other methods early in the learning process. A human reward that does not match both the feedback consistency and the domain may fail to eliminate unnecessary exploration and produce learning rates similar to or worse than the baseline. **Advise** avoided these issues by not converting feedback into a reward.

The remaining three graphs in Fig. 5.2 show one example from each of the non-ideal conditions that we tested: reduced feedback consistency ($\mathcal{L} = 1.0$; $\mathcal{C} = 0.55$), reduced frequency ($\mathcal{L} = 0.1$; $\mathcal{C} = 1.0$), and a case that we call moderate ($\mathcal{L} = 0.5$; $\mathcal{C} = 0.8$). Action Biasing and Reward Shaping[3] performed comparably to **Advise** in two cases. Action Biasing does better than Advise in one case in part because the feedback likelihood is high enough to counter Action Biasing's overly influential feedback policy. This gives the agent an extra push toward the goal without becoming detrimental to learning (e.g., causing loops). In its current form, **Advise** makes no assumptions about the likelihood the human will provide feedback.

The cumulative reward numbers in Table 5.1 show that **Advise** always performed near or above the BQL baseline, which indicates robustness to reduced feedback frequency and consistency. In contrast, Action Biasing, Control Sharing, and Reward Shaping blocked learning progress in several cases with reduced consistency (the most extreme example is seen in column 3 of Table 5.1). Control Sharing performed worse than the baseline in three cases. Action Biasing and Reward Shaping both performed worse than the baseline in one case.

Thus having a prior estimate of the feedback consistency (the value of $\mathcal{C}$) allows **Advise** to balance what it learns from the human appropriately with its own learned policy. We could have provided the known value of $\mathcal{C}$ to the other methods, but doing so would not have helped set r_h, b, or d. These parameters had to be tuned since they only slightly

[3]The results with Reward Shaping are misleading because it can end up in infinite loops when feedback is infrequent or inconsistent with the optimal policy. In frogger we had this problem for $r_h > 1.0$, which forced us to use $r_h = 1.0$. This was not a problem in Pac-Man because the ghost can drive Pac-Man around the map; instead of roaming the map on its own Pac-Man oscillated between adjacent cells until the ghost approached.

correspond to $\mathcal{C}$. We manually selected their values in the ideal case, and then used these same settings for the other cases. However, different values for r_h, b, and d may produce better results in the cases with reduced $\mathcal{L}$ or $\mathcal{C}$. We tested this in our next experiment.

5.5.2 How The Reward Parameter Affects Action Biasing

In contrast to **Advise**, Action Biasing and Control Sharing do not use an explicit model of the feedback consistency. The optimal values to r_h, b, and d for learning with consistent feedback may be the wrong values to use for learning with inconsistent feedback. Here, we test how Action Biasing performed with a range of values for r_h for the case of moderate feedback ($\mathcal{L} = 0.5$ and $\mathcal{C} = 0.8$), and for the case of reduced consistency ($\mathcal{L} = 1.0$ and $\mathcal{C} = 0.55$). Control Sharing was left out of this evaluation because changing r_h did not affect its learning rate. Reward Shaping was left out of this evaluation due to the problems mentioned in Section 5.5.1. The conversion from feedback into reward was set to either $r_h = 500$ or 1000. Using $r_h = 0$ is equivalent to the BQL baseline.

The results in Fig. 5.3 show that a large value for r_h is appropriate for more consistent feedback; a small value for r_h is best for reduced consistency. This is clear in Pac-Man when a reward of $r_h = 1000$ led to better-than-baseline learning performance in the moderate feedback case, but decreased learning rates dramatically below the baseline in the reduced consistency case. A reward of zero produced the best results in the reduced consistency case. Therefore, r_h depends on feedback consistency.

This experiment also shows that the best value for r_h is somewhat robust to a slightly reduced consistency. A value of either $r = 500$ or 1000, in addition to $r = 100$ (see Fig. 5.2.d), can produce good results with moderate feedback in both Pac-Man and Frogger. The use of a human influence parameter $\mathrm{B}[s, a]$ to modulate the value for r_h is presumably meant to help make Action Biasing more robust to reduced consistency. The value for

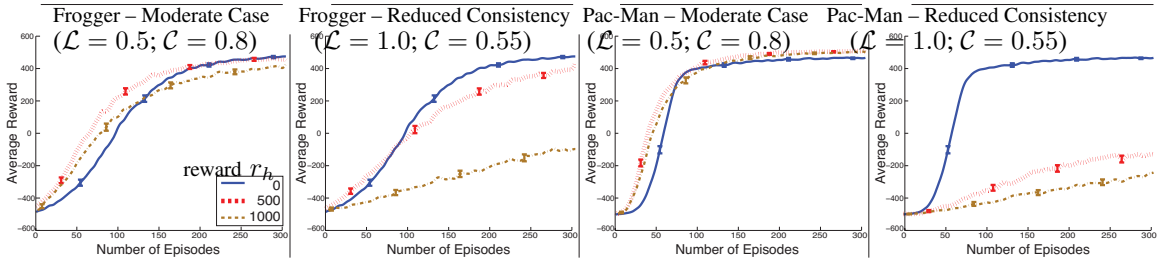


Figure 5.3: How different feedback reward values affected BQL + Action Biasing. Each line shows the average and standard error of 500 learning curves over a duration of 300 episodes. Reward values of $r_h = 0$, 500, and 1000 were used for the experiments. Results were computed for the moderate feedback case ($\mathcal{L} = 0.5; \mathcal{C} = 0.8$) and the reduced consistency case ($\mathcal{L} = 1.0; \mathcal{C} = 0.55$).

$\mathrm{B}[s, a]$ is, however, increased by b whenever feedback is received, and reduced by d over time; b and d are more a function of the domain than the information in accumulated feedback. Our next experiment demonstrates why this is bad for IRL.

5.5.3 How Domain Size Affects Learning

Action Biasing, Control Sharing, and Reward Shaping use a 'human influence' parameter, $\mathrm{B}[s, a]$, that is a function of the domain size more than the amount of information in accumulated feedback. To show this we held constant the parameter values and tested how the algorithms performed in a larger domain. Frogger was increased to a 6×6 grid with four cars (see Fig. 5.4). An oracle was created automatically by running BQL to 50,000 episodes 500 times, and then for each state choosing the action with the highest value. The oracle provided moderate feedback ($\mathcal{L} = 0.5; \mathcal{C} = 0.8$) for the 33360 different states that were identified in this process.

Figure 5.4 shows the results. Whereas **Advise** still has a learning curve above the BQL baseline (as it did in the smaller Frogger domain; see the last column in Table. 5.1), Action Biasing, Control Sharing, and Reward Shaping all had a negligible effect on learning, performing very similar to the BQL baseline. In order for those methods to perform as well as they did with the smaller version of Frogger, the value for $\mathrm{B}[s, a]$ needs to be set higher and decayed more slowly by manually finding new values for b and d. Thus, like r_h, the

optimal values to b and d are dependent on both the domain and the quality of feedback. In contrast, the estimated feedback consistency, $\hat{\mathcal{C}}$, used by **Advise** only depends on the true feedback consistency, $\mathcal{C}$. For comparison, we next show how sensitive **Advise** is to a suboptimal estimate of $\mathcal{C}$.

5.5.4 Using an Inaccurate Estimate of Feedback Consistency

Interactions with a real human will mean that in most cases **Advise** will not have an exact estimate, $\hat{\mathcal{C}}$, of the true feedback consistency, $\mathcal{C}$. It is presumably possible to identify a value for $\hat{\mathcal{C}}$ that is close to the true value. Any deviation from the true value, however, may be detrimental to learning. This experiment shows how an inaccurate estimate of $\mathcal{C}$ affected the learning rate of **Advise**. Feedback was generated with likelihood $\mathcal{L} = 0.5$ and a true consistency of $\mathcal{C} = 0.8$. The estimated consistency was either $\hat{\mathcal{C}} = 1.0$, 0.8, or 0.55.

The results are shown in Fig. 5.5. In both Pac-Man and Frogger using $\hat{\mathcal{C}} = 0.55$ reduced the effectiveness of **Advise**. The learning curves are similar to the baseline BQL learning curves because using an estimate of $\mathcal{C}$ near 0.5 is equivalent to not using feedback at all. In general, values for $\hat{\mathcal{C}}$ below $\mathcal{C}$ decreased the possible gains from feedback. In contrast, using an overestimate of $\mathcal{C}$ boosted learning rates for these particular domains and case of feedback quality. In general, however, overestimating $\mathcal{C}$ can lead to a suboptimal policy especially if feedback is provided very infrequently. Therefore, it is desirable to use $\hat{\mathcal{C}}$ as the closest overestimate of its true value, $\mathcal{C}$, as possible.

5.6 Summary and Discussion

Overall, our experiments indicate that it is useful to interpret feedback as a direct comment on the optimality of an action, without converting it into a reward or an expected action utility. **Advise** was able to outperform tuned versions of Action Biasing, Control Sharing, and Reward Shaping. The performance of Action Biasing and Control Sharing was not as good as **Advise** in many cases (as shown in Table 5.1) because they use feedback as policy

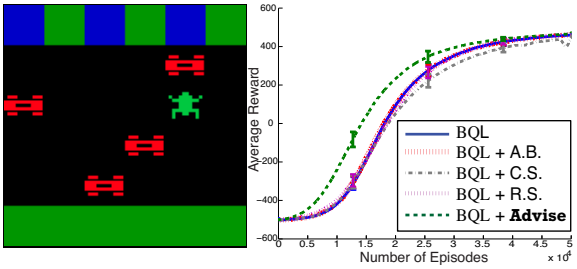


Figure 5.4: The larger Frogger domain and the corresponding learning results for the case of moderate feedback ($\mathcal{L} = 0.5$; $\mathcal{C} = 0.8$). Each line shows the average and standard error of 160 learning curves over a duration of 50,000 episodes.

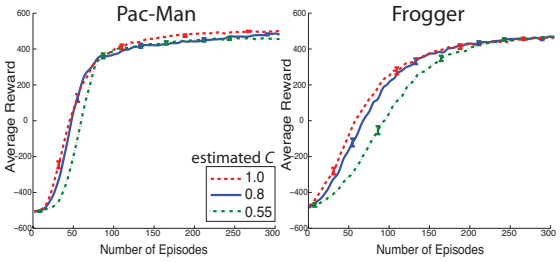


Figure 5.5: The affect of over and under-estimating the true feedback consistency, $\mathcal{C}$, on BQL + **Advise** in the case of moderate feedback ($\mathcal{L} = 0.5$, $\mathcal{C} = 0.8$). A line shows the average and standard error of 500 learning curves over a duration of 300 episodes.

information only after it has been converted into a reward.

Action Biasing, Control Sharing, and Reward Shaping suffer because their use of 'human influence' parameters is disconnected from the amount of information in the accumulated feedback. Although b and d were empirically optimized before the experiments, the optimal values of those parameters are dependent on the convergence time of the underlying RL algorithm. If the size of the domain increased, for example, $\mathrm{B}[s, a]$ would have to be decayed more slowly because the number of episodes required for BQL to converge would increase. Otherwise Action Biasing, Control Sharing, and Reward Shaping would have a negligible affect on learning. Control Sharing is especially sensitive to how well the value of the feedback influence parameter, $\mathrm{B}[s, a]$, approximates the amount of information in both policies. Its performance bottomed out in some cases with infrequent and inconsistent feedback because $\mathrm{B}[s, a]$ overestimated the amount of information in the feedback policy. However, even if $\mathrm{B}[s, a]$ is set in proportion to the exact probability of the correctness of each policy (i.e., calculated using **Advise**), Control Sharing does not allow an agent to simultaneously utilize information from both sources.

Advise has only one input parameter, the estimated feedback consistency, $\hat{\mathcal{C}}$, in contrast to three. $\hat{\mathcal{C}}$ is a fundamental parameter that depends only on the true feedback consistency, $\mathcal{C}$, and does not change if the domain size is increased. When it has the right value for $\hat{\mathcal{C}}$, **Advise** represents the exact amount of information in the accumulated feedback in each

state, and then combines it with the BQL policy using an amount of influence equivalent to the amount of information in each policy. These advantages help make **Advise** robust to infrequent and inconsistent feedback, and fair well with an inaccurate estimate of C.

A primary direction for future work is to investigate how to estimate $\hat{C}$ during learning. That is, a static model of C may be insufficient for learning from real humans. An alternative approach is to compute $\hat{C}$ online as a human interacts with an agent. We are also interested in addressing other aspects of human feedback like errors in credit assignment. A good place to start is the approach described in [83] which is based on using gamma distributions. Another direction is to investigate **Advise** for knowledge transfer in a sequence of reinforcement learning tasks (*cf.* [105]). With these extensions, **Advise** may be especially suitable for learning from humans in real-world settings.

This work defined the Policy Shaping paradigm for integrating feedback with Reinforcement Learning. We introduced **Advise**, which tries to maximize the utility of feedback using a Bayesian approach to learning. **Advise** produced results on par with or better than the current state of the art Interactive Reinforcement Learning techniques, showed where those approaches fail while **Advise** is unaffected, and it demonstrated robustness to infrequent and inconsistent feedback. With these advancements, it may help to make learning from human feedback an increasingly viable option for intelligent systems.

CHAPTER 6

EXPLORATION IN MONTE CARLO TREE SEARCH USING ACTION ABSTRACTIONS

Monte Carlo Tree Search (MCTS) is a family of methods for planning in large domains. It focuses on finding a good action for a particular state, making its complexity independent of the size of the state space. However such methods are exponential with respect to the branching factor. Effective application of MCTS requires good heuristics to arbitrate action selection during learning. In this work we present a policy-guided approach that utilizes action abstractions, derived from human input, with MCTS to facilitate efficient exploration. We draw from existing work in hierarchical reinforcement learning, interactive machine learning and show how multi-step actions, represented as stochastic policies, can serve as good action selection heuristics. We demonstrate the efficacy of our approach in the PacMan domain and highlight its advantages over traditional MCTS.

6.1 Introduction

Monte Carlo Tree Search (MCTS) [67] algorithms have been used to address problems with large state spaces. They focus on solving the policy for a single state—the state the agent is in—making the planning time independent of the total number of the states. MCTS covers a family of algorithms including Sparse Sampling [25] and its successors, UCT [65] and FSSS [66]. It has grown rapidly in visibility due to its early successes in the boardgame Go and by winning AAAI's General Game Playing competitions [106, 107]. More recently, with the growth of deep learning research [68], MCTS methods have been combined with function approximation using deep learning to achieve state-of-the-art performance in Atari games [69] and Go [8]. These results have highlighted the use of MCTS for planning in large domains.

The successes however have their share of costs. Tree search methods are, in general, exponential in their depth, with a branching factor that depends on the number of possible actions and subsequent states at each node. Thus to make MCTS effective requires the use of heuristics that help action selection during tree search and roll-out execution. Existing methods (UCT, FSSS) utilize confidence bounds on the value function[14], by tracking state-action pair visitations, to decide which actions to explore and exploit. These methods are sample intensive and pay a substantial computational cost for every step of action selection.

In parallel, researchers have focused on how to leverage human help to improve learning and planning, including work in learning by demonstration [71], imitation learning [108], and interactive machine learning [109]. The motivation for these works stems from the observation that (1) human help is often available, and (2) humans excel at some important tasks that automated methods have difficulty with. Application of human input has yielded promising results such as helicopter flying [110], teaching a AIBO robot basic soccer skills [111] and played an important role in the success of AlphaGo [50]. Of particular importance in this work is the ability of humans to help autonomous agents explore promising parts of the state space [112] and the use of human input to construct action abstractions that can decompose complex problems in simpler subparts [63].

In this chapter, we show how we can leverage recent work in utilizing action abstractions for reinforcement learning [113] to help satisfy the requirements of MCTS without incurring the expensive computational costs. Action abstractions like Options [51] and Constraints [52] represent multi-step policies that can significantly speed up planning in reinforcement learning. This form of knowledge allows the agent to lookahead over multiple timesteps, obtain better estimates of the utility of an action and propagate this information to multiple states. We note the use of constraints as actions While options and constraints are similar at a high-level, these abstractions differ in their respective definitions and tackle different aspects of the problem. Options represent abstractions that capture

goals to achieve in a task while constraints capture *situations to avoid*. These ideas have been successfully combined and utilized in Q-learning [55] to solve gridworld domains. To our knowledge, this is the first method that utilizes constraints as actions abstractions for MCTS. In this work we characterize specific properties of these action abstractions and show how they can used as action pruning heuristics and high quality roll-out policies for MCTS to solve large problems. In particular, we show that:

- Options offer coherent, near-optimal action sequences for solving sub-tasks. They allow us to increase the effective search depth of MCTS methods.
- Constraints complement options by identifying actions not to follow. They can act as both a form of pruning and a way to encapsulate an intelligent roll-out policy.

We leverage these properties to develop a novel approach, Policy-Guided Sparse Sampling (PGSS), that can effectively use such abstractions to overcome some of its limitations and plan efficiently. Using the PacMan domain, we show how PGSS satisfies the requirements for efficient exploration in MCTS [114].

6.2 Approach

Here we discuss properties of Monte Carlo Tree Search (MCTS) for action-value estimation, and our method of improving it with auxiliary information in the form of action abstractions.

6.2.1 Policy-Guided Sparse Sampling

As discussed in Section 6.1, the key property in determining the efficiency of MCTS is the implicit tree-search policy of the algorithm. Non-interactive approaches to designing this search bias are value-based, and require the agent to visit states multiple times in order to compute the relevant statistics for directing future search. This requirement can be

intractable for a number of reasons, including a high branching factor, strict realtime deadlines, a γ close to 1, or a transition model that is expensive to query (*e.g.,* requires running a physics simulation). This motivates the main idea behind Policy-Guided Sparse Sampling (PGSS): to construct the search policy explicitly by using a combination of different action abstractions with desirable properties.

Action Abstractions

We noted in Section 6.1 that MCTS presents two points for biasing action selection: 1) during tree search and 2) during roll-out. This suggests the use of two different and complementary policy classes: options and constraints. The use of these policies in MCTS is highlighted in Figure 6.1.

Options. The first policy class will serve to augment the set of primitive actions, allowing deeper look-ahead in the tree. Following [51], an option is a sub-policy with clearly defined initiation and termination conditions, and is generally used to encapsulate sub-tasks in a planning problem. Options allow the planner to make large jumps in the state space: assuming the options' policies are locally optimal for their subtask, searching at the level of options increases the effective branching depth of the planner by a factor of d_o, where d_o is the expected length of the option.

Constraints. The second type of abstraction [52] encodes a bias to *disallow* certain actions, and has two modes of operation: (1) as a action-pruning heuristic during tree expansion, and (2) as a roll-out policy for obtaining value estimates for the leaf nodes. In an uninformed implementation of MCTS, the roll-out policy is a random policy, significantly underestimating the actual value of leaf nodes. By comparison, a policy that avoids terminal states where one cannot escape negative reward will generally provide a better estimate of the value of the leaf nodes. We refer to a policy designed to achieve this survivor effect as a *constraint*, to indicate that it restricts the agent from executing actions that result in

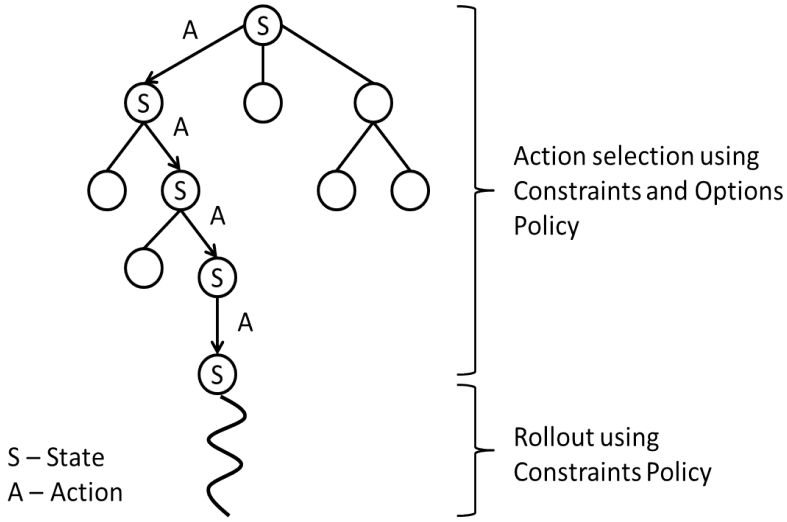


Figure 6.1: Monte Carlo Tree Search highlighting where we use both constraints and options for effective exploration.

terminal states. A constraint is represented by a policy that satisfies its conditions, along with an initiation set that indicates when the policy of the constraint should be taken into account. We find the constraint policy to be useful not only for biasing action selection during the tree search, but also as a self-contained roll-out policy. As we discuss in the next section, this provides a soft form of tree pruning to remove branches unlikely to lead to high value states.

Policies as Heuristics

When considered within MCTS, options and constraints provide ideal heuristics to help bias tree search, allowing for deeper or more accurate value estimation. We first show how to incorporate constraint policies. Because a constraint explicitly represents the permissible actions for all states, it can be used for pruning at each node. In order to prevent constraints from filtering out optimal actions, and thereby removing the theoretical guarantees of MCTS, we apply the softmax operation using an auxiliary β parameter to define a

Algorithm 4 Policy-Guided Sparse Sampling

PGSS(s, d, O)
if $d = H$ **then** return 0
end if
if $O = \emptyset$ **OR** $T_O(s) > rand$ **then**
    % Sample an available option
    $O \sim \{O : I_O(s) = true\}$
end if
if $O \neq \emptyset$ **AND** $d < d_{max}$ **then**
    % Sample from constrained option
    $a \sim P(a|s) \propto P_{\pi_O}(a|s) \times \frac{P_{\pi_c}(a|s)^\beta}{\sum_{a \in A} P_{\pi_c}(a|s)^\beta}$
else
    % Sample directly from constraint
    $a \sim \frac{P_{\pi_c}(a|s)^\beta}{\sum_{a \in A} P_{\pi_c}(a|s)^\beta}$
end if
$s' \sim P(s'|s, a)$
$Q_{ss}(s, a) = R(s, a) + \gamma PGSS(s', d + 1, O)$
return $\max_{a \in A} Q_{ss}(s, a)$

probability distribution over actions for each state:

$$P(a|s) = \frac{P_{\pi_c}(a|s)^\beta}{\sum_{a \in A} P_{\pi_c}(a|s)^\beta} \tag{6.1}$$

π_c represents the constraint policy and β controls how peaked the distribution is over the preferred action, controlling how much to "trust" the constraint. Note that constraints are represented as typical policies, but encode a preference for "safe" actions, with entropy proportional to β. By incorporating the soft-maxed constraint, we can achieve an arbitrarily safe union of policies.

For options, we first review the basic theory of offline planning with options (*e.g.*, value iteration) [51]. An option is defined as a tuple $< I, T, \pi >$ representing the set of states I_o where the option can be initiated, a distribution T_o over states for terminating the option, and the option policy π_o itself. Traditional approaches are based on Bellman-updates over primitive actions, so planning with options requires an expected reward and terminal state

for each option.

$$E[R|\pi_o(s)], E[s'|\pi_o(s)] \tag{6.2}$$

We can extend MCTS to incorporate options by adding them as additional actions to all states in their respective initiation sets I_O, and terminate them during each step according to their respective termination probabilities $T_O(s)$. In this way, MCTS performs the option evaluation. With the expected length of the option counting towards the total depth reached by the agent, options are serving to bias search towards specific trajectories that we have *a priori* reason to believe are useful.[1]

Here we emphasize the need for options and constrains to be handled differently. In our formulation, an option always suggests an action to take while a constraint rarely prefers an action to take unless the agent is about to enter a dire circumstance. More specifically, the use of constraints at the leaves of the tree keeps the agent "alive" by avoiding low expected utility and out-performs options at that task. Similarly options drive one towards goals and out-perform constraints at those tasks. This characteristic makes them qualitatively different and therefore should be managed differently in order to exploit their unique properties.

Algorithm 4 is our approach to Policy-Guided Sparse Sampling. The algorithm recursively constructs a search tree to branching depth d_{max}, and performs constraint policy roll-outs to the horizon H. π_c is the constraint policy, π_O is an option policy, $I_O(s)$ returns true if option O can be initiated from state s, and $T_O(s)$ is the probability of terminating option O in state s. Our implementation branches over primitive actions only when there are no valid options for the current state. This was a reasonable restriction for our experiments, since the options fully covered the set of appropriate actions for all time-steps. However, in general we would typically branch over primitives as well.

[1]The availability of the constraint puts a minor modification on the option's roll-out: since the constraint can preempt the option, we're actually taking samples of a hybrid option+constraint policy for each option

Combining Multiple Constraints

In domains where multiple constraints are required to be satisfied, they can be combined in a straightforward manner. For any given state s we create a list of the constraints that are activated there and then generate a set by taking a union of all the actions the constraints suggest to take. We then reweigh the probabilities of this action set according to the outcome of disobeying each individual actions suggested by the respective constraints. We now have a stochastic distribution over actions that takes into account information of multiple constraints. We draw from it and proceed down the tree to the next node. More details are available in [52].

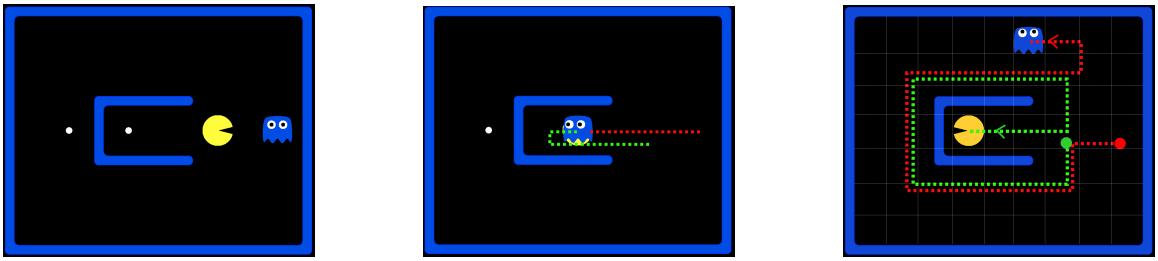
6.3 Experiments

In this section we present empirical evaluation of our approach by instantiating it on the PacMan[2] domain. PacMan naturally lends itself to be abstracted by hierarchical decompositions and is a domain which poses difficulties for tree search methods due to its long horizon. For example, in our experiments, a 25x25 grid with four ghosts and four power pellets has a total of over 10^{15} states with an effective depth of 340 steps. We implemented the necessary abstractions using human interaction.

6.3.1 Information From Humans

In Tokadli and Feigh [2015], the authors describe useful action abstractions for the PacMan domain and motivate how humans naturally provide this information when interacting with the domain. Using this work as motivation, we leverage existing interactive learning methods to learn options [61] and constraints [52]. These approaches use human input in the form of demonstrations to efficiently learn probabilistic policies that define the necessary heuristics.

[2]The version of PacMan we used is an open-source implementation available online at http://www-inst.eecs.berkeley.edu/ cs188/pacman/pacman.html



(a) Start (R = 0)

(b) Without Constraint (R = -495)

(c) With Constraint (R = 497)

Figure 6.2: Starting map configurations for the dead-end problem (left), terminal state for flat MCTS agent (middle), and optimal solution discovered by PGSS (right). Total reward shown in parentheses

In our tests, we learn the heuristics from human interaction and refer to existing work [53] to confirm their utility for learning to solve PacMan. As a result of this, we learned the options *eatFood* and *eatCapsule*, and the constraint *avoidGhost* (avoids the nearest one).

We first describe a simple experiment that illustrates the advantages of using a policy biased approach in MCTS and then show how our approach scales to problems of increased horizon depths.

6.3.2 The Dead-End Experiment

The dead-end experiment is a simple problem designed to provide intuition about the utility of constraint policies in the context of Monte Carlo search. By explicitly asking the question "is this leaf node a state that I can survive in?", the constraint gives the agent a significant advantage in look-ahead. In particular, a constraint policy provides a *more optimistic lower bound* than a random policy for the values of leaf nodes in the search tree. We used a small PacMan grid shown in Figure 6.2a with an effective horizon depth of 18 steps. The ghosts move directionally towards the agent.

As Figures 6.2a-6.2c show, a flat MCTS agent sees the nearest food and goes for it, not realizing that it's a dead-end. By doing an *avoidGhost* roll-out from this state, the constraint agent discovers that it is eventually terminal, backs up that reward to the start state, and chooses to go around instead. When using a random rollout policy, the agent is

unlikely to escape the ghost regardless of whether Pacman is trapped. Therefore this agent is less capable of discriminating between the trap and the open space, and is more likely to make the wrong choice.

We note that the inclusion of options as actions that the agent can branch over is a significant advantage as it enables deeper lookahead during rollouts. Overall the PGSS agent can rollout the *eatFood* option policy to obtain reward from the food pellet and at the same time use the constraint to avoid the ghost. This combination allows PGSS to perform optimally using very small search depths.

6.3.3 Scalability

In this experiment we investigate 1) how action abstractions compare to each other and 2) their performance on problems of increasing horizons. We achieve this by implementing several policy-based variants of the PGSS agent in PacMan domains of different sizes. We note that by increasing the size, the effective horizon increases making it significantly harder for MCTS algorithms. We use four variants of PGSS agents. The original sparse-sampling algorithm which branches only over primitive actions, as well as three policy-guided variants: using only options, using only constraints, and using both. We also compare the performance of these agents with that of an average human player. We show the results of this experiment in Figure 6.3a. The average rewards were computed over 5 trials. We limited search depth to 34 steps, after which we evaluated the constraint as a rollout policy 3 times. Inside the constraint the Boltzmann temperature value was 10. In these experiments the ghost directions were random. The agents' decisions were made in real-time.

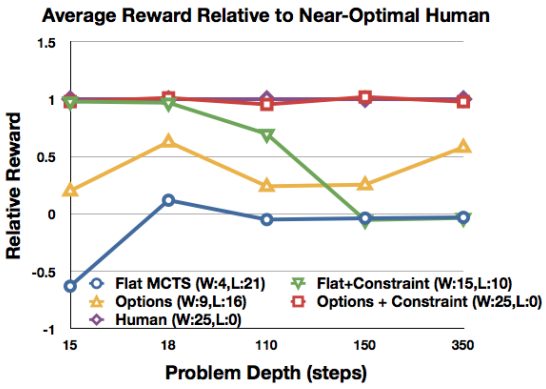
Unsurprisingly, the flat agent proved to be the worst in terms of reward, outcome (win/lose), and runtime. While a small look-ahead is sufficient to win for tiny domains, we found that larger maps required a tree depth that was prohibitively expensive to compute (due to the physics engine). Adding the options extended the effective look-ahead,

and significantly increased the average reward per episode on larger maps; however, options also frequently led to bad terminal states, and this agent eventually died in 4 out of 5 trials on the largest map.

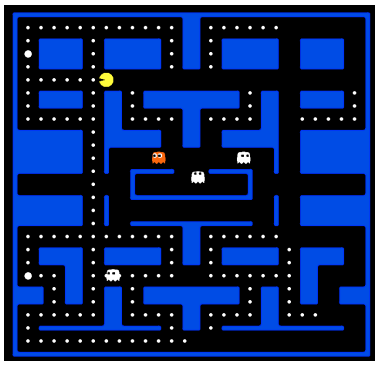
Replacing the options with the constraint meant the agent was less likely to die prematurely, but sacrificed the look-ahead depth of options. While this agent performed well in smaller maps, it frequently became disconnected from regions of reward (food) in larger maps, and wandered randomly until trapped or chased away by a ghost. As shown before, by reflecting whether the agent can stay alive from the leaf nodes of tree, the constraint is essentially a dead-end detection mechanism. Using only the constraint, we observed that the agent ate all the food in a neighborhood and then couldn't "see" outside the sample horizon of the constraint and so wandered randomly. Eventually a ghost would either chase him towards a good region or, especially in the big maps, a dead-end.

Fortunately, the strengths and weaknesses of our constraint and options agents are complementary: the options roll-outs find deep action trajectories that are likely to be good, and the constraints help ensure that they do not lead to undesirable states. We found the options+constraints agent to be the superior policy across all problem sizes in terms of speed, total reward, and final outcome. Taking a closer look at these episodes, it seemed that the primary motif this agent excelled at, as compared to the others, was eating ghosts. Ghosts can only be eaten for narrow windows after eating a power pellet, and it typically requires a long and specific sequence of actions to achieve this result. The probability of an uninformed search discovering this full trajectory by chance was too low to observe for ghosts more than a couple steps away from PacMan. In addition to achieving the best reward, the options+constraint agent produced the only policy that could reliably beat the largest map with a effective horizon depth of 350 steps. (Figure 6.3a).

We have also tested the PGSS algorithm on other related domains (for example Cat and Mouse) that lend themselves to action abstractions and were able to achieve similar results.



(a) Relative Reward



(b) Episode on Largest Map

Figure 6.3: Average reward obtained per trial versus map size for different configurations (left), and a sample run on the largest map (right). The numbers in brackets indicate the win/loss ratio.

6.4 Summary and Discussion

Our experiments yield insights about the use of human-derived action abstractions in MCTS and we highlight them here. An interesting observation is that when sampling the constraint policy, it is possible for us to reach the goal state. In these cases, the computed value for constraint evaluation will be more informative as it includes information about the reward at the goal state. The effect of using such constraints is that it allows us to learn a good policy with a smaller tree depth. We note that this might not be true in all scenarios; however when constructing constraints for a domain, we believe that knowledge of constraints potentially reaching the goal can be utilized to perform more efficient planning.

We view the applicability of PGSS as a way of addressing the class of MDPs in which not only is it intractable to compute a policy for the entire state space, but even for a single state. In Section 6.2.1 we explained that modern MCTS algorithms like UCT and FSSS assume the agent can afford to explore certain parts of the space quite extensively. In fact, FSSS only terminates after closing all nodes in its search tree, which requires visiting every possible state-action transition out to the problem horizon H. This implies that the time required by FSSS to return an action for the current state is *exponential* in the problem depth. Clearly there are many MDPs in which this is infeasible, such as in

our PacMan results from Figure 6.3a. These were obtained in *real-time*, which was only possible by shifting to policy-based heuristic that relaxed the need to explore the search-tree exhaustively.

In our tests on instantiating action abstractions, we find that interactive learning approaches provide abstractions more suited for PGSS than autonomous learning methods. We also note that incorporating action abstractions in MCTS as in PGSS provides a general framework that is applicable to other variants of MCTS as well (UCT, FSSS). These methods would only stand to gain performance speed-ups from the use of domain heuristics in the form of temporally extended actions.

In this work we have described the compatibility between action abstractions learned from humans and the requirements of MCTS. We presented a unifying framework that combines two different kinds of action abstractions and used them as pruning heuristics and intelligent roll-out policies in MCTS. Our experiments in the PacMan domain show that the PGSS algorithm can be used to solve problems of non-trivial horizon depths and thus have a dramatic effect on the performance of the planner. PGSS can also be applied to other domains, ones that can benefit from action abstractions, in a straightforward manner. We would like to highlight that our approach can be viewed as an addendum to existing tree search algorithms, i.e. integrating them with action abstractions in a specific manner and showing its advantages. Extending it to other state-of-the-art techniques in MCTS literature like UCT is a promising area of future work. We are also interested in exploring other kinds of action abstractions that PGSS can utilize.

CHAPTER 7
CONCLUSION AND FUTURE WORK

In this chapter we summarize the work presented in the dissertation along with the proposed hypotheses and goals. We cover what was achieved in each contribution and highlight key findings relevant to the thesis.

7.1 Overview

In this dissertation we focused on the topic of exploration for reinforcement learning. The exploration-exploitation trade off is a central challenge in RL domains and there are several methods in the literature that serve to tackle this under different conditions and using different heuristics. We proposed several policy-based approaches where the primary goal is to learn an adaptive exploration policy separate from the optimal control policy to guide exploration autonomously as well as with the help of human input.

We hypothesized that an approach focused on learning to explore as policies directly helps overcome some of the computational challenges involved in existing approaches. Additionally we hypothesize that such an approach when used in the interactive setting with information from people can be robust to noisy information from humans.

We presented policy-based methods that serve to

1. bias an RL agent's exploration to cover the search space of the domain efficiently
2. balance the exploration-exploitation trade-off for an RL agent learning from human signals

We restate the thesis of the dissertation: **policy-guided exploration for reinforcement learning agents leads to faster convergence to the optimal policy than automatic value-**

based and state-of-the-art learning from demonstration methods and is robust to noisy human signals

7.2 Summary of What Was Achieved

We researched several approaches to learn an exploration policy that is useful from the perspective of the agent's learning algorithm, using statistical measures of regression methods. We use this to design an interactive and autonomous learning approach and were able to show its benefits in variety of domains with long horizons and sparse rewards. In one of the methods, we were also able to relax the requirement of optimal human input.

We presented a Bayes' optimal approach of combining human binary input with Bayesian reinforcement learning and how the combined approach is robust to noisy human signals. Finally we presented a method that uses human demonstrations as action abstractions to improve exploration for Monte Carlo tree search methods with informed constraints on the action selection.

7.3 Main Contributions

Here we highlight the individual research projects that support the claims and contributions made in this dissertation.

7.3.1 Agent-guided Exploration from Human Demonstration

In this work we use statistical measures of regression methods, leverage and discrepancy, as metrics useful for exploration. Leverage specifies if an observation is an outlier or if it is within the convex hull of the observations already experienced. Discrepancy allows us to measure how much the trained model error depends on the datapoint. Together these measures inform the agent how influential each datapoint is towards the model being learned.

Using this approach we capture areas of uncertainty directly from the agent's perspective without the need to design domain specific heuristics. When such a datapoint (state-

action pair) is flagged as influential, we propose that to achieve good exploration, we should encourage the agent to visit the datapoint to collect more information thereby reducing its leverage and its discrepancy. To facilitate this, we use human input to guide the agent towards these areas.

Human input in this case is not required to examples of optimal behavior. Instead the human is only required to be aware of the dynamics of the domain and the knowledge of how to use the actions available to the agent to guide it to the influential datapoint.

We learn an exploration policy using these demonstrations which in turn encourage visitation to the important datapoints. Note that once the datapoint has been visited enough number of times, two outcomes are likely: a) its influence reduces and b) the agent has a better chance of further extending the boundaries of known datapoints.

We implemented our approach on an instructional gridworld to highlight the utility of the statistical measures and on Frogger to show that learned exploration policies lead to faster convergence to the solution than learning from optimal demonstration and model-free exploration strategies. We were able to show the effectiveness of this approach on variants of the Frogger domain which lead to a high-dimensional, long goal horizons and sparse reward domain.

7.3.2 Autonomous Agent-guided Exploration

We build on the work on learning exploration policies using human demonstrations to show how we can learn the exploration policies autonomously. The agent, guided by the same statistical measures, solves for the exploration policy as a separate MDP to help the agent explore.

When an observation is recognized as influential, an auxiliary learning problem is started where the policy to be learned is to reach influential datapoint. Once the policy is learned, additional samples are gather to reduce the influence of the datapoint and help the agent visit other parts of the search space.

This approach removes the requirement of human input which in many domains is unavailable or even infeasible, i.e. where the human is unable to provide actions to help the agent explore due to abstract action space of the agent (say robot joint angles). The rate is convergence is slower than with human input, however we show that is able to converge to the optimal policy efficiently with improved sample complexity over existing value-based approaches.

More recently, deep reinforcement learning has had a large body of success and we show how this work can be integrated into existing off-policy methods to facilitate exploration without significant changes. We highlighted the performance of the agent in classical control problems, a high-dimensional game domain and a popular Atari game with image input.

7.3.3 Policy Shaping with Humans

The previous work learns an exploration policy autonomously or from human input. However it does not directly tackle the problem of balancing the trade off between exploration and exploitation in a principled manner. In this work, we take a closer look at approaches in this direction.

We present a probabilistic approach to combining human signals with a reinforcement learning model. We model human feedback as a policy signal with an estimate of the quality and quantity of the input for the behavior to be learned. This input along with Bayesian RL presents a Bayes' optimal approach to combining information from these sources without the need for any heuristics.

Frequent high quality human inputs reinforce the suggested behavior while noisy human data shifts the balance towards the Bayesian rl algorithm's prediction. We implement the methods on popular game domains PacMan and Frogger and when compared with state-of-the-art interactive learning methods, were able to show the approach is robust to noisy human input and less sensitive to parameter selections.

7.3.4 Exploration in Monte Carlo Tree Search using Action Abstractions

The previous work used human input as demonstrations for model-free off-policy reinforcement learning algorithms. In this work, we provide an alternate interpretation to exploratory human demonstrations.

We show how human demonstrations, when instantiated as temporal action abstractions, can be used to overcome the difficulties of Monte Carlo reinforcement learning methods. Monte Carlo methods rely on the quality of the metric used to guide action selection during the expansion step and the information from rollout policies to evaluate the long-term utility of taking an action in a state.

In this work we show how human can provide action abstractions in the form of options and constraints. These instantiations can then be used in MCTS to bias action selection during node expansion (with options) and as an informative rollout policy (with constraints) to solve the domain more efficiently than withouts.

To test this approach, we experimented with scaled up version of PacMan which requires deeper search in order to perform well and were able to show our approach outperforms MCTS methods. We note that the addition of human input here formulates domain knowledge and provides the basis for the improvement in performance. With this observation, the goal of the work is to motivate the use of action abstractions in this manner as domain knowledge in MCTS methods.

7.4 Limitations and Future Work

With the main contributions stated, we now look at the limitations of the approaches and highlight ways in which they can be address in future work.

Human Input The work on EfD and Policy Shaping require a human to provide inputs to the algorithms to facilitate the learning of an exploration or the optimal policy. As mentioned, this is often not feasible due to a number of reasons. Primary among them is the

unavailability of a human to provide input. Even in the case where a human is available, the human might not be familiar with the domain and its dynamics to provide inputs useful to the underlying algorithms. Further in domains where the agent has access to a knowledgeable human, the mode of interaction with the human might not be conducive for critique or demonstrations. For example if the agent is a robot operating in joint space. It could be hard for a human to provide a demonstration of what the robot should by controlling the robot in joint space.

We tackle some of these questions using the autonomous policy exploration method. However we believe there is interesting work in this space related to designing user interfaces to acquire the necessary human information across a variety of domains. Secondly there is an exciting area of research focusing on learning action abstractions automatically and from humans which could more directly assist Monte Carlo methods.

Bayesian RL In our work we were able to combine binary human critique with Bayesian RL and showed its advantages. The limitation here is that Bayesian RL methods do not scale well as the problems become more complex. As such the applicability of this approach is currently limited in this setup.

A promising direction to take this work is to relax the strict requirement of using Bayesian RL and instead use other probabilistic approaches which a principle manner to measure uncertainty. One such approach is using Gaussian Processes (GPs). Combining human input with GPs will allow us to extend the applications of Policy Shaping beyond the domains shown in this dissertation.

Continuous Control The methods described in this dissertation learn exploration policies using statistical measures setup for continuous state and discrete action problems. This is currently a limitation as it would not be applicable to be used with actor-critic methods that learn policies for continuous control. Applications involving robotics primary would be outside the scope of the algorithm.

To further extend the autonomous approach, as future work we would have to modify influence measures to take into account the action followed by learning the exploration policy to reach similar datapoints to gain more information about them. This is a straightforward extension mathematically, however the primary cost is in the computational resources required to keep track of influential state and action pairs which in continuous space are harder to visit often. This would require further empirical research.

7.5 Final Remarks

In this dissertation we showed the advantages and limitations of learning and utilizing an exploration policy autonomously and from human input in a variety of problems with varying characteristics. We were able to show our approaches lead to faster convergence to the optimal policy than value-based methods and are robust to noisy human input. The work presented takes a step towards realizing solutions to reinforcement learning on complex problems in a sample efficient manner. Furthermore this line of work raises several questions that we believe will serve to make the methods usable across a wider class of problems while maintaining its desirable properties.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[4] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[7] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[8] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, *Mastering atari, go, chess and shogi by planning with a learned model*, 2019. arXiv: `1911.08265 [cs.LG]`.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[10] K. Arulkumaran, A. Cully, and J. Togelius, "Alphastar," *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO 19*, 2019.

[11] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for robocup soccer keepaway," *Adaptive Behaviour*, vol. 13, no. 3, pp. 165–188, 2005.

[12] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[13] I. Giannoccaro and P. Pontrandolfo, "Inventory management in supply chains: A reinforcement learning approach," *International Journal of Production Economics*, vol. 78, no. 2, pp. 153–161, 2002.

[14] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, pp. 397–422, Mar. 2003.

[15] A. L. Strehl and M. L. Littman, "An analysis of model-based interval estimation for markov decision processes.," *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1309–1331, May 5, 2009.

[16] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in neural information processing systems*, 2016, pp. 1471–1479.

[17] I. Osband and B. Van Roy, "Bootstrapped thompson sampling and deep exploration," *arXiv preprint arXiv:1507.00300*, 2015.

[18] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[19] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2005, pp. 1281–1288.

[20] S. Mohamed and D. J. Rezende, "Variational information maximisation for intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2015, pp. 2125–2133.

[21] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer, "Exploration in model-based reinforcement learning by empirically estimating learning progress," in *Advances in neural information processing systems*, 2012, pp. 206–214.

[22] T. Hester, M. Lopes, and P. Stone, "Learning exploration strategies in model-based reinforcement learning," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013, pp. 1069–1076.

[23] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis (4th ed.)* Hoboken: Wiley & Sons, Jul. 2006, ISBN: 0471754951.

[24] C. Szepesvári, *Algorithms for reinforcement learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

[25] M. Kearns, Y. Mansour, and A. Ng, "A sparse sampling algorithm for near-optimal planning in large markov decision processes," *Machine Learning*, vol. 49, no. 2, pp. 193–208, 2002.

[26] M. T. Ronen I. Brafman, "R-max - a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, pp. 213–231, 2002.

[27] S. Bubeck, R. Munos, and G. Stoltz, "Pure exploration in multi-armed bandits problems.," in *ALT*, ser. Lecture Notes in Computer Science, vol. 5809, Springer, Sep. 30, 2009, pp. 23–37, ISBN: 978-3-642-04413-7.

[28] R. Dearden, N. Friedman, and S. Russell, "Bayesian Q-learning," in *Proc. of the 15th AAAI*, 1998, pp. 761–768.

[29] J. Pazis and R. Parr, "PAC optimal exploration in continuous space markov decision processes," in *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[30] T. Hester, M. Lopes, and P. Stone, "Learning exploration strategies in model-based reinforcement learning," in *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '13, 2013, pp. 1069–1076, ISBN: 978-1-4503-1993-5.

[31] T. Hester and P. Stone, "TEXPLORE: Real-time sample-efficient reinforcement learning for robots," *Machine Learning*, vol. 90, no. 3, 2013.

[32] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, 2013, pp. 1037–1044.

[33] O. Şimşek and A. G. Barto, "An intrinsic reward mechanism for efficient exploration," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06, Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 833–840, ISBN: 1-59593-383-2.

[34] A. Epshteyn, A. Vogel, and G. DeJong, "Active reinforcement learning.," in *ICML*, ser. ACM International Conference Proceeding Series, vol. 307, ACM, Aug. 14, 2008, pp. 296–303, ISBN: 978-1-60558-205-4.

[35] T. Akiyama, H. Hachiya, and M. Sugiyama, "Efficient exploration through active learning for value function approximation in reinforcement learning," *Neural Networks*, vol. 23, no. 5, pp. 639 –648, 2010.

[36] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82, 2010.

[37] T. Hester and P. Stone, "Intrinsically motivated model learning for developing curious robots," *Artificial Intelligence*, vol. 247, pp. 170–186, 2017.

[38] P.-Y. Oudeyer, J. Gottlieb, and M. Lopes, "Intrinsic motivation, curiosity, and learning: Theory and applications in educational technologies," in *Progress in brain research*, vol. 229, Elsevier, 2016, pp. 257–284.

[39] Ö. Şimşek and A. G. Barto, "An intrinsic reward mechanism for efficient exploration," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 833–840.

[40] A. G. Barto, "Intrinsic motivation and reinforcement learning," in *Intrinsically motivated learning in natural and artificial systems*, Springer, 2013, pp. 17–47.

[41] N. Bougie and R. Ichise, "Skill-based curiosity for intrinsically motivated reinforcement learning," *Machine Learning*, vol. 109, no. 3, pp. 493–512, 2020.

[42] A. Aubret, L. Matignon, and S. Hassas, "A survey on intrinsic motivation in reinforcement learning," *arXiv preprint arXiv:1908.06976*, 2019.

[43] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[44] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[46] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[48] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *Advances in neural information processing systems*, 2016, pp. 4026–4034.

[49] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 16–17.

[50] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[51] R. S. Sutton, D. Precup, and S. P. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1-2, pp. 181–211, 1999.

[52] A. J. Irani, "Utilizing negative policy information to accelerate reinforcement learning," PhD thesis, Georgia Institute of Technology, 2015.

[53] G. Tokadlı and K. M. Feigh, "Application of abstraction hierarchies to incorporate human knowledge for machine learning a general form for mario bros. & pac-man," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, SAGE Publications, vol. 59, 2015, pp. 657–661.

[54] D. Precup, "Temporal abstraction in reinforcement learning," PhD thesis, UMass Amherst, 2000.

[55] J. Rosalia, G. Tokadli, C. L. Isbell Jr, A. L. Thomaz, and K. M. Feigh, "Discovery, evaluation, and exploration of human supplied options and constraints," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 1873–1874.

[56] K. Rohanimanesh and S. Mahadevan, "Learning to take concurrent actions," in *NIPS*, 2002, pp. 1619–1626.

[57] ——, "Decision-theoretic planning with concurrent temporally extended actions," in *UAI*, 2001, pp. 472–479.

[58] Z. Luo, D. A. Bell, and B. McCollum, "Skill combination for reinforcement learning," in *IDEAL*, 2007, pp. 87–96.

[59] A. Bai, S. Srivastava, and S. J. Russell, "Markovian state and action abstractions for mdps via hierarchical MCTS," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2016, pp. 3029–3039.

[60] E. Brunskill and L. Li, "Pac-inspired option discovery in lifelong reinforcement learning," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 316–324.

[61] K. Subramanian, C. Isbell, and A. Thomaz, "Learning Options through Human Interaction," in *Workshop on Agents Learning Interactively from Human Teachers at IJCAI*, (IJCAI), 2011.

[62] G. Konidaris, S. Kuindersma, A. G. Barto, and R. A. Grupen, "Constructing skill trees for reinforcement learning agents from demonstration trajectories," in *NIPS*, 2010, pp. 1162–1170.

[63] P. Zang, P. Zhou, D. Minnen, and C. L. I. Jr., "Discovering options from example trajectories," in *ICML*, 2009, p. 153.

[64] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," in *ICML*, 2004.

[65] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*, 2006, pp. 282–293.

[66] T. Walsh, S. Goschin, and M. Littman, "Integrating sample-based planning and model-based reinforcement learning," in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, 2010.

[67] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[68] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," Book in preparation for MIT Press, 2016.

[69] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., 2014, pp. 3338–3346.

[70] G. Chaslot, C. Fiter, J.-B. Hoock, A. Rimmel, and O. Teytaud, "Adding expert knowledge and exploration in monte-carlo tree search," in *ACG*, 2009, pp. 1–13.

[71] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[72] B. Price and C. Boutilier, "Accelerating reinforcement learning through implicit imitation.," *Journal of Artificial Intelligence Research (JAIR)*, vol. 19, pp. 569–629, Jun. 11, 2010.

[73] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. L. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," in *Neural Information Processing Systems 26*, 2013, pp. 2625–2633.

[74] W. B. Knox and P. Stone, "Combining manual feedback with subsequent MDP reward signals for reinforcement learning," in *Proc. of the 9th Intl. Conf. on AAMAS*, 2010, pp. 5–12.

[75] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. of the 16th ICML*, 1999, pp. 341–348.

[76] C. L. Isbell, C. Shelton, M. Kearns, S. Singh, and P. Stone, "A social reinforcement learning agent," in *Proc. of the 5th Intl. Conf. on Autonomous Agents*, 2001, pp. 377–384.

[77] H. S. Chang, "Reinforcement learning with supervision by combining multiple learnings and expert advices," in *Proc. of the American Control Conference*, 2006.

[78] W. B. Knox and P. Stone, "Tamer: Training an agent manually via evaluative reinforcement," in *Proc. of the 7th IEEE ICDL*, 2008, pp. 292–297.

[79] A. Tenorio-Gonzalez, E. Morales, and L. Villaseor-Pineda, "Dynamic reward shaping: Training a robot by voice," in *Advances in Artificial Intelligence–IBERAMIA*, 2010, pp. 483–492.

[80] P. M. Pilarski, M. R. Dawson, T. Degris, F. Fahimi, J. P. Carey, and R. S. Sutton, "Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning," in *Proc. of the IEEE ICORR*, 2011, pp. 1–7.

[81] C. L. Isbell, M. Kearns, S. Singh, C. R. Shelton, P. Stone, and D. Kormann, "Cobot in LambdaMOO: An Adaptive Social Statistics Agent," *JAAMAS*, vol. 13, no. 3, pp. 327–354, 2006.

[82] A. L. Thomaz and C. Breazeal, "Teachable robots: Understanding human teaching behavior to build more effective robot learners," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 716–737, 2008.

[83] W. B. Knox and P. Stone, "Reinforcement learning from simultaneous human and MDP reward," in *Proc. of the 11th Intl. Conf. on AAMAS*, 2012, pp. 475–482.

[84] R. Maclin and J. W. Shavlik, "Creating advice-taking reinforcement learners," *Machine Learning*, vol. 22, no. 1-3, pp. 251–281, 1996.

[85] L. Torrey, J. Shavlik, T. Walker, and R. Maclin, "Transfer learning via advice taking," in *Advances in Machine Learning I, Studies in Computational Intelligence*, J. Koronacki, S. Wirzchon, Z. Ras, and J. Kacprzyk, Eds., vol. 262, Springer Berlin Heidelberg, 2010, pp. 147–170.

[86] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters, "Active reward learning," in *Proceedings of Robotics: Science & Systems (R:SS)*, 2014.

[87] A. F. K. Judah and T. Dietterich, "Active imitation learning via state queries," *Proceedings of the ICML Workshop on Combining Learning Strategies to Reduce Label Cost*, 2011.

[88] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proc. of the 17th ICML*, 2000.

[89] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. of the 21st ICML*, 2004.

[90] M. Cakmak and M. Lopes, "Algorithmic and human teaching of sequential decision tasks," in *AAAI Conference on Artificial Intelligence*, 2012.

[91] C. Atkeson and S. Schaal, "Learning tasks from a single demonstration," in *Proc. of the IEEE ICRA*, 1997, pp. 1706–1712.

[92] M. Taylor, H. B. Suay, and S. Chernova, "Integrating reinforcement learning with human demonstrations of varying ability," in *Proc. of the Intl. Conf. on AAMAS*, 2011, pp. 617–624.

[93] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning.," in *AISTATS*, ser. JMLR Proceedings, vol. 15, JMLR.org, 2011, pp. 627–635.

[94] C. Wirth, R. Akrour, G. Neumann, and J. Fürnkranz, "A survey of preference-based reinforcement learning methods," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 4945–4990, 2017.

[95] M. Geist and O. Pietquin, "Algorithmic survey of parametric value function approximation," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 24, no. 6, pp. 845–867, 2013.

[96] T. G. Dietterich, "Machine learning for sequential data: A review," in *Structural, Syntactic, and Statistical Pattern Recognition*, Springer-Verlag, 2002, pp. 15–30.

[97] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," in *Journal of Machine Learning Research*, vol. 4, 2003, pp. 1107–1149.

[98] J. A. Boyan, "Technical update: Least-squares temporal difference learning," *Machine learning*, vol. 49, no. 2-3, pp. 233–246, 2002.

[99] T. Hester, *Texplore: Temporal difference reinforcement learning for robots and time-constrained domains*. Springer, 2013.

[100] G. Konidaris, S. Osentoski, and P. Thomas, "Value function approximation in reinforcement learning using the fourier basis," in *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[101] C. Watkins and P. Dayan, "Q learning: Technical note," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[102] T. Matthews, S. D. Ramchurn, and G. Chalkiadakis, "Competing with humans at fantasy football: Team formation in large partially-observable domains," in *Proc. of the 26th AAAI*, 2012, pp. 1394–1400.

[103] C. Bailer-Jones and K. Smith, "Combining probabilities," GAIA-C8-TN-MPIA-CBJ-053, 2011.

[104] M. L. Littman, G. A. Keim, and N. Shazeer, "A probabilistic approach to solving crossword puzzles," *Artificial Ingelligence*, vol. 134, no. 1-2, pp. 23–55, 2002.

[105] G. Konidaris and A. Barto, "Autonomous shaping: Knowledge transfer in reinforcement learning," in *Proc. of the 23rd ICML*, 2006, pp. 489–496.

[106] S. Gelly and Y. Wang, "Exploration exploitation in go: Uct for monte-carlo go," in *Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006)*, Citeseer, 2006.

[107] H. Finnsson and Y. Bjornsson, "Simulation-based approach to general game playing," in *Proceedings of the 23rd national conference on Artificial intelligence*, 2008, pp. 259–264.

[108] R. Aler, O. Garcia, and J. Valls, "Correcting and improving imitation models of humans for robosoccer agents," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, 2005.

[109] M. Cakmak and A. Thomaz, "Optimality of human teachers for robot learners," in *Proceedings of the IEEE International Conference on Development and Learning (ICDL)*, 2010.

[110] A. Ng, H. Kim, M. Jordan, S. Sastry, and S. Ballianda, "Autonomous helicopter flight via reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 16, 2004.

[111] D. Grollman and O. Jenkins, "Dogged learning for robots," in *IEEE International Conference on Robotics and Automation*, Citeseer, 2007, pp. 2483–2488.

[112] M. Dorigo and M. Colombetti, "Robot shaping: Developing autonomous agents through learning," *Artif. Intell.*, vol. 71, no. 2, pp. 321–370, 1994.

[113] R. Sutton and A. Barto, *Reinforcement learning: An introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998, ISBN: 9780262193986.

[114] "Efficient exploration in monte carlo tree search using human action abstractions," in *Workshop on Future of Interactive Machine Learning at NIPS*, (NIPS), 2016.