Directed Exploration for Improved Sample Efficiency in Reinforcement Learning

Zhaohan Daniel Guo

CMU-CS-18-126

February 2019

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Emma Brunskill, Chair
Drew Bagnell
Ruslan Salakhutdinov
Remi Munos, Google DeepMind

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant number IIS1350984, the Office of Naval Research under grant number N000141612241, the Department of Education under grant number R305A130215, Microsoft, Google, and Yahoo.

Keywords: Reinforcement learning, exploration, artificial intelligence, sample complexity

Abstract

A key challenge in reinforcement learning is how an agent can efficiently gather useful information about its environment to make the right decisions, i.e., how can the agent be sample efficient. This thesis proposes using a new technique called directed exploration to construct new sample efficient algorithms for both theory and practice. Directed exploration involves repeatedly committing to reach specific goals within a certain time frame. This is in contrast to dithering which relies on random exploration or optimism-based approaches that implicitly explore the state space. Using directed exploration can yield provably efficient sample complexity in a variety of settings of practical interest: when solving multiple tasks either concurrently or sequentially, algorithms can explore distinguishing state–action pairs to cluster similar tasks together and share samples to speed up learning; in large, factored MDPs, repeatedly trying to visit lesser known state–action pairs can reveal whether the current dynamics model is faulty and which features are unnecessary. Finally, directed exploration can also improve sample efficiency in practice for the deep reinforcement learning by being more strategic than dithering-based approaches and more robust than reward-bonus based approaches.

Acknowledgments

I would like to thank my thesis committee: Drew Bagnell, Ruslan Salakhutdinov, Remi Munos, and Emma Brunskill, for their valuable feedback and guidance throughout the thesis process. Their sharp critique and insight have inspired me and pushed me to make this thesis even stronger.

I would like to thank my collaborator, Yao Liu, for the joint work done in the sequential task setting in chapter 3. He has been a great collaborator in sharing ideas and especially in coming up with the theoretical machinery to validate those ideas.

I would also like to thank all of my other labmates from Emma's lab and the various post-docs throughout the years for our discussions and collaborations that have helped develop and solidify research ideas and papers.

All the thanks to my PhD advisor Emma Brunskill, who has guided me and helped me on my journey as a PhD student. I am very blessed to have had her continuous support all this time. She is an amazing and dedicated researcher who has been a constant inspiration to me. From the beginning when I was just a nascent researcher, she has carefully helped build up my foundations and filled in my gaps in skills such as helping with brainstorming and developing research ideas to helping write conference papers. Because of her, I have built up the necessary skills and confidence as well as retained my initial enthusiasm and motivation to continue to do research and advance science.

Finally, I would like to thank my parents Sheila and Jaff for their unending hope and support during my PhD. They have helped me persevere and push through even the toughest of times.

Preface

The goal of this thesis is to combine together several of my work that share and build on the same basic idea of directed exploration. The main work presented in the central chapters of this thesis are based on published and in-submission work.

Chapter 2 is based on

> Zhaohan Guo and Emma Brunskill. Concurrent PAC RL. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015b

Chapter 3 is based on

> Yao Liu, Zhaohan Guo, and Emma Brunskill. Pac Continuous State Online Multitask Reinforcement Learning with Identification. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 438–446. International Foundation for Autonomous Agents and Multiagent Systems, 2016

Chapter 4 is based on

> Zhaohan Daniel Guo and Emma Brunskill. Sample Efficient Learning with Feature Selection for Factored MDPs. In *EWRL*, 2018

Chapter 5 is based on work currently in-submission.

During my PhD, I have also worked on other projects that have not been included in the main text of this thesis.

- Zhaohan Daniel Guo, Shayan Doroudi, and Emma Brunskill. A PAC RL Algorithm for Episodic POMDPs. In *Artificial Intelligence and Statistics*, pages 510–518, 2016

- Zhaohan Guo, Philip S Thomas, and Emma Brunskill. Using Options and Covariance Testing for Long Horizon Off-Policy Policy Evaluation. In *Advances in Neural Information Processing Systems*, pages 2492–2501, 2017

- Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Pion, Bernardo A. Pires, Toby Pohlen, and Rémi Munos. Neural Predictive Belief Representations. *arXiv preprint arXiv:1811.06407*, 2018

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Directed Exploration 6

1.3 Basic Background 7

1.4 Overview 8

2 Sample Efficient Concurrent MDPs **11**

2.1 Introduction 11

2.2 Background 13

2.3 Concurrent RL in Identical Environments 13

2.4 CMBIE Experiments 18

2.5 Concurrent RL in Different Environments 19

2.5.1 Algorithm 19

2.5.2 Assumptions 20

2.5.3 Theory 21

2.6 Clustering CMBIE Experiments 27

2.7 Follow-Up Work 28

3 Sample Efficient Transfer Learning in a Finite Set of Continuous MDPs **29**

3.1 Introduction 29

3.2 Setting 31

3.3 Algorithm . . . 31
3.4 Assumptions and definitions . . . 32
3.5 Phase 1 . . . 35
3.6 Clustering and Informative State–action Pairs . . . 36
3.7 Phase 2 . . . 36
3.8 Motivation for Using Q-functions . . . 38
3.9 Theoretical guarantees . . . 39
3.10 Lemmas . . . 41
3.11 Proof of Main Theorem . . . 50
3.12 Experiments . . . 51

4 Sample Efficient Feature Selection for Factored Markov Decision Processes 55
4.1 Introduction . . . 55
4.2 Setting . . . 58
4.3 Related Work . . . 59
4.4 Main Challenge of Feature Selection with Strategic Exploration . . . 60
4.5 Algorithm . . . 62
4.5.1 Overview . . . 62
4.5.2 LearnAndSelect . . . 64
4.6 Theoretical Analysis . . . 67
4.6.1 Assumptions . . . 67
4.6.2 Discussion of Assumption 2 . . . 67
4.6.3 Small Lemmas . . . 69
4.6.4 LearnAndSelect . . . 71
4.6.5 Directed Exploration . . . 73
4.6.6 Main Theorem . . . 76

5 Directed Exploration in RL with Function Approximation 79
5.1 Introduction . . . 79

5.2 Background . . . 81
5.2.1 DQN . . . 81
5.2.2 DDPG . . . 81
5.2.3 UVFA . . . 82
5.2.4 HER . . . 82
5.3 Directed Exploration . . . 82
5.3.1 Directed Exploration Outline . . . 82
5.3.2 Directed Exploration with Function Approximation . . . 83
5.4 Robustness Example . . . 86
5.5 Tabular Experiment . . . 86
5.6 Function Approximation Experiment in a Small Domain . . . 88
5.6.1 Experimental Details . . . 90
5.7 Function Approximation Experiment in a Large Domain . . . 91
5.7.1 Experimental Details . . . 93
5.8 Discussion of Results . . . 94
5.9 Related Work . . . 96

6 Conclusion **97**
6.1 Overview . . . 97
6.2 Future Work . . . 98

Bibliography **101**

List of Figures

1.1 Reinforcement learning loop. The agent take an action in the environment, which then transitions to a new state and gives as feedback a reward value. 2

1.2 Simple Chain Domain. The agent is free to move left and right across the chain, but moving up or down teleports the agent back to the starting state S. The left side of the chain can be quite long. Rewards at left and right ends of the chain are 5 and 1 respectively, and are initially unknown. All other states give zero reward. 3

1.3 An example of a posterior distribution over the unknown left end reward which consists of four possible values of 0, 1, 2, or 3. 4

1.4 Two examples of an augmented reward for the left end state. In the left case, the agumented reward is 1.1, and in the right case, the augmented reward is slightly lower at 0.9. Reward bonus approaches are very sensitive to this slight difference in magnitude, which can lead to neglecting to explore the left end entirely if it believes that the average reward is less than 1. 5

2.1 CMBIE Experiments 14

2.2 ClusterCMBIE Experiments 27

3.1 Averaged reward per task. The bar is standard deviation over 40 rounds. . 52

5.1 The different components of directed exploration with function approximation. 83

5.2 Comparison on gridworld with teleportation. Q-Learning with e-greedy and reward bonus (Q) takes much longer to converge than directed exploration (DE) or MBIE. These results are averaged over 100 runs, and show the evaluation performance when we run the greedy policy. 87

5.3 MountainCar Environment [Brockman et al., 2016] 88

5.4 Episode Reward on Mountain Car. Result is over 10 runs, and show the evaluation performance when we run the greedy policy. 89

5.5 Percent of visited on Mountain Car over a discretized 10x10 grid. Result is over 10 runs. 90

5.6 FetchPush Domain, image from [Plappert et al., 2018]. 91

5.7 Percentage of success of reaching the true goal on FetchPush when comparing regular DDPG, DDPG and reward bonus, and DDPG with directed exploration both with picking random goals and top 100 most uncertain goals. Results are over 5 runs. 92

5.8 Percentage of visitation of possible discretized goal states on FetchPush when comparing regular DDPG, DDPG and reward bonus, and DDPG with directed exploration both with picking random goals and top 100 most uncertain goals. Results are over 5 runs. 93

Chapter 1

Introduction

1.1 Motivation

One key aspect of an intelligent agent is the ability to explore and learn new things. As babies, we crawl and interact with all kinds of unknown objects in order to learn about the world. As children we retain that curiosity and try to conduct all kinds of experiments to figure out how the world works. Exploration is an essential component for filling in the gaps in our knowledge, as well as for discovering new and interesting things.

Even after growing up, we continue to explore the unknown. Suppose you want to eat dinner at a restaurant, but there are many restaurants that have recently opened that you have not tried before. So you do some research and look at online reviews as well as their menus to construct a subjective evaluation of which restaurants look promising. Then you may decide to try out the most promising restaurant, or maybe randomly pick one of the top 3 most promising restaurants. This is an example of performing exploration in a more informed fashion based on prior knowledge and personal taste. This more informed exploration is key to exploring efficiently. If there were 20 restaurants that you have not tried before, and you decided to try them out randomly, it could take more than 10 tries before you find a good restaurant. However, looking up reviews and menus and judging according to your own tastes can greatly increase the chance of finding a good restaurant in just 2 or 3 tries. Using more informed exploration can greatly speed up exploration to target the most promising areas.

This thesis proposes a new technique called directed exploration that, at a high-level, mimics this informed exploration process. In particular, directed exploration consists of first using an informed measure to evaluate and priortize the unknown, and then directly

tries them out according to that measure. Now we will go into more details about how this works.

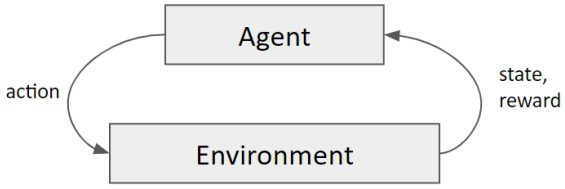


Figure 1.1: Reinforcement learning loop. The agent take an action in the environment, which then transitions to a new state and gives as feedback a reward value.

To train intelligent, autonomous agents, we use the reinforcement learning framework (Figure 1.1). In reinforcement learning, a branch of machine learning, an agent interacts with an environment and the only feedback is through a real-valued (possibly sparse) reward. At every timestep, the algorithm takes an action in the environment, observes how the environment state changes, and gets a reward. Each step can be considered as gathering one new sample data point. The algorithm then tries to change its actions to maximize its cumulative reward. This processes reflects many real world problems where it is difficult to specify the right answer ahead of time, but easy to identify when some goal has been achieved. Robotic domains fall under this specification, for example, trying to get a robot to walk on two legs; it can be very hard to specify how to precisely move the joints to ensure a stable walk on real world uneven terrain, but we can give a positive reward to the robot as long as it has not fallen. Many domains involving humans also fall under this category, for example, trying new restaurants, a tutoring program that tries to teach students fractions, conversation bots, and much more.

A key challenge in reinforcement learning is how to be sample efficient. An agent would like to use the smallest number of samples (steps) it needs in order to learn the optimal policy, i.e., the optimal actions to take in each environment state. Due to the sequential nature of reinforcement learning, the agent has to try to explore the environment and take actions that could lead the agent to learn more about the dynamics of the environment rather than always trying actions with the highest rewards. Efficient exploration is a crucial aspect of any sample efficient algorithm.

Recently, there have been many exciting advances in practical reinforcement learning through the use of deep reinforcement learning. From achieving superhuman performance in video games [Mnih et al., 2013], to learning how to move in robotics domains [Schulman et al., 2015], deep reinforcement learning has been achieving many successes in many domains. However in order to continue to make reinforcement learning practical, sample

efficiency becomes quite important for many problems. These early algorithms suffer from high sample complexity, e.g. it can take up to 13 million steps to learn how to play the Atari game Pong, which translates to over 60 hours of game time [Schaul et al., 2015b]. A human player, on the other hand, may take less than one hour to learn how to play.

Collecting samples can be very costly when testing real robots, running health care trials, or in general scenarios where interaction with the real world and people is required. Even when data is relatively easy to collect, like in video games, using algorithms like DQN without sophisticated exploration strategies in the Atari game Montezumas Revenge can results in no learning whatsoever [Mnih et al., 2013]. Being able to efficiently explore and collect the most useful samples becomes incredibly important in making algorithms more sample efficient.

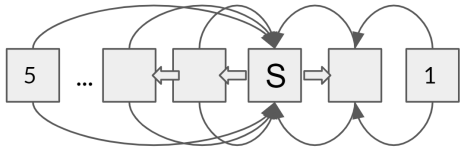


Figure 1.2: Simple Chain Domain. The agent is free to move left and right across the chain, but moving up or down teleports the agent back to the starting state S. The left side of the chain can be quite long. Rewards at left and right ends of the chain are 5 and 1 respectively, and are initially unknown. All other states give zero reward.

Early success in deep reinforcement learning relied on using a simple exploration technique known as dithering. Dithering consists of adding randomness to actions; in the discrete action setting, dithering can be taking a random action with a certain probability, i.e., e-greedy [Mnih et al., 2013]; in the continuous action setting, dithering could be adding Gaussian noise to actions [Schulman et al., 2015, 2017]. However because of its simplicity, dithering can be extremely inefficient in many domains. A simple counter-example is shown in Figure 1.2, which is a simple chain domain. In this chain domain, the agent is free to move left and right across the states of the chain, but moving up or down teleports the agent back to the starting state. There is a large but unknown reward of 5 at the left end of the chain, and a smaller reward of 1 at the right end. There is no reward anywhere else. Through dithering, an agent would be able to easily discover the reward of 1 at the right end, but unfortunately it would take an exponential number of steps to be able to reach the unknown reward at the left end. This is because through taking random actions, the agent must avoid taking the up or down actions that teleport back to the center and try to continuously go left. This means that at the very least, there is a 0.5 of taking either up and down and teleporting back, and thus the probability of avoiding this outcome will be

at most 0.5^N where N is the length number of states in the chain to the left. Thus, the probability of succesfully reaching the left end is exponentially small i.e., it will take an exponential number of steps to reach the left end with high probability.

One interesting extension of dithering is to inject noise to the paramters of your model rather than directly in the action space. This has been show to have some mixed results, resulting in improvements in some domains [Fortunato et al., 2017].

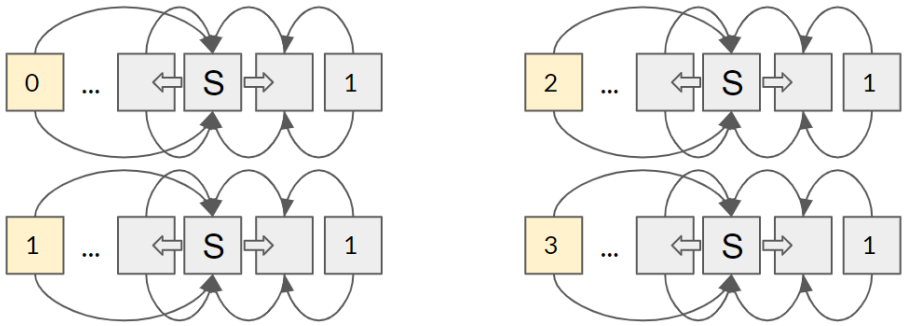


Figure 1.3: An example of a posterior distribution over the unknown left end reward which consists of four possible values of 0, 1, 2, or 3.

A more sophisticated exploration strategy that has been implemented is Thompson (posterior) sampling [Osband et al., 2013]. This technique involves maintaining a posterior over the unknown parameters of the environment, and then repeatedly sampling one potential set of parameters and acting according to the specified environment for some period of time. This technique has been shown to have polynomial Bayesian sample complexity bound in the simple tabular setting where the states and actions are finite and discrete [Osband et al., 2013]. The chain example here falls under the tabular setting. A sample complexity bound is a bound on the number of steps that an agent makes a mistake and takes an action that does not lead to near-optimal performance. If such a bound is polynomial, then it can be considered to be a good bound and the agent can be considered to be sample efficient. A regret bound, which is an alternative bound to the sample complexity bound that is concerned with the rate of improvement of the agent, has also been shown [Agrawal and Jia, 2017].

With thompson sampling, we can get much more efficient exploration in the case of the simple chain domain. Suppose we have a posterior distribution that either assigns a value of 0, 1, 2, or 3 to the left end reward (Figure 1.3). Then as long as we sample the model where the left end reward is higher than 1, then we would end up trying to reach the left end and finding out that the true reward is 5. It no longer takes an exponential number of steps

to explore the left end and thus this is much more efficient. While Thompson sampling has shown promising empirical results in the tabular and other simple settings [Chapelle and Li, 2011, Osband and Van Roy, 2017], scaling it up to more practical, high-dimensional domains has yielded mixed results[Osband et al., 2016, 2018]. This is because in high-dimensional domains, the posterior distribution must now be approximated, and it is a challenge to approximate it accurately and sample from it efficiently.

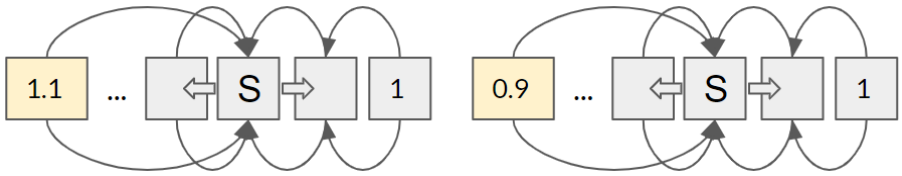


Figure 1.4: Two examples of an augmented reward for the left end state. In the left case, the agumented reward is 1.1, and in the right case, the augmented reward is slightly lower at 0.9. Reward bonus approaches are very sensitive to this slight difference in magnitude, which can lead to neglecting to explore the left end entirely if it believes that the average reward is less than 1.

Currently, the most successful exploration strategy in practice has been through the use of reward bonuses, by adding a bonus to the reward function to encourage the agent to visit less visited states. This approach originated in small settings under the idea of Optimism in the Face of Uncertainty (OFU). This idea is about constructing confidence intervals around the unknown environment parameters, and then acting according to the most optimistic instantiation of those parameters i.e., the instantiation that would give the most reward. A simple implementation of this idea is to add a bonus to the estimated reward function, where the bonus is correlated with the size of the confidence intervals. A larger confidence interval means that there is more uncertainty, so a larger bonus is given to encourage the agent to explore that area and collect samples to learn more and shrink the uncertainty. This idea has results in many sample effcient algorithms with polynomial sample complexity bounds in the tabular setting Brafman and Tennenholtz [2003], Strehl and Littman [2008].

However when scaling up to more practical domains, computing a bonus or even tracking parameter uncertainty often becomes computationally infeasible. Many different techniques have been tried to compute an approximate and practical bonus, from keeping approximate visitation counts [Tang et al., 2017], to prediction loss of a dynamics model [Burda et al., 2018], or even the prediction gain of an approximate state density model [Ostrovski et al., 2017]. The main drawback is that reward bonuses are approximate, non-stationary and change as the agent explores, which may hinder exploration. For example,

consider the chain domain where we use an approximate reward bonus that sometimes assigns a bonus of 1.1 and other times a bonus of 0.9 to the left end (Figure 1.4). Even though 1.1 and 0.9 are close in magnitude to each other and close to the right end reward of 1, if more often than not, the reward bonus is 0.9, then the algorithm will end up thinking that the left end reward is on average less than 1. This means the algorithm has no incentive to go to the left, and completely misses out on exploring the left end.

1.2 Directed Exploration

This thesis builds on using the same kind of uncertainty that the reward bonus approach uses and proposes using a new technique called directed exploration and makes the following contributions:

1. We construct new provably sample efficient algorithms that rely on directed exploration to tackle several settings of practical interest that have not had sample efficient bounds before, specifically the concurrent task setting, the sequential task setting, and feature selection in factored settings

2. We also construct a new, practical algorithm based on directed exploration to use with function approximation that can be more sample efficient and robust than the reward bonus approach in some settings

The focus of this thesis is not on computing this bonus, but rather on an alternate method of utilizing this bonus, different from augmenting the reward function, and thereby sidestepping the issue of non-stationarity altogether. We rely on prediction loss of a dynamics model [Burda et al., 2018] to comptue an approximate bonus.

Directed exploration, more precisely, consists of the following:

1. Given a source of uncertainty, sample goal states with the highest uncertainty

2. Learn and use a goal-conditioned policy $\pi(s, g)$ to try to reach the sampled goal within a time limit

3. Repeat from step 1

Directed exploration is more explicit about trying to reach goal states as opposed to the reward bonus approach in which we rely implicitly on the augmented reward function to visit promising states. Note that the goal-conditioned policy $\pi(s, g)$ is independent of

the measure of the uncertainty, and is thus completely stationary and does not need to be learned again when the uncertainty changes. In the case of the chain domain with the stochastic reward bonus (Figure 1.4), due to the fact that the magitude of the left end reward is close to the reward of the right end, directed exploration would end up picking the left end as a goal just as often as the right end, and thus end up trying to explore both ends reasonably often.

However directed exploration is only one part of a reinforcement learning algorithm, which is to explore. We combine directed exploration with other components in order to yield algorithms with provably efficient sample complexity in a variety of settings of practical interest, and also improve sample efficiency in practice for reinforcement learning with function approximation compared with the reward bonus approach by being more strategic and robust.

1.3 Basic Background

In this section is a basic specification of reinforcement learning. More detailed background is in each chapter.

A basic formulation of the reinforcement learning process is represented by a Markov decision process (MDP). An MDP is a tuple $\langle S, A, T, R, \gamma \rangle$ where S is a set of states of the environment, A is a set of actions, T is a transition model where $T(s'|s, a)$ is the probability of the environment transitioning to state s' given current state s, and the algorithm taking action a. $R(s, a) \in [0, 1]$ is the expected reward received in state s upon taking action a, and γ is an (optional) discount factor. A policy π is a mapping from states to actions. The value $V^\pi(s)$ of a policy π is the expected sum of discounted rewards obtained by following π starting in state s: $V^\pi(s) = \sum_{t=1}^{\infty} \gamma^{t-1} r_t$. The optimal policy π^* for an MDP is the one with the highest value function, denoted $V^*(s)$. In reinforcement learning, the transition and reward models are typically unknown. A reinforcement learning algorithm for solving MDPs would take actions to gather samples of (s, a, s') tuples, and use them to maximize its expected value.

To theoretically analyze RL algorithms, we use the Probably Approximately Correct (PAC) RL framework [Kearns and Singh, 2002b, Brafman and Tennenholtz, 2002, Strehl et al., 2006]. An RL algorithm is considered to be PAC if the number of steps t in which

$$V(s_t) + \epsilon < V^*(s_t) \tag{1.1}$$

is bounded by a polynomial function of the problems' parameters, with high probability. In other words, the number of sub-ϵ-optimal steps, i.e. mistakes, is at most polynomial.

1.4 Overview

Here we provide a brief outline of each chapter. A more detailed introduction is present in each chapter.

In chapter 2, we consider the situation in which a decision maker may make decisions across many separate reinforcement learning tasks in parallel, which we denote as Concurrent RL. An example of this could be many students concurrently attending an online class, and we would like to personalize the course materials for each student. We introduce two new concurrent RL algorithms that rely on directed exploration and prove polynomial sample complexity bounds i.e. that they are PAC. We prove that under some mild conditions, both when the agent is known to be acting in many copies of the same MDP, and when they are not the same but are taken from a finite set, we can gain linear improvements in the sample complexity over not sharing information, with the main idea behind the algorithm using directed exploration to find states where we can distinguish between different types and cluster the same types together quickly. Our experiments confirm this result and show preliminary empirical benefits.

In chapter 3, we consider the case where an agent will be performing a series of reinforcement learning tasks with a continuous state space and a discrete action space. This serial set of tasks comes up in many applications, including robotics, consumer marketing, and healthcare. We introduce a new algorithm (Continuous State Online Multitask RL with Identification a.k.a COMRLI) for online multi-task learning across a series of continuous-state, discrete-action RL tasks with the same state and action spaces that are drawn from a finite set. We prove that this algorithm is PAC and that its sample complexity is much smaller due to being able to reuse past data to solve future tasks. The key point of our algoritm is in how we continually refine a set of possible distinguishing states, and rely on directed exploration to gather more data from those states until we narrow down to exactly the states that can help us identify the type of the new task and reuse previous task experience. Again, our exploration involves trying to directly visit a special subset of states that is useful for distinguishing tasks. We also provide encouraging, preliminary empirical performance on a standard domain where our algorithm exceeds a state-of-the-art continuous-state multitask RL algorithm.

In chapter 4, we consider how to improve sample complexity through feature selection. In many machine learning and AI control problems, choosing which features to represent the state of the domain is critical. Using a set of carefully hand-designed features can greatly enhance performance, but this process can be expensive, requiring domain experts to select the features, and may easily miss relevant features resulting in sub-optimal performance. We present a new reinforcement learning algorithm and prove that its sample

complexity scales with the complexity of only the relevant features needed to learn the optimal policy, and not on all features; then there would be no need to hand-design features and instead this algorithm would automatically find the best subset of features, even though which features are necessary is unknown. Our key insight is to use directed exploration and leverage negative information from failing to reach goals to identify when our estimated model of the MDP must be wrong, and to be able to identify at least one aspect of the model that is wrong. This idea allows us to make progress, eventually eliminating features for which we have poor models when computing our optimal policy. We are not aware of prior work that uses negative information during reinforcement learning, and we believe that this insight may have practical benefits for new algorithmic developments in future work.

Finally, in chapter 5, we bring the idea of directed exploration to more practical settings, specifically to the deep RL setting. One of the major challenges in deep RL has been the difficulty of doing efficient exploration[Mnih et al., 2013]. Currently, uncertainty-based methods have been shown to be the most effective and most promising to tackle hard exploation domains [Houthooft et al., 2016, Ostrovski et al., 2017, Tang et al., 2017, Burda et al., 2018]. These uncertainty-based methods use a reward bonus approach, where they compute a measure of uncertainty and transform that into a bonus that is then added into the reward function. Unfortunately, as previously mentioned, this reward bonus approach is non-stationary which can lead to slow convergence of function approximators as well as being very sensitive to the relative magnitudes of the reward bonuses (Figure 1.4). We introduce a practical form of deep RL with directed exploration that repeatedly tries to visit promising goal states with high uncertainty. We show in our experiments that directed exploration is more robust to noisy and inaccurate uncertainty measures, and is more efficient at exploration than the reward bonus approach.

We conclude this thesis with a discussion on possible future directions for directed exploration.

Chapter 2

Sample Efficient Concurrent MDPs

This work was done jointly with Emma Brunskill and accepted at AAAI 2015 [Guo and Brunskill, 2015a].

2.1 Introduction

In many real-world situations a decision maker may make decisions across many separate reinforcement learning tasks in parallel, yet there has been very little work on concurrent RL. Building on the efficient exploration RL literature, we introduce two new concurrent RL algorithms and bound their sample complexity. We show that under some mild conditions, both when the agent is known to be acting in many copies of the same MDP, and when they are not the same but are taken from a finite set, we can gain linear improvements in the sample complexity over not sharing information, with the main idea behind the algorithm being directed exploration. This is quite exciting as a linear speedup is the most one might hope to gain. Our preliminary experiments confirm this result and show empirical benefits.

There has been a number of papers (e.g. [Evgeniou and Pontil, 2004, Xue et al., 2007]) on supervised concurrent learning (referred to as multi-task learning). In this context, multiple supervised learning tasks, such as classification, are run in parallel, and information from each is used to speed learning. When the tasks themselves involve sequential decision making, like reinforcement learning, prior work has focused on sharing information serially across consecutive related tasks, such as in transfer learning (e.g. [Taylor and Stone, 2009, Lazaric and Restelli, 2011]) or online learning across a set of tasks [Brunskill and Li, 2013a]. Note that multi-agent literature considers multiple agents acting in

a single environment, whereas we consider the different problem of one agent / decision maker simultaneously acting in multiple environments. The critical distinction here is that the actions and rewards taken in one task do not directly impact the actions and rewards taken in any other task (unlike multi-agent settings) but information about the outcomes of these actions may provide useful information to other tasks, if the tasks are related.

One important exception is recent work by Silver et al. (2013) on concurrent reinforcement learning when interacting with a set of customers in parallel. This work nicely demonstrates the substantial benefit to be had by leveraging information across related tasks while acting jointly in these tasks, using a simulator built from hundreds of thousands of customer records. However, this paper focused on an algorithmic and empirical contribution, and did not provide any formal analysis of the potential benefits of concurrent RL in terms of speeding up learning.

We draw upon literature on Probably Approximately Correct (PAC) RL [Kearns and Singh, 2002b, Brafman and Tennenholtz, 2002, Kakade, 2003], and bound the sample complexity of our approaches, which is the number of steps on which the agent may make a sub-optimal decision, with high probability. Interestingly, when all tasks are identical, we prove that simply by applying an existing state-of-the-art single task PAC RL algorithm, MBIE [Strehl and Littman, 2008], we can obtain, under mild conditions, a linear improvement in the sample complexity, compared to learning in each task with no shared information. We next consider a much more generic situation, in which the presented tasks are sampled from a finite, but unknown number of discrete state–action MDPs, and the identity of each task is unknown. Such scenarios can arise for many applications in which an agent is interacting with a group of people in parallel: for example, Lewis [Lewis, 2005] found that when constructing customer pricing policies for news delivery, customers were best modeled as being one of two (latent) types, each with distinct MDP parameters. We present a new algorithm for this setting that uses directed exploration to try and cluster the MDPs into the latent types, and prove that under fairly general conditions, that if any two distinct MDPs differ in their model parameters by a minimum gap for at least one state–action pair, and the MDPs have finite diameter, that we can also obtain essentially a linear improvement in the sample complexity bounds across identical tasks. Our approach incurs no dominant overhead in sample complexity by having to perform a clustering amongst tasks, implying that if all tasks are distinct, the resulting (theoretical) performance will be equivalent to as if we performed single task PAC RL in each task separately. These results provide an interesting counterpart to the sequential transfer work of Brunskill and Li [Brunskill and Li, 2013a] which demonstrated a reduction in sample complexity was possible if an agent performed a series of tasks drawn from a finite set of MDPs; however, in contrast to that work that could only gain a benefit after completing many tasks, clus-

tering them, and using that knowledge for learning in later tasks, we demonstrate that we can effectively cluster tasks and leverage this clustering during the reinforcement learning of those tasks to improve performance. We also provide small simulation experiments that support our theoretical results and demonstrate the advantage of carefully sharing information during concurrent reinforcement learning.

2.2 Background

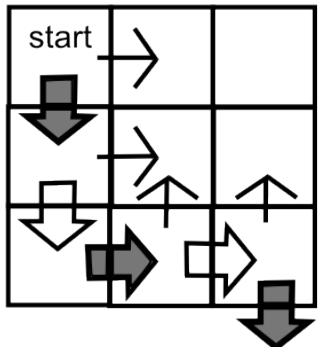
We rely on Probably Approximately Correct (PAC) RL methods [Kearns and Singh, 2002b, Brafman and Tennenholtz, 2002, Strehl et al., 2006] to guarantee the number of steps on which the agent will make a less than ϵ-optimal decision, the sample complexity, is bounded by a polynomial function of the problems' parameters, with high probability. Sample complexity can be viewed as a measure of the learning speed of an algorithm, since it bounds the number of possible mistakes the algorithm will make. We will similarly use sample complexity to formally bound the potential speedup in learning gained by sharing experience across tasks.

Our work builds on MBIE, a single-task PAC RL algorithm [Strehl and Littman, 2008]. In MBIE the agent uses its experience to construct confidence intervals over its estimated transition and reward parameters. It computes a policy by performing repeated Bellman backups which are optimistic with respect to its confidence intervals, thereby constructing an optimistic MDP model, an optimistic estimate of the value function, and an optimistic policy. This policy will drive the agent towards little experienced state–action pairs or state–action pairs with high reward. We chose to build on MBIE due to its good sample complexity bounds and very good empirical performance.

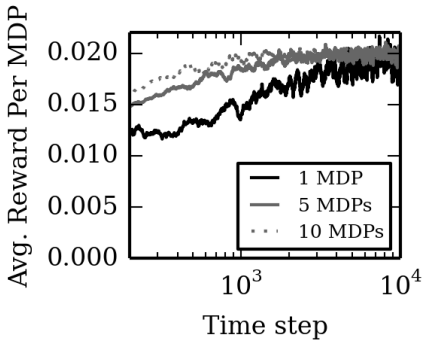
We think it will be similarly possible to create concurrent algorithms and analysis building on other single-agent RL algorithms with strong performance guarantees, such as recent work by Lattimore, Hutter and Sunehag [Lattimore et al., 2013], but leave this direction for future work.

2.3 Concurrent RL in Identical Environments

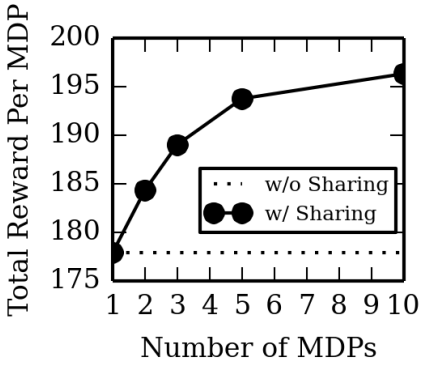
We first consider a decision maker (a.k.a agent) performing concurrent RL across a set of K MDP tasks. The model parameters of the MDP are unknown, but the agent does know that all K tasks are the same MDP. At time step t, each MDP k is in a particular state s_{tk}. The decision maker then specifies an action for each MDP $a_1, \ldots, a_K$. The next state of



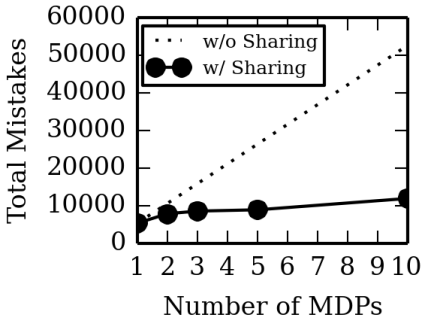
(a) Skinny/filled thick/empty thick arrows yield reward 0.03/0.02/1 with prob 1/1/0.02.



(b) Average reward per MDP per time step for CMBIE, when running in 1, 5, or 10 copies of the same MDP. A sliding window average of 100 steps is used for readability.



(c) Total cumulative reward per MDP after 10000 time steps versus number of MDPs.



(d) The number of total mistakes made after 10000 time steps versus number of MDPs.

Figure 2.1: CMBIE Experiments

each MDP then is generated given the stochastic dynamics model $T(s'|s,a)$ for the MDP and all the MDPs synchronously transition to their next state. This means the actual state (and reward) in each task at each time step will typically differ. [1]. In addition there is no interaction between the tasks: imagine an agent coordinating the repair of many identical-make cars. Then the state of repair in one car does not impact the state of repair of another car.

We are interested in formally analyzing how sharing all information can impact learning speed. At best one might hope to gain a speedup in learning that scales exactly linearly with the number of MDPs K. Unfortunately such a speedup is not possible in all circumstances, due to the possibility of redundant exploration. For example, consider a small MDP where all the MDPs start in the same initial state. One action transitions to a part of the state space with low rewards, and another action to a part with high rewards. It takes a small number of tries of the bad action to learn that it is bad. However in the concurrent setting, if there are many many MDPs, then the bad action will be tried much more than necessary because the rest of the states have not yet been explored. This potential redundant exploration is inherently due to the concurrent, synchronous, online nature of the problem, since the decision maker must assign an action to each MDP at each time step, and can't wait to see the outcomes of some decisions before assigning other actions to other MDPs.

Interestingly, we now show that a trivial extension of the MBIE algorithm is sufficient to achieve a linear improvement in the sample complexity for a very wide range of K, with no complicated mechanism needed to coordinate the exploration across the MDPs. Our concurrent MBIE (CMBIE) algorithm uses the MBIE algorithm in its original form except we share the experience from all K agents. Directed exploration is not necessary for this particular setting.

We now give a high-probability bound on the total sample complexity across all K MDPs. As at each time step the algorithm selects K actions, our sample complexity is a bound on the total number of non-ϵ-optimal actions selected (not just the number of steps).

Theorem 1. *Given ϵ and δ, and K agents acting in identical copies of the same MDP, Concurrent MBIE (CMBIE) will select an ϵ-optimal action for all K agents on all but at most*

$$\tilde{O}\left(\frac{1}{\epsilon(1-\gamma)^2}\left[\frac{S^2A}{\epsilon^2(1-\gamma)^4} + SA(K-1)\right]\right) \tag{2.1}$$

actions, with probability at least $1-\delta$, where $\tilde{O}$ neglects multiplicative log factors.

[1] We suspect it will be feasible to extend to asynchronous situations but for clarity we focus on synchronous execution and leave asynchronous actions for future work.

Theorem 1 shows that there is only a dependence on the number of agents K in the lower order terms of the bound. We now compare this to the alternative where the agents share no information. Using the MBIE bound for a single MDP [Strehl and Littman, 2008], $\tilde{O}\left(\frac{S^2A}{\epsilon^3(1-\gamma)^6}\right)$, no-sharing MBIE yields a sample complexity of $\tilde{O}\left(\frac{S^2AK}{\epsilon^3(1-\gamma)^6}\right)$. Taking the ratio of this with our CMBIE sample complexity bound yields potential improvement factor of $\tilde{O}\left(\frac{SK\frac{1}{\epsilon^2(1-\gamma)^4}}{\left[\frac{S}{\epsilon^2(1-\gamma)^4}+K\right]}\right)$. We now consider how this scales as a function of K. If $K \leq \frac{S}{\epsilon^2(1-\gamma)^4}$, the speedup is approximately K. This suggests that until K becomes very large, we can ignore the lower order terms and gain a linear improvement in the sample complexity as more agents are added. If $K \gg \frac{S}{\epsilon^2(1-\gamma)^4}$, then unfortunately we cannot get much speedup, but this is ultimately unavoidable we always must make at least K mistakes.

Nevertheless, this is quite encouraging, as it implies that by performing concurrent RL across a set of identical MDPs, we expect to get a linear speedup in the sample complexity as a function of the number of concurrent MDP copies/agents compared to not sharing information. Furthermore, the larger the MDP, the higher the threshold is for K for us to obtain a linear speedup.

Proof. Before we proceed we need to define a few variants of the world MDP $M = \langle S, A, T, R, \gamma \rangle$ that the agent is acting in. Let Q^* denote the optimal state-action values of M. We define a state action pair to be known if it has been experienced (across all MDPs) m times. Let M_{K_t} be a known state–action MDP which is identical to the optimistic MDP M_t on all unknown state–action pairs, and identical to the true MDP M's model parameter values for all known state–action pairs K_t. Finally let $\hat{M}_{K_t}$ be the empirical known-state action MDP which is the same as M_{K_t} except the transition and reward models for the known state–action pairs are set to be the maximum likelihood value given the observed counts. Let Q_t/V_t be the optimal (state) action values for the empirical known state–action MDP $\hat{M}_{K_t}$, π_t be the optimal policy for $\hat{M}_{K_t}$ and $V^{\pi_t}_{M_{K_t}}$ be the value of policy π_t in MDP M_{K_t}.

We use Strehl, Li and Littman 2006's generic PAC-MDP theorem to analyze the sample complexity of our approach: as discussed previously [Brunskill and Li, 2013a] while this proof was originally developed for single-task RL, it works with no major modifications for multi-task RL. This theorem requires that our approach satisfies three conditions. If we set

$$m = \Theta\left(\frac{S}{\epsilon^2\left(1-\gamma\right)^4}\log\left(\frac{kSA}{\delta_1}\right)\right)$$

then the first two conditions (optimism and accuracy) hold directly from MBIE, due to

the construction of the set of known states and related MDPs M_t and M_{K_t}. The key contribution is to bound the learning complexity, and thereby satisfy the third condition of Proposition 1 in [Strehl et al., 2006] and show that this yields the resulting sample complexity bound.

First define $T = \frac{1}{1-\gamma} \log\left(\frac{4}{\epsilon(1-\gamma)}\right)$. Then for all policies π and states s, finite horizon expected reward over T steps is $\epsilon/4$-close to the expected infinite horizon reward (as previously proved in Kearns and Singh 2002b):

$$|V^{\pi}_{M_{K_t}}(s,T) - V^{\pi}_{M_{K_t}}| \leq \frac{\epsilon}{4}, \tag{2.2}$$

Let the escape event A_i be the event that in task i we encounter an unknown state–action (unknown in world i) during the next T steps. Also define $V_M^{\mathcal{A}_t}(s_t,T)$ as the expected reward we will obtain in task i over the next T steps when following our concurrent MBIE algorithm. Then

$$V_M^{\mathcal{A}_t}(s_t,T) \geq V^{\pi_t}_{M_{K_t}}(s_t,T) - \Pr(A_i)/(1-\gamma) \tag{2.3}$$

holds in each task i since following our CMBIE algorithm yields the same policy as following π_t in M_{K_t} as long as A_i doesn't occur. Following a similar derivation as in Strehl and Littman 2008 we can prove that for each task i

$$V_M^{\mathcal{A}_t}(s_t) \geq V_M^*(s_t) - \Pr(A_i)/(1-\gamma) - 3\epsilon/4. \tag{2.4}$$

If $\Pr(A_i) \leq \epsilon(1-\gamma)/4$, then $V_M^{\mathcal{A}_t}$ is ϵ-close to the optimal value (obtained by combining the above expression with the optimism and accuracy conditions), indicating that we will follow an ϵ-optimal policy in task i with high probability. We now bound the number of time steps on which $\Pr(A_i) > \epsilon(1-\gamma)/4$.

The number of times an unknown state–action pair can be visited, across all tasks, is at most $m + K - 1$. This situation occurs when in one task we visit a state–action pair $m-1$ times and then in all K tasks we happen to visit the same state and choose the same action (since it will not yet be known). Therefore the total number of escape events is at most $SA(m+K-1)$. Given the lower bounds on the probability of A_i, with probability at least $1-\delta_2$, after $\frac{8}{\epsilon(1-\gamma)}(SA(m+K-1)+\log\frac{1}{\delta_2})$ trials all $SA(m+K-1)$ escape events will have occurred (see Lemma 56 in [Li, 2009a]). As each T-step episode potentially contributes T mistakes, that yields at most $\frac{8T}{\epsilon(1-\gamma)}\left(SA(m+K-1)+\log\frac{1}{\delta_2}\right)$ actions on which the decision maker may make a non-ϵ-optimal decision. This translates into a bound of

$$\tilde{O}\left(\frac{1}{\epsilon(1-\gamma)^2}\left(\frac{S^2A}{\epsilon^2(1-\gamma)^4} + SA(K-1)\right)\right) \tag{2.5}$$

when dropping log terms and substituting in the expressions for m and T. To show high probability, we simply use a union bound for the error probabilities and set $\delta_1 + \delta_2 = \delta$, but they disappear with the log terms. □

2.4 CMBIE Experiments

Prior work by Silver et al. 2013 has already nicely demonstrated the potential empirical benefits of concurrent reinforcement learning for a large customer marketing simulation. Our primary contribution is a theoretical analysis of the potential benefits of concurrent RL for efficient RL. However, sample complexity bounds are known to be quite conservative. We now illustrate that the benefits suggested by our theoretical results can also lead to empirical improvements in a small simulation domain.

We use a 3x3 gridworld (Figure 2.1(a)). The agent moves in the 4 cardinal directions deterministically. Hitting a wall results in no change, except moving down in the bottom-right state will transition to the top-left start state. The arrows display different reward dynamics for those actions. The optimal path to take is along the thick arrows, which will give expected reward of 0.02 per step.

Silver et al. obtained encouraging empirical rewards by using an ϵ-greedy style approach for concurrent RL in identical continuous-state environments, but we found that MBIE performed much better than ϵ-greedy for our discrete state–action MDP, so we focused on MBIE.

Each run was for 10000 time steps and all experiments were averaged over 100 runs. As is standard, we treat the size of the confidence sets over the reward and transition parameter estimates (given the experience so far) as tunable parameters. We tuned the confidence interval parameters to maximize the cumulative reward for acting in a single task, and then used the same settings for all concurrent RL scenarios. We set $m = \infty$, which essentially corresponds to always continuing to improve and refine the parameter estimates (fixing them after a certain number of experiences is important for the theoretical results but empirically it is often best to use all available experience).

Figure 2.1(b) shows that CMBIE achieves a significant increase in how fast it learns the best policy. Figure 2.1(c) also shows a significant gain in total rewards as more sharing is introduced. A more direct measure of sample complexity is to evaluate the number of mistakes (when the agent does not follow an ϵ-optimal policy) made as the agent acts. CMBIE incurs a very small cost from 1-10 agents, significantly better than if there is no sharing of information (Figure 2.1(d)), as indicated by the dotted line. These results

provide preliminary empirical support that concurrent sample efficient RL demonstrates the performance suggested by our theoretical results, and also results in higher rewards in practice.

2.5 Concurrent RL in Different Environments

Up to this point we have assumed that the agent is interacting with K MDPs, and it knows that all of them are identical. We now consider a more general situation where the K MDPs are drawn from a set of N distinct MDPs (with the same state–action space), and the agent does not know in advance how many distinct MDPs there are, nor does it know the identity of each MDP (in other words, it does not know in advance how to partition K into clusters where all MDPs in the same cluster have identical parameters). As an example, consider a computer game being played in parallel by many users. We may cluster the users as being 1 of N different types of how they may overcome the challenges in the game.

2.5.1 Algorithm

We propose a two-phase algorithm (ClusterCMBIE) to tackle this setting:

1. Run PAC-EXPLORE (Algorithm 1) in each MDP.
2. Cluster the MDPs into groups of identical MDPs.
3. Run CMBIE for each cluster.

For the first phase, we introduce a new algorithm, PAC-EXPLORE. The sole goal of PAC-EXPLORE is to explore the state–action pairs in each MDP sufficiently well so that the MDPs can be accurately clustered together with other MDPs that have identical model parameters. It does not try to act optimally, and is similar to executing the explore policy in E^3 [Kearns and Singh, 2002b], though we use confidence intervals to compute the policy which works better in practice. Specifically, PAC-EXPLORE takes input parameters m_e and T, and will visit all state–action pairs in an MDP at least m_e times. PAC-EXPLORE achieves this through directed exploration. It divides the state–action space into those pairs which have been visited at least m_e times (the known pairs) and those that have not. It constructs a MDP $\widehat{M}_K$ in which for known pairs, the reward is 0 and the transition probabilities are the estimated confidence sets (as in MBIE), and for the unknown pairs

the reward is 1 and the transitions are deterministic self loops. This effectively treats the subset of unknown pairs as the goal states. It then constructs an optimistic (with respect to the confidence sets) T-step policy for $\widehat{M}_K$ which will try to reach the unknown pairs. This T-step policy is repeated until an unknown pair is visited, then which the least tried action is taken (balanced wandering), which is then repeated until a known pair is visited, at which point a new optimistic T-step policy is computed and this procedure repeats. The use of episodes was motivated by our theoretical analysis, and has the additional empirical benefit of reducing the computational burden of continuous replanning. Furthermore, the use of directed exploration as opposed to relying on prior algorithms such as MBIE or E^3 is significant, as prior algorithms cannot guarantee that everything is explored within a polynomial number of steps; prior algorithms focus on achieving near-optimality, which may end up not visiting some states at all.

Once phase 1 finishes, we compute confidence intervals over the estimated MDP model parameters for all state–action pairs for all MDPs. For any two MDPs, we place the two in the same cluster if and only if their confidence intervals overlap for all state–action pairs. The clustering algorithm proceeds by comparing the first MDP with all the other MDPs, pooling together the ones that don't differ from the first MDP. This creates the first cluster. This procedure is then repeated until all MDPs are clustered (which may create a cluster of cardinality one). This results in at most $\binom{N}{2} \leq N^2$ checks.

We now show our approach can yield a substantially lower sample complexity compared to not leveraging shared experience.

2.5.2 Assumptions

Our analysis relies on two quite mild assumptions:

1. Any two distinct MDPs must differ in their model parameters for at least one state–action pair by a gap Γ (e.g. the L1 distance between the vector of their parameters for this state–action pair must be at least Γ).

2. All MDPs must have a finite diameter [Jaksch et al., 2010a] (denoted by D).

This is a very flexible setup, and we believe our gap condition is quite weak. If the dynamics between the distinct MDPs had no definite gap, we would incur little loss from treating them as the same MDP. However, in order for our algorithm to provide a benefit over a no sharing approach, Γ must be larger than the $\epsilon(1-\gamma)$ accuracy in the model parameter estimates that is typically required in single task PAC approaches (e.g. Strehl,

Li and Littman [Strehl et al., 2006]). Intuitively this means that it is possible to accurately cluster the MDPs before we have sufficient experience to uncover an ϵ-optimal policy for a MDP. This assumption, while mild in nature, is significant and the key to being able to make any improvements at all. In general, it takes $O(1/\Gamma^2)$ amount of data to detect a gap of size Γ. This means that without this assumption, the gap could possibility be something like $2\epsilon(1-\gamma)$. Since we want to be ϵ-optimal and not 2ϵ-optimal, we would have to try to detect this gap to cluster, in which case we would need $O(1/(\epsilon(1-\gamma))^2)$ data, which means we would already be spending the sample complexity of learning an ϵ-optimal policy. Then we would not be able to get any speedup in sample complexity from clustering. Furthermore, if there is no assumption, then we do not know what the gap is in advance. Then it becomes necessary to expect the worse and thus always spend the samples to try to detect the small gap just in case, even if the true gap is quite large. Unfortunately the PAC framework makes it necessary to consider the worst case gap scenario, which means if we try to change the algorithm to dynamically cluster according to available data, we would still end up with the same PAC bound. But, some interesting future work could be to consider the regret framework which may allow for the possibility of more smoothly and dynamically changing up the clusters as we get more data.

Our second assumption of a finite diameter is to ensure we can explore the MDP without getting stuck in a subset of the state–action space. The diameter of an MDP is the expected number of steps to go between any two states under some policy. With this diameter assumption and the usage of directed exploration, we are able to guarantee that all states can be explored within a polynomial number of steps.

2.5.3 Theory

We first present a few supporting lemma, before providing a bound on how long phase 1 will take using our PAC-EXPLORE algorithm. We then use this to provide a bound on the resulting sample complexity. For the lemmas, let M be an MDP. Let $\widehat{M}$ be a generalized, approximate MDP with the same state–action space and reward function as M but whose transition functions are confidence sets, and each action also includes picking a particular transition function for the state at the current timestep.

Lemma 1. *Generalized Undiscounted Simulation Lemma*

Suppose the transition confidence sets of $\widehat{M}$ are ϵ-approximations to M (i.e. any possible transition function is within ϵ in L1 distance). Then for all T-step policies π, states s, and times $t < T$ we have that $|V_{\pi,t,\widehat{M}}(s) - V_{\pi,t,M}(s)| < \epsilon T^2$ where $V_{\pi,t,M}(s)$ is the expected undiscounted cumulative reward from time t to T when following π in M.

Proof. Let $S_{T-1} = (s_t, s_{t+1}, \ldots, s_{T-1})$ be a $T-t$ length sequence of states from time t to $T-1$. Let S_τ be a prefix up to time τ. Let $\Pr(S_\tau)$ and $\widehat{\Pr}(S_\tau)$ be the probability of these state sequences in M and $\widehat{M}$ under policy π respectively. Let $R(S_\tau)$ be the cumulative expected reward of that subsequence. Then

$$
\begin{aligned}
&|V_{\pi,t,\widehat{M}}(s) - V_{\pi,t,M}(s)| \\
&= |\sum_{S_{T-1}} (\Pr(S_{T-1}) - \widehat{\Pr}(S_{T-1}))R(S_{T-1})| && (2.6)\\
&\leq \sum_{S_{T-1}} |\Pr(S_{T-1}) - \widehat{\Pr}(S_{T-1})|(T-t) && (2.7)
\end{aligned}
$$

since the reward is bounded between 0 and 1. The sum is over all sequences that start with $s_t = s$.

We now show that the probabilities are bounded

$$\sum_{S_{T-1}} |\Pr(S_{T-1}) - \widehat{\Pr}(S_{T-1})| \leq \epsilon T \tag{2.8}$$

Let $P(s'|S_\tau)$ and $\widehat{P}(s'|S_\tau)$ denote the next state probabilities given the current sequence of states and the action chosen by policy π. (So for $\widehat{M}$, since this is after picking the action, the transition function has also been picked). The ϵ-approximation condition implies that $P(s'|S_\tau)$ and $\widehat{P}(s'|S_\tau)$ have L1 distance at most ϵ. For any $\tau < T-1$ it follows that

$$
\begin{aligned}
&\sum_{S_{\tau+1}} |\Pr(S_{\tau+1}) - \widehat{\Pr}(S_{\tau+1})| && (2.9)\\
&= \sum_{S_\tau, s'} |\Pr(S_\tau)P(s'|S_\tau) - \widehat{\Pr}(S_\tau)\widehat{P}(s'|S_\tau)| && (2.10)\\
&\leq \sum_{S_\tau, s'} |\Pr(S_\tau)P(s'|S_\tau) - \widehat{\Pr}(S_\tau)P(s'|S_\tau)| + \\
&\quad |\widehat{\Pr}(S_\tau)P(s'|S_\tau) - \widehat{\Pr}(S_\tau)\widehat{P}(s'|S_\tau)| && (2.11)\\
&= \sum_{S_\tau} |\Pr(S_\tau) - \widehat{\Pr}(S_\tau)| \sum_{s'} P(s'|S_\tau) + \\
&\quad \sum_{S_\tau} \widehat{\Pr}(S_\tau) \sum_{s'} |P(s'|S_\tau) - \widehat{P}(s'|S_\tau)| && (2.12)\\
&\leq \sum_{S_\tau} |\Pr(S_\tau) - \widehat{\Pr}(S_\tau)| + \epsilon && (2.13)
\end{aligned}
$$

and recursing on this gives the final result. This lemma and proof was based on Lemma 8.5.4 from Sham's thesis. □

Lemma 2. *Optimistic Undiscounted Values*

Let $P_T(s,a)$ be the confidence set of transition probabilities for state–action (s,a) in $\widehat{M}$, and assume they contain the true transition probabilities of M. Then performing undiscounted value iteration with the update rule

$$Q_{t,T,\widehat{M}}(s,a) = R(s,a) + \max_{T' \in P_T} \left(\sum_{s'} T'(s'|s,a) V_{t+1,T,\widehat{M}} \right)$$

results in an optimistic q–value function i.e. $Q_{t,T,\widehat{M}}(s,a) \geq Q^*_{t,T,M}(s,a)$ for all state–actions (s,a) and time steps $t \leq T$.

Proof.

$$Q^{OPT}_{T,T,\widehat{M}}(s,a) = R(s,a) = Q^*_{T,T,M}(s,a) \tag{2.14}$$

for the very last step of the episode since they have the same rewards. Then for $t < T$

$$\begin{aligned}
&Q^{OPT}_{t,T,\widehat{M}}(s,a) \\
&= R(s,a) + \max_{T^{OPT} \in P_T} \left(\sum_{s'} T^{OPT}(s'|s,a) V^{OPT}_{t+1,T,\widehat{M}} \right) && (2.15) \\
&\geq R(s,a) + \left(\sum_{s'} T(s'|s,a) V^{OPT}_{t+1,T,\widehat{M}} \right) \\
&\text{where T is the true transitions} && (2.16) \\
&\geq R(s,a) + \left(\sum_{s'} T(s'|s,a) V^*_{t+1,T,M} \right) \text{ by induction} && (2.17) \\
&= Q^*_{t,T,M}(s,a) && (2.18)
\end{aligned}$$

So we can use this value iteration step to compute an optimistic T-step policy, by picking

$$\text{argmax}_a(Q^{OPT}_{t,T,M}(s,a))$$

at time t for state s. □

Theorem 2. *PAC-EXPLORE will visit all state–action pairs at least m_e times in no more than $\tilde{O}(SADm_e)$ steps, with probability at least $1-\delta$, where $m_e = \tilde{\Omega}(SD^2)$.*

Proof. Consider the very start of a new episode. Let K be the set of state–action pairs that have been visited at least $m_e = \Omega(S \log(S/\delta)/\alpha^2))$ times, the known pairs, where α will later be specified such that $m_e = \tilde{\Omega}(SD^2)$. Then the confidence intervals around the model parameter estimates will be small enough such that they are α-approximations (in L1 distance) to the true parameter values (by Lemma 8.5.5 in Kakade 2003). Let M_K be the same as $\widehat{M}_K$ except the transition model for all known state–action pairs are set to their true parameter values. Since the diameter is at most D, that means in the true MDP M, there exists a policy π that takes at most expected D steps to reach an unknown state (escape). By Markov's inequality, we know there is a probability of at least $(c-1)/c$ that π will escape within cD steps and obtain a reward of dD in $(c+d)D$ steps in M_K. Then the expected value of this policy in a $T = (c+d)D$ length episode is at least $\frac{(c-1)dD}{c}$. Now we can compute an optimistic T-step policy $\widehat{\pi}$ in $\widehat{M}_K$. Then $\widehat{\pi}$'s expected value in $\widehat{M}_K$ is also at least $\frac{(c-1)dD}{c}$ (Lemma 2). Applying Lemma 1, $\widehat{\pi}$'s expected value in M_K, which can be expressed as $\sum_{t=1}^{T} \Pr(\text{escapes at } t)(T-t)$, is at least $\frac{(c-1)dD}{c} - \alpha((c+d)D)^2$. Then the probability of escaping at any step in this episode, p_e is at least

$$\begin{aligned} p_e = \sum_{t=1}^{T} \Pr(\text{escapes at } t) &\geq \sum_{t=1}^{T} \Pr(\text{escapes at } t) \frac{(T-t)}{T} \\ &\geq \frac{(c-1)dD}{cT} - \frac{\alpha((c+d)D)^2}{T} \\ &= \frac{(c-1)d}{c(c+d)} - \alpha((c+d)D). \end{aligned}$$

Setting α as $\Theta(1/D)$ results in a function of c and d. For example, picking $c = 2, d = 1, \alpha = 1/(36D)$ results in $p_e = 1/12$, a constant.

Every T-step episode has at least probability p_e of escaping. There are at most SAm_e number of escapes before everything is visited m_e times, so we can bound how many episodes there are until everything is known with high probability. We use Lemma 56 from [Li, 2009a] to yield $O\left(\frac{1}{p_e}(SAm_e + \ln(1/\delta))\right) = \tilde{O}(SAm_e)$ for the number of episodes. The total timesteps is $\tilde{O}\left(SADm_e\right)$. □

Algorithm 1 PAC-EXPLORE

Input: S, A, R, T, m_e,
while some (s, a) hasn't been visited at least m_e times **do**
  Let s be the current state
  if all a have been tried m_e times **then**
    This is the start of a new T-step episode
    Construct $\widehat{M}_K$
    Compute an optimistic T-step policy $\widehat{\pi}$ for $\widehat{M}_K$
    Follow $\widehat{\pi}$ for T steps or until reach an unknown s–a
  else
    execute a that has been tried the least
  end if
end while

We now bound the sample complexity of ClusterCMBIE.

Theorem 3. *Consider N different MDPs, each with K_N copies, for a total of NK_N MDPs. Given ϵ and δ, ClusterCMBIE will select an ϵ-optimal action for all $K = NK_N$ agents on all but at most*

$$\tilde{O}\Bigg(SAm_eNK_N\left(D - \frac{1}{\epsilon(1-\gamma)^2}\right) + \frac{SAN}{\epsilon(1-\gamma)^2}\left(\frac{S}{\epsilon^2(1-\gamma)^4} + (K_N - 1)\right)\Bigg) \tag{2.19}$$

actions, with probability at least $1 - \delta$.

Proof. For phase 1, the idea is to set m_e just large enough so that we can use the confidence intervals to cluster reliably. By our condition, there is at least one state–action pair with a significant gap Γ for any two distinct MDPs, so we just need confidence intervals that are half that size. Using the reward dynamics as an example, an application of Hoeffding's inequality gives $1-\delta'$ confidence intervals when $m_e = \Omega\left(\frac{1}{\Gamma^2}\log\left(\frac{1}{\delta'}\right)\right)$ of that size. Similar confidence intervals can be derived for transition dynamics. Using a union bound over all state–action pairs, we can detect, with high probability, when they are different, and when they are the same. Another union bound over the $O(N^2)$ checks ensures in high probability that we cluster correctly.

Therefore clustering requires $m_e = \tilde{\Theta}(1/\Gamma^2)$. However recall that PAC-EXPLORE requires $m_e = \tilde{\Omega}(SD^2)$ and so we set m_e as $\max\left(\tilde{\Theta}(1/\Gamma^2), \tilde{\Theta}(SD^2)\right)$. Then the total

sample complexity is the sample complexity of PAC-EXPLORE plus the sample complexity of CMBIE.

$$\tilde{O}\left(SADm_eNK_N\right) + \\ \tilde{O}\left(\frac{SAN}{\epsilon(1-\gamma)^2}\left(\frac{S}{\epsilon^2(1-\gamma)^4} + (K_N - 1) - m_eK_N\right)\right) \tag{2.20}$$

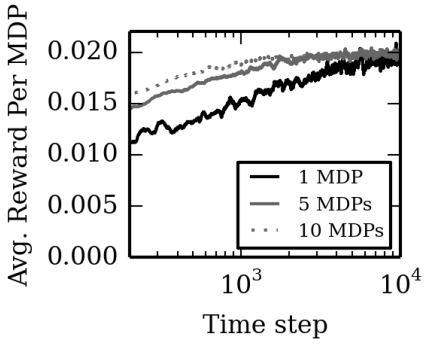
The $-m_eK_N$ term arises above because the samples that are obtained with PAC-EXPLORE are used immediately in CMBIE to contribute to the total samples needed, so CMBIE needs less samples in the second phase. Combining these terms and rearranging yields the bound. □

We now examine the potential improvement in the sample complexity of clustering CMBIE over standalone agents.

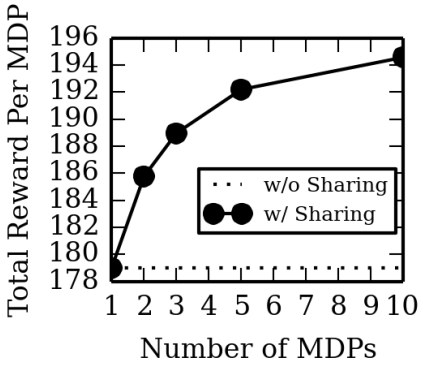
From the CMBIE analysis, we already know if $K_N \leq \frac{S}{\epsilon^2(1-\gamma)^4}$ then we can get a linear speedup for each cluster, resulting in a linear speedup overall if we increase the number of MDPs of each type. The new tradeoff is the overhead in PAC-EXPLORE, specifically in the term $SAm_eNK_N\left(D - \frac{1}{\epsilon(1-\gamma)^2}\right)$, which looks to be a strange tradeoff, but actually what is being compared is the length of an episode. For PAC-EXPLORE, each episode is $\Theta(D)$ steps, whereas for CMBIE, the analysis uses $\Theta(1/(1-\gamma))$ as the episode/horizon length. The additional $1/(\epsilon(1-\gamma))$ comes from the probability of escape for CMBIE, whereas the probability of escape for PAC-EXPLORE is constant. This difference is from the different analyses as CMBIE does not assume anything about the diameter, thus CMBIE is more general than PAC-EXPLORE. We believe analysing CMBIE with a diameter assumption is possible but not obvious. Strictly from the bounds perspective, if D is much smaller than $\frac{1}{\epsilon(1-\gamma)^2}$, then there will not be any overhead. The lack of overhead makes sense as all of the samples in the first exploratory phase are used in the second phase by CMBIE.

However, more significantly, there's an implicit assumption that $K_Nm_e \leq m$ (m is the total number of samples needed for each state–action pair to achieve near-optimality) in the bound. Otherwise if, for example, $m_e = m$, then all the mistakes are made in the PAC-EXPLORE phase with no sharing, so there is no speedup at all. This is where Γ and D matters. Γ must be larger than the $\epsilon(1-\gamma)$ accuracy, as mentioned before, and D must also not be too large, in order to keep $K_Nm_e \leq m$. If $K_Nm_e \leq m$ holds, then after the clustering, all the samples are shared at once. Therefore in many situations we expect the nice outcome that the initial exploration performed for clustering will perform no redundant exploration, and the resulting sample complexity will essentially be the same as CMBIE, where we know in advance which MDPs are identical.

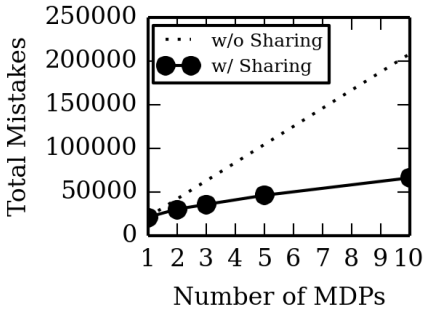
2.6 Clustering CMBIE Experiments



(a) Average reward per MDP per time step for clustering CMBIE, when running in 1, 5, or 10 copies of each MDP type. A sliding window average of 100 steps is used for readability.



(b) Total cumulative reward per MDP after 10000 time steps versus number of MDPs.



(c) The number of total mistakes made after 10000 time steps versus number of MDPs.

Figure 2.2: ClusterCMBIE Experiments

We now perform a simple simulation to examine the performance of our ClusterCMBIE algorithm. We assume there are 4 underlying MDPs: the same grid world domain as in our prior experiments, and rotated and reflected versions. This leads to at least one skinny arrow action being distinct between any two different MDPs, and it means very little exploration is required.

The experiments mirrored the CMBIE experiment parameters where applicable, including using the same parameters for CMBIE (which were optimized for single task MBIE), and fixed for all runs. The PAC-EXPLORE algorithm was optimized with $m_e = 1$ and $T = 4$, and fixed for all runs. With these parameters, the first phase of exploration is quick.

Figure 2.2(a) shows that ClusterCMBIE still achieves a significant increase in how fast it learns the best policy. Figure 2.2(b) also still shows a significant gain in total rewards as more sharing is introduced. In terms of mistakes, again there is a significant improvement with regards to no clustering/sharding, with only a very small overhead in mistakes as the number of MDPs goes up (Figure 2.2(c)). We also saw similar results when we used just 2 distinct, underlying MDPs. These results mirror the results for plain CMBIE when it is known all MDPs are identical, owing to how quickly the exploration phase is. All together these results provide preliminary empirical support for our theoretical results, and their performance in practice.

2.7 Follow-Up Work

Since this work has been published, there has been interesting follow-up work that have extended this work to the continuous state case [Pazis and Parr, 2016]. There has also been work that has shown that more coordination between the agents can improve performance in practice [Dimakopoulou and Van Roy, 2018, Dimakopoulou et al., 2018]. Theoretically, we hypothesize that it may be possible that adding coordination may improve and decrease the dependence on the number of agents in the lower order terms, leading to better practical performance.

Chapter 3

Sample Efficient Transfer Learning in a Finite Set of Continuous MDPs

This work was done jointly with Yao Liu and Emma Brunskill and accepted at AAMAS 2016 [Liu et al., 2016].

3.1 Introduction

While the previous chapter was about many concurrent tasks being done at the same time, this chapter instead tackles the setting when there is a series of tasks one after another. We consider how an agent can leverage prior experience from performing reinforcement learning in order to learn faster in future tasks. We introduce the first, to our knowledge, probably approximately correct (PAC) RL algorithm COMRLI for sequential multi-task learning across a series of continuous-state, discrete-action RL tasks. We assume tasks are sampled from a finite number of clusters of Markov decision processes, and provide a bound on the number of steps on which the algorithm makes a suboptimal decision that is substantially smaller on later tasks. We also provide preliminary evidence to suggest our approach may be useful in practice, by showing encouraging simulation performance in a standard domain where it compares favorably to a state-of-the-art algorithm.

A key feature of an intelligent, autonomous agent is to be able to learn from past experience to improve future performance. In many applications, including robotics, consumer marketing, and healthcare, such an agent will be performing a series of reinforcement learning (RL) tasks modeled as Markov Decision Processes (MDPs) with a continuous state space and a discrete action space. In this chapter, we are interested in formally quan-

tifying the amount of experience needed for the agent to learn to make good decisions while learning from past experience in a series of such tasks.

Our work falls into the broader class of transfer, lifelong and multi-task sequential decision making. There has been significant interest in this area for the past two decades, and novel algorithms have been introduced with promising empirical performance. Transfer learning has typically focused on leveraging experience from a single source task to improve performance on a single target task. Lifelong learning typically refers to doing a series of tasks and improving performance on later tasks. Multi-task learning often refers to having experience from a set of distinct tasks, and using that experience to improve performance on a new target task. This can be done in either a batch setting, or an online setting where the new target task is added to the set of source tasks to accelerate performance on later tasks. This final setting is the one we consider in this chapter. Note that the topic of curriculum learning, where an agent may choose the order in which to complete tasks, is a very interesting question that is outside the scope of our current work: here we assume that the stochastic environment selects the next task to provide to an agent.

Encouraging recent work has shown substantially improved empirical performance on online multi-task continuous state RL [Ammar et al., 2014, Bou-Ammar et al., 2015]. However, there has been very little theoretical analysis of this setup. Exceptions include work by Lazaric et al. [Lazaric and Restelli, 2011] that shows a bound on the estimated value function used by transferring samples from prior tasks, but they consider the batch setting (where one has a set of samples from a target task) and do not handle online exploration/exploitation. Very recent work by Eaton et al. [Ammar et al., 2015] uses a policy search method and provides regret bounds, but the regret bounds are with respect to the best policy in their policy class with additional structural assumptions, rather than the true optimal policy. To our knowledge there has been no work on bounding the amount of experience needed to make good decisions in later continuous-state RL tasks by leveraging prior experience in tasks with the same state and action space, but different dynamics and/or reward models.

Here, we help to fill this gap by introducing a PAC RL algorithm (Continuous-State Online Multitask RL with Identification a.k.a COMRLI) for online multi-task learning across a series of continuous-state, discrete-action RL tasks with the same state and action spaces. We assume that each task is drawn from a stationary distribution over C MDP clusters: though for simplicity most of our analysis will assume that there is a single MDP within each cluster, our results also extend to when each cluster consists of MDPs with very similar transition and reward models. We prove that the number of steps on which the agent may take a non ϵ-optimal action, known as the sample complexity, will be substantially smaller in later tasks, scaling as a function of C rather than the size

(covering number) of the state-action space. Our work builds on recent advances in PAC discrete-state online multi-task RL algorithms [Brunskill and Li, 2013b], single task PAC continuous-state reinforcement learning [Pazis and Parr, 2013], and directed exploration. We also provide encouraging, preliminary empirical performance on a standard domain where our algorithm exceeds a state-of-the-art continuous-state multitask RL algorithm.

Our algorithm proceeds in two stages. In the first stage, our approach learns a good estimate of the optimal Q-function of every given task (in our setting, after a bounded number of tasks, all C tasks will have been encountered with high probability). In the second stage, we leverage this prior information to speed learning in new tasks by using a novel method that includes directed exploration to quickly identify the new task. Intuitively we leverage a mixing assumption on the underlying MDPs to enable us to eliminate possible MDPs by checking that we can reach different states in a bounded number of steps with high probability. This identification process, paired with assuming that tasks are sampled from a stationary (not adversarial) distribution, enables our overall sample complexity to scale as a function of the number of tasks rather than the size of the state-action space.

3.2 Setting

In our continuous multi-task RL setting, we assume tasks are drawn sequentially and iid over a finite set $\mathcal{C}$ of C tasks (we will mention in section 4 how this can be relaxed to C clusters of tasks). Each task is a Markov Decision Process (MDP). We assume all tasks have the same state-action space $S \times A$, discount factor γ, and task length H; but, the parameters (T, R, b_0) can differ. The overall objective of the algorithm is to maximize the value function $V(s)$ for all tasks.

While we acknowledge that our assumptions are not suitable for all situations, we believe our setting captures a general set of important domains such as user modeling where different groups of users may have similar behavior (see examples from [Liu and Koedinger] or [Nikolaidis et al., 2015]).

We prove a high probability, polynomial bound on the sample complexity of our algorithm (PAC bound). The sample complexity is the total number of steps on which the expected value of algorithm is less than ϵ-optimal i.e. formally, the number of steps t where $V(s_t) \leq V^*(s_t) - \epsilon$ for all tasks.

3.3 Algorithm

Algorithm 2 Continuous State Online Multitask RL with Identification (COMRLI)

Require: $T_1, \bar{C}, L_Q, \epsilon, \Gamma, H$
  for $t = 1, 2, \ldots T_1$ **do**
    Receive an unknown MDP $M_t \in \mathcal{M}$
    Run algorithm 3 on M_t with (Γ, D)-*known*.
    For all remaining steps until H steps, execute C-PACE algorithm on M_t
    Store all samples as a set $Sample_{M_t}$
  end for
  Cluster all tasks into $\widehat{C} \leq \bar{C}$ groups and combine their sample sets.
  for $t = T_1 + 1, \ldots T$ **do**
    Receive unknown MDP $M_t \in \mathcal{M}$
    Run algorithm 4 on M_t
    if M_t is identified **then**
      Combine samples from M_t to the group
    end if
  end for

We divide the sequence of tasks into two phases. In phase 1, in each task, we try to explore the whole state space efficiently to learn a good estimate of the optimal Q-function of every task over the entire state space. At the end of phase 1, we cluster the tasks according to the estimated Q-functions. Since tasks are sampled iid from C possible tasks, we can set the length of phase 1 (see Lemma 4) so that, with high probability, all C types are encountered. In phase 2, at the beginning of each task we try to identify which of the C task clusters it belongs to. After identifying it, we run the single task C-PACE algorithm using the all the previous samples collected from all the tasks within that cluster. Phase 2 is used for all subsequent tasks. We will shortly prove that this approach allows us to improve the sample complexity without incurring negative transfer in terms of our theoretical bounds.

3.4 Assumptions and definitions

Before we illustrate details of our algorithm, we need to introduce some assumptions:

1. There exists a distance metric $d[\cdot, \cdot]$ (e.g. Euclidean in our experiments) in which the optimal Q-function $Q_{M_i}(s, a)$ is Lipschitz continuous over (s, a), for any M_i in $\mathcal{C}$.

2. Tasks are sampled from a finite set of C distinct MDPs, and $\bar{C}$ is a known upper

bound on C.

3. The optimal Q function for each of the C MDPs must differ in at least one state-action pair, e.g. for any two MDPs M_i and M_j, there must exist at least one (s, a) pair such that $||Q_i(s, a) - Q_j(s, a)||_2 \geq \Gamma > 0$.

4. There is a finite diameter $D(\epsilon)$: any (s, a) pair's neighborhood (radius at most ϵ) is reachable from any other (s, a) pair in at most an expected $D(\epsilon)$ steps.

The first assumption is a smoothness property over the Q-functions for any MDP. Intuitively, this ensures that close-by state-action pairs can only differ by a bounded amount in their Q-values. This is essential for near optimal learning to be efficient: if any two arbitrarily close state-action pairs can differ an arbitrary amount in their value, then it is necessary to visit all state–action pairs in the space to learn a near-optimal policy; this is an impossible task to do in finite time because the state space is infinite and continuous. Note that the Q-funtion is also Lipschitz continuous over the action space if we use proper a metric (e.g. our experiments). This assumption also implies that one could approximate such an MDP using a finite set of states and actions, because near-by states can only differ a finite amount in their resulting state–action values. This assumption was used and resulted in the tabular representation of the Q-function by a prior paper that introduced a PAC RL algorithm for single task continuous-state MDPs[Pazis and Parr, 2013].

The second assumption describes our multi-task setting. For simplicity, we assume that there are C distinct MDP clusters with only one MDP in each cluster i.e. only C distinct MDPs, and that each task is one of the MDPs. It is straightforward to relax the assumption to the case where each cluster is made up of MDPs with similar dynamics. If we have C clusters of MDPs, where all MDPs in the same cluster have highly similar transition and reward models, then, our results can be extended; our results immediately apply following 5 if MDPs in the i-th cluster (of C clusters), satisfy the following property: for any two MDP M_1, M_2 in cluster i, for any (s, a) pair,

$$|r_{M_1}(s, a) - r_{M_2}(s, a)| < \frac{\epsilon(1-\gamma)}{16}$$
$$\int_{\mathcal{S}} |T_{M_1}(s'|s, a) - T_{M_2}(s'|s, a)| \mathrm{d}s' < \frac{\epsilon(1-\gamma)^2}{16}$$

The third assumption says that we require tasks that are from distinct MDPs to differ in their optimal Q-values in at least one state–action pair. This is quite mild: if two tasks do not differ in their optimal Q-values in any state–action pair, then they have identical Q-values and optimal policies, and are likely to have the same model parameters.

The fourth assumption is perhaps the strongest, and represents a restriction on the mixing property of the MDPs considered. A similar assumption has been employed in

discrete state–action spaces[Auer et al., 2009, Brunskill and Li, 2013b, Guo and Brunskill, 2015a]. Intuitively, it ensures that it is possible to go between two regions of state–action space under some policy in a bounded number of steps. This assumption, coupled with directed exploration, is needed for our algorithm to perform identification of the current task during phase 2. Fortunately there do exist a number of domains which have bounded diameter. For example, many interesting RL domains are episodic. In episodic settings, the diameter can be no more than twice the episode length for any two states that are reachable within one episode from the starting distribution. Also note that in an episodic domain, only the states that are reachable within one episode from the starting state are relevant to the optimal policy and value function.

Our algorithm also depends on the *Q-gap* between distinct optimal Q-functions.

Definition 1. *The Q-gap,* Γ*, is a positive constant that satisfies that for any two MDP clusters*[1] $M_i,\ M_j \in \mathcal{M}$*, there exist some state–action pair* (s, a) *such that*

$$|Q^*_{M_i}(s,a) - Q^*_{M_j}(s,a)| \geq \Gamma$$

.

Note that according to the third assumption, the *Q-gap* must always be greater than 0. We do assume our algorithm has an estimate of a lower bound on Γ. In many real world domains, we don't know Γ in advance; in this case the lower bound can be set to $\epsilon/2$, since if the Q-functions between two tasks are closer than $\epsilon/2$, the ϵ-optimal policy for one is still an ϵ-optimal policy for the other and the two tasks would be most likely almost identical anyway. However in most cases, Γ would be much larger than ϵ.

Some more definitions used in algorithm are clarified here:

Definition 2. *A state-action is known if it has k visited neighbors, which means*

$$L_Q d[(s,a),(s_i,a_i)] \leq \frac{\epsilon(1-\gamma)}{8}$$

, where $k = O(\frac{1}{\epsilon^2(1-\gamma)^2})$ *according to theorem 3.16 in [Pazis and Parr, 2013]. This kind of knownness is called as* ϵ*-known. The knownness in phase 1,* (Γ, D)*-known, is different and will be discussed later.*

Definition 3. *For an MDP* $M = < S, A, T, R, \gamma, H, b_0 >$*, let* K *be the set of known state-action pairs. The known MDP* $M_K = < K \cup sa_{uk}, T, R, \gamma, H, b_0 >$ *is defined as follows.*

[1]Note that different MDP clusters have distinct Q-functions, given our assumption 3 and lemma 5.

sa_{uk} is an additional s-a pair to denote all the unknown s-a pairs. For state-action pairs in K, $r(s,a) = 0$, and for the sa_{uk} state-action, $r(s,a) = 1$. All the unknown state-action pairs are merged into sa_{uk} and it is an absorbing state-action pair. Transitions to unknown pairs in M are redirected to sa_{uk}, and transitions within K are identical to those defined in the original MDP M.

Definition 4. *For a Q-function Q_{M_i}, we use π_{M_i} to denote the greedy policy introduced by Q_{M_i}.*

Algorithm 3 Phase 1: Continuous PAC Explore

Require: T_e, L_Q, Γ

Set the neighborhood radius to $\frac{min\{\Gamma/4,1/24\}}{L_Q}$

while some (s,a) is unknown (see Def.7) **do**

This is a start of new T_e-step episode.

Find a T_e-step undiscounted optimistic Q-function Q_{0,T_e} by:

Initialize: $Q_{T_e,T_e}(s,a) = 0$

for $t = T_e - 1, \ldots, 0$ **do**

if (s,a) is known **then**

Find k-NN of (s,a): $(s_j, a_j, r_j, s_j^{'})_{j=1}^{k}$

$$Q_{t,T_e} = \frac{1}{k}\sum_{j=1}^{k}\left(r_j + \max_a Q_{t+1,T_e}\left(s_j^{'}, a\right) + L_Q d_{ij}\right)$$

else

$Q_{t,T_e} = (T_e - t)$

end if

end for

Take greedy policy of $Q_{0,T_e}(s,a)$ for next T_e steps.

if (s,a) is unknown **then**

Add (s,a,r,s') to the sample set

end if

end while

3.5 Phase 1

In the first phase, we efficiently explore each task to get a good estimate of their optimal Q-functions. We only need the estimated Q-functions to be (Γ, D) known, which means $O(\Gamma)$ (the precise value is set later in the theory section) within the true Q-function, in

order to correctly cluster the tasks. Note that Γ may be much bigger than ϵ so phase 1 may not yet give us an ϵ-optimal policy for the tasks. We extend the PAC-EXPLORE algorithm [Guo and Brunskill, 2015a] to continuous spaces while maintaining the same guarantees, and use it to explore (see Algorithm 3). To achieve efficient exploration, we construct a new MDP called *known MDP* (see definition 3). Then we find a T_e-step undiscounted optimistic Q-function on this MDP through value iteration, where the notation Q_{t,T_e} means the T_e-step optimistic function in the t^{th} step. The policy introduced by this Q-function will direct exploration towards the less explored (unknown) state-action pairs. After the Q-function is (Γ, D) known, we then execute C-PACE for remaining steps of the task. The k nearest neighbors are defined by the distance used in the Lipschitz property. Note that the number of tasks in Phase 1 is defined in advance, in order to ensure all underlying C tasks are experienced with high probability.

3.6 Clustering and Informative State–action Pairs

At the end of each task in phase 1, we solve the fixed-point equations (equation (1) in [Pazis and Parr, 2013]) in C-PACE to estimate the optimal Q-function of that task given the data gathered. Our analysis in the next section shows that the distance between the estimated Q-functions and the true optimal Q-functions for each task is no more than $\frac{\Gamma}{4}$. This allows our algorithm to cluster together all tasks whose true optimal Q-function differ by no more than a fixed threshold (defined below). After the clustering completes, all the samples within a cluster are merged to form a single sample set. The algorithm then runs the fixed-point method from C-PACE to estimate the optimal Q-function for each cluster. Note that depending on the amount of experience in the cluster, this estimated function may or may not be ϵ-optimal yet.

Once the algorithm computes an estimated optimal Q-function for each cluster, these functions are used to compute a set of informative state–action pairs.

Definition 5. *Given a set of Q-functions, a state-action (s, a) is informative if there exist at least two distinct Q-functions, Q_i and Q_j, such that $|Q_i(s, a) - Q_j(s, a)| \geq \frac{7\Gamma}{8}$.*

These informative pairs are the state–action pairs where Q-functions significantly differ, and so distinguish between distinct Q-functions. We will make heavy use of these pairs to identify which cluster new tasks will belong to in phase 2.

3.7 Phase 2

After clustering all the tasks from phase 1, we end up with $\widehat{C}$ clusters, each of which consist of a (merged) sample set and an estimated optimal Q-function. In this phase, we try to identify which cluster each new task belongs to, and once we do we can use that cluster's sample set to jumpstart learning – we may even immediately get a near-optimal policy if we have enough data in the cluster's sample set.

Identifying the current task requires some care since we only know that the current task has an optimal Q-function that matches one of the $\widehat{C}$ clusters. Of course, if we compute an approximate Q-function in the current task of sufficient accuracy, that is enough to identify which cluster it belongs to; however by the time we have enough data to achieve that accuracy in computing the Q-function, we will only get a little benefit to leveraging prior samples. This will only reduce the exploration needed to being a function of Γ instead of ϵ, but it will not impact the dependence on the size of the state–action space covering number.

Instead, we will rely on informative pairs for identification. For each task in phase 2, we start with a set of $\widehat{C}$ candidate clusters that the new task potentially belongs to. Then we can use their associated Q-functions to compute the set of informative pairs I; we will need at most one informative pair for every pair of distinct clusters. We then use directed exploration to repeatedly try to visit informative state-action pairs and compute an estimate of its Q-value for the current task. Each informative pair distinguishes between two clusters' Q-functions. Then we will compare the estimate to the two optimal Q-functions to determine which cluster should be eliminated as a candidate. Then we move onto another informative pair to eventually eliminate another candidate. This process is repeated until there is only one candidate left. More details follow.

Given an informative set of pairs, we define an informative MDP for the current task:

Definition 6. *For an MDP $M = < S \times A, T, R, \gamma >$, let I be the set of informative state-action pairs. The MDP $M_{inform} = < S \times A \setminus I \cup sa_{inform}, T, R, \gamma >$ is defined in a similar way as known MDP in definition 3, replacing the unknown state-action pairs with informative pairs.*

To start, we need to quickly reach an informative pair. The main issue with trying to visit informative pairs quickly is that the identity of the new task is unknown. If we knew the identity, then we could create an M_{inform} based on the associated cluster, and execute the resulting policy. Because of our diameter assumption, this policy will quickly reach the target informative pair. But since we actually don't know the identity of the new task, we need to do something else. What we do is try to use the M_{inform} of every candidate cluster. If we fail to reach the target informative pairs, we just move on and try to use M_{inform} of the next candidate cluster. Since we know that one of candidate cluster is the

true cluster, we will eventually use its M_{inform}, and successfully reach the target pair. This will require trying to use the M_{inform} of each candidate cluster at most once. With this process, first we can create M_{inform} for each candidate with all the informative pairs I to try to reach any informative pair. After that, we can create M_{inform} for each candidate with a single target informative pair to try to return to that particular informative pair.

Next, we need to use the informative pair to try to eliminate candidates. Note that an informative (s,a) pair is only guaranteed to be informative for a particular pair of candidates (the pair may actually be informative for more than two candidates, but for simplicity we only consider two). After we reach an informative state-action pair, for example where Q_{M_i} and Q_{M_j} is sufficiently different, we run T_i trajectories of π_{M_i} and π_{M_j} from here (using the previous process to return to the informative pair after each trajectory) for an estimate of this task's Q-value at this pair. This is to test which one of these two Q-functions is different with the current new task's Q-function. Since Q_{M_i} and Q_{M_j} are sufficiently different, we know at least one of them will differ with the current new task's Q-value. If the estimate of the Q-value of the current task is not close to the Q-value computed from its candidate, this candidate will be eliminated. If both policies being tested have estimates that are close enough to what they are supposed to be, we will eliminate the candidate with the smaller computed Q-value. Once we eliminate one or both of these candidates, we pick another informative pair belonging to two other candidates to test. Every informative pair leads to eliminating at least one candidate, so we will eventually result in only one candidate left i.e. a successful identification.

After a successful identification it may immediately result in an ϵ-optimal policy if there are enough samples from the cluster; otherwise the data of the current task is collected and merged with the cluster's data. Later on, after encountering tasks from the same cluster over and over again there will eventually be enough data for an ϵ-optimal policy. Thereafter, every new task will get an ϵ-optimal policy immediately after identification. Since identification does not depend on the covering number of the state-action space, the sample complexity of new tasks will then also no longer depend on the size (covering number) of the state-action space. This is the source of the sample efficiency.

3.8 Motivation for Using Q-functions

In prior work [Brunskill and Li, 2013b], the difference between model parameters (the transition and reward dynamics) is used to distinguish between different tasks and to identify a new task. However complications arise if we try the same approach here due to a continuous state space. The transition function becomes a probability density and is hard

to represent. We can parameterize it, but that restricts the structure of the transition function. Alternatively, we can use a nonparametric method such as Gaussian Processes (GPs) like in [Grande et al., 2014], however that introduces extra computational complexities, as opposed to using a simple table to maintain a Q-function. Also, using a GP results in slower learning for single tasks than using the Q-function based method C-PACE [Grande et al., 2014]. Our approach of using Q-functions is simpler since the representation is just a table, and this same representation is also used for the policy. The only smoothness assumption is on the Q-function, and not on the model parameters.

Using Q-functions rather than model parameters to cluster also opens a new opportunity. Without assuming that different MDPs have different Q-functions, it may be possible that different MDPs end up having the same Q-function and can be clustered together. However we would no longer be able to identify new tasks quickly. Our identification relies on quickly reaching informative states. Computing these policies to reach informative states requires that the dynamics of the new task resemble the dynamics of the cluster it belongs to. If we allowed different MDPs with the same Q-function to be clustered together, we would no longer be able to guarantee that we can reach an informative state quickly.

3.9 Theoretical guarantees

To start the analysis of the sample complexity, we need some additional assumptions and definitions.

1. Each task has at least $p_{min} > 0$ probability to be drawn in phase 1, and phase 1 has at least $\frac{\ln \bar{C}}{p_{min}\delta}$ tasks.

2. The covering number $\mathcal{N}_{\mathcal{SA}}(L_Q, \epsilon)$ is the size of the largest minimal covering set S_c of the state-action space, which means that for any (s, a) there exist k points in S_c, such that for any one of the k points (s_i, a_i): $L_Q d\left[(s, a), (s_i, a_i)\right] \leq \epsilon(1-\gamma)$

3. Given an input δ, all tasks in phase 1 and 2 should be executed for more than H_{min} steps, where:

$$H_{min} = O\left(max\left\{D^2 \ln\left(\frac{\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)}{\delta}\right), \frac{Q_{max}^2}{\Gamma^2(1-\gamma)^2} \ln\left(\frac{T_1 \mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)}{\delta}\right)\right\} \mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma) D\right)$$

The first assumption makes sure we will encounter all distinct tasks in phase 1, and is a common assumption in previous work [Brunskill and Li, 2013b]. The second assumption is to ensure we can sufficiently explore to create good enough estimates of the Q-functions in phase 1. The main PAC bound for our algorithm is described in the theorem below.

Theorem 4. *For any ϵ and δ, if we run Algorithm 2 for T sequential tasks, each for $H \geq H_{min}$ steps, where H_{min} meets the requirement in the third assumption above. Then the algorithm will select an ϵ-optimal policy on all but at most*

$$O\left(T_1 \max\left\{H_{min}, \min\left\{\zeta_s, H\right\}\right\} + \bar{C}\zeta_s \right.$$
$$\left. + (T - T_1)\frac{Q_{max}^2 \bar{C}}{\Gamma^2} \ln \frac{\bar{C}}{\delta}\left(\bar{C}D \ln \frac{\bar{C}}{\delta} + \log_\gamma \Gamma\right)\right)$$

steps, with probability $1-\delta$. ζ_s is the sample complexity of a single task C-PACE algorithm, which is at most $O\left(\frac{Q_{max}^2}{\epsilon^2(1-\gamma)^2} \ln\left(\frac{1}{\epsilon(1-\gamma)}\right) \ln\left(\frac{\mathcal{N}_{\mathcal{SA}}(L_Q,\epsilon)}{\delta}\right) \mathcal{N}_{\mathcal{SA}}(L_Q, \epsilon)\right)$.

Note that the sample complexity after the first phase no longer depends on the sample complexity of a single task ζ, and only depends on how fast it is to identify. If Γ is much larger than ϵ, then this is a significant decrease in sample complexity for the second phase.

T_1 is the number of tasks in phase 1. Before going into the proofs, we are going to examine the improvement in sample complexity of our algorithm over a single-task algorithm.

First, consider the sample complexity of phase 1, which is the first term of the bound. We want to compare H_{min} and ζ_s. If H_{min} is less than or equal to ζ_s, then the single-task complexity ζ_s dominates and we do no worse than the single task case. In most cases, $T_1 < \mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)$, so H_{min} can be simplified to

$$O\left(max\left\{D^2, \frac{Q_{max}^2}{\Gamma^2(1-\gamma)^2}\right\} \ln\left(\frac{\mathcal{N}_{\mathcal{SA}}(L_Q,\Gamma)}{\delta}\right) D\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)\right)$$

Note that the $\mathcal{N}_{\mathcal{SA}}(L_Q, \epsilon)$ term for C-PACE can be much larger than the $\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)$ term for H_{min} due to Γ being larger than ϵ. If D is at most $O\left(\frac{1}{\epsilon^{\frac{2}{3}}}\right)$ and $\frac{D}{\Gamma^2}$ is at most $O\left(\frac{1}{\epsilon^2}\right)$, then the sample complexity of phase 1 is no more than C-PACE.

Now we compare the total sample complexity of our algorithm with a single-task algorithm. Even in the worst case, the total sample complexity of our algorithm is about

$$\widetilde{O}\left(\left(\bar{C} + T_1 D\right)\zeta_s + (T - T_1)\frac{\bar{C}^2 D Q_{max}^2}{\Gamma^2}\right)$$

Since in most cases $\bar{C} << \mathcal{N}_{\mathcal{SA}}$ and $\bar{C} + T_1 D$ is constant as T increases, assuming again that $\frac{D}{\Gamma^2}$ is at most $O\left(\frac{1}{\epsilon^2}\right)$ we can get a notable improvement over single task algorithms whose sample complexity is linear with respect to $\mathcal{N}_{\mathcal{SA}}$.

Now we start to prove the theorem. We firstly divide the suboptimal steps of our algorithm into 3 parts, and bound each part. The first part consists of steps in phase 1; the second part consists of steps in phase 2 before identification succeeds; and the third part consists of suboptimal steps after identification succeeds. Besides analyzing the sample complexity, we also need to prove the accuracy of clustering and the correctness of identification. Before that, we introduce some supporting lemmas.

3.10 Lemmas

Proposition 1. *(lemma 4.5 in [Kakade et al., 2003]) There are at most $k\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)$ number of visits to unknown state-action pairs in Algorithm 3, where a known state-action pair means it has k visited neighbors within a distance of $\frac{\Gamma(1-\gamma)}{8L_Q}$.*

Proposition 2. *In algorithm 3, denoting exact Bellman operator by B and the upper bound of Q-value by Q_{max}, if*

$$\frac{4Q_{max}^2}{\epsilon^2} \ln\left(\frac{4\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)}{\delta}\right) \leq k \leq \frac{4\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)}{\delta}$$

and the radius of the neighborhood is no more than $\frac{\Gamma}{2L_Q}$, then w.p. $1 - \delta/2$, for all known (s, a) (Known is defined in proposition 1), we have for any $t < T_e$:

$$|Q_{t,T_e}(s, a) - BQ_{t+1,T_e}(s, a)| \leq \Gamma$$

where T_e is the finite horizon length in algorithm 3.

Proof. By proposition 1, there should be at most $k\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)$ unknown samples in algorithm 3. By lemma 3.13 in [Pazis and Parr, 2013], we have that if $\frac{Q_{max}^2}{\Gamma^2} \ln\left(\frac{4\mathcal{N}_{\mathcal{SA}}(L_Q,\Gamma)}{\delta}\right) \leq k \leq \frac{4\mathcal{N}_{\mathcal{SA}}(L_Q,\Gamma)}{\delta}$, then w.p. $1 - \delta/2$ for any known (s, a) and t,

$$-\Gamma/2 \leq \widehat{B}Q_{t,T_e}(s, a) - BQ_{t,T_e}(s, a) \leq \Gamma/2$$

We also have

$$0 \leq \widetilde{B}Q_{t,T_e}(s, a) - \widehat{B}Q_{t,T_e}(s, a) \leq \Gamma/2$$

and $Q_{t,T_e} = \widetilde{B} Q_{t+1,T_e}$ by definition. Then combining them together we get

$$-\Gamma \leq Q_{t,T_e}(s,a) - BQ_{t+1,T_e}(s,a) \leq \Gamma$$

□

Proposition 3. *Assume $R \in [0,1]$. Suppose in Algorithm 3,*

$$\left|Q_{t,T_e}(s,a) - BQ_{t+1,T_e}(s,a)\right| \leq \Gamma$$

, π is a T_e-step greedy policy introduced by Q_{t,T_e}, and Q^{π}_{t,T_e} is the Q value of this policy. Then $\forall$ known (s,a) (Known is defined in proposition 1), t¡T, $\left|Q^{*}_{t,T_e}(s,a) - Q^{\pi}_{t,T_e}(s,a)\right| \leq 2(T_e - t)\Gamma$, and

$$\left|V^{*}_{t,T_e}(s) \leq V^{\pi}_{t,T_e}(s)\right| \leq 2(T_e - t)\Gamma$$

Proof. Firstly we need to clarify some notations: B is the exact Bellman operator. $\hat{B}$ is the approximate Bellman operator which is defined in [Pazis and Parr, 2013]:

$$\hat{B}Q(s,a) = \frac{1}{k}\sum_{i=1}^{k}\left(r_i + \gamma V(s_i^{'})\right)$$

where $(s_i, a_i, r_i, s_i^{'})$ ranges over the k nearest neighbor tuples. $\tilde{B}$ denotes the approximate Bellman operator defined by the right side of equation 1 in [Pazis and Parr, 2013]:

$$\tilde{B}Q(s,a) = \frac{1}{k}\sum_{i=1}^{k}\left(r_i + \gamma V(s_i^{'}) + L_Q d_i\right)$$

where L_Q is the Lipschitz constant and d_i is the distance between (s,a) and (s_i,a_i). Then, we will prove

$$\left|Q^{*}_{t,T_e}(s,a) - Q_{t,T_e}(s,a)\right| \leq (T_e - t)\Gamma$$

for all $0 \leq t \leq T_e$ by induction. For $t = T_e$, $Q^{*}_{t,T_e}(s,a) = Q_{t,T_e}(s,a) = 0$. Then assuming the inequality holds for t+1, we want to prove the result also holds for t:

$$\begin{aligned}
\left|Q^{*}_{t,T_e} - Q_{t,T_e}\right| &\leq \left|BQ^{*}_{t+1,T_e} - BQ_{t+1,T_e}\right| \\
&\quad + \left|BQ_{t+1,T_e} - Q_{t,T_e}\right| \\
&\leq \Bigg|\sum_{s'} P(s'|s,a)\left(max_{a'}Q^{*}_{t+1,T_e}(s',a')\right. \\
&\quad \left. - max_{a'}Q_{t+1,T_e}(s',a')\right)\Bigg| + \Gamma \\
&\leq (T_e - t - 1)\Gamma + \Gamma \\
&= (T_e - t)\Gamma
\end{aligned}$$

The first step follows from triangle inequality and the fact $Q^*_{t,T_e} = BQ^*_{t+1,T_e}$. The second line follows from the assumption in the proposition. The third line follows from the assumption of induction hypothesis. Now we have

$$\left|Q^*_{t,T_e}(s,a) - Q_{t,T_e}(s,a)\right| \leq (T_e - t)\,\Gamma$$

Then we will bound the difference between Q^π_{t,T_e} and Q_{t,T_e} in a similar way. Here we define a new Bellman operator B^π:

$$B^\pi Q_{t,T_e}(s,a) = R(s,a) + \sum_{s' \in \mathcal{S}} P(s'|s,a) Q_{t+1,T_e}(s', \pi(s'))$$

From the definition, we know $Q^\pi_{t,T_e} = B^\pi Q^\pi_{t+1,T_e}$. Since π is the greedy policy over Q, we have $BQ = B^\pi Q$. Replace the B operator above with B^π, then following similar inequalities we have $\left|Q^\pi_{t,T_e}(s,a) - Q_{t,T_e}(s,a)\right| \leq 2\,(T_e - t)\,\Gamma$. Then by triangle inequality we get the result. After we have the bound on Q, the the bound on V immediately follows. □

Before we introduce the following lemmas, we need to introduce a new concept of knownness, (Γ, D)-known.

Definition 7. *A state-action pair is (Γ, D)-known when it has at least*

$$max\left\{D^2 \ln\left(\frac{\mathcal{N}_{\mathcal{SA}}(L_Q,\Gamma)}{\delta}\right),\ \frac{Q^2_{max}}{\Gamma^2(1-\gamma)^2}\ln\left(\frac{T_1\mathcal{N}_{\mathcal{SA}}(L_Q,\Gamma)}{\delta}\right)\right\}$$

visited neighbors within a distance of $\frac{\Gamma(1-\gamma)}{8L_Q}$.

Lemma 3. *After no more than $O\left(\left(k\mathcal{N}_{SA}(L_Q,\Gamma) + \ln\frac{1}{\delta}\right)D\right)$ steps of Algorithm 3, every state-action pair will have at least k visited neighbors, with probability of $1-\delta$, where $k \geq k_{min} = O\left(D^2 \ln\left(\frac{\mathcal{N}_{\mathcal{SA}}(L_Q,\Gamma)}{\delta}\right)\right)$, $\quad k \leq k_{max} = O\left(\frac{\mathcal{N}_{\mathcal{SA}}(L_Q,\Gamma)}{\delta}\right)$.*

Proof. For convenience, we denote a visit to an unknown state-action pair in algorithm 3 as an escape. By the diameter assumption and Markov's inequality, we have that there exists a policy π that will escape within $2D$ steps with a probability of at least 1/2. So such a policy would get a reward of D in $T = 3D$ steps in M_K. So the optimal policy will get at least D reward if we set $T_e = 3D$ steps, which means $V^*_{0,T_e}(s) \geq D$. By applying proposition 3, we get $w.p.\ 1 - \delta/2$, $V^\pi_{0,T_e}(s)$ is at least $D - 2T_e\Gamma$, which could also be expressed as

$$\sum_{t=1}^{T_e} Pr\,(escape\ at\ t)\,(T_e - t)$$

So with probability of $1-\delta/2$ we get the probability of escape in T steps, p_e, could be at least a constant such as $\frac{1}{4}$ by using a Γ smaller than 1/24:

$$\begin{aligned} p_e &= \sum_{t=1}^{T_e} Pr\left(escape\ at\ t\right) \geq \sum_{t=1}^{T_e} Pr\left(escape\ at\ t\right) \frac{T_e - t}{T_e} \\ &\geq \frac{D}{T_e} - 2\Gamma = \frac{1}{3} - 2\Gamma \geq \frac{1}{4} \end{aligned}$$

Every T_e-step episode has at least probability p_e of escaping. Since there are at most $k\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma)$ number of escapes before everything is known, we can bound how many episodes there are until everything is known with high probability $(1-\delta/2)$. Lemma 56 from [Li, 2009b] yields $O\left(k\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma) + \ln\frac{1}{\delta}\right)$ for the number of episodes. Then the total number of time-steps required is $O\left(\left(k\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma) + \ln\frac{1}{\delta}\right) D\right)$. The lower bound of p_e holds with probability $1-\delta/2$, so the whole theorem holds by a union bound with probability $1-\delta$. Note that the Q_{max} in the constraint of k is no more that $T = O\left(D\right)$. $\square$

Lemma 4. *(lemma 1 in [Brunskill and Li, 2013b]) If $T_1 \geq \frac{\ln\frac{\bar{C}}{\delta}}{p_{min}}$ then w.p. $1-\delta$, all distinct MDPs will be encountered in phase 1.*

Lemma 5. *If M_1 and M_2 are 2 MDPs s.t. for any (s,a),*

$$\begin{aligned} |r_{M_1}(s,a) - r_{M_2}(s,a)| &< \tfrac{\epsilon(1-\gamma)}{2} \\ \textstyle\int_{\mathcal{S}} |T_{M_1}(s'|s,a) - T_{M_2}(s'|s,a)| \mathrm{d}s' &< \tfrac{\epsilon(1-\gamma)^2}{2} \end{aligned}$$

then the optimal Q-functions for M_1 and M_2, Q_{M_1} and Q_{M_2}, satisfy that for any (s,a) pair

$$|Q_{M_1}(s,a) - Q_{M_2}(s,a)| < \epsilon$$

Proof. Let B_1 and B_2 denote the Bellman operators of M_1 and M_2, and Q_0 denotes an arbitrary function over $\mathcal{S} \times \mathcal{A}$. To show the result, it is sufficient to prove that $|B_1^i Q_0(s,a) - B_2^i Q_0(s,a)| \leq \sum_{j=0}^{i} \gamma^j \epsilon(1-\gamma)$, and then take the limit as $i \to \infty$. We prove this by induction. The base case is trivial since $Q_0(s,a) = Q_0(s,a)$. Assuming the statement

holds for i, we consider the case of i+1:

$$
\begin{aligned}
& |B_1^{i+1}Q_0(s,a) - B_2^{i+1}Q_0(s,a)| \\
\leq\ & |r_{M_1}(s,a) - r_{M_2}(s,a)| \\
+\ & \gamma \left| \int_{\mathcal{S}} T_{M_1}(s'|s,a) max_{a'} B_1^i Q_0(s,a) \mathrm{d}s' \right. \\
& \left. - \int_{\mathcal{S}} T_{M_2}(s'|s,a) max_{a'} B_2^i Q_0(s,a) \mathrm{d}s' \right| \\
\leq\ & |r_{M_1}(s,a) - r_{M_2}(s,a)| \\
+\ & \gamma \left| \int_{\mathcal{S}} (T_{M_1}(s'|s,a) - T_{M_2}(s'|s,a)) max_{a'} B_1^i Q_0(s,a) \mathrm{d}s' \right. \\
& + \int_{\mathcal{S}} T_{M_2}(s'|s,a) max_{a'} B_1^i Q_0(s,a) \mathrm{d}s' \\
& \left. - \int_{\mathcal{S}} T_{M_2}(s'|s,a) max_{a'} B_2^i Q_0(s,a) \mathrm{d}s' \right| \\
\leq\ & \frac{\epsilon(1-\gamma)}{2} + \gamma \left| Q_{max} \int_{\mathcal{S}} |T_{M_1}(s'|s,a) - T_{M_2}(s'|s,a)| \mathrm{d}s' \right| \\
+\ & \gamma \left| \int_{\mathcal{S}} T_{M_2}(s'|s,a) |max_{a'} B_1^i Q_0(s,a) - max_{a'} B_2^i Q_0(s,a)| \mathrm{d}s' \right| \\
\leq\ & \frac{\epsilon(1-\gamma)}{2} + \gamma \frac{1}{1-\gamma} \times \frac{\epsilon(1-\gamma)^2}{2} + \gamma \sum_{j=0}^{i} \gamma^j \epsilon(1-\gamma) \\
\leq\ & \epsilon(1-\gamma) + \gamma \sum_{j=0}^{i} \gamma^j \epsilon(1-\gamma) \\
=\ & \sum_{j=0}^{i+1} \gamma^j \epsilon(1-\gamma)
\end{aligned}
$$

The first inequality follows from the definition of Bellman operator. The second inequality follows by adding and subtracting the same thing. The third inequality follows by the triangle inequality. The fourth inequality follows from the condition of the lemma and the inductive assumption. □

This lemma allows us to relax the assumption of C distinct tasks to C fuzzy clusters. It shows if the tasks within a cluster are similar enough, the Q functions would be viewed as the same, in the sense of ϵ-accuracy.

Lemma 6. *If all tasks in phase 1 are run for at least H_{min} steps, with probability $1-\delta$, the following holds:*

1. *For all tasks, any state-action pair is (Γ, D)-known.*
2. *Tasks in phase 1 will be clustered correctly w.h.p..*
3. *For any cluster, the max-norm distance between the approximate Q-function and the true optimal Q-function for any task in this cluster is at most $\frac{5\Gamma}{16}$.*

Proof. The first statement could be proved immediately by lemma 3.

Before we prove the second statement, we firstly clarify how to cluster tasks at the end of phase 1. For each task, we find a fixed-point solution Q_{M_i} of $Q = \widetilde{B}Q$, where $\widetilde{B}$ is defined in C-PACE. Then we check all the state-action pairs in a covering set and put two tasks that have $\frac{\Gamma}{2}$-close value on all the pairs into one cluster.

We are going to prove that after clustering like this, different tasks would be clustered into different groups and the same tasks would be put into the same group. We will first prove there must exist an $\frac{3\Gamma}{4}$ difference between Q^* of different tasks on the covering set, and the Q^* of tasks within one underlying cluster is $\frac{\Gamma}{4}$ close. Then we prove the fixed solution Q_{M_i} is a $\frac{\Gamma}{16}$-close approximation of Q^*. These will prove that the distance of approximate Q-functions from a same underlying cluster should be at most $\frac{3\Gamma}{8} = \frac{\Gamma}{4} + 2 * \frac{\Gamma}{16}$. Thus a threshold of $\frac{\Gamma}{2}$ would ensure the correctness of clustering.

By the assumption of a Q-gap Γ and the Lipschitz smoothness, we could say for any two distinct tasks i, j, there exists at least a state-action pair such that the optimal Q-function is different in its neighborhood. By the definition of a covering set, there must exists one point in the covering set in such a neighborhood. Therefore for such a point, the optimal Q function has a gap of at least $\frac{3\Gamma}{4}$.

Note that when we define the underlying MDP cluster, we guarantee that their parameters are within a distance of $\frac{\epsilon(1-\gamma)^2}{16}$, and Γ is at least $\frac{\epsilon}{2}$. Thus by lemma 5, the max-norm distance of optimal Q functions within one true underlying cluster are at most $\frac{\Gamma}{4}$.

Then we prove that $\left|Q_{M_i}(s,a) - Q^*_{M_i}\right| \leq \frac{\Gamma}{16}$ for all (s,a) in the covering set and M_i in the T_1 tasks. We have at least $k = \frac{16^2 Q^2_{max}}{\Gamma^2(1-\gamma)^2} \ln\left(\frac{\mathcal{N}_{\mathcal{SA}} T_1}{\delta}\right)$ neighbors for any point in the covering set S_c. Note the size of the covering set is $O(\mathcal{N}_{\mathcal{SA}}(L_Q, \Gamma))$ and we certainly have T_1 tasks. By Lemma 3.14 in [Pazis and Parr, 2013], the fixed point solution Q satisfies that

$$|Q_{M_i} - BQ_{M_i}| \leq \frac{\Gamma}{16(1-\gamma)}$$

for any M_i. Then using Proposition 4.1 in [Williams and Baird, 1993], we have $|Q^* - Q| \leq \frac{\Gamma}{16}$ for all the T_1 tasks.

Now we have already proved the third statement: The distance of approximate Q functions of any MDP with its optimal Q function is at most $\frac{\Gamma}{16}$, and the distance between optimal Q of MDPs within a cluster is $\frac{\Gamma}{4}$. So the distance between approximate Q-function with the optimal Q-functions for any MDP in the same cluster is $\frac{\Gamma}{4} + \frac{\Gamma}{16} \leq \frac{5\Gamma}{16}$. □

Lemma 7. *Assume every state-action pair is (Γ, D)-known. Then given any start state and desired state-action pair, it is possible to visit the desired state-action pair's neighborhood in no more than $\widetilde{O}(D)$ steps with high probability.*

Proof. Firstly, we construct an MDP M_{inform} such that the desired state-action pairs have unit reward and all others have 0 reward. The desired pair is a self-loop and other transition probabilities are inherited from the true MDP dynamics. We know which point is desired, so we can modify the original samples to become samples in the new MDP M_{inform}. Because now every pair is (Γ, D)-known, so we have $O(D^2)$ samples in every state-action's neighborhood in this M_{inform}. Following a similar analysis of lemma 3, we could find a policy whose probability of reaching the desired region within $3D$ steps is at least $\frac{1}{4}$. Then after $3D \log_{\frac{3}{4}} \delta$ steps the policy could reach the desired region with probability of $1 - \delta$. □

Lemma 8. *When we face an unknown task in phase 2, we could reach any desired state-action pair within $O(\bar{C} D \ln \frac{\bar{C}}{\delta})$ steps with probability $1 - \frac{\delta}{\bar{C}}$.*

Proof. First, consider the case where we know the diameter D. The unknown task must be one of the $\bar{C}$ tasks from phase 1. Our algorithm tries to run each policy in $3D \ln \frac{\bar{C}}{\delta}$ steps to the desired state-action pair using the samples from one of the $\bar{C}$ tasks, thus lemma 8 holds with probability $1 - \frac{\bar{C}}{\delta}$. By trying all policies from the $\bar{C}$ tasks, we will encounter the one policy that corresponds to the same task and reach the desired region with high probability.

If we don't know the diameter, we could use the doubling trick to find an upper bound on D without an increase in the sample complexity. First we try the whole process with $\widetilde{D} = 1$. If we fail, we double the $\widetilde{D}$ and begin a new trial. When $\widetilde{D}$ is bigger than the true value of D, the rest of the analysis is the same as when we know the true diameter. □

Lemma 9. *If T_i in algorithm 4 is at least $O\left(\frac{Q_{max}^2}{\Gamma^2} \ln \frac{\bar{C}}{\delta}\right)$ and n is at least $O(\log_\gamma \Gamma)$, we could compute an approximate Q value of policy π over current task M': $\widehat{Q}_{M'}^{\pi_i}(s,a) = R_i$ such that for any (s,a), $\left|\widehat{Q}_{M'}^{\pi_i}(s,a) - Q_{M'}^{\pi_i}(s,a)\right| \leq \frac{\Gamma}{16}$ with probability $1 - \frac{\delta}{\bar{C}}$.*

Proof. By setting n to at least $\log_\gamma \frac{\Gamma(1-\gamma)}{32}$ we have that for each episode t,

$$\|R_{it} - \sum_{l=0}^{\infty} \gamma^l r_l\| \leq \frac{\Gamma}{32}$$

. Note that the expectation of $\sum_{l=0}^{\infty} \gamma^l r_l$ is $Q^{\pi_i}_{M'}(s,a)$. We bound the error as follows:

$$\begin{aligned}
& P\left(\left|\widehat{Q}^{\pi_i}_{M'}(s,a) - Q^{\pi_i}_{M'}(s,a)\right| \geq \frac{\Gamma}{16}\right) \\
= \quad & P\left(\left|\frac{1}{T_i}\sum_{t=1}^{T_i} R_i t - Q^{\pi_i}_{M'}(s,a)\right| \geq \frac{\Gamma}{16}\right) \\
\leq \quad & P\left(\left|\frac{1}{T_i}\sum_{t=1}^{T_i}\left(\sum_{l=0}^{\infty}\gamma^l r_l\right) - Q^{\pi_i}_{M'}(s,a)\right| \geq \frac{\Gamma}{32}\right) \\
\leq \quad & 2exp\left\{-\frac{2T_i\Gamma^2}{32^2 Q^2_{max}}\right\}
\end{aligned}$$

The first step follows from the definition of $\widehat{Q}^{\pi_i}_{M_i}(s,a)$. The second step follows from the fact $\|R_{it} - \sum_{l=0}^{\infty}\gamma^l r_l\| \leq \frac{\Gamma}{32}$. The third step follows from the Hoeffding inequality. Setting the probability above to $\frac{\delta}{C}$ and solving for T_i, we have that

$$T_i = O\left(\frac{Q^2_{max}}{\Gamma^2}\ln\frac{\bar{C}}{\delta}\right)$$

is sufficient to guarantee our desired result. □

Lemma 10. *If both model M_i and M_j haven't been eliminated by $\left|R_g - Q_{M_g}(s,a)\right| \geq \frac{\Gamma}{8}$ in algorithm 4, then the true model would have a greater R_g with high probability.*

Proof. Without loss of generality, we assume M_i is in the same underlying cluster with current task M'. Then:

$$\begin{aligned}
& \left|\widehat{Q}^{\pi_i}_{M_i}(s,a) - Q^*_{M'}(s,a)\right| \\
\leq \quad & \left|\widehat{Q}^{\pi_i}_{M_i}(s,a) - Q^{\pi_i}_{M_i}(s,a)\right| + \left|Q^{\pi_i}_{M_i}(s,a) - Q_{M_i}(s,a)\right| \\
& + \left|Q_{M_i}(s,a) - Q^*_{M'}(s,a)\right| \\
\leq \quad & \frac{\Gamma}{16} + \frac{\Gamma}{16} + \frac{5\Gamma}{16} \\
= \quad & \frac{7\Gamma}{16}
\end{aligned}$$

The first step follows from triangle inequality. In the second step, the first replacement follows from Lemma 9, the second replacement follows from proposition 4.1 in [Williams and Baird, 1993], and the third replacement follows from Lemma 6. Because M_j also hasn't been eliminated:

$$\begin{aligned} & \left|Q^{\pi_j}_{M'}(s,a) - Q_{M_j}(s,a)\right| \\ \leq \quad & \left|Q^{\pi_j}_{M'}(s,a) - \widehat{Q}^{\pi_j}_{M'}(s,a)\right| + \left|\widehat{Q}^{\pi_j}_{M'}(s,a) - Q_{M_j}(s,a)\right| \\ \leq \quad & \frac{\Gamma}{16} + \frac{\Gamma}{8} \\ = \quad & \frac{3\Gamma}{16} \end{aligned}$$

The first inequality is from the triangle inequality. The second inequality follows from Lemma 9 (for the first erm) and the elimination condition at line 16 in Algorithm 4 (for the second term). We know there is a gap in the Q-function between M_i and M_j because (s,a) is an informative state-action pair: $\left|Q_{M_i}(s,a) - Q_{M_j}(s,a)\right| \geq \frac{7\Gamma}{8}$. Then $Q_{M_j}(s,a)$ must be smaller than $Q_{M_i}(s,a)$. Otherwise it implies that $Q^{\pi_j}_{M_i}(s,a)$ is larger than $Q^*_{M'}(s,a)$. However that is impossible because $Q^*_{M'}$ is the optimal policy's Q value. Therefore,

$$\begin{aligned} & \widehat{Q}^{\pi_i}_{M'}(s,a) - \widehat{Q}^{\pi_j}_{M'}(s,a) \\ = \quad & (\widehat{Q}^{\pi_i}_{M'}(s,a) - Q_{M_i}(s,a)) + (Q_{M_j}(s,a) - \widehat{Q}^{\pi_j}_{M'}(s,a)) \\ + \quad & (Q_{M_i}(s,a) - Q_{M_j}(s,a)) \\ \geq \quad & -\frac{\Gamma}{8} - \frac{\Gamma}{8} + \frac{7\Gamma}{8} \\ \geq \quad & \frac{5\Gamma}{8} \end{aligned}$$

The first inequality's first two terms follows from the elimination condition at line 16 in Algorithm 4. The third term follows because (s,a) is an informative pair and the fact that we just showed that $Q_{M_i}(s,a) > Q_{M_j}(s,a)$, so $Q_{M_i}(s,a) - Q_{M_j}(s,a) \geq \frac{7\Gamma}{8}$. Now we have shown that the approximate Q value of the true model's policy must be larger than the other. Thus when we eliminate a candidate with the smaller Q-value, we will not eliminate the true candidate of the current task. □

Lemma 11. *After* $O\left(\frac{Q^2_{max}}{\Gamma^2}\bar{C}\ln\frac{\bar{C}}{\delta}\left(\bar{C}D\ln\frac{\bar{C}}{\delta} + \log_\gamma \Gamma\right)\right)$ *steps in phase 2, we could correctly identify the new task w.p.* $1-\delta$.

Proof. From Lemma 9, $O\left(\frac{Q^2_{max}}{\Gamma^2}\bar{C}\ln\frac{\bar{C}}{\delta}\log_\gamma \Gamma\right)$ total steps (across multiple trajectories) is sufficient to closely estimate $Q^{\pi_i}_{M'}(s,a)$ for each $i \in \mathcal{C}$ with probability at least $1-\frac{\delta}{2\bar{C}}$. Then

by Lemma 10, we eliminate at least one candidate model per one informative state-action pair. However, each possible trajectory must start at the same informative state–action pair.

From Lemma 8 $O(\bar{C}D)$ steps are sufficient to return to a desired informative state–action pair with high probability. Therefore the total number of steps required to identify the current task is bounded by

$$O\left(\frac{Q_{max}^2}{\Gamma^2}\bar{C}\ln\frac{\bar{C}}{\delta}\left(D\ln\frac{\bar{C}}{\delta}+\log_\gamma\Gamma\right)\right)$$

where we have applied the union bound to ensure the final bound holds with probability at least is $1-\delta$. □

3.11 Proof of Main Theorem

Proof. Recall that we divide the sample complexity into 3 parts. We will show the bound of each part to prove the whole sample complexity.

The first part consists of the steps in phase 1. H_{min} is the lower bound of steps we need to run algorithm 3 in phase 1. After we finish the exploration, we can run C-PACE for remaining steps of the episode $(H - H_{min})$. Note that the unknown state-action pairs of algorithm 3 is actually a subset of the unknown state-action pairs in ϵ-known C-PACE (since $\Gamma \geq \epsilon$ so more states are unknown under the threshold of ϵ-known). So we could view algorithm 3 as part of the initial exploration done in C-PACE. The total sample complexity in phase 1 would be no more than $max\left\{H_{min}, min(\zeta_s, H)\right\}$.

The second part consists of the identification during phase 2. Following from lemma 11 we need

$$O\left((T-T_1)\frac{Q_{max}^2\bar{C}}{\Gamma^2}\ln\frac{\bar{C}}{\delta}\left(\bar{C}D\ln\frac{\bar{C}}{\delta}+\log_\gamma\Gamma\right)\right)$$

samples in $T - T_1$ tasks in phase 2.

The third part is after the identification in phase 2. If the cluster has gathered enough samples we will get an ϵ-optimal policy immediately. If not, we still need to gather more. But since the samples are gathered across all the tasks in one cluster and the number of clusters is at most $\bar{C}$, this only yields an additional sample complexity of $\bar{C}\zeta_s$. Note that to run the C-PACE algorithm, we set all $\epsilon_s, \epsilon_T, \epsilon_d, \epsilon_K$ in C-PACE to $\frac{\epsilon(1-\gamma)}{8}$ so that we could get an ϵ-optimal policy. □

3.12 Experiments

Though our primary contribution is to prove online multi-task RL enables a lower sample complexity in later continuous-state tasks, we also tested its empirical performance on a popular simulated domain, the spring mass damper system.

This domain is characterized by 3 parameters: the spring constant k, the damping constant d and the mass m. The ranges are: $k \in [1, 10]$ $d \in [0.01, 0.2]$ $m \in [0.5, 5]$, which are mirrored from [Ammar et al., 2014]. The state is parameterized by the position and velocity of the mass, and the action is a horizontal force on the mass. Due to the complexity of performing a maximization over a continuous set of actions, we considered a discrete action set of $\{-10, -1, -0.1, 0, 0.1, 1, 10\}$. [2] The transition dynamics are characterized by an ODE system and simulated by the Euler method in our experiment.

The goal is to control the mass starting at $(1, 0)$ to stay in state $(0, 0)$, and the reward is the negative $l2$-norm distance between current state and the goal.

We performed simulation rounds, where each round consists of 50 tasks. Tasks are generated by randomly sampling from three distinct spring-mass MDPs (k,d,m). Each task consists of 100 episodes with each episode having 150 steps: these settings were informed by recent work[Ammar et al., 2014]. We repeat this process for 40 rounds to assess average performance across a series of tasks.

We compare against 3 other baselines. The first is single-task C-PACE: the PAC RL algorithm C-PACE is run for each task independently, yielding no information sharing across tasks. In the second baseline, C-PACE (mixture of all MDPs), is executed across all tasks without distinguishing them. This is a naive form of transfer that treats all tasks as identical. Finally, our third baseline is the state-of-the-art online policy gradient method for multi-task RL, PG-ELLA[Ammar et al., 2014]. PG-ELLA learns a shared basis set across different tasks. Unlike our work, PG-ELLA assumes as input the true identification of each task. PG-ELLA runs 5000 episodes rather than 100 for each task. That's because they need more samples to update policy parameters.

The considered algorithms all have input parameters. For C-PACE (and the parts of our algorithm that use C-PACE), we let $k = 1$ since this domain is deterministic. The Lipschitz constant is set to 0.1, and Q_{max} is set to zero. For our algorithm, we set the number of tasks in phase 1 to 15 according to Lemma 4 with uniform task sampling and $\delta = 0.02$. The Q-gap Γ is set to 2. The number of trajectories we run in phase 2 to approximate the

[2] For computational efficiency and to focus on the key impact of transferring knowledge from multiple tasks, we chose this discrete action space. Alternatively we could use the Lipschitz constant to define a discretization.

Q value, T_i, is set to 1 because the domain is deterministic. For PG-ELLA, the learning rate is selected automatically in their code.

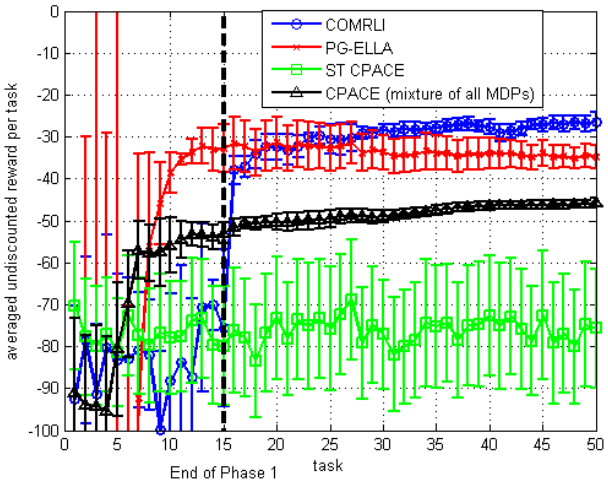


Figure 3.1: Averaged reward per task. The bar is standard deviation over 40 rounds.

Figure 3.1 shows the average reward for each of the 50 tasks for the 4 algorithms. As expected, our approach performs comparably to single task C-PACE in phase 1, with a slight loss in performance because we try to fully explore the state–action space. However our algorithm significantly improves in phase 2, successfully accelerating learning by identifying tasks quickly. In phase 2, the gap between our method and single-task C-PACE implies the superiority of transferring samples from previous tasks, which also confirms the analysis of sample complexity in an empirical way. For single task C-PACE could not collect enough samples to converge in just one task, its standard deviation is quite big as, as it was shown in the figure. Our algorithm is also better than the C-PACE algorithm that treats all tasks the same, which shows the benefit of learning a separate optimal policy per distinct task instead of a single policy for all tasks.

During phase 2 our algorithm exceeds the performance of PG-ELLA. Note that PG-ELLA uses the first several tasks to learn a shared basis, which we show in the figure, but is omitted in the graphs of their paper. We consider this quite encouraging, since PG-ELLA is provided the identity of each task, compared to our approach which does not have that information. Of course, our algorithm is operating in a domain in which tasks have Q functions that are quite well separated, as the Q-gap assumption.

These preliminary results suggest that our approach can perform well in standard benchmark simulation, with a significant advantage over single task algorithms and naive

transfer, and equivalent or slightly improved performance over a state-of-the-art method, PG-ELLA. As our approach has rigorous guarantees on sample complexity, which PG-ELLA lacks, these empirical results are quite encouraging.

Algorithm 4 Phase 2: Continuous Identify

Require: $\Gamma, \epsilon, \bar{C}, H, T_i, n$
  Initialize version space: $\mathcal{C} \leftarrow \{1, \ldots, \bar{C}\}$
  while $h < H$ **do**
    for $c \in \mathcal{C}$ **do**
      Use algorithm 3 with samples from M_c to find a informative pair (s, a) s.t. $\left|Q^{\pi_i}_{M_i}(s,a) - Q^{\pi_j}_{M_j}(s,a)\right| \geq \frac{7\Gamma}{8}$
      if informative pair (s, a) is reached **then**
        Break the loop.
      end if
    end for
    for $g = \{i, j\}$ **do**
      for $t = 1 \ldots T_i$ (Monte Carlo estimate) **do**
        Run the greedy policy of Q_{M_g}, π_g, for n steps
        $R_{gt} \leftarrow r(s,a) + \sum_{l=1}^{n} \gamma^l r_l, h \leftarrow h + n$
        $\widetilde{D} \leftarrow 1$
        while Haven't returned to (s, a) **do**
          for $c \in \mathcal{C}$ **do**
            Use M_c to create an informative MDP, and try to go back to (s, a) within $12\widetilde{D} \ln \frac{\bar{C}}{\delta}$ steps.
            if get back to (s, a) **then**
              Break the loop.
            end if
          end for
          $\widetilde{D} \leftarrow 2\widetilde{D}$
        end while
      end for
      $R_g \leftarrow \frac{1}{T_i} \sum_{t=1}^{T_i} R_{gt}$
      if $\left|R_g - Q_{M_g}(s,a)\right| \geq \frac{\Gamma}{8}$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \backslash \{g\}$
      end if
    end for
    if Both i,j haven't been eliminated **then**
      Eliminate the model k with a smaller R_g.
    end if
    if Only one group left **then**
      Combine the sample sets and run C-PACE
    end if
  end while

Chapter 4

Sample Efficient Feature Selection for Factored Markov Decision Processes

This work was done jointly with Emma Brunskill [Guo and Brunskill, 2018].

4.1 Introduction

Previous chapters have looked at the struture of how many tasks are tackled in the real world, whether it be concurrently or sequentially. This chapter takes a closer look at how a single task is better represented and how we can be more sample efficient by optimizing the representation itself.

In reinforcement learning, state is often represented by feature vectors. Prior sample complexity bounds scale with the complete set of features and their complexity. However, not all features may be necessary for learning a good policy. Therefore it is of significant interest to understand if the sample complexity can scale with the complexity of necessary features instead of all features. We answer this in the affirmative for at least one important case of interest: factored Markov Decision Processes. We show that is possible to eliminate unnecessary features by using directed exploration and leveraging the negative information from failing to reach desired states. Under mild assumptions, this is sufficient to show there exists an RL algorithm whose sample complexity scales with the cardinality of the parent sets of the necessary features, rather than the parent sets of all features. This yields an exponential improvement in sample complexity bounds when the maximum cardinality of the parent sets of the necessary features is smaller than for all features.

In many machine learning and AI control problems, choosing which features to represent the state of the domain is critical. Since the best representation is typically unknown, it is appealing to start with raw sensory input (like the pixels in a video game snapshot) or all possible features that might be relevant. Recent work in deep reinforcement learning [Mnih et al., 2013] has shown that it is possible to obtain great performance in some domains by using such representations; however the resulting methods typically require an enormous amount of training examples. Although in some simulated environments this is not a critical limitation, in many high stakes domains (such as customer marketing, healthcare, education, and robotics) sample efficiency is very important. While in some cases deep learning methods may be sample efficient, very little is known formally about where and when that would occur. Indeed in prior theoretical analysis, the samples needed typically scales exponentially with the cardinality of the parent sets of the features [Strehl et al., 2007, Diuk et al., 2009, Chakraborty and Stone, 2011]. Hence, in many such RL settings, good performance relies on using a small set of carefully hand-designed features. This process can be expensive, requiring domain experts to select the features, and may easily miss relevant features resulting in sub-optimal performance. Ideally, a reinforcement learning algorithm would have its sample complexity depend only on the relevant features needed to learn the optimal policy, and not on all features; then there would be no need to hand-design features and instead this algorithm would automatically find the best subset of features. This effort can be viewed as part of a recent interest in democratizing artificial intelligence: the importance of making it easier for domain experts (rather than machine learning experts) to leverage machine learning and AI for scientific discovery and other progress.

In this chapter we present theory that takes a step towards this goal, showing it is possible for an online RL algorithm, in the factored Markov Decision Processes setting, to efficiently achieve near-optimal average performance with sample complexity that only depends on the complexity of the necessary features, even though which features are necessary is unknown.

Factored Markov Decision Processes (FMDPs) use feature vectors to represent states, enabling a compact encoding of real world domains. The sample complexity (the number of steps on which the algorithm may make non-near-optimal decisions) of RL algorithms for tabular MDPs scales at least linearly with the size of the state space [Strehl et al., 2009], which is exponential in the number of features if applied to FMDPs. Fortunately in an FMDP, the dynamics of each feature can depend on a small parent set of other features, so the sample complexity scales exponentially only with the size of the largest parent set (known as the in-degree) [Kearns and Koller, 1999].

However, there exist many domains where some features' dynamics may be both com-

plex to model, and irrelevant to the reward and (any) value function for the domain. For example, when making tea, modeling the sky beyond the window may involve weather predictions, or knowledge of the date and time, but the value function of any policy for making tea may only rely on a sensor for water temperature. Video games often dedicate many pixels in order to show a complete scene, but most of the time only a small subset of the objects actually influence the value of chosen policies, and their dynamics may be much simpler than some other distracting elements. In such domains, the set of features necessary to learn the optimal value function may have a much smaller parent set compared to all descriptive environment features. If we could know these features in advance, it may be possible to substantially speed reinforcement learning.

We prove that it is possible to do Probably Approximately Correct RL where the sample complexity scales as an exponential function of only the in-degree (number of parents) of the (initially unknown) necessary features. Our result is an exponential improvement over prior FMDL PAC RL algorithms [Chakraborty and Stone, 2011] that do no feature selection, if the in-degree of the necessary features is smaller than the in-degree of the full feature set. We prove this result by presenting an algorithm that achieves this bound with no knowledge of which nor how many features are necessary, nor any knowledge of the structure of the underlying Dynamics Bayesian network of the factored MDP.

Initially this may seem an impossible problem: without knowledge of which features are needed or even how many they are, nor a bound on how complex the dynamics models are (e.g. in terms of a bound on the number of possible parents), how can we hope to learn a near optimal policy in a number of samples that scales with this necessary but unknown set of features? Fortunately under some mild assumptions, we can show a positive result.

Our key insight is to leverage negative information to identify when our estimated model of the MDP must be wrong, and to be able to identify at least one aspect of the model that is wrong. This idea allows us to make progress, eventually eliminating features for which we have poor models when computing our optimal policy. We are not aware of prior work that uses negative information during reinforcement learning, and we believe that this insight may have practical benefits for new algorithmic developments in future work.

More precisely, a key part of our algorithm is to use directed exploration: the algorithm repeatedly tries to visit specific states. Prior work [Guo and Brunskill, 2015a, Liu et al., 2016] has shown that directed exploration can yield PAC RL algorithms that are competitive with or in some cases improve upon prior PAC results. In this work we take a different stance: under the mild assumption that the domain has a finite diameter (which has also been assumed in multiple other theoretical RL approaches [Jaksch et al., 2010b, Bartlett and Tewari, 2009]), failure to reach a state implies that we must have a poor dynamics

model of at least one feature. In order to identify at least one feature that is not adequately modeled, we make another mild assumption, which we call the Superset Assumption. It ensures we can detect if adding additional true parents yields a better model for a particular feature, even if we do not consider the full set of that feature's true parents. While it is possible to construct pathological models or single algorithm runs which could fail these assumptions, we believe that in almost all practical cases of interest both will be satisfied.

Directed exploration combined with these assumptions allows us to fully utilize relevant information in a sample efficient way for learning dynamics and eliminating unnecessary features. Our approach builds on work for RL in factored MDPs that does not require knowledge of the in-degree of an FMDP [Chakraborty and Stone, 2011] but goes significantly beyond this to tackle the feature selection problem during online learning. Our key contribution is to show the significant improvement in sample complexity that can be obtained even if the necessary features are unknown: our algorithm can be viewed as a tool in support of this analysis, and is not designed to be practical. We hope that some of our insights, in particular the idea of leveraging negative information, may have potential benefit for the future design of empirically-oriented algorithms.

4.2 Setting

A finite FMDP is defined by a tuple (S, A, P, R), where S is a finite set of states, A is a finite set of actions, P is the transition distribution and R is the reward distribution. Each state s is a feature vector $(x_1, x_2, \ldots, x_n)$ where $x_i \in Dom_i$ and $|Dom_i| = d$ [Kearns and Koller, 1999]. The transition distribution factors over the state space i.e. $P(s_{t+1}|s_t, a_t) = \prod_i P(x_{i,t+1}|s_t, a_t) = \prod_i P_i(x_{i,t+1}|Par_i(s_t), a_t)$. Each P_i is the transition probability for feature x_i, dependent on its parent set of features Par_i. The notation $Par_i(s_t)$ denotes filtering the feature vector s_t to only the features present in the set Par_i. The reward is defined as $R(s, a) = \sum_j^{|R|} R_j(s, a)$, where each R_j is an individual reward function. Like transitions, each reward function R_j is also dependent on a parent set of features Par_j. $P(R_j|Par_j(s), a)$ is also a discrete distribution with a domain of size d just like a feature. Since features and rewards both utilize the same discrete representation, the same approach can be used to learn feature transition dynamics and rewards.

The in-degree of a factored MDP is the size of the largest parent set over all features/rewards i.e. $\max_i |Par_i|$. Let F' denote the set of all features. Given a factored MDP, assume there exists a set of necessary features F, such that the parents of features in F are in F, and the parents of the rewards are also in F. This implies that a factored MDP defined only over the set F has an identical value function to the value function in the orig-

inal factored MDP for any policy, where the original FMDP's value function is constant over the unnecessary features (Lemma 12). As in some prior work, we also assume the FMDP has a finite diameter D [Auer and Ortner, 2007]. A diameter D means that for any two states s_1, s_2, the expected number of steps to go from s_1 to s_2 is at most D.

In this setting, the problem is to interact with a factored MDP where the transition and reward dynamics are unknown (i.e. parent sets are unknown), the in-degree is unknown, and the necessary features are unknown, and execute a policy whose performance is ϵ-close to the best possible policy. Interaction proceeds in steps, where in each step an algorithm takes an action, and observes the (stochastic) next state and reward. We measure performance of a policy π using the average reward notion, where $U^\pi(s) = \lim_{T\to\infty} U^\pi(s,T) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}(\sum_{t=1}^{T} r_t | s_1 = s, \pi)$ [Kearns and Singh, 2002a]. Like prior factored MDP work, we also assume $U^\pi(s)$ is independent of s and can be denoted as just U^π [Chakraborty and Stone, 2011]. Since we are working with finite samples, we assume the ϵ-return mixing time T_ϵ is given, same as in prior work [Chakraborty and Stone, 2011]. T_ϵ is such that for any policy π and $T' \geq T_\epsilon$, $|U^\pi - U^\pi(T')| \leq \epsilon$ i.e. T_ϵ is long enough to see ϵ-optimal average reward.

4.3 Related Work

Prior work on factored MDP RL with formal theoretical bounds include Met-RMax [Diuk et al., 2009] and LSE-RMax [Chakraborty and Stone, 2011]; however, such work does not perform feature selection. More recent work has significantly reduced the sample complexity of learning factored MDPs [Hallak et al., 2015], but requires strong structural assumptions and only apply to the batch setting, which does not account for the trade-off between exploration and exploitation.

Prior work for feature selection for factored MDPs performs it as a post-processing step after solving it and uses the learned features for transfer learning [Kroon and Whiteson, 2009]. In contrast, our algorithm learns the necessary features while doing online reinforcement learning, and is more sample efficient as there is no need to fully learn the dynamics for all features. We also provide a formal theoretical analysis which is the first, to our knowledge, for this setting. Other prior work focus more on practical algorithms without formal guarantees such as using multinomial regression with LASSO [Nguyen et al., 2013].

There also exists work for feature selection for value function estimation but under the assumption of a linear value function [Painter-Wakefield and Parr, 2012, Geramifard et al., 2011].

In the bandit setting, there has been work that formally shows how sample complexity can be reduced assuming the reward is a sparse linear function [Abbasi-Yadkori et al., 2012], where the sample complexity becomes smaller with more sparsity.

Feature selection can also be viewed as a form of model selection, where each model is a particular selection of features. Prior work has theoretical bounds for model selection such as the OAMS algorithm [Ortner et al., 2014]; however those bounds depend on the square root of the number of models. For factored MDPs the number of models grows doubly exponential with the number of features.

More generally for MDPs without a feature vector representation, the concept of feature selection translates to state abstraction – ignoring features is equivalent to clustering all the states that match on the necessary features. An example is the U-Tree algorithm [McCallum, 1996]. While many state abstraction algorithms perform well empirically, they lack formal guarantees.

4.4 Main Challenge of Feature Selection with Strategic Exploration

If we know the in-degree of the necessary features, or if we know which features are necessary, then we can simply use prior PAC algorithms for factored MDPs [Diuk et al., 2009, Strehl et al., 2007, Chakraborty and Stone, 2011]. However, because we do not know the in-degree nor which features are necessary, we run into issues when trying to explore and estimate the transition and reward dynamics. since we want our sample complexity to scale with the in-degree of just the necessary features, we cannot afford to gather enough data to also learn the dynamics of the unnecessary features. This means our estimates for the unnecessary feature dynamics will most likely be incorrect and stay incorrect. But if our dynamics model is incorrect, we may fail to explore properly, as we do not know which features are necessary or unnecessary so we cannot focus on simply exploring the necessary features. Therefore, our key insight is to take advantage of the failure to explore. We use directed exploration combined with the diameter assumption to try to explore by repeatedly targeting various target states. Then we can explicitly detect when we fail to explore by failing to reach the target within an expected number of steps. If we fail to explore, this implies our dynamics model is incorrect. However just knowing that it is incorrect is not enough, we also need to find out exactly which feature's estimated dynamics our model is predicting incorrectly, in order make progress in improving our model. For this we make a new assumption called the Superset Assumption, which we describe in more detail later in section 4.5.2 as well as formally define it in Assumption 2.

Now we go into more detail on why exploration can fail. First we go into some intuition about how to estimate the transition (and reward) dynamics. Suppose we know that the size of the parent set for feature x_i is K, but not which features are in it. We can consider trying out all potential parent sets of size K, and estimate the transition (and reward) conditioned on all those parent sets. Then as we get more data, we can compare different parent sets P and Q by comparing estimates of $P(x_i|P)$ and $P(x_i|Q)$ to estimates $P(x_i|P \cup Q)$. The idea is that if P is the true parent set for x_i, then further conditioning on any other parents Q should not change the probability ($P(x_i|P) = P(x_i|P \cup Q)$); if there is a significant difference, then P would be incorrect and we can eliminate it as a potential parent set. By checking this for all potential parent sets P and Q, we will be able to eliminate many incorrect parent sets, leaving only the true parent set as well as any other parent sets that just happens to give the same estimated probability as the true parent set. Regardless, we are left with an accurate estimate of the true transition (and reward) probability.

Next, we will go into detail about the issues with feature selection and exploration through a more concrete example. Consider a factored MDP with three binary features (x_1, x_2, x_3). The parent set for x_1 is $\{x_1\}$, for x_2 is also $\{x_2\}$, and for x_3 is $\{x_1, x_2, x_3\}$. Thus the in-degree of all features is 3. Suppose features x_1 and x_2 are necessary, and x_3 is unnecessary, then the in-degree of the necessary features is 1. During learning, we will be gathering data and estimating the dynamics. For example, for x_1, we would estimate $P(x_1' = 1|x_1)$, $P(x_1' = 1|x_2)$, and $P(x_1' = 1|x_3)$, which correspond to the 3 possible parent sets (there are actually 6 probabilities to estimate, 2 for each parent set where the parent feature can take on a value of either 0 or 1). Additionally, we would also estimate $P(x_1' = 1|x_1, x_2)$, $P(x_1' = 1|x_2, x_3)$, and $P(x_1' = 1|x_3, x_1)$ in order to conduct the pairwise comparison of different potential parent sets to eliminate incorrect parent sets. We would then estimate similarly for features x_2 and x_3. Note that our goal is a PAC algorithm whose sample complexity only scales with the in-degree of the necessary features, which in this case is 1. This means our algorithm cannot afford to consider parent sets of size 3 (the in-degree of all features), because that would require too many samples; we would need samples from all possible instantiations of the values of the parent set of size 3, which would be 2^3 in this case and exponential in the parent set size in general. Thus we would be unable to find the true parent set for the unnecessary feature x_3.

Being unable to find the true parent set for x_3 can cause exploration to fail to gather the samples that one would need for accurate estimates of the dynamics. Because all of the parent sets considered for x_3 are wrong, all of our estimates for the dynamics of x_3 could also be incorrect, leaving us with an incorrect dynamics model. Not being able to learn the dynamics for x_3 can mean that we fail to efficiently explore and gather samples for estimating $P(x_1' = 1|x_3)$, which would slow progress on finding the correct parent set for

x_1. For example, this may happen if the model is incorrect on how to transition from $x_3 = 0$ to $x_3 = 1$, and thus we are unable to efficiently gather samples from states where $x_3 = 1$. This would leave us using more than expected number of steps of exploration to estimate $P(x'_1 = 1|x_3 = 1)$. In general these effects may cascade and hinder the learning of the dynamics for necessary features, and resulting in high sample complexity. This is why we make the Superset Assumption. Briefly, this assumption constrains pathological cases where all of the data we get from visiting states where our dynamics model is incorrect is completely useless until we have collected past our limit on our sample complexity. We expect that in most cases, this data is actually very useful in quickly detecting which of our potential parent sets is giving us the wrong estimate and eliminating those incorrect parent sets.

4.5 Algorithm

In this section, we present the existence of an algorithm that can learn a factored MDP with sample complexity that scales exponentially with the in-degree of the necessary features and not with the in-degree of all features. Note that we do not present this as a practical algorithm, but more of an existence proof that it is possible to achieve the desired sample bounds.

4.5.1 Overview

Algorithm 5 shows the high-level structure of our approach. F' is the set of all features. m, H, M are parameters that will be defined later in the theory section. Algorithm 5 proceeds by fixing a possible in-degree K, starting with $K = 1$ up to the total number of features. For each K, the algorithm assumes that the in-degree of the necessary features is K, and proceeds accordingly. Incrementally increasing K is a simple approach to handle the case of not knowing the true in-degree [Chakraborty and Stone, 2011]. Algorithm 1 achieves near-optimal performance on average as soon as K is at least as large as the true in-degree of the necessary features, and continues to be near-optimal on average for all larger K. Thus our algorithm achieves near-optimal performance on average after a number of samples exponential in the true in-degree of only the necessary features.

For each considered in-degree K, Algorithm 1 executes the LearnAndSelect (Algorithm 6) subprocedure. Directed exploration and feature selection is done in LearnAndSelect. When LearnAndSelect finishes, it returns a subset of features F, and the learned model $\widehat{T}$ for those features. Then Algorithm 1 computes the optimal policy for that fea-

ture subset, and executes it for M steps. Our algorithm never explicitly discovers the true in-degree, and therefore LearnAndSelect is executed considered K. Our bound will be over the average performance over all steps for every K. Once K is at least as large as the in-degree of the necessary features, then LearnAndSelect will return a superset of the necessary features as well as correctly learned dynamics for the necessary features (with high probability), and we exploit enough steps such that the average error per step is near-optimal. We now discuss LearnAndSelect in more detail.

Algorithm 5

Input: m, H, M
for $K = 1$ to $|F'|$ **do**
  $F, \widehat{T}$=LearnAndSelect(K,m,H)
  Compute π^* for $F, \widehat{T}$
  Execute π^* for M steps
end for

Algorithm 6 LearnAndSelect

Input: K, m, H
Initialize F to all features
$G \leftarrow$ all possible feature-value vectors of size $2K$
while Exists a element g of G not visited m times **do**
  $\forall s\ R_{tmp}(s) = 0$, $R_{tmp}(g) = 1$
  Compute optimistic policy π_o using R_{tmp}
  stuck=True
  for $t = 1$ to H **do**
    Execute π_o
    If a feature-value vector that has not yet been visited m times is visited, set stuck=False and break out of loop
  end for
  if stuck **then**
    Run Superset Test to eliminate incorrect parent sets
    Eliminate f from F if all possible parent sets for it are eliminated
    Remove feature-value vectors from G whose features are no longer in F
  end if
end while
Return remaining F

4.5.2 LearnAndSelect

The purpose of LearnAndSelect (Algorithm 6) is twofold. One, it explores to gather samples for learning an accurate model. Two, under the diameter assumption and Superset Assumption, using directed exploration, it detects features whose dynamics cannot be accurately modeled using a parent set of size K and eliminates them. This results in a learned model over a subset of features.

Directed Exploration

To achieve both goals, Algorithm 6 performs directed exploration: it computes a policy to try to reach a target feature-value vector of size $2K$ (line 4–17)[1]. A feature-value vector is a particular instantiation of values for a set of features. We illustrate Algorithm 6 through a concrete example. Consider a factored MDP with 3 binary features (x_1, x_2, x_3). Suppose $K = 1$. Then all feature-value vectors of size 2 for (x_1, x_3) are simply $(x_1 = 0, x_3 = 0), (x_1 = 0, x_3 = 1), (x_1 = 1, x_3 = 0), (x_1 = 1, x_3 = 1)$. Then Algorithm 6 proceeds by first forming the set G of all possible feature-value vectors for all subsets of 2 features: $(x_1, x_2), (x_1, x_3), (x_2, x_3)$. The next step is to randomly select a feature-value vector from G as a target. Let $g = (x_1 = 0, x_2 = 1)$ be the first target. To reach $(x_1 = 0, x_2 = 1)$, Algorithm 6 uses a reward function where for any state that matches g, the reward is 1 i.e. states $(0, 1, 0), (0, 1, 1)$ (line 6). The reward for all other states is 0. Then an optimistic policy π_o is computed and followed for up to H steps to try to visit states that match the target feature-value vector (line 8–11).

Throughout LearnAndSelect, we are accumulating samples (s, a, s', r) which we use to learn a model. We use the samples to estimate probabilities $P(x_i|P, a)$, $P(x_i|Q, a)$, and $P(x_i|P \cup Q, a)$ where P, Q are potential parent sets for feature x_i, and use these estimates to narrow down which parent sets could be correct. We start with all parent sets being possible, and eliminate incorrect parent sets as more samples are accumulated. In particular, we use the Adaptive k-Meteorologists Algorithm [Diuk et al., 2009].

The optimistic policy (line 6) is then computed by picking a parent set out of the remaining possible parent sets for every feature such that results in the largest optimal value.

[1]The choice of $2K$ will become clear in the theoretical analysis

Selecting Features

If directed exploration is always successful and ends up visiting all state-action pairs at least m times, and K is at least as large as the in-degree of necessary features, then LearnAndSelect will have enough samples to learn a near-optimal model for the necessary features. However, since the true in-degree is unknown, K could be incorrect, and directed exploration could fail to visit everything. This negative information will be used to eventually eliminate features whose dynamics cannot be accurately represented with parent sets of size K.

The diameter assumption guarantees that all states are reachable in expected D steps under at least one policy. If our learned model is correct, then the optimistic policy π_o would be one such policy, and we can visit the target g in $H > O(D^2)$ steps with high probability (the precise value of H is specified in the theory section). If π_o fails, then we can conclude that our learned model indeed has an error. Specifically, one of the parent sets that was used to compute π_o is incorrect. Furthermore, because the policy π_o is optimistic, we will keep visiting state–action pairs where that parent set is incorrect. However, because we do not know which parent set is incorrect, we now rely on our Superset Assumption to figure it out.

Superset Assumption

In order to pinpoint which parent set was incorrect, we use the following observation. Suppose in our dynamics model we use the parent set P for some feature and it is incorrect. Suppose the parent set Q is the true parent set. Then we can easily detect that P is incorrect by gathering m samples of the dynamics for the parent set $P \cup Q$ and compare. The challenge is that we do not know Q in advance, nor do we know the size $|Q|$. We would need to try out all possible Q, which can take an exponential number of steps, where the exponent is $|Q|$. Thus if $|Q|$ is larger than the in-degree of the necessary features, we can no longer ensure sample complexity that is solely exponential in the in-degree of the necessary features.

Now consider the parent set $P \cup Q'$ where $Q' \subseteq Q$. In practice, we expect that by adding some of the true parents Q' into P, we may detect a difference in their predictions without adding all of Q. Of course a pathological case is still possible where no matter what proper subset $Q' \subseteq Q$ we add to P, the prediction stays the same, and it is only when we try adding the whole Q where we can finally see a difference. However this seems unlikely in practice. Thus we make our Superset Assumption which states that there exists a Q' where $|Q'| \leq 2K$ and we can see a difference in the prediction of the dynamics. Note

that we make no further assumptions on Q'.

Superset Test

We perform the Superset Test (line 13) to figure out which parent sets are incorrect and eliminate them. For every feature f in F, the Superset Test will compare the predictions of its parent set in the current model to the predictions of all supersets of size $2K$, using the samples gathered in the H steps. The Superset Assumption guarantees that the Superset Test will always find at least one incorrect parent set to eliminate.

As an example, suppose we have explored for some time and our current target is now $g = (x_2 = 1, x_3 = 1)$. Suppose we get stuck. Then after H steps, we would detect that we got stuck and then perform the Superset Test. Suppose we first look at feature x_2 and the possible parents remaining are x_2 and x_3 i.e. x_1 has been eliminated. The Superset Test will test the predictions of each parent set with all of their possible supersets. For the parent set $(x_2 = 0)$, the supersets of size 2 are $(x_2 = 0, x_3 = 0)$ and $(x_2 = 0, x_3 = 1)$. Let $\hat{P}(x_2 = 0|x_2 = 0)$ be the estimated transition for x_2 with parent set $(x_2 = 0)$. Let $\hat{P}(x_2 = 0|x_2 = 0, x_3 = 0)$ be the estimated transition for x_2 with the parent set $(x_2 = 0, x_3 = 0)$. The Superset Test will check whether $|\hat{P}(x_2 = 0|x_2 = 0, x_3 = 0) - \hat{P}(x_2 = 0|x_2 = 0)|$ is above some threshold. If it is, then it would mean that x_2 is not a parent of x_2 and so both parent sets $(x_2 = 0)$ and $(x_2 = 1)$ would be eliminated. The Superset Test then continues with the other possible parents and their supersets, and then will check the other features in the same way. If all possible parent sets for a feature x_i has been eliminated, then x_i itself is eliminated (line 14).

Each time Algorithm 6 targets a feature-value vector for exploration, either some target that has not yet been visited m times will get visited, or the Superset Test will eliminate a potential parent set for a feature. Thus eventually the algorithm will terminate. Once Algorithm 6 terminates, we are left with the remaining features F for which we have visited all targets at least m times. If K is at least as large as the in-degree of necessary features, then F will be a superset of the necessary features, and by visiting all targets m times the learned model $\widehat{T}$ of the necessary features will be accurate. There may be some unnecessary features in F as well, but as long as the model of the necessary features are accurate, it is sufficient to calculate a near-optimal policy.

4.6 Theoretical Analysis

This section presents the supporting lemmas and main theorem for the performance of Algorithm 5. In the first section (Section 4.6.1), we present the assumptions we make.

4.6.1 Assumptions

Assumption 1. *(Diameter Assumption) Let s, s' be any two states. Let $D_\pi(s, s')$ be the random variable for the number of steps it takes policy π to start at s and reach s' the first time. Let $D(s, s') = \min_\pi \mathbb{E}(D_\pi(s, s'))$. A diameter D means that $D \geq \max_{s,s'} D(s, s')$. It is an upper bound on the expected number of steps it takes to go between any two states. We assume D is known.*

Assumption 2. *(Superset Assumption) Let π be a policy computed from assuming a particular parent set for every feature, where some of the parent sets are incorrect. Suppose π visits a state–action pair for which the predicted dynamics of an incorrect parent set are 2ϵ off from the true dynamics in $O(D^2)$ steps, with probability $1 - \delta$. This would be the case if we get stuck (line 13) in Algorithm 6.*

Let that feature with an incorrect parent set be x_i. Let W of size K be the wrong parent set (of features) for feature f_i. Let U be the true parent set (of features). Consider supersets of W of size $2K$. There are up to $\binom{n}{K} \leq O(n^K)$ different superset sets of size $2K$. Each superset has d^{2K} instantiations of values. By the pigeonhole principle, after $d^{2K}m$ steps, at least one instantiation of every possible superset has been visited at least m times, since we gather data for all supersets at every step.

The assumption we make is that after $O(d^{2K}m)$ steps, there exists some superset W' out of all possible supersets of size $2K$ such that one of its instantiations has been visited at least m times and for that instantiation, $|P(x_i|W', a) - P(x_i|W, a)|_1 \geq O(\epsilon)$ for some action a.

4.6.2 Discussion of Assumption 2

The challenge with creating a Superset-like assumption comes from the intricate dependence on the policy. When the parent set for a feature is incorrect, the MLE estimate according to that incorrect parent set is completely determined by the state distribution induced by the policy. Because of this, it is extremely difficult to make a Superset-like assumption that only depends on the factored MDP. There can exist adversarial policies for

which it is near impossible to detect that a parent set is incorrect without a massive amount of data. Thus our assumption attempts to use a favorable policy where we are repeatedly visiting informative states where the incorrect parent set has an incorrect transition model.

Comparison to G–SCOPE

For the G–SCOPE algorithm [Hallak et al., 2015], 3 assumptions are made: Strong Parent Superiority, Non-Parent Conditional Weakness, and Conditional Diminishing returns. Our Superset Assumption is similar to Strong Parent Superiority, and we do not make the other 2 assumptions. Because G–SCOPE is an offline algorithm, a suitable behavior policy as well as suitable domain dynamics are needed to satisfy these 3 assumptions. For our algorithm, the Superset Assumption is similarly dependent on the policy and dynamics, but the dependence on the policy is weaker because we have control over the policy; we are using a policy in which we know we are visiting state–action pairs for which our dynamics has incorrect predictions, gathering samples from those pairs, and so we expect in practice that this exploration policy is sufficient to satisfy the Superset Assumption.

Satisfying the Superset Assumption

Here, we give some intuition for how it can be satisfied for simple settings.

The Stock Trading domain used in many prior PAC-FMDP algorithms [Strehl et al., 2007] satisfies the Superset Assumption. In the Stock Trading domain, the state is specified by a set of sectors, each sector having a number of stocks. Stocks are binary features indicating whether they are rising or falling. Sectors are binary variables indicating whether they have been bought or not. The probability of a stock rising is $P(rising) = 0.1 + 0.8 \times (\text{fraction of stocks rising in same sector})$. The dynamics of the stocks are not affected by actions. Thus in this domain, no matter what the policy is, it will be possible to reach any stock state. Suppose there are 3 sectors and 3 stocks per sector. Suppose we mistakenly assume the in-degree is 1, whereas the true in-degree is 3. Then we will only be able to try parent sets with one stock, and end up aliasing multiple states. There is a significant difference in $P(rising)$ when the number of rising stocks is off by one; that difference is exactly $0.8/3$. Consider a parent set where the stock is currently not rising. The prediction of this parent would be the average of the predictions from 4 aliased states (the state of the other 2 stocks in the sector, which could have 0, 1, or 2 stocks rising). Comparing this to a superset of size 2, where both stocks are current not rising; in this case there are 2 aliased states, which is just whether the 3rd stock is rising or not. The probability of rising predicted by the superset of size 2 will be a fixed quantity lower than

what is predicted by the parent set of size 1. If the requested accuracy of near-optimality, ϵ is small, then we will be able to detect this difference, no matter the policy, and thus rely on the superset assumption to pinpoint that this parent set is incorrect. Generalizing to more stocks and sectors with higher in-degree (the in-degree is the number of stocks per sector), the superset test will consider parent sets of k stocks as well as supersets of $2k$ stocks. There is a large difference in $P(rising)$ between k stocks and $2k$ stocks, with the additional k stocks all being in the rising state, and we will be able to detect that difference.

The Taxi domain [Dietterich, 2000, Hallak et al., 2015] also satisfies the Superset Assumption under many policies. Consider a uniform random exploration policy. Then all 25 locations in the 5x5 grid world would be visited with reasonable probability (there is no location that is particularly hard to get to). Movement actions have in-degree 2, so comparing a parent set of size 1 to supersets of size 2 will definitely work, since the true parent set is also size 2. The pick up and drop off actions have in-degree 3; they depend on both the row and column of the location of the taxi, as well as another feature representing the location of the passenger. With an incorrect parent set of size 1, we would be aliasing states for which the pickup/dropoff actions are both successful and not. However if we consider supersets of size 2, then there are feature-values where pickup/dropoff always fail. Thus we can detect the difference between sometimes being successful and always failing, and learn that the parent set of size 1 is incorrect.

4.6.3 Small Lemmas

In this section, we present some small lemmas that will be used later on.

Lemma 12. *(Necessary Feature Lemma) For any policy π*

$$Q^\pi(s, a, T) = Q^\pi(z, a, T) \tag{4.1}$$

where z is the state s restricted to only the necessary features from F i.e. Q-functions only depend on the necessary features. Furthermore, the transition dynamics of the unnecessary features have no effect on Q-functions. This means that as long as the dynamics for the necessary features are learned accurately, dynamics for unnecessary features may be arbitrary.

Proof. Let π be given. Let $s = (z, y)$ be the state decomposed into necessary features z and unnecessary features y. Initialize $Q(\cdot, \cdot, 0)$ to 0. We will perform induction. The base case for $Q(\cdot, \cdot, 0)$ is trivially true since it is a constant.

By induction

$$
\begin{aligned}
& Q(s,a,T+1) && (4.2)\\
&= R(s,a) + \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a',T) && (4.3)\\
&= \sum_i R_{i,a}(s) && (4.4)\\
&+ \sum_{s'} \prod_i P_{i,a}(s'|Par_{i,a}(s)) \max_{a'} Q(s',a',T) && (4.5)\\
&= \sum_i R_{i,a}(z) && (4.6)\\
&+ \sum_{s'} \prod_i P_{i,a}(s'|Par_{i,a}(s)) \max_{a'} Q(z',a',T) && (4.7)\\
&= \sum_i R_{i,a}(z) && (4.8)\\
&+ (\sum_{z'} \prod_{i \in z'} P_{i,a}(s'|Par_{i,a}(s)) \max_{a'} Q(z',a',T) && (4.9)\\
&\cdot \sum_{y'} \prod_{i \in y'} P_{i,a}(s'|Par_{i,a}(s))) && (4.10)\\
&= \sum_i R_{i,a}(z) && (4.11)\\
&+ \sum_{z'} \prod_{i \in z'} P_{i,a}(s'|Par_{i,a}(s)) \max_{a'} Q(z',a',T) && (4.12)\\
&= Q(z',a',T) && (4.13)
\end{aligned}
$$

Note that the dynamics of the unnecessary features make no difference.

□

Lemma 13. *(Simulation Lemma) [Kearns and Koller, 1999] Let M be a factored MDP over n state variables with l entries in conditional probability table of the transition model. Let M' be an approximation to M where all the CPTs differ by at most $\alpha = O((\epsilon/T^2 l R_{\max})^2)$. Then for any policy π, $|U^\pi_M(T) - U^\pi_{M'}(T)| \leq \epsilon$.*

Lemma 14. *(Explore or Exploit Lemma) Fix a policy π. Let M and M_K be MDPs such that M and M_K agree on some states, but differ in dynamics and rewards for other states. Then $|U^\pi_{M_K}(T) - U^\pi_M(T)| \leq T R_{\max} P(escape)$ where $P(escape)$ is the probability of visiting a state in which the two models differ.*

Proof. Let τ_1 denote trajectories that stay within states where the two models agree and τ_2 denote trajectories where there are escapes to other states. Then

$$|U^{\pi}_{M_K}(T) - U^{\pi}_{M}(T)| \tag{4.14}$$

$$= \frac{1}{T}|\sum_{\tau,|\tau|=T} P_M(\tau)R(\tau) - \sum_{\tau,|\tau|=T} P_{M_K}(\tau)R(\tau)| \tag{4.15}$$

$$\leq \frac{1}{T}|\sum_{\tau_1} P_M(\tau_1)R(\tau_1) - \sum_{\tau_1} P_{M_K}(\tau_1)R(\tau_1)| \tag{4.16}$$

$$+ \frac{1}{T}|\sum_{\tau_2} P_M(\tau_2)R(\tau_2) - \sum_{\tau_2} P_{M_K}(\tau_2)R(\tau_2)| \tag{4.17}$$

$$\leq \frac{1}{T}|\sum_{\tau_2} P_M(\tau_2)R(\tau_2) - \sum_{\tau_2} P_{M_K}(\tau_2)R(\tau_2)| \tag{4.18}$$

$$\leq \frac{1}{T}\sum_{\tau_2} |P_M(\tau_2)R(\tau_2) - P_{M_K}(\tau_2)R(\tau_2)| \tag{4.19}$$

$$\leq \frac{1}{T}TR_{\max}\sum_{\tau_2} |P_M(\tau_2) - P_{M_K}(\tau_2)| \tag{4.20}$$

$$= R_{\max}P(escape) \tag{4.21}$$

Because non-escapes result in exactly the same trajectories with the same dynamics, so the probability of escaping to the other states is the same in both M and M_K. □

Corollary 1. *Suppose π_1 is the optimal policy for M_k and π_2 is the optimal policy for M. Suppose $U^*_{M_K}(T) \geq U^*_M(T)$ i.e. M_K is optimistic. Then $U^{\pi_1}_{M_K} \geq U^{\pi_2}_M - TR_{\max}P(escape)$.*

Proof.

$$U^{\pi_1}_M \geq U^{\pi_1}_{M_K} - R_{\max}P(escape) \tag{4.22}$$

$$\geq U^{\pi_2}_M - R_{\max}P(escape) \tag{4.23}$$

□

4.6.4 LearnAndSelect

We now prove several results about the LearnAndSelect (Algorithm 6) subprocedure.

First, note that throughout the whole subprocedure, we are accumulating samples (s, a, s', r) which we use to learn a dynamics model. There exists multiple algorithms

which can take samples and learn a dynamics model for a factored MDP, and in general they use the samples to estimate probabilities $P(x_i|P,a)$, $P(x_i|Q,a)$, and $P(x_i|P \cup Q,a)$ where P, Q are potential parent sets for feature x_i, and use these estimates to narrow down which parent sets could be correct. In particular, one can use the Adaptive k-Meteorologists Algorithm [Diuk et al., 2009].

The optimistic policy (line 6) is then computed by picking a parent set out of the remaining possible parent sets for every feature such that results in largest optimal value.

Determining m

In this section we determine a sufficient value for m, the number of samples that LearnAndSelect tries to obtain from all feature-value targets of size $2K$.

Lemma 15. *Let F be a subset of features with in-degree K. Suppose we have*

$$m = O\left(|A|K\frac{d}{\epsilon_1^2}\log(d|F|/\delta_1)\right)$$

samples from every feature-value target of size $2K$ of F. Then we can construct a dynamics model for F with an L_1 accuracy of ϵ_1 with probability $1-\delta_1$.

Proof. The Adaptive k-Meteorologists Algorithm [Diuk et al., 2009] is a sample efficient online learning algorithm for learning the dynamics of a factored MDP. Given any (nonstationary) exploration policy and a known in-degree, it will learn a near-optimal dynamics model making only a finite number of sub-optimal mistakes (prediction accuracy), with high probability.

By Hoeffdings, we need $O(\frac{d}{\epsilon_1^2}\log(d/\delta_1))$ samples to learn each set of values of a parent set to an L_1 accuracy of ϵ_1 (we apply Hoeffdings d times, learning the probability of each outcome with Hoeffdings) with probability $1-\delta_1$. Since there are $|A|$ actions, the number of samples we need is

$$O\left(|A|\frac{d}{\epsilon_1^2}\log(d/\delta_1)\right) \tag{4.24}$$

By Adaptive k-Meteorologist, we need $O\left(\frac{1}{\epsilon_1^2}\log\frac{k}{\delta_1}\right)$ samples from $P(x_i|P \cup Q,a)$, i.e. the pair of parent sets P, Q, in order to eliminate whichever P or Q is more incorrect (k is the number of possible feature-value vectors of parents). Therefore if we have that many visits for all possible pairs $P \cup Q$, i.e. all feature-value pairs of size $2K$, then all

incorrect parent sets will be eliminated, and we are left with parent sets whose predictions are all ϵ_1 accurate (and we can just pick one arbitrarily).

There are $\binom{|F|}{K}$ possible parent sets in total. Each parent set has d^K possible value instantiations, thus $k = O\left((d|F|)^K\right)$. Then a sufficient value for m is

$$m = O\left(|A|K\frac{d}{\epsilon_1^2}\log(d|F|/\delta_1)\right)$$

where we do a union bound for the error probability with $\delta_1 = \frac{1}{\delta\binom{|D|}{K}}$ and bound $\binom{|F|}{K} \leq O(|F|^K)$.

□

4.6.5 Directed Exploration

In this section, we present several results on the directed exploration of LearnAndSelect.

Lemma 16. *(Exploration Episode Lemma) The following holds w.p. $1 - \delta_1$. At the end of each iteration of the while loop (line 4 – line 17) in the LearnAndSelect algorithm (Algorithm 6), one of two things will happen: either the target g or another feature-value vector that has not been visited m times will be visited, or some potential parent set will be eliminated as a possible parent for some feature.*

Proof. The idea behind the directed exploration is the Explore or Exploit lemma (Lemma 14) and the diameter assumption. The diameter assumption allows the algorithm to potentially reach g with high probability. The Explore or Exploit lemma allows the algorithm to either reach g or end up gathering new data, or fail. If it fails, then we take advantage of the Superset assumption to eliminate an incorrect parent set.

First, we compute how large H needs to be in order for a good policy to reach g with high probability. By the diameter assumption, there exists a policy expected to reach g within D steps, thereby obtaining a reward of 1 from the artificially defined reward function R_{tmp}. By the Markov Inequality, the probability of reaching the goal within $2D$ steps is at least $\frac{1}{2}$. Thus the optimal average value within $2D$ steps is at least $\frac{1}{4D}$. If we used an ϵ_2-optimal policy, it would have an expected value of at least $\frac{1}{4D} - \epsilon_2$. Let τ be trajectories of length $2D$. Then $\frac{1}{4D} - \epsilon_2 = \frac{1}{2D}\sum_\tau Pr(escape)Pr(\tau|escape)TotalReward(\tau)$. The probability that the ϵ_2-optimal policy reached the goal (escapes) can be lower bounded by the worst case scenario: every escape trajectory has every step giving a reward. That means the probability of reaching the goal (escape) is at least $\frac{1}{4D} - \epsilon_2$. Then probability

of failing to reach the goal is at most $1 - \frac{1+2D\epsilon_2}{4D}$. Then by repeating this 2D-step trial N times, the error probability is upper bounded by $\left(1 - \frac{1+2D\epsilon_2}{4D}\right)^N$. Then that means if we want to have a failure probability of δ_1, we would need to repeat this $2D$-step sub-episode $\frac{\log(\delta_1)}{\log(1-\frac{1+2D\epsilon_2}{4D})}$ times. We can simplify the denominator $\log(1 - \frac{1+2D\epsilon_2}{4D})$ by the upper bound $\log(1 - \frac{1}{4D})$. Note that the $\log$ function is concave, so we can upper bound it with its first order approximation around $\log(1)$ i.e. by $O(-\frac{1}{D})$. Simplifying the whole fraction becomes $O(D\log(1/\delta_1))$. Thus we end up with

$$O(D^2 \log(1/\delta_1)) \tag{4.25}$$

as the number of steps we need before reaching the goal with high probability. Thus this is a lower bound for H and we also know in this case the while loop will terminate early after these many steps.

Now we consider the case when our optimistic policy is bad i.e. the probability of escaping is at least ϵ_2. Then either we get lucky and visit some other feature-value target that has not yet been visited m times, or we get stuck. If we get stuck, then we need the data in these H steps for the Superset test (line 13).

For the Superset test, we apply the Superset assumption (Assumption 2). We know that we got stuck so the data that we have is where the escape probability is high, thus meeting the superset assumption requirements of visiting distinguishing states (states where our model is incorrect).

Now we count how much data we need from getting stuck to perform the Superset Test. We need to gather new data for the prediction of supersets of size $2K$. By Assumption 2 we will need $O(d^{2K}m)$ steps. However since we only have an escape probability of ϵ_2, we need to add repeats to escape with high probability, just like we did earlier.

This means we need an additional factor of $O(\frac{R_{\max}}{\epsilon_2}\log(1/\delta_1))$. So we need

$$H > O\left(\frac{d^{2K}mR_{\max}}{\epsilon_2}\log(1/\delta_1)\right)$$

. Combining this with earlier means a sufficient H is

$$H = O\left(\frac{D^2 d^{2K}mR_{\max}}{\epsilon_2}\log(1/\delta_1)\right) \tag{4.26}$$

By Lemma 15, we have an ϵ_1-accurate dynamics model, so together with the Simulation Lemma (Lemma 13) that means $\epsilon_1 = O\left(\frac{\epsilon_2^2}{\max(D,T_\epsilon)^4 R_{\max}^2 n^2 |A|^2 d^{2K}}\right)$ where n is the total

number of features. So the updated value for m in terms of ϵ_2 is

$$m = O\left(\frac{KdR_{\max}^4 n^4 |A|^5 d^{4K} \max(D, T_\epsilon)^8}{\epsilon_2^4} \log(dn \max(D, T_\epsilon) R_{\max} |A|/\delta_2)\right) \quad (4.27)$$

□

Lemma 17. *(LearnAndSelect Lemma) The following holds w.p. $1 - \delta_1$. After LearnAndSelect (Algorithm 6) is finished, all targets g will either have been visited m times, or one of its features will have been eliminated. If $K \geq J$ where J is the in-degree of the necessary features, all necessary features will remain in F. This will take*

$$O\left(\frac{D^2 K (dn)^{3K} m^2 R_{\max}}{\epsilon_2} \log(dn/\delta_1)\right)$$

steps.

Proof. Now we will count how many steps LearnAndSelect (Algorithm 6) will take. From Lemma 16, every while loop iteration (line 4 – line 17) contributes to one of two cases: visiting a feature-value vector that has not yet been visited m times, or the Superset Test.

Since there are at most $O((dn)^K)$ superset tests (each test eliminates at least one parent set), and we know how much data each superset test requires (equation 4.26), we combine those to get a total of

$$O((dn)^K H) \quad (4.28)$$

$$= O\left(\frac{D^2 (dn)^{2K+1} m R_{\max}}{\epsilon_2} \log(1/\delta_1)\right) \quad (4.29)$$

steps towards superset tests.

Our targets are subsets of features and values of size $2K$, thus there are $O((dn)^{3K})$ targets. Each target needs to be visited m times, thus $O((dn)^{3K} mH)$ total steps are needed. Then the number of steps this all takes is

$$O((dn)^K H + (dn)^{3K} mH) \quad (4.30)$$

$$= O\left(\frac{D^2 K (dn)^{3K} m^2 R_{\max}}{\epsilon_2} \log(dn/\delta_1)\right) \quad (4.31)$$

where this includes a union bound over all $O((dn)^{3K})$ targets and $O((dn)^K)$

Thus eventually the while loop will have visited all possible targets from the remaining set of features F. If $K \geq J$, where J is the in-degree of the necessary features, then the

necessary features must remain in F because they will never be eliminated through the superset test, since $K \geq J$.

□

4.6.6 Main Theorem

In this section, we give the main theorem and its proof.

Theorem 5. *Given ϵ, δ. Let $T_{\epsilon,D} = \max(D, T_\epsilon)$. Let J be the in-degree of the necessary features. Let n be the total number of features. Then the following is true of Algorithm 5 with probability $1 - \delta$*

1. *The total number of steps taken up to $K = J$ is*

$$O\left(\frac{J^2 D^2 (dn)^{11J+10} |A|^{10} R_{\max}^{10} T_{\epsilon,D}^{16}}{\epsilon^{10}} \log^2(dn T_{\epsilon,D} R_{\max} |A| / \delta)\right)$$

2. *For all $K \geq J$ i.e. at least as large as the in-degree of the necessary features, the average per-step reward is ϵ-optimal i.e. $|U - U^*| \leq \epsilon$*

Proof. From Lemma 17, LearnAndSelect will take $O\left(\frac{D^2 K (dn)^{3K} m^2 R_{\max}}{\epsilon_2} \log(dn/\delta_1)\right)$ steps. Once $K \geq J$, the necessary features will remain in the returned F, and by Lemma 15, the returned dynamics of the necessary features will be ϵ_1 accurate. Due to the value of m (eqn 4.27), the optimal policy computed from the learned dynamics will be ϵ_2-optimal, since we can ignore the dynamics of any unnecessary features in F by Lemma 12.

Thus after LearnAndSelect, Algorithm 5 will execute an ϵ_2-optimal policy for M steps. Counting all of the steps of LearnAndSelect as mistakes and setting as B, the average error per-step for any single $K \geq J$ is

$$\frac{R_{\max} B + \epsilon_2 M}{B + M} \tag{4.32}$$

So to bound this by ϵ, we need $\epsilon_2 < \epsilon$, so we'll use $\epsilon_2 = \epsilon/2$. Also we need to make M

large enough. Then

$$\frac{2R_{\max}B + \epsilon M}{2B + 2M} \leq \epsilon \tag{4.33}$$

$$\iff 2B(\frac{R_{\max}}{\epsilon} - 1) \leq M \tag{4.34}$$

$$\impliedby M \geq \frac{2BR_{\max}}{\epsilon} \tag{4.35}$$

$$\impliedby M = O\left(\frac{BR_{\max}}{\epsilon}\right) \tag{4.36}$$

where $B = O\left(\frac{D^2K(dn)^{3K}m^2R_{\max}}{\epsilon_2}\log(dn/\delta_1)\right)$. So for each K, Algorithm 5 runs for $M+B$ steps, which is

$$O\left(\frac{D^2K(dn)^{3K}m^2R_{\max}^2}{\epsilon^2}\log(dn/\delta_1)\right) \tag{4.37}$$

Then we count all $K < J$ as mistakes, which is

$$O\left(\frac{J^2D^2(dn)^{3K}m^2R_{\max}^2}{\epsilon^2}\log(dn/\delta_1)\right) \tag{4.38}$$

Finally plugging in m (equation 4.27) results in the final sample complexity of

$$O\left(\frac{J^2D^2(dn)^{11J+10}|A|^{10}R_{\max}^{10}T_{\epsilon,D}^{16}}{\epsilon^{10}}\log^2(dnT_{\epsilon,D}R_{\max}|A|/\delta)\right) \tag{4.39}$$

where we use a union bound to bound the error for each K with $\delta_1 = \delta/n$. □

Chapter 5

Directed Exploration in RL with Function Approximation

This work was done jointly with Emma Brunskill and is in-submission.

5.1 Introduction

Up until this point, this thesis has focused on theoretically analyzing various settings that relate to how reinforcement learning is represented in practice. The algorithms proposed have used directed exploration as an effective tool to obtain sample efficient bounds. But theory is only part of the solution. In this chapter, we investigate how directed exploration can be used to improve sample efficiency in practice.

Neural networks have been shown to be an effective function approximator for large, complex functions in many domains. In order to scale up reinforcement learning, function approximators are a key component, and recent work combining RL with deep learning has shown promising results; algorithms such as DQN, PPO, A3C have been shown to work in domains such as Atari games, and robot locomotion and manipulation [Mnih et al., 2013, Schulman et al., 2017, Mnih et al., 2016]. However, most approaches only use simple exploration strategies that add some simple randomness to the actions. DQN uses e-greedy, whereas PPO and A3C follow the policy gradient with policies that add some random noise to the actions. The lack of more sophisticated exploration has hindered progress in more complex domains with sparse rewards.

Recently, there has been work towards more sophisticated exploration strategies in

deep RL. Noisy networks add e-greedy exploration in the parameter space rather than the action space have been shown to be better [Fortunato et al., 2017]. Bootstrapping and ensembles have been used to approximate posterior sampling for better exploration [Osband et al., 2016]. Currently, uncertainty-based methods have been shown to be the most effective and most promising to tackle hard exploation domains [Houthooft et al., 2016, Ostrovski et al., 2017, Tang et al., 2017, Burda et al., 2018]. These uncertainty-based methods use a reward bonus approach, where they compute a measure of uncertainty and transform that into a bonus that is then added into the reward function.

Unfortunately this reward bonus approach has some drawbacks. We focus on the model-free case where we either approximate a value function or a policy since the state of the art approaches in deep RL are either model-free or a hybrid combination of model-based and model-free; regardless, there exists a model-free component that must take in the augmented reward function. The main drawback is that reward bonuses may take many, many updates before they propagate and change agent behavior. This is due to two main factors: 1) the function approximation itself needs many updates before converging; 2) the reward bonuses are non-stationary and change as the agent explores, meaning the function approximator needs to update and converge to a new set of values every time the uncertainties change. This makes it necessary to ensure that uncertainties do not change too quickly, in order to give enough time for the function approximation to catch up and propagate the older changes before needing to catch up to the newer changes. If the reward bonuses change too quickly, or are too noisy, then it becomes possible for the function approximator to prematurely stop propagation of older changes and start trying to match the newer changes, resulting in missed exploration opportunies or even converging to a sub-optimal mixture of old and new uncertainties. Non-stationarity has already been a difficult problem for RL in learning a Q-value function, which the DQN algorithm is able to tackle by slowing down the propagation of changes through the use of a target network [Mnih et al., 2013]. These two factors together result in slow adaptation of reward bonuses and lead to less efficient exploration.

As we have seen in past chapeters, directed exploration is an alternative approach for using uncertainty for exploration, and is able to avoid the issue of non-stationarity altogether. Note that in most prior chapters we focused on the tabular setting, where planning given a goal (selected via directed exploration) was straightforward. Now with function approximation, computing a policy to reach a desired goal is non-trivial. To do this, we learn a goal-conditioned policy $\pi(s, g)$ that would enable us to try to reach any goal states we specify [Schaul et al., 2015a, Andrychowicz et al., 2017]. We would then repeatedly pick states that have high uncertainty and set them as goals and use the goal-conditioned policy $\pi(s, g)$ to reach them. This results in an algorithm that is completely stationary,

because the goal-conditioned policy is independent of the uncertainty. Also, because we can pick and commit to reaching goal states, we are much more robust to the case where uncertainty estimates are changing and noisy, which can negatively impact and slow down reward bonus approaches. We show in our experiments that directed exploration is more robust to noisy and inaccurate uncertainty measures, and is more efficient at exploration than the reward bonus approach.

This idea of directed exploration is closely related to the hierarchical reinforcement learning literature. It can be considered a particular instantiation of goal generation from higher level policies, and then trying to reach those goals through lower level policies. Prior work has focused on uniformly random or expert guided goal generation for learning sub task and task structure, but we look into uncertainty-based goal generation explicitly for better exploration [Held et al., 2018, Nachum et al., 2018]. We also do not require learning a hierarchical policy, allowing our method to be more easily adapted to existing algorithms.

5.2 Background

We now outline several basic components that we will build on to construct a practical directed exploration algorithm with function approximation.

5.2.1 DQN

Deep Q-Networks (DQN) is a deep reinforcement learning algorithm that has been shown to successfully achieve superhuman performance in variety of domains, including multiple Atari games [Mnih et al., 2013], and is suited to use with discrete action domains. It is a variation of Q-learning and is able to make use of off-policy data, i.e. data that came from following a different policy than the current greedy policy associated with the learned Q-network. This is important as when we are doing directed exploration, we will be following different policies to try to reach different goal states, which will be different from the current best policy.

5.2.2 DDPG

Deep deterministic policy gradient (DDPG) is an off-policy, deep reinforcement learning algorithm to use with continuous state and action spaces based on policy gradient [Lillicrap

et al., 2015].

5.2.3 UVFA

Universal value function approximators (UVFA) extends the idea of a value function to a goal conditioned value function $V(s, g)$ that represents the value when trying to reach the goal g starting from state s [Schaul et al., 2015a]. Given such a value function, we can then navigate to any goal g, in particular we can use this to facilitate directed exploration by picking goals g that are useful for exploration. DQN can be naturally extended to implement UVFA by adding the goal g as an additional input to learn the goal-conditioned Q-value function $Q(s, a, g)$, and DDPG can be naturally extended to learn the goal-conditioned policy $\pi(s, g)$. To unify both approaches, since we can extract out a policy from the Q-value function, we now will refer to both DQN and DDPG being able to learn a goal-conditioned policy $\pi(s, g)$.

5.2.4 HER

Hindsight experience replay (HER) is a technique that can speed up the learning of an UVFA by making use of trajectories that fail to reach the goal [Andrychowicz et al., 2017]. Given a trajectory that tried and failed to reach g, instead we can pretend that we were actually trying to reach s', where s' is a state that we actually visited during the trajectory. HER turns a negative example for a goal g into data for a positive example for a reached goal s'.

5.3 Directed Exploration

5.3.1 Directed Exploration Outline

The general steps behind directed exploration is outlined in Algorithm 7. The main idea is to repeatedly try to visit the states with the largest uncertainty according to some uncertainty measure U, and then take one step of random action as exploration. We then rely on an off-policy RL algorithm to learn from these directed exploration trajectories. This algorithm is a slight modification of the directed exploration that is done in chapters 2-4. Tt is specificically only considering single states as possible goals rather than sets of states since keeping track of sets of states quickly becomes computationally infeasible.

Algorithm 7 Generic Directed Exploration RL

Input: $A, U, \pi(s, g)$
A is an off-policy RL algorithm
U is a method to compute uncertainty for a state
for Episode $i = 1$ to ∞ **do**
  while episode not ended **do**
    Pick g with largest uncertainty according to U
    Visit g using goal-conditioned policy $\pi(s, g)$
    Take a random action
  end while
  Update A
end for

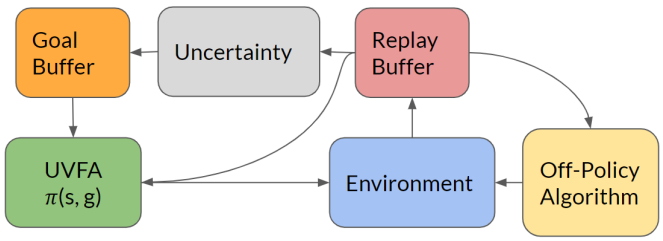


Figure 5.1: The different components of directed exploration with function approximation.

It also includes a step of random exploration, because the uncertainty measure may be noisy and inaccurate, so having a bit of randomness prevents directed exploration from being stuck visiting the exact same states over and over again. While this outline can be implemented exactly in the tabular setting, there are several additional considerations that must be made when we move to the function approximation setting.

5.3.2 Directed Exploration with Function Approximation

Algorithm 8 outlines how to do use directed exploration in the function approximation setting. Figure 5.1 shows the different components that make up the algorithm.

Algorithm 8 Directed Exploration RL with Function Approximation

Input: D, K, N, E, A, U
A, E are off-policy RL algorithms
A is used to learn from the real reward
E is used to learn the UVFA/goal-conditioned policy $\pi(s, g)$
U is a method to compute uncertainty for a state
$G \leftarrow$ FIFO buffer of goal states with capacity N
for Episode $i = 1$ to ∞ **do**
    With probability 0.5, act greedily w.r.t. to current optimal policy
    Otherwise, do directed exploration:
    while episode not ended **do**
        Sample goal state g from top K uncertain goal states in G
        Try to reach g for D steps
        If we reach or D steps are up, then do one step of random action
    end while
    Store episode experience into common replay buffer
    Sample minibatch B from common replay buffer
    Compute uncertainty for each state in B using U and add to G
    Update A with B
    Update E with B using HER
    Update U with B
end for

Learning a Goal-conditioned Policy

The first challenge is to get a goal-conditioned policy $\pi(s, g)$ to use for reaching goal states. We use an off-policy RL algorithm E to learn a UVFA/goal-conditioned policy (the green node in Figure 5.1) through the use of hindsight experience replay (HER), which is simultaenously being trained along with the existing off-policy RL algorithm A (the yellow node in Figure 5.1). Unfortunately, since we are now learning an approximation of $\pi(s, g)$, this means that we may fail to reach our target goal state g, and so we introduce a timestep limit D. We only follow $\pi(s, g)$ for up to D steps, after which we take one random action step. E is trained from the same common replay buffer as the existing algorithm A. In our experiments, we set D to be half or a quarter of the maximum episode length.

Tracking Uncertainty and Goal States

The next challenge is in picking the states with the largest uncertainty. We use a FIFO buffer which stores the states to be used as goal states, as well as their associated uncertainties (orange node in Figure 5.1). This buffer is also updated from minibatches sampled from the common replay buffer, where we use U to compute their uncertainties. For computational efficiency, we do not recompute the uncertainty for any states after they have been added to the buffer. This means that for older states, their associated uncertainty values will be stale; however this is not that detrimental since the consequence will be that there may be a slight delay to wait for older, more uncertain states to be pushed out of the buffer before newer, less uncertain states are picked. The staleness can be controlled by changing the maximum capacity of the buffer, N, though it is also important to keep the buffer large enough so that the uncertain states get many opportunities to be picked as goals. Due to function approximation, we may need to visit the same goal states over and over again to accumulate enough data to train, which is actually in line with keeping stale uncertainty values.

When picking goals from the goal buffer, we will sample uniformly at random from the top K most uncertain goal states. For simple domains where the uncertainty is accurate and not noisy, it is usually sufficient to use $K = 1$ or something very small, and thus always try to reach the most uncertain goal state. However for more complicated domains or much more noisy and inaccurate uncertainty estimates, setting K larger such as $K = 100$ results in much more robust behavior. Sampling uniformly from the top K rather than with probability proportional to their uncertainties allows the algorithm to explore a wider variety of goal states in case some of the uncertainty estimates are very inaccurate.

Computing Uncertainty

The uncertainty measure U (grey node in Figure 5.1) can be something simple, such as count-based bonuses in the tabular setting, or something complex like density models or bayesian models [Houthooft et al., 2016, Ostrovski et al., 2017]. In our experiments, we use a simple, learned, uncertainty measure, as we intend to show how directed exploration can take advantage and be more robust to uncertainty and reward bonus-based approaches. We train a network to predict the next state from the current state and action, and then use the prediction loss as the uncertainty. This is also trained from the same minibatches from the common replay buffer.

On-Policy Mixing

For each episode, we flip a coin to decide whether we will do directed exploration or just execute the current greedy policy from A. The reason we mix in on-policy execution in addition to the off-policy directed exploration samples is because function approximators work much better with on-policy data. Mixing in on-policy data allows for the function approximators to update values for states along the current greedy path and move it towards the optimal policy. The ratio of on-policy vs. directed exploration is a hyperparameter; however we found that 0.5 is a reasonable value to use in general and did not attempt further optimization.

5.4 Robustness Example

Here, we describe a simple example in which directed exploration is much more robust to the uncertainty estimate than reward bonus approaches. Suppose there are two states s_1, s_2 whose uncertainty estimates are close, but s_1 is always more uncertain than s_2. Then, a reward bonus approach would converge to an exploration policy that always explores s_1 and ignores s_2. However, with directed exploration, since both s_1 and s_2 have high uncertainty, they would both be sampled as goal states. Thus directed exploration would end up visiting both of these states.

If the uncertainty estimate is accurate, then it would not be a problem for reward bonus to focus solely on s_1, because eventually after visiting s_1, its uncertainty estimate would go down and drop below the uncertainty of s_2. Then reward bonus would try to visit s_2. However if the uncertainty estimate is not accurate, then it is possible that reward bonus would completely miss out on exploring s_2, whereas directed exploration is robust to these small relative differences and will end up exploring both.

5.5 Tabular Experiment

We first examine the tabular setting where we can compute everything exactly, so we can implement Algorithm 7 directly. All algorithms are implemented exactly using a table of values and we run value iteration till convergence at every step and so there is no approximation error.

Here we use a small toy gridworld to illustrate the benefit of committing to reaching states instead of relying on e-greedy. This can come up in settings where a small mistake

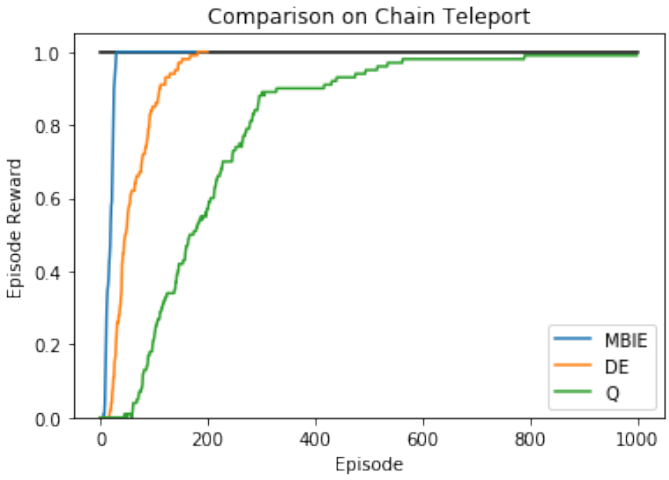


Figure 5.2: Comparison on gridworld with teleportation. Q-Learning with e-greedy and reward bonus (Q) takes much longer to converge than directed exploration (DE) or MBIE. These results are averaged over 100 runs, and show the evaluation performance when we run the greedy policy.

can be very costly, such as a teleport back to the beginning. The gridworld is a chain of 40 states, where the goal state is at the left end and the initial state is in the middle. Taking actions up or down always teleports you back to the middle, thus taking a random action in the process of reaching the goal can immediately send you back and lose all progress.

We compare 3 algorithms, MBIE [Strehl and Littman, 2008], Q-learning+bonus (Q), and directed exploration (DE) (Figure 5.2). MBIE is a near-optimal model-based algorithm in the tabular setting, and is our best-case baseline after tuning confidence intervals. We use exact count-based reward bonuses as our uncertainty measure. Q-learning+bonus simulates Q-learning with infinite replay and count-based reward bonuses (the same bonuses used for MBIE), where we do not have access to a model and thus cannot learn anything about (s, a) pairs that have never been visited. In such a case, e-greedy exploration is necessary in order to discover new (s, a) pairs not visited yet.

Figure 5.2 shows, unsurprisingly, that MBIE is the most efficient. Q-learning+bonus (Q) takes a long time to become consistent because there are many runs in which a random exploratory action results in a teleport back to the beginning. Thus trying to reach the goal state at the end of the chain becomes very inefficient. On the other hand, for directed exploration (DE), we only take a random exploratory action once we reach a desired goal state, meaning there is no chance to deviate and fail to reach the goal state due to ran-

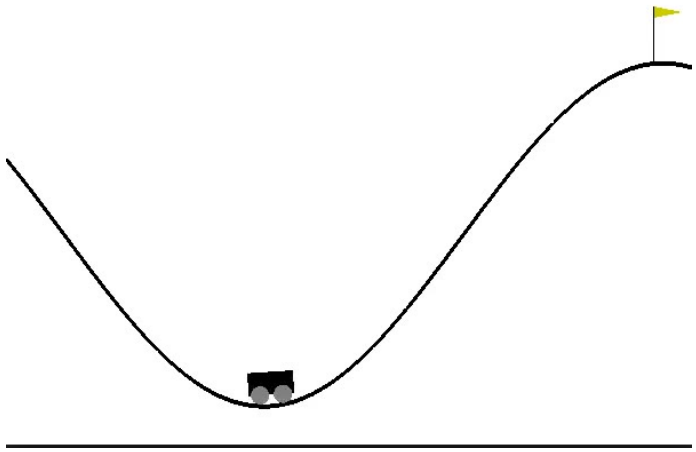


Figure 5.3: MountainCar Environment [Brockman et al., 2016]

domness. This commitment to reaching goal states is another aspect in which directed exploration can be more efficient than reward bonus approaches.

5.6 Function Approximation Experiment in a Small Domain

We now move into the function approximation setting by using DQN in the standard Mountain Car domain with discrete actions [Brockman et al., 2016]. We use the double DQN algorithm as the off-policy RL algorithm [Van Hasselt et al., 2016, Dhariwal et al., 2017] to learn the UVFA with HER, and as the second, regular DQN. To compute the uncertainty of a state, we use the loss from a simple L2 regression task for predicting the next state from the current state and action.

We compare 4 algorithms: DQN, DQN+bonus, DQN+directed with sampling uniformly random goals, and DQN+directed picking the most uncertain goals. For DQN and DQN+bonus, as well as the second regular DQNs present in the directed exploration algorithms, we use e-greedy exploration with a fixed $\epsilon = 0.1$. This is because even though DQN+bonus uses a reward bonus, it cannot assign reward bonuses to states that it has never seen before (bonuses are augmented to minibatches sampled from the replay buffer), and thus must still rely on some form of randomness to discover new states. Directed exploration does not need randomness when trying to reach goal states; instead it takes a step of random action after reaching a goal or reaching the timelimit D.

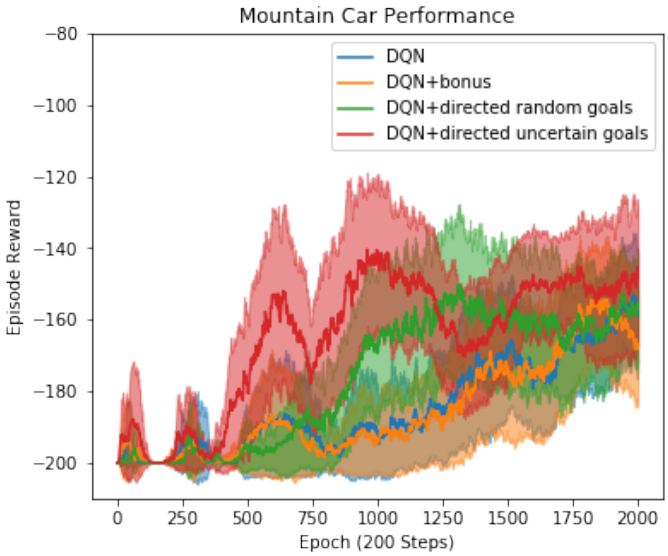


Figure 5.4: Episode Reward on Mountain Car. Result is over 10 runs, and show the evaluation performance when we run the greedy policy.

From Figure 5.4, we see that directed exploration using uncertainty (red) is the fastest at learning. Directed exploration with random goals (green) comes in second. Regular DQN (blue) and DQN with reward bonus (orange) seem to tie and perform the worst. These results indicate that directed exploration by itself, even just using random goals and not using uncertainty, is already better at finding the sparse reward. But the best results come from using the uncertainty to drive directed exploration towards the most uncertain states. This shows that the uncertainty, based on prediction loss, is in fact informative and useful for exploration. Reward bonus does not seem to be able to take advantage of this uncertainty at all.

To gather more insight, Figures 5.5 shows how much of the state space (after discretization) has been visited at least once. We see a much clearer difference in terms of how the algorithms are exploring. Both versions of directed exploration are much more efficient at covering the state space, whereas reward bonus seems to not be able to explore more than regular DQN. But the uncertainty is clearly important, as directed exploration using uncertainty to pick goals is significantly faster at exploration than when picking goals randomly. These results directly correlate with the actual performance of these algorithms in Figure 5.4.

The uncertainty based on prediction loss that we use is a very simple and natural ap-

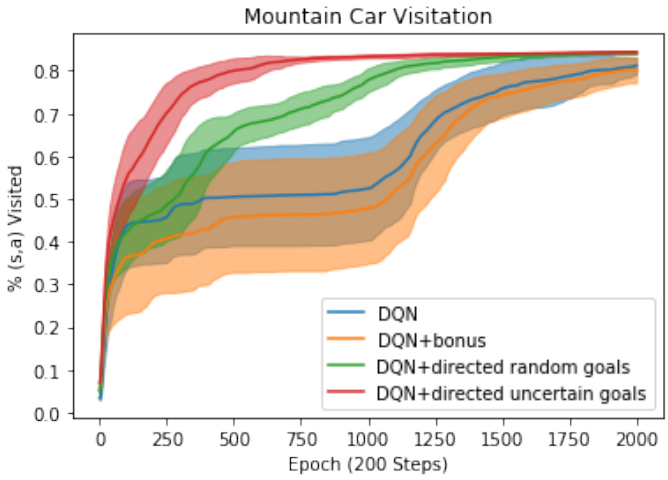


Figure 5.5: Percent of visited on Mountain Car over a discretized 10x10 grid. Result is over 10 runs.

proach that has been explored in prior works [Burda et al., 2018]. It is not a sophisticated, state-of-the-art measure of uncertainty, and thus is more noisy and less accurate. The reward bonus approach struggles and is not able to explore using this uncertainty. However, directed exploration is much more robust and is still able to take advantage of such a measure of uncertainty to explore extremely efficiently.

5.6.1 Experimental Details

Hyperparameters used for DQN were two hidden layers of size 64 with RELU activations. We used the Adam optimizer with a learning rate of 0.0001, target network is updated every 1000 steps. Prioritized replay is used with $\alpha = 0.4$, and $\beta = 1.0$. We used double DQN with a huber loss function. For the UVFA, we had a separate DQN with the same architecture, except we used a learning rate of 0.001 and the target is updated every 30 steps, and we used uniform replay. We found that the UVFA network could sustain a high learning rate and therefore converge much faster, unlike the regular DQN network which sometimes diverged with higher learning rates. For e-greedy, we used a fixed $\epsilon = 0.1$.

For reward bonus, we found that normalizing the uncertainty using running exponential averages of minimum and maximums, and then scaling by the reciprocal of the square root of the number of timesteps helps stablize performance.

For directed exploration, we set $K = 1$, meaning we always pick the most uncertain

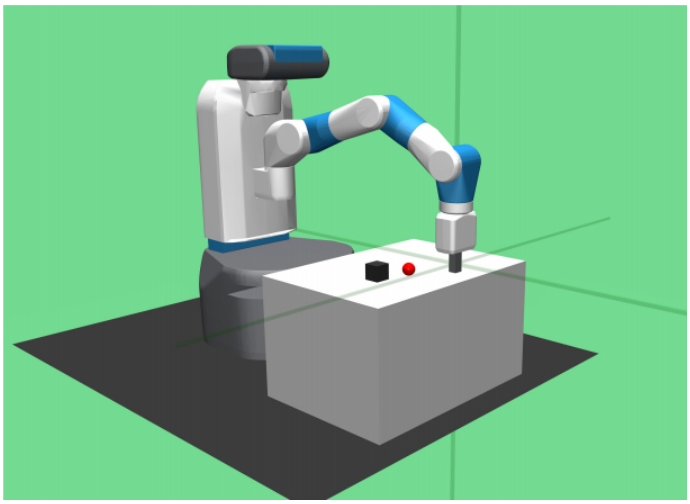


Figure 5.6: FetchPush Domain, image from [Plappert et al., 2018].

state from the goal buffer. We set the maxmimum length of a directed exploration attempt, $D = 50$, which is a quarter of the maximum episode length. The goal buffer has a capacity of 10000.

5.7 Function Approximation Experiment in a Large Domain

We implemented directed exploration in the FetchPush domain [Andrychowicz et al., 2017, Plappert et al., 2018] (Figure 5.6). This domain consists of a robotic arm in the middle of a table. The objective is to push the box to a desired goal position (3 dimensional) on the table. The initial position of the box is random. In the original formulation, the goal position is known and also randomized every episode, which is done in order to perform multi-goal learning. We have modified the domain to a fixed but unknown goal position to turn it into an ordinary MDP environment. For directed exploration, we also use object positions as goal states to be consistent with the domain. We rely on DDPG as the off-policy RL algorithm since actions are continuous Plappert et al. [2018]. The uncertainty we use is the mean squared loss from predicting the next object position from the current state and action.

Figure 5.7 compares the performance of regular DDPG (blue), DDPG with reward bonus (orange), DDPG with directed exploration but samples uniformly random goals

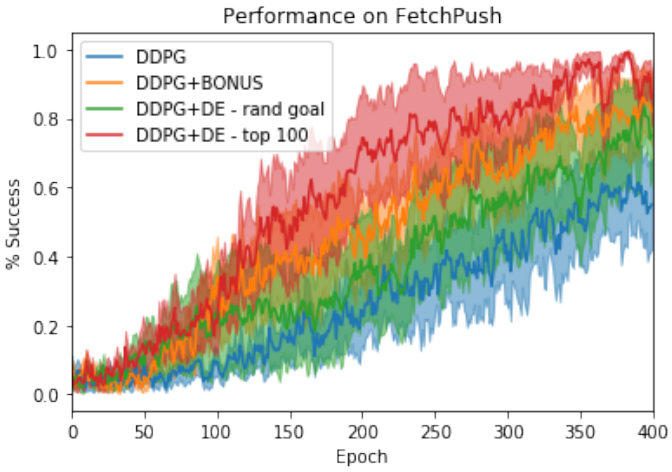


Figure 5.7: Percentage of success of reaching the true goal on FetchPush when comparing regular DDPG, DDPG and reward bonus, and DDPG with directed exploration both with picking random goals and top 100 most uncertain goals. Results are over 5 runs.

from the goal buffer (green), and DDPG with directed exploration but samples the top 100 most uncertain goals from the goal buffer (red). The performance is measured by evaluating the learned policy on 10 new episodes and calculating the percentage that successfully reach the true fixed goal [Andrychowicz et al., 2017].

We see from Figure 5.7 that directed exploration with sampling from the top 100 most uncertain goals (red) is able to learn the fastest compared to the others. Notably, it is more efficient than directed exploration with sampling random goals from the goal buffer (green). This indicates that the uncertainty we are using is actually meaningful, and driving exploration towards uncertain goals is more efficient than random goals. The reward bonus approach using the same uncertainty (orange) is still better than regular DDPG (blue), further confirming that the uncertainty is informative for exploration, but is not as efficient as directed exploration. Furthermore, for the reward bonus approach, we had to normalize the bonus by keeping a running minimum and maximum, and rescale it to decay as the reciprical of the number of timesteps in order to obtain the current performance; using the raw mean squared prediction loss simply results in identical performance to regular DDPG. On the other hand, for directed exploration, we used the raw losses and stored them in the goal buffer to use for sampling. These results show that directed exploration is able to much better take advantage and extract out meaningful information from the uncertainty estimate, making it much more robust than the reward bonus approach.

To develop more insight, we have also tracked how many different object positions

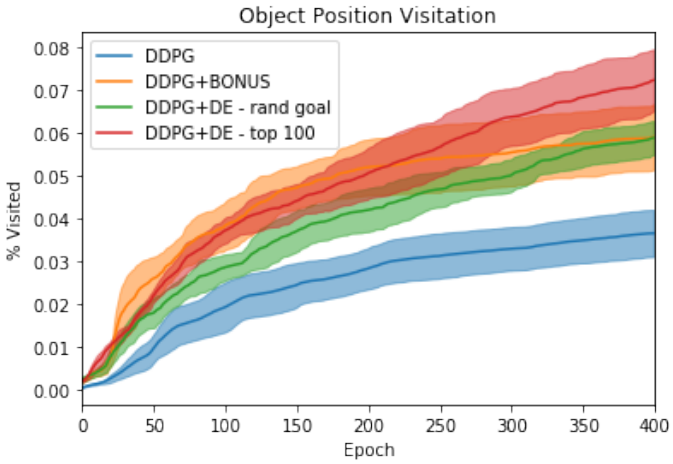


Figure 5.8: Percentage of visitation of possible discretized goal states on FetchPush when comparing regular DDPG, DDPG and reward bonus, and DDPG with directed exploration both with picking random goals and top 100 most uncertain goals. Results are over 5 runs.

have been explored in Figure 5.8. We discretized the possible object positions and tracked what percentage of total positions have been visited at least once. The results show that directed exploration using the uncertainty (red) is able to explore more than directed exploration with random goals (green), which confirms the effectiveness of using unertainty to guide exploration. The reward bonus approach (orange) initially falls short of directed exploration with uncertainty, before slightly overtaking it in the middle, but eventually still falls short. This is because once the reward bonus approach has found the sparse reward, then it will gradually stop exploration and start exploitation. It seems that even though in the middle, reward bonus is able to match and slightly exceed the amount of exploration that directed exploration is able to achieve, it is slower to start, and thus slower at first encountering the sparse reward. It would seem that the non-stationary reward bonuses slow down convergence to the optimal policy, whereas directed exploration avoids non-stationarity and is able to converge faster.

5.7.1 Experimental Details

We use the DDPG and HER implementations from Plappert et al. [2018], along with the same hyperparameters. Due to technical limitations, we did not run 19 parallel threads, but only 1. For directed exploration, we use two DDPG agents; one is goal-conditioned and is trained using HER; the second is regular unconditioned DDPG. The performance is

evaluated solely using the second regular DDPG.

For predicting the next object position from the current state and action, we use a neural network that takes as input the current state and action, passes it through 2 hidden layers of with 256 units each and RELU activations, and a final linear output layer that predicts the next object position (3 dimensional). This is trained using Adam with a learning rate of 0.001 and a batch size of 256, in tandem with DDPG.

For reward bonus, we keep a running exponential average of the minimum and maximum uncertainty from this mean squared loss, and rescale the raw loss to be between 0 and 1, before dividing it by the square root of the number of minibatch updates so far to better emulate a count-based bonus. This turns out to be much more effective than using the raw mean squared loss directly.

For directed exploration, the goal buffer has capacity 50000. We set $K = 100$, meaning we sample uniformly from the top 100 most uncertain states in the goal buffer. We set the maxmimum length of a directed exploration attempt, D, to be 25, half of the episode length. When we use the goal-conditioned DDPG, we use the greedy policy and disable any action noise.

5.8 Discussion of Results

The tabular setting showed how committing to reaching goal states with high uncertainty was much more efficient than the combination of e-greedy and reward bonus. However, if e-greedy could be avoided, like using a model-based method such as MBIE, then it is possible to achieve more efficient exploration. But in the function approximation case, learning a good dynamics model is a big challenge, and the current most successful methods actually combine model-based and model-free, resulting in still using some form of randomization in the actions [Nagabandi et al., 2018]. Thus, commitment in directed exploration may still result in increased efficiency for exploration. However, commitment only has the largest impact if small mistakes are extremely costly, and this is not always the case for many domains. If small mistakes are not that bad, then commitment wil not result in a drastic increase in efficiency for exploration.

The function approximation experiments showed how directed exploration is more efficient at exploration, but more importantly, can take advantage of a simple uncertainty measure to explore even better. Before we discuss the uncertainty, we will first discuss directed exploration itself. The results on Mountain Car when sampling goal states at random do show that directed exploration itself is already faster at exploration. However the

results on FetchPush show that sampling random goal states is not as good at exploration as reward bonus. This difference highlights one of the biggest challenges for directed exploration: learning the goal-conditioned policy $\pi(s, g)$. In Mountain Car, due to the simplicity of the domain, we are able to learn $\pi(s, g)$ very efficiently, and thus directed exploration is able to start exploring efficiently very early on. On FetchPush, it takes much longer (Figure 5.8), and so directed exploration with random goals is not able to explore a lot until later on in training. We hypothesize that if we are able to train a $\pi(s, g)$ faster, whether using a different architecture or improving the training process, then directed exploration may become much more efficient at exploration. This also highlights one of the limitations of directed exploration, in that if we are using a model-based algorithm like in the tabular setting, then there might not be a strict benefit over the reward bonus approach because the augmented reward function is being updated and changes are always fully propagated; the only potential benefit for directed exploration would be its robustness to the uncertainty measure due to committing to goals.

We will now discuss directed exploration with uncertainty. For both Mountain Car and FetchPush, using the uncertainty to sample goals greatly increased the efficiency of exploration over just using random goals. This indicates that the uncertainty we used is in fact informative about what parts of the state space we need to explore, even though our uncertainty is simply the prediction loss from predicting the dynamics. However when we compare performance to reward bonus, we see that in Mountain Car, reward bonus is not able to take advantage of the uncertainty at all. In FetchPush, the reward bonus is able to take some advantage of the uncertainty and improve performance, but not as much as directed exploration. Even though in FetchPush, the goal-conditioned policy $\pi(s, g)$ is being learned quite slowly, it seems that trying to reach those goal states with high uncertainty makes a significant difference to exploration efficiency. These results show that directed exploration is robust to the uncertainty measure and can better take advantage to explore more than reward bonus. Reward bonus seems to be much slower to update and propagate the uncertainty.

There is an additional computation cost for directed exploration since we need to maintain two off-policy RL algorithms. But there could be ways to reduce the cost, such as potentially sharing parts of the network between the two algorithms. Alternatively, the two algorithms can easily be updated in parallel, taking advantage of multithreading to speed up computation.

5.9 Related Work

Recently, Uber introduced the Go-Explore algorithm [Ecoffet et al., 2019], which has achieved incredible results on two hard exploration Atari games: Montezuma's Revenge and Pitfall. One of the key components of their algorithm is to use directed exploration to revisit states that looked promising for further exploration. However their approach differs from ours as they are not learning a general goal-conditioned policy to figure out how to reach states, but they are replaying past trajectories due to being in a deterministic environment. Our approach is general and fully support stochastic environments. Furthermore their algorithm also has many other components put together and it is not clear the effect of directed exploration itself.

Directed exploration can be considered a particular instantiation of goal generation from higher level policies in hierarchical reinforcement learning. Prior work has focused on uniformly random or expert guided goal generation for learning sub task and task structure, but we look into uncertainty-based goal generation explicitly for better exploration [Held et al., 2018, Nachum et al., 2018]. Another difference is that our directed exploration component is completely separated from the existing off-policy algorithm, and it only contributes samples to the common replay buffer. This allows our component to be more modular, and more easily added to existing off-policy algorithms.

Chapter 6

Conclusion

6.1 Overview

This thesis has shown how directed exploration can be a useful exploration approach that can lead to new algorithms that are provably sample efficient in various settings of practical interest, and a useful practical tool that can lead to new practical algorithms with function approximation that are sometimes more efficient and robust than using the reward bonus approach.

In chapter 2, we have the first formal analysis of RL in concurrent tasks. Our theoretical results indicate that with sharing samples from copies of the same MDP we can achieve a linear speedup in the sample complexity of learning, and our simulation results show that such speedups can also be realized empirically. We can also achieve a speedup under the relaxation that there are a finite number of different types of MDPs with mild separability and diameter assumptions, where we can use directed exploration to find distinguishing states and cluster identical MDPs together, even without knowing the number of distinct MDPs nor which are identical.

In chapter 3, we have introduced the COMRLI algorithm, which to our knowledge the first PAC RL algorithm for learning across a series of tasks drawn from a set of continuous-state, discrete action Markov decision processes. Its bound on the sample complexity can be significantly smaller in later tasks, and is independent of the covering number for the state–action space. This shows that transferring knowledge in continuous-state RL can provably reduce the amount of experience needed to make good decisions. We also provided preliminary but encouraging evidence that our approach may be helpful in practice, showing good performance in one benchmark domain against a state-of-the-art

policy gradient method for multi-task RL.

In chapter 4, we presented the first algorithm for performing feature selection while solving factored MDPs whose sample complexity scales exponentially with the in-degree of the necessary features, potentially an exponential improvement over scaling with the in-degree of all features.

In chapter 5, we presented a general directed exploration algorithm for RL with function approximation, and showed how it can be sometimes more efficient at exploration and faster at learning than using a reward bonus when using the approximate uncertainty measure of next-state prediction loss. This is due to directed exploration committing to reach states, and being more robust to noisy and inaccurate uncertainty estimates.

6.2 Future Work

The proposed algorithms in chapters 2-4 rely on various assumptions that could potentially be further relaxed and lead to more general algorithms. In the concurrent MDP setting, the algorithm depends on the assumption of a definite and large gap in the transition and reward dynamics between different types of MDPs is quite important, but also a limitation of the algorithm. Perhaps there may be some method to avoid making such an assumption, though most likely it would require using an alternative framework such as the regret framework instead of PAC since PAC is binary when it comes to classify a timestep as a mistake or near-optimal. In the factored MDP setting with feature selection, the algorithm relies on the superset assumption to guarantee progress in eliminating unnecessary features. Again, while the superset assumption is not too strong, perhaps with a different analysis or framework, it may be possible to relax such an assumption even more.

In addition to the 3 settings in chapters 2-4, there may be many other settings in which some form of directed exploration can help derive a novel sample efficient algorithm. Directed exploration can allow us to choose which states are visited, potentially simplifying theoretical analyses to focus on just those states of interest.

Furthermore, the idea from feature selection of leveraging negative information through the failure to reach expected reachable goals has much more potential. It might be helpful for practical algorithms to help diagnose an error in its model model class, an error in an assumption, or perhaps that its model does not have the capacity to accurately represent the environment.

Also, in practice, directed exploration has a lot of potential for more efficient exploration. The work in chapter 5 can still be greatly expanded and generalized. While it is

not clear that all settings may benefit from directed exploration, there may be some larger and more complex domains, perhaps even partially observable domains where directed exploration can greatly improve exploration and performance. However, the challenge will be how to efficiently learn a goal-conditioned policy, which is itself also an interesting direction to pursue. A challenge that comes up in the partially observable setting is how to define goal states, since the underlying states are no longer observable; it then becomes important to be able to learn a suitable state representation as well.

This thesis has shown how directed exploration is a useful exploration technique that can lead to sample efficient bounds and better practical performance, with much potential for future expansions.

Bibliography

Yasin Abbasi-Yadkori, David Pal, and Csaba Szepesvari. Online-to-confidence-set conversions and application to sparse stochastic bandits. In *AISTATS*, volume 22, pages 1–9, 2012. 4.3

Shipra Agrawal and Randy Jia. Optimistic posterior sampling for reinforcement learning: worst-case regret bounds. In *Advances in Neural Information Processing Systems*, pages 1184–1194, 2017. 1.1

Haitham B. Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 1206–1214. JMLR Workshop and Conference Proceedings, 2014. URL `http://jmlr.org/proceedings/papers/v32/ammar14.pdf`. 3.1, 3.12

Haitham Bou Ammar, Rasul Tutunov, and Eric Eaton. Safe policy search for lifelong reinforcement learning with sublinear regret. *The Journal of Machine Learning Research (JMLR)*, 2015. 3.1

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017. 5.1, 5.2.4, 5.7, 5.7

Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. *Advances in Neural Information Processing Systems*, 19:49, 2007. 4.2

Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 89–96, 2009. 3.4

Peter L Bartlett and Ambuj Tewari. Regal: A regularization based algorithm for reinforcement learning in weakly communicating mdps. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 35–42. AUAI Press, 2009. 4.1

Haitham Bou-Ammar, Jose Marcio Luna, Eric Eaton, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015. 3.1

Ronen I. Brafman and Moshe Tennenholtz. R-max—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3: 213–231, October 2002. 1.3, 2.1, 2.2

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003. 1.1

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. (document), 5.3, 5.6

E. Brunskill and L. Li. Sample complexity of multi-task reinforcement learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013a. 2.1, 2.3

Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence*, 2013b. 3.1, 3.4, 3.8, 3.9, 4

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018. 1.1, 1.2, 1.4, 5.1, 5.6

Doran Chakraborty and Peter Stone. Structure learning in ergodic factored MDPs without knowledge of the transition function's in-degree. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 737–744, 2011. 4.1, 4.2, 4.3, 4.4, 4.5.1

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011. 1.1

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017. 5.6

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303, 2000. 4.6.2

Maria Dimakopoulou and Benjamin Van Roy. Coordinated exploration in concurrent reinforcement learning. *arXiv preprint arXiv:1802.01282*, 2018. 2.7

Maria Dimakopoulou, Ian Osband, and Benjamin Van Roy. Scalable coordinated exploration in concurrent reinforcement learning. *arXiv preprint arXiv:1805.08948*, 2018. 2.7

Carlos Diuk, Lihong Li, and Bethany R Leffler. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 249–256. ACM, 2009. 4.1, 4.3, 4.4, 4.5.2, 4.6.4, 4.6.4

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019. 5.9

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, 2004. 2.1

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017. 1.1, 5.1

Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 881–888, 2011. 4.3

Robert Grande, Thomas Walsh, and Jonathan How. Sample efficient reinforcement learning with gaussian processes. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1332–1340. JMLR Workshop and Conference Proceedings, 2014. URL `http://jmlr.org/proceedings/papers/v32/grande14.pdf`. 3.8

Zhaohan Guo and Emma Brunskill. Concurrent PAC RL. In *AAAI*, pages 2624–2630, 2015a. 2, 3.4, 3.5, 4.1

Zhaohan Guo and Emma Brunskill. Concurrent PAC RL. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015b.

Zhaohan Guo, Philip S Thomas, and Emma Brunskill. Using Options and Covariance Testing for Long Horizon Off-Policy Policy Evaluation. In *Advances in Neural Information Processing Systems*, pages 2492–2501, 2017.

Zhaohan Daniel Guo and Emma Brunskill. Sample Efficient Learning with Feature Selection for Factored MDPs. In *EWRL*, 2018. 4

Zhaohan Daniel Guo, Shayan Doroudi, and Emma Brunskill. A PAC RL Algorithm for Episodic POMDPs. In *Artificial Intelligence and Statistics*, pages 510–518, 2016.

Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Pion, Bernardo A. Pires, Toby Pohlen, and Rémi Munos. Neural Predictive Belief Representations. *arXiv preprint arXiv:1811.06407*, 2018.

Assaf Hallak, COM François Schnitzler, Timothy Mann, and Shie Mannor. Off-policy model-based learning under unknown factored dynamics. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 711–719, 2015. 4.3, 4.6.2, 4.6.2

David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *ICML*, 2018. 5.1, 5.9

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016. 1.4, 5.1, 5.3.2

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010a. 2

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010b. 4.1

Sham Kakade, Michael Kearns, and John Langford. Exploration in metric state spaces. In *ICML*, volume 3, pages 306–312, 2003. 1

Sham M. Kakade. *On the Sample Complexity of Reinforcement Learning.* . PhD thesis, University College London, 2003. 2.1, 2.5.3

Michael Kearns and Daphne Koller. Efficient reinforcement learning in factored mdps. In *IJCAI*, volume 16, pages 740–747, 1999. 4.1, 4.2, 13

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002a. 4.2

Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2–3):209–232, 2002b. 1.3, 2.1, 2.2, 2.3, 2.5.1

Mark Kroon and Shimon Whiteson. Automatic feature selection for model-based reinforcement learning in factored MDPs. In *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*, pages 324–330. IEEE, 2009. 4.3

Tor Lattimore, Marcus Hutter, and Peter Sunehag. The sample-complexity of general reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 28–36, 2013. 2.2

Alessandro Lazaric and Marcello Restelli. Transfer from multiple mdps. In *Advances in Neural Information Processing Systems*, pages 1746–1754, 2011. 2.1, 3.1

M. Lewis. A dynamic pricing approach to customer relationship pricing. *Management Science*, 51(6):986–994, 2005. 2.1

Lihong Li. *A Unifying Framework for Computational Reinforcement Learning Theory*. PhD thesis, Rutgers University, New Brunswick, NJ, 2009a. 2.3, 2.5.3

Lihong Li. *A unifying framework for computational reinforcement learning theory*. PhD thesis, Rutgers, The State University of New Jersey, 2009b. 3.10

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 5.2.2

Ran Liu and Kenneth R Koedinger. Variations in learning rate: Student classification based on systematic residual error patterns across practice opportunities. 3.2

Yao Liu, Zhaohan Guo, and Emma Brunskill. Pac Continuous State Online Multitask Reinforcement Learning with Identification. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 438–446. International Foundation for Autonomous Agents and Multiagent Systems, 2016. 3, 4.1

Andrew Kachites McCallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester, 1996. 4.3

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013. 1.1, 1.1, 1.4, 4.1, 5.1, 5.2.1

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 5.1

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 3307–3317. Curran Associates, Inc., 2018. 5.1, 5.9

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018. 5.8

Trung Thanh Nguyen, Zhuoru Li, Tomi Silander, and Tze-Yun Leong. Online feature selection for model-based reinforcement learning. In *ICML (1)*, pages 498–506, 2013. 4.3

Stefanos Nikolaidis, Ramya Ramakrishnan, Keren Gu, and Julie Shah. Efficient model learning from joint-action demonstrations for human-robot collaborative tasks. In *HRI*, pages 189–196, 2015. 3.2

Ronald Ortner, Odalric-Ambrym Maillard, and Daniil Ryabko. Selecting near-optimal approximate state representations in reinforcement learning. In *Algorithmic Learning Theory*, pages 140–154. Springer, 2014. 4.3

Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2701–2710. JMLR. org, 2017. 1.1

Ian Osband, Dan Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. In *NIPS*, pages 3003–3011, 2013. 1.1

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016. 1.1, 5.1

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *arXiv preprint arXiv:1806.03335*, 2018. 1.1

Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017. 1.1, 1.4, 5.1, 5.3.2

Christopher Painter-Wakefield and Ronald Parr. Greedy algorithms for sparse reinforcement learning. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 1391–1398, New York, NY, USA, July 2012. Omnipress. ISBN 978-1-4503-1285-1. 4.3

Jason Pazis and Ronald Parr. Pac optimal exploration in continuous space markov decision processes. In *AAAI Conference on Artificial Intelligence*. Citeseer, 2013. 3.1, 3.4, 2, 3.6, 3.10, 3.10, 3.10

Jason Pazis and Ronald Parr. Efficient pac-optimal exploration in concurrent, continuous state mdps with delayed updates. In *AAAI*, pages 1977–1985, 2016. 2.7

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018. (document), 5.6, 5.7, 5.7.1

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015a. 5.1, 5.2.3

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b. 1.1

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015. 1.1, 1.1

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 1.1, 5.1

David Silver, Leonard Newnham, David Barker, Suzanne Weller, and Jason McFall. Concurrent reinforcement learning from customer interactions. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 924–932, 2013. 2.1, 2.4

Alexander L. Strehl and Michael L. Littman. An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8): 1309–1331, 2008. 1.1, 2.1, 2.2, 2.3, 2.3, 5.5

Alexander L. Strehl, Lihong Li, and Michael L. Littman. Incremental model-based learners with formal learning-time guarantees. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 485–493, 2006. 1.3, 2.2, 2.3, 2.5.2

Alexander L Strehl, Carlos Diuk, and Michael L Littman. Efficient structure learning in factored-state mdps. In *AAAI*, volume 7, pages 645–650, 2007. 4.1, 4.4, 4.6.2

Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444, 2009. 4.1

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017. 1.1, 1.4, 5.1

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009. 2.1

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016. 5.6

Ronald J Williams and Leemon C Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Citeseer, 1993. 3.10, 3.10

Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-task learning for classification with dirichlet process priors. *The Journal of Machine Learning Research*, 8:35–63, 2007. 2.1