SCALING UP REINFORCEMENT LEARNING WITHOUT SACRIFICING OPTIMALITY BY CONSTRAINING EXPLORATION

A Dissertation

by

TIMOTHY ARTHUR MANN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Yoonsuck Choe
Committee Members,	Robin Murphy
	Ricardo Gutierrez-Osuna
	John Buchannan
	Peter Stone
Department Head,	Duncan Walker

December 2012

Major Subject: Computer Science

Copyright 2012 Timothy Arthur Mann

ABSTRACT

The purpose of this dissertation is to understand how algorithms can efficiently learn to solve new tasks based on previous experience, instead of being explicitly programmed with a solution for each task that we want it to solve. Here a task is a series of decisions, such as a robot vacuum deciding which room to clean next or an intelligent car deciding to stop at a traffic light. In such a case, state-of-the-art learning algorithms are difficult to employ in practice because they often make thousands of mistakes before reliably solving a task. However, humans learn solutions to novel tasks, often making fewer than a couple of mistakes, which suggests that efficient learning algorithms may exist. One advantage that humans have over state-of-the-art learning algorithms is that, while learning a new task, humans can apply knowledge gained from previously solved tasks. The central hypothesis investigated by this dissertation is that learning algorithms can solve new tasks more efficiently when they take into consideration knowledge learned from solving previous tasks. Although this hypothesis may appear to be obviously true, what knowledge to use and how to apply that knowledge to new tasks is a challenging, open research problem.

I investigate this hypothesis in three ways. First, I developed a new learning algorithm that is able to use prior knowledge to constrain the exploration space. Second, I extended a powerful theoretical framework in machine learning, called Probably Approximately Correct, so that I can formally compare the efficiency of algorithms that solve only a single task to algorithms that consider knowledge from previously solved tasks. With this framework, I found sufficient conditions for using knowledge from previous tasks to improve efficiency of learning to solve new tasks and also identified conditions where transferring knowledge may impede learning. I

present situations where transfer learning can be used to intelligently constrain the exploration space so that optimality loss can be minimized. Finally, I tested the efficiency of my algorithms in various experimental domains.

These theoretical and empirical results provide support for my central hypothesis. The theory and experiments of this dissertation provide a deeper understanding of what makes a learning algorithm efficient so that it can be widely used in practice. Finally, these results also contribute the general goal of creating autonomous machines that can be reliably employed to solve complex tasks.

DEDICATION

To my wife Laura

ACKNOWLEDGEMENTS

I could not have completed this dissertation without the support of many people, including my advisor Yoonsuck, my committee, my wife, and my family and friends. I am grateful for all the help and patience they have shown me throughout my time as a PhD student (and beyond). My experience as a graduate student would not have been nearly as fun or complete without their help.

My advisor, Yoonsuck, has guided me through many successful and unsuccessful ideas and helped to shape my philosophical and research interests. We initially met when I was a student in the National Science Foundation's 2006 Research Experience for Undergraduates summer program at Texas A&M University. He introduced me to the world of academic research, which later helped me gain admission to the Computer Science & Engineering graduate program. Yoonsuck allowed me to investigate so many interesting ideas and we have had many great conversations that have affected the way that I think about artificial intelligence and consciousness. Working with him has allowed me to satisfy my interest in philosophy, mathematics, and engineering. I am grateful for his help and advice.

My wife Laura and my family have given me the love and support needed while writing my dissertation, which, as any one who has written a dissertation or is married to someone who has, knows is a lot. I am grateful to my parents for raising me to be interested in the world's many mysteries and providing a loving home that allowed me to grow into the person I am today. My wife Laura has encouraged me during the whole process, and I am so thankful that she was willing to leave her job in New York and moved all the way to Texas so that I could pursue a PhD. I am fortunate to have such a wonderful wife.

My committee has been extremely helpful. They have greatly contributed to my graduate education and provided valuable discussion and comments that have helped to make this dissertation stronger. Dr. Gutierrez-Osuna's pattern analysis course was an extremely helpful introduction to a breadth of machine learning techniques. Dr. Murphy's grounded advice and probing questions have helped me to identify some of the limitations of modern machine learning research. Dr. Buchanan has provided me with insight about biologically inspired models of motor control. I am grateful for Dr. Stone's invaluable insights and discussions on reinforcement learning and transfer learning, and the constructive comments and help that I have received from his current and former students.

I have also had many other mentors and professors that have helped me with my education and research. I am grateful for the many professors at SUNY Potsdam that helped to provide me with an excellent undergraduate education and experience. I also benefited greatly by interning at the Naval Research Laboratory where I worked with David Talmage in the Distributed Computing Group. David's mentorship helped me to gain confidence and skills that were critical for success in graduate school.

During my time as a PhD student, I was fortunate to have the opportunity to work on a research project with Professor Minho Lee at Kyungpook National University in Daegu, South Korea as part of the National Science Foundation's East Asia Pacific Summer Institutes Program. I am extremely grateful for the opportunity and fun that I had while working with Professor Lee and his students Sungmoon and Yunjung.

Finally, I would like to acknowledge the financial support that I have been given by the Department of Computer Science & Engineering as a teaching assistant, the Texas A&M Dissertation Fellowship, and the National Science Foundation. Without

the support of their programs, I would no have been able to complete this dissertation.

NOMENCLATURE

MBP	Multiarmed Bandit Problem
MDP	Markov Decision Process
RL	Reinforcement Learning
TL	Transfer Learning
Ω	A Markov Decision Process or Multiarmed Bandit Problem instance
S_Ω	The set of states in task Ω
N_Ω	The number of states in task Ω ($\lvert S_\Omega \rvert$)
A_Ω	The set of actions in task Ω
K_Ω	The number of actions in task Ω ($\lvert A_\Omega \rvert$)
T_Ω	The transition probabilities of task Ω
R_Ω	The (possibly stochastic) reward function of task Ω
R_{MAX}	The maximum possible reward for a single timestep
R_{MIN}	The minimum possible reward for a single timestep
V_{MAX}	The maximum possible long-term value of a state
π	A policy mapping states to actions
π^*_Ω	The optimal policy for task Ω
V^π_Ω	The value function for policy π in task Ω
V^*_Ω	The optimal value function for task Ω
Q^π_Ω	The action-value function for policy π in task Ω
Q^*_Ω	The optimal action-value function for task Ω

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGEMENTS v

TABLE OF CONTENTS ix

LIST OF FIGURES xiii

LIST OF TABLES xx

1. INTRODUCTION 1
   1.1 The Problem 1
   1.2 Motivation for Studying Transfer Learning 2
   1.3 Approach 4
   1.4 Outline of the Dissertation 4

2. BACKGROUND 7
   2.1 Multiarmed Bandit Problems (MBPs) 8
      2.1.1 Algorithms for MBPs 9
      2.1.2 Evaluation of MBP Algorithms 11
      2.1.3 Limitations of the MBP Model 16
   2.2 Markov Decision Processes (MDPs) 17
      2.2.1 Planning (Dynamic Programming) in MDPs 19
      2.2.2 Reinforcement Learning (RL) in MDPs 21
      2.2.3 Evaluation of RL Algorithms in MDPs 26
      2.2.4 Limitations of the MDP Model 32
   2.3 Summary 32

3. REVIEW OF TRANSFER LEARNING 33

3.1 Supervised Transfer Learning 35
3.2 Reinforcement Transfer Learning 37
3.3 Multitask Learning 38
3.4 Transfer via Intertask Mappings 38
3.5 Advantages of Transfer Learning 39
3.6 Challenges 40
3.6.1 What Knowledge should be Transferred? 41
3.6.2 How can Prior Knowledge be Learned? 42
3.6.3 How can Prior Knowledge be Transferred? 43
3.6.4 How should we Evaluate a Transfer Learning System? 43
3.7 Summary 45
4. TARGETED EXPLORATION BY PRUNING STATES 46
4.1 Background 46
4.2 Algorithm: STAR-MAX 48
4.3 Experiment: Simple Case 53
4.4 Experiment: Dropping Arbitrary States 56
4.5 Experiment: Learning from Demonstration 58
4.6 Discussion 58
4.7 Summary 61
5. ANALYSIS OF PRUNING ACTIONS 62
5.1 Background 63
5.2 Explicit Action Pruning 68
5.2.1 R-MAX 70
5.2.2 Delayed Q-learning 75
5.3 Implicit Action Pruning 85
5.3.1 R-MAX 91
5.3.2 Delayed Q-learning 93
5.4 Summary 95

6. ANALYSIS OF ACTION-VALUE TRANSFER 96
6.1 Action-value Transfer Framework 96
6.2 Analysis of Action-value Transfer 97
6.2.1 Analysis of Source Task Sample Complexity 98
6.2.2 Analysis of Target Task Sample Complexity 102
6.3 Experiments & Results 107
6.3.1 Experiment: One State Transfer 112
6.3.2 Experiment: Variable β 117
6.3.3 Experiment: Scaling Up in a Gridworld Tasks 119
6.3.4 Experiment: Scaling Up in an Inverse Kinematic Task 120
6.4 Discussion 127
6.5 Summary 129
7. ANALYSIS OF MULTITASK LEARNING 130
7.1 Background 131
7.2 Learning Objective and Approach 134
7.3 Complexity of a Domain of Tasks 136
7.3.1 Deterministic Measures of Domain Complexity 138
7.3.2 Stochastic Measures of Domain Complexity 141
7.4 Algorithm: Evolving Exploration Tables (EET) 143
7.5 Algorithm: Learning Maximum Values (LMV) 143
7.6 Algorithm: Learning Exploration Tables (LET) 146
7.7 Experiments & Results 154
7.7.1 Experiment: Evolving an Exploration Table 157
7.7.2 Experiment: Learning an Exploration Table 159
7.8 Discussion 164
7.9 Summary 164
8. DISCUSSION 166
8.1 Action Pruning 166
8.1.1 Findings & Contributions 166

8.1.2 Limitations 166
8.1.3 Future Work & Open Questions 167
8.2 Action-value Transfer 167
8.2.1 Findings & Contributions 167
8.2.2 Limitations 168
8.2.3 Future Work & Open Questions 169
8.3 Multitask Learning 169
8.3.1 Findings & Contributions 169
8.3.2 Limitations 170
8.3.3 Future Work & Open Questions 170
9. CONCLUSIONS 172
REFERENCES 174

LIST OF FIGURES

FIGURE Page

1.1 (a) Fetal motor learning problem. (b) Experimental prenatal motor learning problem. 2

1.2 (a) Agent Centered representation where the origin of the coordinate system is part of the agent itself. (b) World Centered representation where the origin is outside of the agent. 3

2.1 A reinforcement learning agent is embedded in an environment. Each time the agent acts, the agent's state in the environment changes and it receives a reward. Adapted from [1]. 7

2.2 An example multiarmed bandit problem with $K = 4$ actions $\{a_1, a_2, a_3, a_4\}$. Each action is associated with a reward distribution. 8

2.3 A simple MDP where every state $\{s_1, s_2, s_3\}$ cannot be reached in a single episode. 29

2.4 A simple MDP where reaching state s_2 from s_1 may take many tries, if η is very small. 29

3.1 An agent constructs an instance of a source task algorithm to interact with a source task. Interaction with the source task generates knowledge, which is stored in the agent's library. The knowledge from the agent's library is transferred in a form that is meaningful in the context of the target task and that information is used to construct a target algorithm. The target algorithm is biased by the transferred knowledge, which will hopefully decrease the time needed to learn. . . 34

3.2 The sample complexity of transfer learning is more complex than sample complexity in a single task setting. The number of samples is distributed over the source and target task. 44

4.1 (*a*) Example gridworld with exploration envelope shaded in gray. The start state is denoted by a green square, the agent by a blue square, and the target state is denoted by a red square. (*b*) Example visitation table by the STAR-MAX algorithm (lighter cells were visited more frequently). (*c*) Comparison of cumulative reward between multiple RL algorithms as the number of states in the gridworld increases. Notice that the cumulative rewards are negative because the task gives negative rewards to encourage the algorithm to move to the target state as quickly as possible. STAR-MAX scales much better than Q-learning and R-MAX. Adapted from Mann and Choe [54]. 54

4.2 As the number of states in the STAR-MAX exploration envelope increase, the cumulative reward decreases linearly. Adapted from Mann and Choe [54]. 55

4.3 The red herring domain introduced by [2]. This gridworld domain contains two red herring states (denoted by R) with small rewards and one goal state (denoted by G) that gives a large reward. Learning algorithms that explore too little may settle on one of the suboptimal red herring states rather than finding the goal state. 56

4.4 Cumulative reward increases simply by randomly dropping states (or state-action pairs) from the exploration envelope in the Red Herring domain [2] up to about 60% of the number of states. Adapted from Mann and Choe [54]. 57

4.5 (*a*) Learning to perform a figure-eight from demonstrations by a teacher. (*b*) A state visitation table generated by a teacher. (*c*) The exploration envelope extracted from (*b*) by selecting the states visited more frequently than the 95th percentile. (*c*) Comparison of average reward at the figure-eight task by Q-learning, R-MAX, and STAR-MAX initialized with the exploration envelope in (*c*). Notice that STAR-MAX quickly achieves high average reward at the task, while R-MAX thoroughly explores before settling on a reasonable policy. Q-learning slowly improves its policy. Adapted from Mann and Choe [54]. 59

5.1 (a) A full exploration table with no state-action pruning. All state-action pairs may need to be explored to guarantee learning a near-optimal policy. (b) The 4-state, 3-action MDP described by the full exploration table. (c) A sparse exploration table with pruned state-action pairs. (d) The 4-state, 3-action MDP described by the sparse exploration table. 68

5.2 The "good" actions at a state are the actions with optimal action-values that are α-close to the optimal action-value at that state. . . . 86

5.3 Example of poorly estimated action-values at a single state. Notice that a greedy policy would select the optimal action b_1 in this case despite the fact that the action-value estimates are extremely poor. . 88

5.4 Weak admissible heuristic applied to a one state task with six actions. The weak admissible heuristic only needs to optimistically initialize the action-value for a single near-optimal action at each state. 89

6.1 A reset MDP is a chain of N states. Action a^* transitions the agent to the next state (or state 1 if the current state is N) with probability 0.8 and stays in the current state with probability 0.2, while all other actions reset the agent to state 1. The agent only receives a reward for taking action a^* in state N. 99

6.2 Transfer from a one-state source task with three actions to a one-state target task with six actions. Despite the transferred action-values severely underestimating the optimal action b_1 and severely overestimating the lowest valued action b_6,, and OFU exploration strategy can still converge to a near-optimal policy (i.e., b_2). 103

6.3 A double reset MDP is similar to a reset MDP (Figure 6.1), except that it has two chains of states. The final state of the first chain ($N/2$) gives reward $R_{\text{MAX}}/2$ when action a^* is executed, while the final state of the second chain N gives reward R_{MAX} when b^* is executed. 109

6.4 The Red Herring task (a) [2] contains a goal state G that gives a reward $+20$ and two red herring states R that give a reward 0, while all other states give the reward -1. The Taxi task (b) [3] requires the agent to pick up a passenger at one of four colored locations and drop the passenger off at its desired location. 110

6.5 Transfer from a two-joint arm (source task) to a three-joint arm (target task). . . . 111

6.6 Comparison between three possible intertask mappings in the one state transfer scenario. (*a*) A poor intertask mapping, denoted BAD. The transferred action-values underestimate the near-optimal actions and overestimate the worst action b_6. (*b*) An intertask mapping that induces an admissible heuristic, denoted AH. The transferred action-values are all overestimated. (*c*) An intertask mapping that induces a weak admissible heuristic, denoted WAH. The transferred action-values overestimate one (but not both) near-optimal action. 113

6.7 Comparison of cumulative reward for 100 different runs of each Delayed Q-learning transfer condition under the one state transfer scenario. Each algorithm was run in the target task for only 45 timesteps. Whiskers indicate 1.5 times the interquartile range. 114

6.8 Having a reasonable intertask mapping helps to eliminate certain state-action pairs from consideration. In the target task, TL(DQL)/BAD and DQL typically explore all six actions, while TL(DQL)/AH and TL(DQL)/WAH typically explore about half (or fewer) of the actions. Error bars indicate ± 1 standard deviation. 115

6.9 Comparison of cumulative reward for 100 different runs of each Q-learning transfer condition under the one state transfer scenario. Each algorithm was run in the target task for only 45 timesteps. Whiskers indicate 1.5 times the interquartile range. 117

6.10 Varying β influences the cumulative reward achieved at the (reset, double reset) transfer scenario. Whiskers indicate 1.5 times the interquartile range. (*a*) When a good intertask mapping is used to transfer action-values, as β increases the cumulative reward decreases. (*b*) If a poor intertask mapping is used, adding a small positive value improves the cumulative reward. However, adding to large a value causes the cumulative reward to decrease. Notice that the penalty for selecting β too small is much worse than selecting a value that is too large. 118

6.11 (a) With a nearly optimal intertask mapping, transferring action-values from the Red Herring domain to the Taxi domain results in higher cumulative reward for both R-MAX and Delayed Q-learning than their respective base algorithms without transfer. (b) With an arbitrarily assigned intertask mapping, the transferred action-values do not result in much loss for either R-MAX or Delayed Q-learning compared to their respective base algorithms without transfer. 119

6.12 (a) Average reward achieved with a conservative intertask mapping. (b) Average reward achieved with an aggressive intertask mapping. . . 121

6.13 Proportion of α-good actions (with $\alpha = 0.1$) in the final learned policy for (a) DQL and (b) TL(DQL) and $\beta = 0$. 122

6.14 Average proportion of states with transferred action-values ($\beta = 0$) that optimistically initialize an α-good action satisfying the weak admissible heuristic (WAH) criterion and the number of states with transferred action-values that are optimistic for every action satisfying the admissible heuristic criterion (which also belongs to WAH). . . . 123

6.15 Comparison between the number of visits made to each state by DQL (the baseline algorithm) and TL(DQL) with $\beta = 0$ averaged over 100 runs. Both DQL and TL(DQL) visit a small fraction of states far more frequently than all of the other states. However, TL(DQL) is more concentrated on a few states than DQL. 124

6.16 Proportion of α-good actions selected by the final learned policy of TL(DQL) over the set of highly visited states. 125

6.17 Average proportion of highly visited states with transferred action-values ($\beta = 0$) that optimistically initialize an α-good action satisfying the weak admissible heuristic (WAH) criterion and the number of states with transferred action-values that are optimistic for every action satisfying the admissible heuristic criterion (which also belongs to WAH). 126

6.18 The percentage of highly visited states conforming to the admissible heuristic or α-weak admissible heuristic conditions where the learned policy at that state selects an α-good action. Whiskers indicate 1.5 times the interquartile range. 127

7.1 Under multitask RL an agent, consisting of a library and algorithm factory, is confronted with a sequence of tasks drawn from the same domain (or distribution over tasks). For each new task, the agent constructs an algorithm by combining knowledge from its library with its algorithm factory to construct a domain specific algorithm. The domain library is updated with each experienced task. 132

7.2 Depicts the minimal hitting set problem for a state s_i over four MDPs. White cells depict α-good actions, while gray depicts actions that are not in $G^{\alpha}_{\Omega_j}(s_i)$ for $j = 1, 2, 3, 4$. Notice that there are multiple minimal hitting sets. 139

7.3 A one-state domain with four actions and the optimal action-values for each task. The red dashes indicate the maximum action-values across all tasks. Notice that maximum action-values implicitly eliminate actions b and d from each task because either action a or c has a higher value than the maximum value of b and d. 144

7.4 A one-state domain with four actions and the optimal action-values for each task. The red dashes indicate the maximum action-values across all tasks. Task Ω_1 has probability mass 0.25 and much higher action-values than the action-values for every other task. The learned maximum action-values are only helpful in task Ω_1. No actions can be pruned in tasks Ω_2, Ω_3, or Ω_4, even though only actions a and c are ever optimal. 146

7.5 Example of one task sampled from a reset domain. 157

7.6 Average of best genomes (with error bars indicating ±1 standard deviation) learned over 40 runs EET on domains (a) $\mathcal{D}_1$, (b) $\mathcal{D}_2$, (c) $\mathcal{D}_3$, (d) $\mathcal{D}_4$, (e) $\mathcal{D}_5$, and (f) $\mathcal{D}_6$. 158

7.7 Average learned exploration tables (with error bars indicating ± 1 standard deviation) by the LET algorithm on domains $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$. The learned exploration tables accurately determine the minimum number of actions that can be explored depending on the domain. 159

7.8 Average learned exploration tables (with error bars indicating ± 1 standard deviation) by the LET algorithm on domains (a) $\mathcal{D}_1$, (b) $\mathcal{D}_2$, (c) $\mathcal{D}_3$, and (d) $\mathcal{D}_4$ expanded to have ten actions instead of four. The learned exploration tables accurately determine the minimum number of actions that can be explored depending on the domain. 160

7.9 Comparison between the cumulative reward earned by LET and MTQL in $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$ extended to 10 actions rather than 4. Whiskers indicate 1.5 times the interquartile range. 161

7.10 The number of entries in exploration tables learned by LET for domains $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$. As the number of tasks with different optimal actions increases. The number of nonzero entries in the learned exploration tables increase. Whiskers indicate 1.5 times the interquartile range. 162

7.11 Comparison between the cumulative reward earned by LET and MTQL in $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$. MTQL is unable to solve the task in a reasonable amount of time due to relying on undirected exploration. LET performs well compared to RMAX when there are few potentially optimal actions, but it begins to drop important actions as the task complexity increases. 163

LIST OF TABLES

TABLE Page

2.1 MBP Experimental Questions 12

2.2 MDP Experimental Questions 26

2.3 MDP Sampling Models 30

6.1 Transfer Outcomes 103

6.2 Algorithm Conditions 107

6.3 (Source Task, Target Task) Pairs 107

6.4 One-State Transfer Expected Rewards 108

7.1 Domain Complexity 137

7.2 $\mathcal{D}_1$: Multiarmed Bandit Domain with Four Actions. 154

7.3 $\mathcal{D}_2$: Multiarmed Bandit Domain with Four Actions. 155

7.4 $\mathcal{D}_3$: Multiarmed Bandit Domain with Four Actions. 155

7.5 $\mathcal{D}_4$: Multiarmed Bandit Domain with Four Actions. 155

7.6 $\mathcal{D}_5$: Multiarmed Bandit Domain with Four Actions. In Task Ω_3 All Actions are Optimal. 156

7.7 $\mathcal{D}_6$: Multiarmed Bandit Domain with Four Actions. 156

1. INTRODUCTION

1.1 The Problem

An important goal of artificial intelligence research and reinforcement learning (RL) is developing autonomous systems that learn to act optimally in complex real-world environments. This goal encompasses three requirements. First, the system should learn a robust solution with few training samples. Second, it should act autonomously, i.e., operate with little intervention from human engineers. Third, it should gracefully handle the complexities of real-world problems. The concept of lifelong learning [4], or learning to learn, supports these requirements by enabling the agent to build up prior knowledge autonomously and apply that body of knowledge to learn novel tasks more efficiently. In other words, the agent learns prior knowledge on its own, instead of relying on human engineers to manually collect and seed the agent with prior knowledge.

Previous research on lifelong learning and transfer learning [5] has primarily emphasized empirical results demonstrating the feasibility of transferring knowledge, but the literature does not offer a full theoretical justification or guidance for applying these techniques. An important measure of an algorithm's efficiency is its sample complexity. Informally, sample complexity is the number of training samples needed to ensure that an algorithm has learned. For RL in a single task the Probably Approximately Correct in Markov Decision Processes framework (PAC-MDP; [6]) provides theoretical guidance for analyzing sample complexity but extensions to settings with multiple tasks have not been fully developed.

In this dissertation, I extend the notion of sample complexity to settings with multiple tasks and use a combination of this theoretical framework with empirical

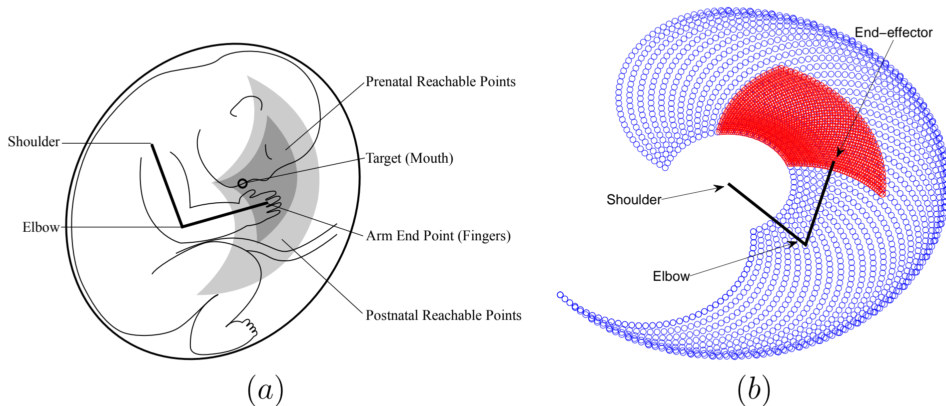


Figure 1.1: (a) Fetal motor learning problem. (b) Experimental prenatal motor learning problem.

experimentation to address the following questions. (1) What kinds of prior knowledge can reduce sample complexity compared to learning from scratch? (2) How does prior knowledge affect sample complexity of learning a new task? (3) How can a learning system acquire and transfer useful prior knowledge autonomously?

1.2 Motivation for Studying Transfer Learning

Our motivation to study transfer learning stems from the many successful applications where transferred knowledge from one task to others has proved useful. In this section, we will outline several works that motivated the research in this dissertation. Taylor and Stone [7], Taylor et al. [8], Taylor and Stone [9], Taylor et al. [10, 11] demonstrated that transferring action-values between two tasks with different with different states and action spaces using a special structure called an intertask mapping.

Mann and Choe [12] considered the possibility of motor learning by human fetuses (Figure 1.1). Fetal motor learning is interesting from a transfer learning perspective because physical conditions before and after birth are quite different. Due to these stark difference one might conclude that there is no reason for motor learning to

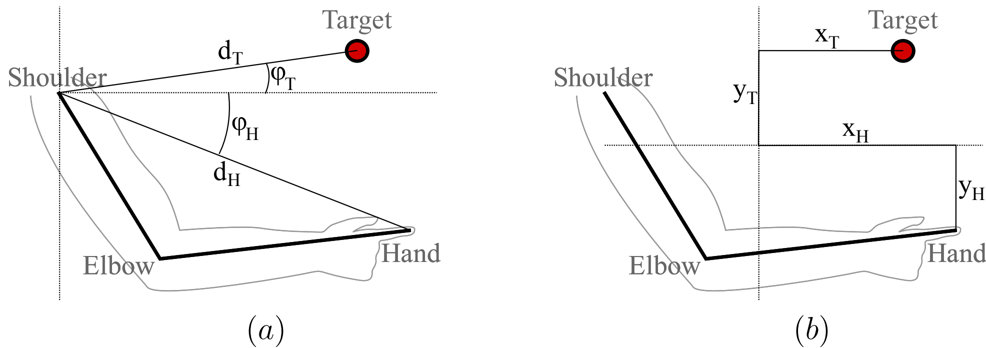


Figure 1.2: (*a*) Agent Centered representation where the origin of the coordinate system is part of the agent itself. (*b*) World Centered representation where the origin is outside of the agent.

occur as a fetus. However, Zoia et al. [13] provide evidence from four-dimensional ultrasound that fetuses plan and execute hand movements to targets such as the mouth and eyes. The main contribution of Mann and Choe [12] was to recognize the importance of problem representation for transfer from a fetal motor learning scenario to motor control after being born. It turns out that when a learning system uses an agent centric coordinate system, like the one shown in Figure 1.2a, it is far easier to generalize motor programs learned as a fetus to life after birth, than using a world centered coordinate system Figure 1.2b. Considering the problems faced by a fetus from a transfer learning perspective helped to understand how a learning system might overcome some of the changes that occur after being born.

Another interesting example that has motivated us to study transfer learning is the problem of autonomously improving perceptual knowledge for use in a wide range of tasks [14]. Raw sensor data is too high dimensional, ambiguous, and noisy to be used to directly solve most control problems. Instead, sensory data must first be translated into higher level perceptual information. Learning perceptions from sensory data can be thought as a task, while the control problems that depend on those percepts can be thought of as target tasks. Mann et al. [14] considered the

problem of autonomously learning accurate binocular depth perception that can be used in object manipulation tasks.

1.3 Approach

The main approach taken throughout this dissertation is to constrain the space searched by an RL algorithm without significantly affecting the learned policy. This can be accomplished through generalization or by reducing the number of states and actions explored. In the majority of this dissertation, we will focus on reducing the number of actions that are explored by an RL algorithm. By decreasing the space searched by an RL algorithm, the time required to learn can be drastically decreased. The main challenge with constraining the exploration space is that many constraints will prohibit RL algorithms from finding a sufficiently optimal policy. Thus constraints must be chosen carefully.

We apply transfer learning to decide which regions of the exploration space can be ignored. We investigate two different structures for transfer called (1) an exploration table and (2) a weak admissible heuristic. It turns out that both of these structures constrain the exploration space by eliminating actions from consideration by an RL algorithm.

We evaluate TL both empirically and from a sample complexity perspective. To this end, we apply transferred knowledge to provably sample efficient RL algorithms R-MAX [15] and Delayed Q-learning [16].

1.4 Outline of the Dissertation

This dissertation is divided into three parts. Sections 1, 2, and 3 introduce and motivate the problem and relevant background related to reinforcement learning and transfer learning. Sections 4, 5, 6, and 7 represent the main work and research contributions of this dissertation. Sections 8 and 9 discuss the main contributions,

limitations, and future work and summarizes the significance of the findings with concluding remarks.

In Section 2, we review details of the reinforcement learning framework including multiarmed bandit problems, Markov decision processes, relevant algorithms and evaluation approaches.

In Section 3, we review previous research on transfer learning from a supervised learning and reinforcement learning perspective. We discuss the advantages of transfer learning, and identify several challenging open questions related to transfer.

In Section 4, we introduce an algorithm, called STAR-MAX, that can restrict its exploration space to a small set of states. We show through experiments that by reducing the searched state-space the STAR-MAX is able to scale more gracefully than other reinforcement learning algorithms that consider the entire state-space.

In Section 5, we develop the theoretical results based on constraining the action-space that are used throughout the rest of this dissertation.

In Section 6, we analyze a transfer learning approach, called action-value transfer, based on the theoretical results presented in Section 5 and empirical results. Both of these results, help to understand when transfer learning will provably succeed at speeding up reinforcement learning in large scale tasks.

In Section 7, we introduce measures of domain complexity and learning algorithms for multitask reinforcement learning. We demonstrate the relationship between the various measures of domain complexity and sample complexity. We determine an efficient algorithm for learning a domain specific algorithm and demonstrate that it is able to learn domain specific reinforcement learning algorithms in practice that outperform general purpose reinforcement learning algorithms.

In Section 8, we discuss the main contributions of our work, its main limitations and areas of future work. Section 9 summarizes the findings of this dissertation and

provides concluding remarks and recommendations.

2. BACKGROUND

Reinforcement learning (RL) is a computational framework for trial-and-error learning [1]. In the RL framework, a learning system or agent, embedded in an initially unknown environment, is faced with a set of possible actions. Each time the agent acts, it receives a reward (Figure 2.1). These rewards are represented by scalar values and may be positive, neutral, or negative (i.e., costs). The objective of the agent is to learn over time how to act in a way that maximizes long term rewards. In other words, the RL agent learns to take actions that are good in the long run instead of actions that offer a payoff now but may lead to poor performance later on.

The critical problem that unifies RL is the exploration/exploitation dilemma. The exploration/exploitation dilemma is a problem faced by any learning agent that needs to act when some information about the environment is unknown. The agent needs to try various actions to find out whether each action is associated with high or low rewards. This process is called exploration. On the other hand, the agent's objective is to maximize long-term rewards, so it needs to use the information that it already has to select the action that it believes is best. This process is called exploitation. The main problem faced by the learning agent is that it must balance

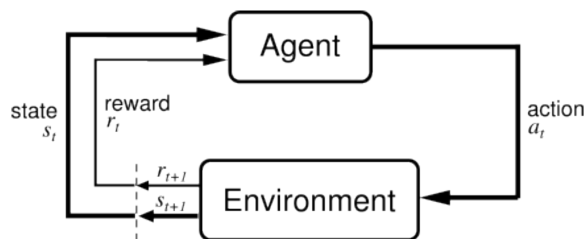


Figure 2.1: A reinforcement learning agent is embedded in an environment. Each time the agent acts, the agent's state in the environment changes and it receives a reward. Adapted from [1].

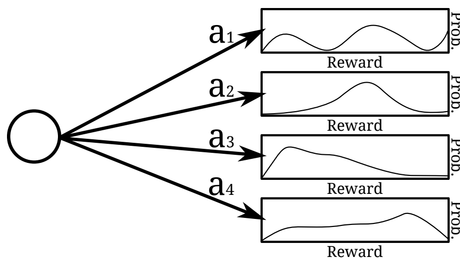


Figure 2.2: An example multiarmed bandit problem with $K = 4$ actions $\{a_1, a_2, a_3, a_4\}$. Each action is associated with a reward distribution.

exploring actions that may or may not lead to high rewards while at the same time exploiting the actions that it believes are best given the data the agent has already collected.

The exploration/exploitation dilemma has primarily been modeled by Multi-armed Bandit Problems and Markov Decision Processes. In this section, we will review these models and related algorithms and theory.

2.1 Multiarmed Bandit Problems (MBPs)

A multi-armed bandit problem (e.g. Figure 2.2), first proposed by Robbins [17], is one of the simplest formulations of the exploration/exploitation dilemma. Multi-armed bandit problems (MBPs) are repeated decision problems, where the learning agent is given the opportunity to select one of $K \geq 2$ actions at each timestep. Each action a_k for $k = 1, 2, \ldots, K$ is associated with a bounded reward distribution $0 \leq R_k \leq 1$. At each timestep, after the agent selects an action a_k, a reward r distributed by R_k is given to the agent. Typically, the reward distributions are assumed to be stationary, meaning that they do not change over time. However, in later sections we will consider generalizations of the MBP that relax this constraint.

If the reward distributions are stationary, then the optimal course of action, called

a policy, is to always choose the action

$$a^* = \arg \max_{k \in \{1,2,\ldots,K\}} E[R_k] \tag{2.1}$$

whose corresponding reward distribution has the highest expected value, where $E[\cdot]$ denotes the expected value operator.

But the action associated with the highest expected reward is unknown. To learn the optimal action, an algorithm must sample each reward distribution by trying the different actions. Once the algorithm has sampled the distributions, it can make an informed decision about which action has the highest expected reward. However, because the reward distributions are stochastic, the expected value of each arm cannot be known with certainty with a finite number of samples. This is where the exploration/exploitation dilemma comes in. No algorithm can eliminate uncertainty, so the algorithm must balance exploring uncertain actions and exploiting the action believed to give the highest expected reward.

2.1.1 Algorithms for MBPs

The literature has explored many algorithms for solving stationary MBPs. These algorithms can be broadly placed into three categories: (1) probabilistic explorers, (2) finite-sample, and (3) upper confidence bounded algorithms.

The two most popular probabilistic exploration algorithms are ε-greedy and softmax. These algorithms construct a probability distribution to decide when to explore and when to exploit. Both algorithms start by choosing each of the K actions once. After that, ε-greedy selects the action with the highest empirical reward with probability $(1 - \varepsilon)$ and randomly selects any other action with probability ε. For this strategy the parameter $\varepsilon \in [0, 1]$ controls the tradeoff between exploration and exploitation. When ε is close to 0, the algorithm rarely explores. In this case, if the

algorithm believes that the wrong action has the highest expected reward, then it may take a long time before it learns to correct its mistake. When ε is close to 1 the algorithm explores most of the time, but rarely uses that information to choose the best action. The softmax algorithm constructs a probability distribution that gives actions with higher empirical mean greater probability and controls the exploration/exploitation trade-off with a single parameter $T > 0$ called the temperature. When the temperature T is high the probabilities are similar. As T moves close to zero, the action with highest empirical mean receives more and more probability mass. One problem with this kind of algorithm is that they continue exploring forever, which is potentially wasteful. However, these simple strategies often work well in practice.

Finite-sample algorithms stop after a finite number of trials and recommend an action. These algorithms initially perform exploration for a finite number of timesteps. One simple algorithm is to try each action $m > 1$ times and select the action associated with the highest empirical mean (Algorithm 1). However, a more sophisticated algorithm, MedianElimination, has been shown to outperform algorithm 1 by dropping actions from consideration that fall below the empirical median [18, 19].

Algorithm 1 Naïve Bandit

Require: K, m

1: **for** $k = 1, 2, \ldots, K$ **do**
2: $\quad r(k) \leftarrow 0$
3: $\quad$ **for** $n = 1, 2, \ldots, m$ **do**
4: $\quad\quad$ Try action a_k
5: $\quad\quad r(k) \leftarrow r(k) + r$
6: $\quad$ **end for**
7: **end for**
8: **return** $\hat{a} = \arg\max_{k \in \{1,2,\ldots,K\}} r(k)$

An alternative to the naïve and MedianEliminiation algorithms are the Upper Confidence Bound (UCB) algorithms. UCB1 and UCB2 construct an upper confidence bound (which is where they get their names from) on the uncertainty of the current estimate [20]. The decision rule for the UCB1 algorithm is

$$\hat{a} = \arg \max_{k \in \{1,2,\ldots,K\}} \frac{r(k)}{n(k)} + \alpha \sqrt{\frac{2 \ln \left(\sum_{i=1}^{K} n(i) \right)}{n(k)}} \tag{2.2}$$

where $r(k)$ is the sum of rewards received after selecting the kth arm, $n(k)$ is the number of times the kth arm has been selected, and α (which is not present in the theoretically pure version of UCB1) is used in practice to adjust how much the algorithm explores. The first term on the right hand side is simply the empirical mean of the reward for the kth arm. The second term serves as an exploration bonus. When added together these two terms upper bound the confidence interval for the expected reward of the kth arm. Actions that have high uncertainty receive a large bonus, while actions with small uncertainty receive a small bonus. This bonus causes the algorithm to explore uncertain actions unless there is an action with expected value so much higher than the other actions that their uncertainty bonuses are dwarfed in comparison.

2.1.2 Evaluation of MBP Algorithms

An important question to ask is: Which MBP algorithm is best? To make an informed judgment about which algorithm is best for a particular task, we need a strategy for evaluating and comparing MBP algorithms.

One strategy is to test MBP algorithms on a set of MBP benchmarks. Empirical testing of algorithms can provide valuable insight about the performance of algorithms. After all, our purpose for inventing algorithms is to apply some of them to

Table 2.1: MBP Experimental Questions

Question	**Options**
What algorithms to compare?	Powerset of all MBP algorithms
What algorithm parameters to use?	Dependent on algorithms
What benchmark problems to compare?	Powerset of all MBPs
How many independent trials?	$\mathbb{Z}^+$
How many decisions to observe?	$\mathbb{Z}^+$
What statistics to record and compare?	Avg. or Cum. Rewards

real-world problems. A useful statistic for comparing MBP algorithms is average, cumulative reward. Cumulative reward is the sum of all of the reward received by an algorithm over a finite number of decisions. Initially algorithms are exploring the different actions and receive small rewards. As the algorithm improves its knowledge of the actions, it should transition to exploiting the action that it believes has the highest expected reward. Algorithms with high cumulative reward must balance the trade-off between exploration and exploitation, because algorithms that explore during the entire finite window will achieve lower cumulative reward than algorithms that spend some of their time exploiting. On the other hand, algorithms that stop exploring too early may settle on an action that is suboptimal, which also results in lower cumulative reward.

One of the main problems with empirical comparison between algorithms is that exhaustive comparison is not possible. For example, Table 2.1 specifies a few of the questions that need to be address to implement an experiment. Notice that many of questions can be answered in infinitely many ways. For this reason, empirical analysis alone is not enough to understand and compare MBP algorithms. In addition to empirical analysis, which typically compare the average or cumulative rewards received over a finite number of timesteps, there are two popular theoretical frameworks for comparing algorithms: regret and sample complexity.

2.1.2.1 Regret Measure

The regret formulation measures the expected loss between the optimal policy and the policy followed by an algorithm. Initially the learning algorithm cannot follow the optimal policy because it does not know the optimal policy. It has to try the different actions multiple times to learn the expected reward associated with each action. Regret is defined by

$$\mathcal{R}_t = \sum_{\tau=1}^{T} \left(E[a^*] - E[a_{k_\tau}]\right) \tag{2.3}$$

where a^* is the action with the highest expected reward, and a_{k_τ} is the action chosen by the learning algorithm at time τ.

Auer et al. [20] derive upper bounds on regret over a finite time period for several algorithms, including a variant of the ε-greedy algorithm and the algorithms UCB1 and UCB2. This study is important because it demonstrates the existence of algorithms that achieve the optimal regret bounds in a finite number of decisions.

Although the regret formulation captures the trade-off between exploration and exploitation in a natural way, it is somewhat unnatural in the sense that it does not specify any time at which the algorithm can be said to have learned the task. In fact, despite having an optimal finite-time bound on regret, UCB1 and UCB2 may act suboptimally no matter how long the algorithm has run [20]. This property of regret analysis is counter-intuitive and may lead to the development of algorithms that are unsuitable in contexts where a learning system needs to act near-optimally after a finite training period.

2.1.2.2 Sample Complexity Measure

An alternative theoretical formulation to regret is sample complexity. Sample complexity measures the the number of actions or timesteps that the learning algorithm needs to try before it can select a near-optimal action with high probability. More formally, for $\epsilon > 0$ and $\delta \in (0, 1]$, the sample complexity of a learning algorithm for solving any MBP is the minimum number of action samples the algorithm needs to observe before the algorithm can select an action $\hat{a}$ such that $\hat{a} \geq a^* - \epsilon$ with probability at least $1 - \delta$ [18]. This notion of sample complexity is similar to Probably Approximately Correct [21] notion of sample complexity in the supervised learning setting but differs in the sense that errors are not misclassifications. Instead an error occurs when an algorithm returns an action with expected reward that is more than ϵ below the optimal action. The parameters ϵ and δ can be chosen to admit any degree of desired accuracy or certainty, respectively, at the cost of increasing sample complexity.

Sample complexity analysis is a useful tool for comparing algorithms over all MBPs. The strategy is to identify upper bounds on the sample complexity of particular algorithms, while at the same time identifying lower bounds on sample complexity that cannot be beaten by any algorithm. If the upper bound on sample complexity of an algorithm $\mathcal{A}$ matches the lower bound, then $\mathcal{A}$ can be said to have optimal sample complexity.

For MBPs, the parameters that control sample complexity are the number of arms K, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$. In other words, as the number of actions K increases the sample complexity also increases. The same is true for the other parameters.

The best reported upper bound on sample complexity for MBPs, due to Even-Dar

et al. [18], is

$$O\left(\frac{K}{\epsilon^2}\ln\left(\frac{1}{\delta}\right)\right)$$

for the MedianElimination algorithm where ln is the natural logarithm. We can compare this to

$$O\left(\frac{K}{\epsilon^2}\ln\left(\frac{K}{\delta}\right)\right)$$

the upper bound on sample complexity for the NaïveBandit algorithm. Notice that the MedianElimination algorithm is superior because it shaves off a $\ln(K)$ dependence [18]. Finally, the MedianElimination algorithm has optimal sample complexity because the upper bound established by [18] matches the lower bound

$$\Omega\left(\frac{K}{\epsilon^2}\ln\left(\frac{1}{\delta}\right)\right)$$

established by [22].

The foundational tools for analyzing sample complexity of MBPs (and later Markov decision processes) are the Hoeffding bound and the union bound.

Theorem 2.1. *(Hoeffding Bound [23]) Let $\epsilon \geq 0$ and $X_1, X_2, \ldots, X_m$ be $m > 0$ independent random variables such that $\Pr\left[a_i \leq X_i \leq b_i\right] = 1$ for $i = 1, 2, \ldots, m$. If $S = \sum_{i=1}^{m} X_m$ and $\mu_S = E\left[S\right]$, then*

$$\Pr\left[\left|\frac{S}{m} - \frac{\mu_S}{m}\right| > \epsilon\right] < 2\exp\left(-\frac{2\epsilon^2 m^2}{\sum_{i=1}^{m}(b_i - a_i)^2}\right) \tag{2.4}$$

where $E[\cdot]$ is the expected value operator.

The importance of the Hoeffding bound is that it bounds the probability that a sum of independent random variables is far from its mean. The remarkable thing about this bound is that the random variables belong to any bounded distribution.

This allows the analysis of sample complexity when the reward distributions associated with each action are unknown. However, tighter bounds may be possible if the reward distributions are known to belong to a specific family.

One limitation of the Hoeffding bound is that it only applies to a single sum of random variables. The union bound complements the Hoeffding bound by allowing us to make claims about the probability of multiple events.

Theorem 2.2. *(Union Bound) If $B_1, B_2, \ldots, B_m$ be $m > 0$ Bernouli random variables (not necessarily independent) with outcomes in the set $\{0, 1\}$, then*

$$\Pr\left[\sum_{i=1}^{m} B_i > 0\right] \leq \sum_{i=1}^{m} \Pr\left[B_i = 1\right] \tag{2.5}$$

which means that the probability that at least one of the m events will occur is bound by the sum of each variables probability of success.

Combining the Hoeffding bound and union bound, allows us to analyze the sample complexity upper bound for many MBP algorithms. Application of these theorems is foundational for sample complexity analysis in general.

2.1.3 Limitations of the MBP Model

The simplicity of the MBP model allows it to capture the essential aspects of many real world decision problems, but it is limited in the sense that many problems have a notion of state. That is, the outcome of an action may depend on the current situation. MBPs traditionally assume that no information about the state of the environment or problem is known to the agent. The reward distribution is assumed to depend only on the selected arm. Due to this limitation, a more sophisticated problem model is needed to describe problems with inherent state dependencies.

2.2 Markov Decision Processes (MDPs)

Although the multiarmed bandit problem (MBP) formulation captures many interesting aspects of the exploration/exploitation dilemma, there are many problems that cannot be fully described as a multiarmed bandit problem. For example, consider the problem faced by an autonomous car traveling from a city A to another city B. Along the journey the which turns the car will make depend on its current location. How fast the car will travel will depend on conditions like the speed limit and how fast other cars are traveling. All of this information can be wrapped up in the notion of state. Markov decision processes extend the MBP formulation to capture the notion of states and are the dominant framework used for analyzing reinforcement learning [1].

A Markov decision process (MDP; [24]) M is typically defined by a 5-tuple $M = \langle S, A, T, R, \gamma \rangle$ where S is a nonempty, finite set of states, A is a nonempty finite set of actions, T is a set of probability distributions governing state changes, $R : S \times A \rightarrow \mathbb{R}$ maps state-action pairs to rewards, and $\gamma \in [0, 1)$ is called the discount factor and determines how preferential high rewards are now compared to high rewards received in the distant future. At each timestep the agent is in a particular state $s \in S$ and selects an action $a \in A$ to be executed. After the selected action is executed the agent transitions to a new state $s' \sim T(\cdot|s, a)$. Throughout this dissertation, we assume that the reward function R is bounded by the interval $[0, 1]$. This is a very minor restriction because bounded reward functions can be scaled and shifted to fit within this interval.

Given an MDP M, the objective of the agent is to learn a policy $\pi : S \rightarrow A$ that

maps the current state to an action, such that the policy maximizes

$$V_M^{\pi}(s_t) = E\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} R(s_t, \pi(s_t))\right] \tag{2.6}$$

where s_t is the current state at timestep t. Throughout this dissertation most policies will be deterministic mappings, however, policies can also be expressed as probability distributions $a \sim \pi(\cdot|s)$ over actions. Equation 2.6 is called the value function of policy π on M with respect to state s (or simply a value function). Due to our assumption that the reward function is bounded by the interval $[0, 1]$, the value function is bounded by the interval $\left[0, \frac{1}{1-\gamma}\right]$. The policy π^* that maximizes (2.6) is called the optimal policy and we denote the optimal policy's value function by $V^*(s)$ for a state $s \in S$.

An important discovery, due to Bellman [25], is that the value function can be written recursively as

$$V_M^{\pi}(s_t) = R(s_t, \pi(s_t)) + \gamma E_{s_{t+1} \sim T(\cdot|s_t, a)}\left[V_M^{\pi}(s_{t+1})\right] \tag{2.7}$$

where the expected value is taken with respect to the the next state distribution of M defined by $T(\cdot|s, a)$. The optimal value function can be written recursively as

$$V_M^*(s_t) = \max_{a \in A} \left\{R(s_t, a) + \gamma E_{s_{t+1} \sim T(\cdot|s_t, a)}\left[V_M^*(s_{t+1})\right]\right\} \quad , \tag{2.8}$$

which has been influential in many dynamic programming and reinforcement learning algorithms.

Another important differentiation of the value function is the action-value func-

tion defined by

$$Q^{\pi}_{M}(s_t, a_t) = R(s_t, a_t) + \gamma E_{s_{t+1} \sim T(\cdot|s_t, a_t)} \left[V^{\pi}_{M}(s_{t+1}) \right] \quad (2.9)$$

where $Q^{\pi}_{M}(s_t, a_t)$ is the long-term value of taking action a_t and thereafter following the policy π. The notion of the action-value function is useful for defining the optimal policy

$$\pi^*(s) = \arg\max_{a \in A} Q^*_M(s, a) \quad (2.10)$$

where Q^*_M denotes the action-value function of the optimal policy.

Algorithms pertaining to MDPs broadly fall into two categories: dynamic programming and reinforcement learning. Dynamic programming algorithms plan an optimal policy based on an MDP. The assumption under dynamic programming is that the MDP is known completely and the problem is simply planning a policy. Dynamic programming problems do not need to explore the environment. Reinforcement learning algorithms, on the other hand, do not assume that the details about the MDP are known *a priori* and must simultaneously learn about the environment as well as plan a policy that achieves high long-term reward (i.e. the exploration/exploitation dilemma).

2.2.1 Planning (Dynamic Programming) in MDPs

Dynamic programming algorithms assume a complete model of the MDP is known *a priori*. We mention these dynamic programming algorithms here because they are often used as subroutines in model-based reinforcement learning algorithms. The two most popular algorithms for dynamic programming are value iteration and policy iteration.

The most popular algorithm for dynamic programming is value iteration. The

Algorithm 2 Value Iteration

Require: $S, A, T, R, \gamma, \theta^*$
1: **for** $s \in S$ **do** {Initialize the value function to 0.}
2: $\quad V(s) \leftarrow 0$
3: **end for**
4: **while** $\theta > \theta^*$ **do** {Run value iteration until the change in V is small.}
5: $\quad \theta \leftarrow 0$
6: $\quad$ **for** $s \in S$ **do**
7: $\quad\quad \hat{V} \leftarrow V(s)$
8: $\quad\quad V(s) \leftarrow \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V(s') \right\}$
9: $\quad\quad$ **if** $|\hat{V} - V(s)| > \theta$ **then**
10: $\quad\quad\quad \theta \leftarrow |\hat{V} - V(s)|$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end while**
14: **for** $s \in S$ **do** {Calculate the policy.}
15: $\quad \pi(s) \leftarrow \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V(s') \right\}$
16: **end for**
17: **return** π

value iteration algorithm (Algorithm 2) is given a fully specified MDP $M = \langle S, A, T, R, \gamma \rangle$ and a threshold parameter θ^* used to determine a stopping point for the algorithm. For each iteration, the algorithm loops over all states and computes a new estimate of the optimal value function using an update rule inspired by equation (2.8). After the values are sufficiently close to optimal, a policy is computed from the approximate value function. Using value iteration it is possible to plan a policy that is arbitrarily close to optimal in time polynomial in the number of states, actions, and accuracy parameter. In fact, value iteration can be used to plan exact optimal policies in polynomial time [26]. This means that arbitrarily accurate policies can be planned efficiently with respect to computation.

Another popular dynamic programming algorithm is policy iteration. Policy iteration starts with an arbitrary policy and alternates between evaluating the current

policy and improving the policy. It can be shown that policy iteration converges to the optimal policy [24]. Policy iteration is also an attractive algorithm because it has often been found to converge faster than value iteration in practice [24, 1].

As stated above, dynamic programming algorithms can only be applied if a complete model of the MDP is known *a priori*. However, in this dissertation we are primarily concerned with the case where the MDP is not known initially.

2.2.2 Reinforcement Learning (RL) in MDPs

RL algorithms do not assume knowledge about the environments transition probabilities T and (possibly) the reward function R. The algorithm must learn to improve their initial policy despite this lack of knowledge. The RL agent must interact with its environment by trying out different actions and observing their consequences to learn to act optimally.

To investigate the exploration/exploitation dilemma, previous research has investigated a number of strategies or policies for balancing exploration and exploitation. Several of the most popular exploration policies are:

1. ε-greedy
2. softmax
3. optimism in the face of uncertainty

Given an estimate Q of the optimal action-values Q^*, the ε-greedy exploration policy selects the action that is believed to be best (i.e., $\arg\max_{a \in A} Q(s, a)$) with probability $(1 - \varepsilon)$ and an action chosen uniformly random with probability ε where ε is typically much smaller than $\frac{1}{2}$. This strategy causes a learning algorithm to choose the action that it believes to be best most of the time but also to devote a small amount of effort to exploring other potentially more rewarding actions.

The softmax exploration policy assigns probabilities

$$\Pr\left[a|s\right] = \frac{\exp\left(Q(s,a)/\tau\right)}{\sum_{i=1}^{K} \exp\left(Q(s,a_i)/\tau\right)} \tag{2.11}$$

to actions based on how "good" the learning algorithm believes that each action is. The parameter τ is called the temperature and controls how extreme the probability distribution is. If an action is believed to have high value, then the probability that that action will be selected is high. If, on the other hand, the action is believed to have low value, then it will be selected with low probability. This strategy allows the agent to explore what it believes to be the best course of action, but, much like ε-greedy, other exploratory actions are selected some of the time. This allows the learning system to explore and exploit.

Optimism in the face of uncertainty is another popular heuristic for handling the exploration/exploitation dilemma. The idea behind optimism in the face of uncertainty is to initially assume that all actions offer the best possible reward and always choose the action that is believed to have the best reward. When actions with low rewards are chosen, the learning agent will eventually discover that its estimate is overly optimistic and lower its belief to a more appropriate value. This heuristic causes the agent to explore thoroughly before settling on a specific policy.

Besides deciding how to explore RL algorithms need to determine what statistics to maintain about the environment. One approach is to learn the transition probabilities and reward function and then applying one of the dynamic programming algorithms to plan a policy. These algorithms are known as model-based, because they construct a complete model of the MDP. Interestingly some algorithms are able to learn a near-optimal policy without explicitly learning the transition probabilities or reward function. These algorithms are referred to as model-free.

2.2.2.1 Model-Based

Model-based RL algorithms learn the transition probabilities T and reward function R of an MDP and then use dynamic programming algorithms to solve the model [27, 15, 28, 29].

Algorithm 3 R-MAX [15]

Require: $S, A, \gamma, m, \epsilon_1$
1: **for** $(s, a) \in S \times A$ **do** {Initialize the algorithm.}
2: $\quad Q(s, a) \leftarrow \frac{1}{1-\gamma}$ {Initialize action-values optimistically.}
3: $\quad n(s, a) \leftarrow 0$ {# visits to (s, a)}
4: $\quad R(s, a) \leftarrow 0$ {Records reward at (s, a)}
5: $\quad$ **for** $s' \in S$ **do**
6: $\quad\quad l(s, a, s') \leftarrow 0$ {# $(s, a) \rightarrow s'$}
7: $\quad$ **end for**
8: **end for**
9: **for** $t = 1, 2, 3, \ldots$ **do** {Main interaction loop.}
10: $\quad$ Observe the current state s at timestep t
11: $\quad$ Select action $a = \arg\max_{a' \in A} Q(s, a')$
12: $\quad$ Execute action a and observe the next state s' and reward r
13: $\quad$ **if** $n(s, a) < m$ **then**
14: $\quad\quad n(s, a) \leftarrow n(s, a) + 1$ {Increment (s, a) counter.}
15: $\quad\quad l(s, a, s') \leftarrow l(s, a, s') + 1$
16: $\quad\quad R(s, a) \leftarrow R(s, a) + r$
17: $\quad\quad$ **if** $n(s, a) = m$ **then**
18: $\quad\quad\quad$ Compute a transition model $\hat{T}$ from n and l
19: $\quad\quad\quad$ Compute a the reward function $\hat{R}$ from n and R
20: $\quad\quad\quad$ Use dynamic programming to update Q with ϵ_1-accurate action-values estimates for the constructed internal model
21: $\quad\quad$ **end if**
22: $\quad$ **end if**
23: **end for**

The R-MAX algorithm [15, 30] is one of the simplest and most popular model-based algorithms. R-MAX (Algorithm 3) initializes its action-value estimates Q for each state-action pair to be the largest possible value $\frac{1}{1-\gamma}$. At each state R-MAX

selects the action with the largest action-value

$$\pi(s) = \arg\max_{a \in A} Q(s, a) \tag{2.12}$$

which is sometimes called the greedy policy with respect to the action-value estimates Q. After an action is executed the algorithm observes that it transitioned to some state $s' \in S$ and receives a reward $r \in [0, 1]$. R-MAX counts the number of times it has tried an action a in state s using the counter $n(s, a)$. It also counts the number of times trying (s, a) transitions the agent to state s' in the counter $l(s, a, s')$. Once $n(s, a)$ reaches m, the algorithm can approximate the transition probability distribution for the state action pair (s, a) by

$$\hat{T}(s'|s, a) = \frac{l(s, a, s')}{n(s, a)} \tag{2.13}$$

where $\hat{T}(s'|s, a)$ is an approximation of the probability that state s' will be transitioned to by executing action a in state s. The reward function

$$\hat{R}(s, a) = \frac{R(s, a)}{n(s, a)} \tag{2.14}$$

where $\hat{R}(s, a)$ is an approximation of the reward function at (s, a).

These two pieces of information ($\hat{T}$ and $\hat{R}$) complete the model of the MDP and a dynamic programming algorithm such as value iteration (Algorithm 2) can be used to plan a policy.

The main trick behind R-MAX is that the model parameters for state-action pairs that have not been visited m times are given clever default values. For example, [30] assigned state-action pairs that have not been visited m times absorbing transition probabilities (so that the state-action pair transitions to the state it was already in)

and the default reward received is $R_{\text{MAX}} = 1$ (which is where the algorithm gets its name). By assigning these clever defaults, any state-action pair that has not been visited at least m times will appear to give the highest possible value $\frac{1}{1-\gamma}$ in the model. When dynamic programming is used to solve the model it plans to visit these high valued but unknown state-action pairs until a suitable policy for the true MDP is learned.

2.2.2.2 Model-Free

Model-free RL algorithms manage to improve their policy without explicitly learning the transition probabilities or reward function. The most popular model-free RL algorithm is Q-learning [31]. The idea behind Q-learining is to use samples of the reward and next state to accurately approximate the action-value function (or Q-function) at each state. Q-learning (Algorithm 4) has a simple update rule

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha\left(R(s,a) + \gamma \max_{a' \in A} Q_t(s',a')\right) \tag{2.15}$$

based on equation 2.8, where Q_t is the action-value estimate before the update, Q_{t+1} is the action value estimate after the update, s is the current state, a is the current action, α is the learning rate, and s' is the next state. It can be shown that Q-learning converges to the optimal action-values provided that every state-action pair is visited infinitely often [31]. One immediate problem with the Q-learning algorithm is that it sidesteps the exploration/exploitation dilemma by not specifying an exploration policy π_e.

Strehl et al. [16] introduces the Delayed Q-learning algorithm (Algorithm 5). Delayed Q-learning is a model-free RL algorithm in that its per timestep computational complexity is small and its memory usage is $O(NK)$ where N is the number of states and K is the number of actions, which is less memory than is needed to specify the

Algorithm 4 Q-learning [31]

Require: $S, A, \gamma, \pi_e, \alpha$

1: Initialize $Q(s, a)$ arbitrarily for all $(s, a) \in S \times A$
2: **for** $t = 1, 2, 3, \ldots$ **do**
3: Observe the state s at timestep t
4: Select action $a = \pi_e(s)$
5: Execute action a and observe the next state s' and reward r
6: $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a')\right)$
7: **end for**

Table 2.2: MDP Experimental Questions

Question	**Options**
What algorithms to compare?	Powerset of all MDP algorithms
What algorithm parameters to use?	Dependent on algorithms
What benchmark problems to compare?	Powerset of all MDPs
How does the algorithm sample?	Batch, Exploration, etc.
How many independent trials?	$\mathbb{Z}^+$
How many decisions to observe?	$\mathbb{Z}^+$
What statistics to record and compare?	Avg. or Cum. Rewards

transition model T of an MDP. The algorithm gets its name from the fact that it does not use samples to update the action-values immediately. Instead Delayed Q-learning waits until it has collected $m > 1$ samples before attempting to update an action-value. Unlike the Q-learning algorithm, Delayed Q-learning specifies an exploration strategy. It initializes its action-values to $\frac{1}{1-\gamma}$ (the highest possible action-value) and uses the optimism in the face of uncertainty heuristic to guide exploration.

2.2.3 Evaluation of RL Algorithms in MDPs

An important question to ask about RL algorithms is: which algorithm is best? Similar to the previous section on MBPs, the options for comparing algorithms are empirical experiments and theoretical analysis.

Empirical experimentation is critical for comparing algorithms. However, Table 2.2 points out the many questions that needed to be answered while designing an ex-

Algorithm 5 Delayed Q-learning [16, 6]

Require: $S, A, \gamma, m, \epsilon_1$

1: **for** $(s,a) \in S \times A$ **do** {Initialize the algorithm.}
2: $\quad Q(s,a) \leftarrow \frac{1}{1-\gamma}$ {Optimistically initialize action-values.}
3: $\quad U(s,a) \leftarrow 0$ {Records value for attempted updates.}
4: $\quad n(s,a) \leftarrow 0$ {# visits to (s,a)}
5: $\quad b(s,a) \leftarrow 0$ {Beginning timestep of most recent attempted update.}
6: $\quad LEARN(s,a) \leftarrow true$ {Used to determine when to stop learning.}
7: **end for**
8: $\tau \leftarrow 0$ {Timestep of the most recent successful update.}
9: **for** $t = 1, 2, 3, \ldots$ **do** {Main interaction loop.}
10: $\quad$ Observe the current state s at timestep t
11: $\quad$ Execute $a = \arg\max_{a' \in A} Q(s,a')$ and observe the next state s' and reward r
12: $\quad$ **if** $b(s,a) \leq \tau$ **then**
13: $\quad\quad LEARN(s,a) \leftarrow true$
14: $\quad$ **end if**
15: $\quad$ **if** $LEARN(s,a) = true$ **then**
16: $\quad\quad$ **if** $n(s,a) = 0$ **then**
17: $\quad\quad\quad b(s,a) \leftarrow t$
18: $\quad\quad$ **end if**
19: $\quad\quad n(s,a) \leftarrow n(s,a) + 1$ {Increment (s,a) counter.}
20: $\quad\quad U(s,a) \leftarrow U(s,a) + (r + \gamma \max_{a' \in A} Q(s', a'))$
21: $\quad\quad$ **if** $n(s,a) = m$ **then**
22: $\quad\quad\quad$ **if** $Q(s,a) - U(s,a)/m \geq 2\epsilon_1$ **then**
23: $\quad\quad\quad\quad Q(s,a) \leftarrow U(s,a)/m + \epsilon_1$
24: $\quad\quad\quad\quad \tau \leftarrow t$
25: $\quad\quad\quad$ **else if** $b(s,a) > \tau$ **then**
26: $\quad\quad\quad\quad LEARN(s,a) \leftarrow false$
27: $\quad\quad\quad$ **end if**
28: $\quad\quad\quad U(s,a) \leftarrow 0; n(s,a) \leftarrow 0$
29: $\quad\quad$ **end if**
30: $\quad$ **end if**
31: **end for**

periment. Many of the answers to these questions can take an infinite number of options, which rules out exhaustive empirical comparison. Again, empirical comparison is important, but theoretical analysis is needed to compare algorithms over the entire class of MDPs.

2.2.3.1 Convergence

The first and weakest theoretical guarantee is convergence to the optimal policy. Convergence guarantees say that an algorithm, given access to an infinite number of samples, will learn the optimal policy. For example, it is well known that the Q-learning algorithm will converge to the optimal action-value function (and as a consequence the optimal policy) if the algorithm visits every state-action pair infinitely often [31]. Convergence results offer an important first step towards analyzing an algorithm. However, no algorithm can ever observe an infinite amount of data in practice, so we need to take a step further and determine how quickly an algorithm converges toward the optimal policy.

2.2.3.2 Regret

The notion of regret for MDPs is similar to the notion of regret for MBPs. The analysis of the regret of RL algorithms in MDPs has so far been restricted to ergodic MDPs [32], where every state is reachable from every other state in a small number of timesteps (in expectation). Similar to the case of MBPs, regret seems to capture our intuitive notion of the exploration/exploitation dilemma, but has the awkward property that there is no time at which we can say the system has learned a solution.

2.2.3.3 Sample Complexity

Analysis of sample complexity under MDPs is more complicated than analysis of sample complexity under MBPs. The main problem is exploration. In MBP

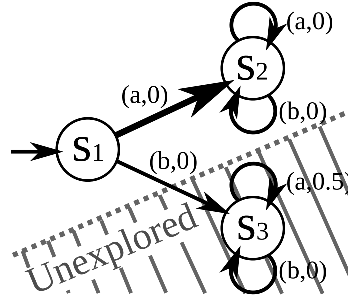


Figure 2.3: A simple MDP where every state $\{s_1, s_2, s_3\}$ cannot be reached in a single episode.

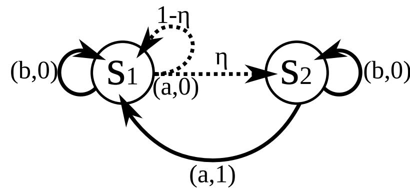


Figure 2.4: A simple MDP where reaching state s_2 from s_1 may take many tries, if η is very small.

problems every action could be tried just as easily as any other action. In MDPs, some states may be difficult or impossible to reach from other states. For example, in the MDP in Figure 2.3 no algorithm can visit state s_2 and s_3 in a single episode. In Figure 2.4 the probability of reaching state s_2 from s_1 by taking action a may be very small. It may take many tries to reach s_2 if η is very small. Therefore it may not always be possible to learn an optimal policy from a single stream of experience.

Another difficulty of analyzing sample complexity under MDPs is planning. The value function is defined over a sequence of actions. When we say that a policy is optimal we mean that it makes sequences of optimal decisions. Optimality depends on multiple actions and cannot be easily measured in practice.

By sample we mean that the agent tries an action a in a state s and observes the next state s' and reward r. The tuple (s, a, s', r) can be considered a single sample. We can imagine a number of ways for an algorithm to gain access to samples. The

Table 2.3: MDP Sampling Models

Model	What's Learned?	Description
Batch Model	Near-Optimal π wrt Data	Previously recorded samples are given to the algorithm.
Generative Model	$\pi \approx \pi^*$	The algorithm can observe a transition and sample a reward from any state-action pair.
Reset Model	π such that $V^{\pi}(s_0) \geq V^*(s_0) - \epsilon$	Learning occurs in episodes. The algorithm must perform a sequence of actions to transition to any particular state-action pair. At any time the algorithm may use a special reset action to return to an initial state s_0.
Explore Model	$V^{\mathcal{A}_t}(s) \geq V^*(s) - \epsilon$	Learning occurs in one infinite stream of actions. The algorithm must perform a sequence of actions to transition to any particular state-action pair. The algorithm *cannot* reset to an initial state.

interface by which an algorithm gains access to samples is called a sampling model [30]. There are four popular sampling models:

1. Batch Model
2. Generative Model
3. Reset Model
4. Explore Model

These sampling models are briefly described in Table 2.3.

In the batch model, samples are prerecorded by an oracle. These samples are given to the algorithm as a single batch and the problem of exploration is avoided. The main problem with the batch model is that the quality of the output policy depends on the data collected by the oracle. If the oracle does not collect a representative data set, then it may be impossible to learn a good policy. Interestingly, [33] have

developed generalization bounds that extend to states not sampled by the oracle under some restrictive settings, but the fundamental problem is that the derived policy depends on the samples provided by the oracle.

A generative model allows a learning algorithm to directly sample any state-action pair in the MDP without transitioning to the corresponding state. This sampling model is the most powerful of the discussed sampling models because it allows the learning algorithm to observe every state-action pair with little effort. With a generative model, a learning algorithm can learn a near-optimal policy with at most $\tilde{O}(NK)$ samples where N is the number of states and K is the number of actions [34], where $\tilde{O}$ suppresses log factors. The main problem with the generative model is that it may not be realistic for a learning algorithm to have this level of access to many real-world problems.

In the reset model, the algorithm learns in a series of episodes. State-action pairs can only be sampled by executing a sequence of actions to reach the desired state-action pair. However, at any time the algorithm can execute a special reset action that transitions the algorithm to an initial state s_0 with probability 1. In this model, algorithms learn a policy π that is near optimal with respect to the initial state s_0.

In the explore model, the algorithm is initialized in an arbitrary state and learns from an uninterrupted sequence of experiences. The algorithm never resets to an initial state or initial state distribution. In this setting, optimality must be redefined because it may not be possible to visit every state-action pair. Kakade [30] states that an algorithm $\mathcal{A}$ executed on an MDP M is acting ϵ-optimally if the policy $\mathcal{A}_t$ followed by algorithm $\mathcal{A}$ at timestep t satisfies

$$V^{\mathcal{A}_t}(s_t) \geq V^*(s_t) - \epsilon \tag{2.16}$$

from the current state s_t for some selected $\epsilon > 0$. This means that an algorithm is acting ϵ-optimally. Algorithms whose sample complexity is polynomial in the number of states N, the number of actions K, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$ are referred to as Probably Approximately Correct in Markov Decision Processes (PAC-MDP, [6]).

2.2.4 Limitations of the MDP Model

While MDPs are more realistic than MBPs, they still make simplifying assumptions, which limit their ability to appropriately model many real-world problems. First, the MDP model assumes that the current state is always known with complete certainty. This is not always true, for example, in a robotics domain where the agent observes a limited regions of the environment with noisy sensors. Second, the MDP model assumes that the transition probability distributions depend only on the current state and executed action. In many real-world problems, the optimal course of action may depend on which states have been visited previously.

2.3 Summary

We have defined the two most popular frameworks (MBPs and MDPs) for modeling real-world decision problems. Learning in MBPs and MDPs both requires a strategy for managing the exploration/exploitation dilemma. If a learning algorithm always selects the action that it believes to be best (exploiting), then it may miss out on learning about some better alternative action. On the other hand, if a learning algorithm executes actions over and over to reduce its uncertainty, it may be wasting valuable resources.

MBP algorithms and MDP algorithms can be evaluated empirically and theoretically. While empirical evaluation is critical, theoretical analysis is need to generalize results over a broad class of problems.

3. REVIEW OF TRANSFER LEARNING

Transfer learning (TL) is a process by which a learning algorithm can use information gathered by working on previously experienced tasks to help the algorithm learn in novel tasks. The information acquired from previous tasks can bias the algorithm so that it learns more quickly [35]. In this case, the algorithm modifies itself to improve its quality when learning solutions to novel tasks.

The intuition behind learning to learn [4] and TL (Figure 3.1) is to reuse knowledge acquired while learning a solution to a previous task to speed up learning a solution to a novel task [36]. The previous tasks are known as source tasks because they are the source that the learning agent acquires knowledge from. The task that knowledge is transferred to is called the target task. The process of transfer learning can be broken down into four steps:

1. Initialize a source task learning algorithm.

2. Acquire knowledge by interacting with the source task.

3. Transfer the acquired knowledge to the target task learning algorithm.

4. Run the target task learning algorithm on the target task (hopefully reducing learning time compared to learning from scratch).

In the first step, the agent initializes a learning algorithm for the source task. If the agent's library of task knowledge is not empty, this information could be used by the agent to construct a more efficient algorithm for learning in the source task. Otherwise the agent initializes a learning algorithm without prior knowledge. In the second step, the agent interacts with one or more source tasks and acquires knowledge

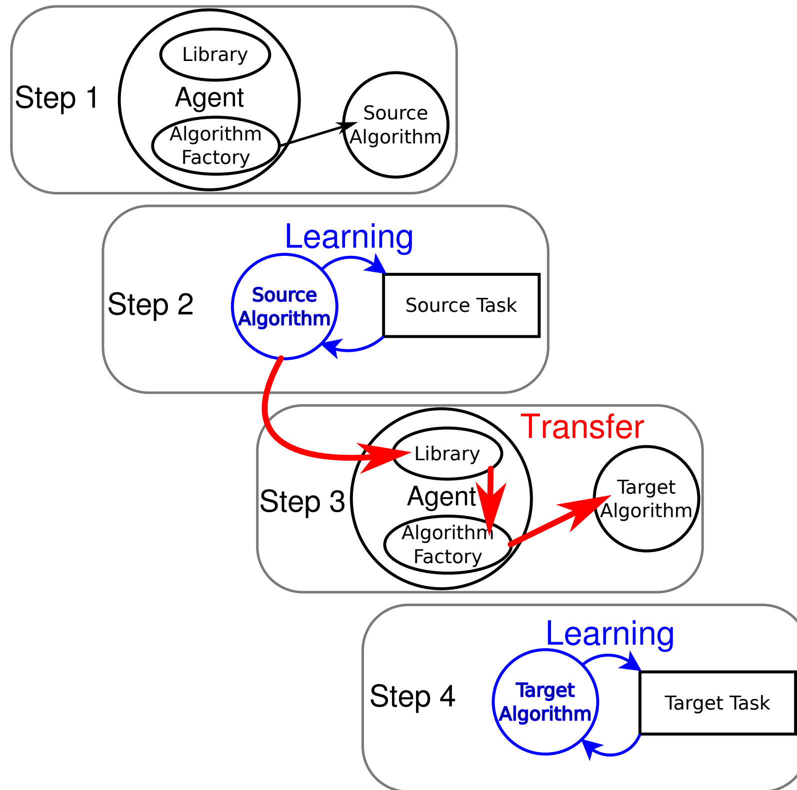


Figure 3.1: An agent constructs an instance of a source task algorithm to interact with a source task. Interaction with the source task generates knowledge, which is stored in the agent's library. The knowledge from the agent's library is transferred in a form that is meaningful in the context of the target task and that information is used to construct a target algorithm. The target algorithm is biased by the transferred knowledge, which will hopefully decrease the time needed to learn.

about these source tasks. Because the acquired knowledge is not directly from the target task it may need to be translated into a form that is meaningful for the target task. For example, the source task and the target task may have different state-action spaces [5] or different (but possibly overlapping) feature spaces [37]. The third step translates source task knowledge to a form that is meaningful in the target task and uses this translated knowledge to initialize a learning algorithm for the target task. Finally the knowledge is used by the agent to improve its learning performance in the target task.

TL has been applied to both supervised learning settings and reinforcement learning settings. It is worth discussing the terminology and ideas developed for the supervised setting because they have been influential in applying TL to the RL setting.

3.1 Supervised Transfer Learning

Most of the literature on TL focuses on the supervised learning setting. In a simple supervised learning setting, there exists an unknown concept function

$$c : X \to \{-1, 1\} \tag{3.1}$$

such that

$$c(x) = \begin{cases} 1 & \text{if } x \text{ is a member of the concept} \\ -1 & \text{otherwise} \end{cases} \tag{3.2}$$

for all $x \in X$. The concept function represents a high level concept such as "images containing Y" and "images without Y". The objective of a learning algorithm is to select a hypothesis $h : X \to \{-1, 1\}$ from a hypothesis space H to minimize

$$L(h) = \int_X ||c(x) - h(x)|| \tag{3.3}$$

which is the loss due to selecting hypothesis h where $||\cdot||$ is a norm function. To learn the function c, the learning algorithm is given a set containing $m \geq 1$ training examples $\{x_i, c(x_i)\}_{i=1}^{m}$ where $x_i \in X$. This setting is called supervised learning because the algorithm is given a set of training examples with the correct labels by a teacher.

In a supervised transfer learning setting, the algorithm needs to learn two or more concept functions. For simplicity, assume that the algorithm is supposed to learn unknown concepts c_1 and c_2. If these two concepts are arbitrarily different, then learning the concepts together will probably not reduce the complexity of learning. However, if the two concepts share a relationship, then learning them together may be helpful.

Caruana [38] investigated learning several real-world classification problems simultaneously. Artificial neural networks trained with backpropagation and classification trees were simultaneously trained on different but related classification tasks. Caruana [38] found better classification performance when a single learning algorithm was trained on multiple concepts simultaneously, because the algorithms were able to learn a mutually beneficial internal representation. These experiments also provide support for the more general hypothesis that many real-world tasks share common structure that can be exploited when learning multiple concepts together.

Baxter [35] developed a theory of inductive bias. In this paper the author investigates a model for automatically learning inductive bias. The important contribution of [35] is that the author formalizes inductive bias learning and describes the process as restricting a hypothesis space. This description fits nicely with other literature on machine learning that classifies the sample complexity of algorithms based on their hypothesis class [21, 39].

Ben-David et al. [40] present a theory for combining multiple sources of labeled

data sampled from different distributions to minimize loss. Their formulation expresses the trade-off between bias introduced by data sampled from a non-target distribution and the potential reduction in variance achieved by increasing the size of the data set.

The main limitation of TL in a supervised setting is that tasks are restricted to concept learning and regression. In this dissertation, we are interested in learning without a supervisor providing the answers. Instead we are interested learning policies for acting optimally in complex environments. To accomplish this goal, we need to consider transfer applied to RL.

3.2 Reinforcement Transfer Learning

Research on TL in a supervised learning setting set the stage for applying transfer to RL. However, applying TL to RL is considerably more complex, because policies are evaluated over a sequence of timesteps, instead of evaluating the class label assigned to a single input. Task similarity is also more complex due to the notion of states and actions in RL.

In RL learning tasks are represented by MDPs. Although it is possible to consider transfer from multiple source tasks to a target task, we consider only a single source task for clarity and simplicity. Given a source task $\Omega_{\mathrm{SRC}} = \langle S_{\mathrm{SRC}}, A_{\mathrm{SRC}}, T_{\mathrm{SRC}}, R_{\mathrm{SRC}}, \gamma \rangle$ and a target task $\Omega_{\mathrm{TRG}} = \langle S_{\mathrm{TRG}}, A_{\mathrm{TRG}}, T_{\mathrm{TRG}}, R_{\mathrm{TRG}}, \gamma \rangle$, it is not difficult to understand why transfer in this setting is more complex than supervised TL. The problem is that tasks can differ by the number of states they have (i.e. S_{SRC} may not equal S_{TRG}). The action sets may differ (i.e. A_{SRC} may not equal A_{TRG}). The transition probabilities T_{SRC} and T_{TRG} may differ, and even the reward functions R_{SRC} and R_{TRG} may differ. What this means is that the source and target tasks may be completely different. A successful approach to transfer learning must provide

mechanisms for dealing with a subset or all of these potential differences.

TL approaches applied to reinforcement learning can be broadly categorized as either multitask learning or general transfer learning.

3.3 Multitask Learning

The key assumption of multitask learning is that new tasks are sampled from a distribution over MDPs and that the distribution favors a subset of MDPs that share exploitable structure [41, 5]. Although not necessary, most research on multitask learning has assumed that the tasks share the same state-action space and focus on differences in the transition probabilities or reward functions.

For example, Fernández et al. [42] focused on learning a library of reusable policies for a set of tasks. The key assumption is that a small subset of policies generally capture the desired behavior for the set of tasks.

Tanaka and Yamamura [41] examine learning action-value statistics for tasks sampled from a distribution. They use these values to bias the intial action-values when learning a novel task.

Multitask learning is a useful subclass of transfer learning because it assumes that the tasks of interest are distributed according to some distribution, however, it is limited by the fact that not all tasks with similar structure share the same state-action space. Next we will discuss general transfer learning methods.

3.4 Transfer via Intertask Mappings

General transfer learning methods attempt to share information between tasks that do not necessarily share the same state-action space but nevertheless share similar underlying structure. In this setting it is not immediately clear how information from the source task should be related to the target task. Taylor and Stone [43]

introduce the concept of an intertask mapping

$$h : S_{\text{TRG}} \times A_{\text{TRG}} \rightarrow S_{\text{SRC}} \times A_{\text{SRC}} \tag{3.4}$$

that maps state-action pairs from the target task to state-action pairs from the source task. An intertask mapping defines a relationship between two tasks even if they do not share the same state-action space.

3.5 Advantages of Transfer Learning

Learning to learn and lifelong learning offers a number of critical advantages over learning solutions to each task separately. The main advantages of learning to learn (compared to learning each task separately) come in three varieties:

1. Learning with fewer samples
2. Learning with less computation
3. Learning more safely

Sample efficient learning is extremely critical for applying algorithms to real world problems because the exploration that learning algorithms perform is undesirable. Whenever possible we would rather have an algorithm that already knows how to act optimally. We only apply learning when the optimal policy is not easy or impossible to determined *a priori*. Increasing sample efficiency reduces the exploration performed by the learning algorithm. By transferring knowledge from source tasks it may be possible to learn a near-optimal policy for a target task with fewer samples (or observations). The reason for this is that prior knowledge may help to constrain the solution space.

Prior knowledge can also help to improve computational efficiency. Computational efficiency is related to sample efficiency. If the sample complexity of an algo-

rithm is small, then the algorithm can sometimes have lower computational demands. Information about previous tasks can expose potential computational shortcuts for planning policies or previously acquired knowledge may be structurally similar to knowledge about the new task. For example, the action-values between two tasks may be similar [8] or the policy from a previously solved task may be similar to a new task [44].

Another interesting advantage of learning to learn that has received considerably less attention is that learning to learn may enable a learning agent to learn more safely over time. The basic idea is that the agent could learn about unsafe or dangerous situations in one task that are universally unsafe in future tasks. Under these conditions the agent could learn to avoid these situations even while it is learning in a new setting.

These advantages offer powerful motivation for investigating learning to learn. However, before learning to learn can be applied to real-world problems there are a number of challenges to overcome.

3.6 Challenges

Despite the motivating advantages behind learning to learn, there are a number of difficulties associated with implementing the concept successfuly on real-world systems. Although there are many potential challenges, the main problems that we focus on in this dissertation are:

1. What knowledge should be transferred?

2. How can prior knowledge be learned?

3. How can prior knowledge be transferred?

4. How should we evaluate a transfer learning system?

3.6.1 What Knowledge should be Transferred?

Transfer learning requires learning knowledge from source tasks and then transferring that knowledge to the target task. What prior knowledge should be learned and transferred? Previous research has investigated transferring many kinds of knowledge. For example,

1. Action-Values [45, 46, 8]

2. Policies [44, 42]

3. Models [4]

4. Instances [36, 10]

One important lesson discovered by experiments in transfer learning is that some transferred knowledge is detrimental to learning. This phenomenon is called negative transfer [5, 43]. The problem is that the same kind of knowledge is sometimes useful and other times counterproductive. For example, in a TL setting where the source task and the target task are both maze navigation problems, if the maze in the source task is completely different than the maze in the target task, knowledge transferred from the source task to the target task may be misleading. If the target task learning algorithm makes assumptions about the target task based on its knowledge of the source task, then it will likely perform worse than if the algorithm ignores source task knowledge.

The kind of knowledge transferred also affects the kinds of learning algorithms that can be used [5]. For example, if information about the action-values of the target task can be learned from source tasks, this information can only be exploited by learning algorithms that use action-values. On the other hand, if part of the

optimal policy is transferred, this information could be exploited by almost any RL algorithm.

In this dissertation, we look to previous empirical studies of transfer learning to suggest plausible kinds of knowledge to transfer. The main lessons learned from experimenting with applying TL to RL is that the kind of knowledge to transfer depends on the relationship between the tasks and the learning objective.

3.6.2 How can Prior Knowledge be Learned?

Some prior knowledge would be very useful for an agent to have, but if it is difficult or impossible to learn, then that knowledge may not be practical for TL. We break this problem down into three questions:

1. How can the knowledge be learned?
2. How does the source task algorithm perform empirically?
3. What is the sample complexity of learning this knowledge?

First off, we need to identify an algorithm for learning the knowledge. If no such algorithm exists, this kind of knowledge is not a plausible candidate for learning to learn, no matter how useful it is for learning a novel task.

Once an algorithm for learning knowledge is established, we need to test the algorithm empirically. This process can help to determine whether or not the knowledge can be learned efficiently. It can also help to generate intuition about what conditions cause the algorithm to succeed or fail.

However, no number of empirical experiments will identify whether the algorithm always succeeds. The algorithm should be theoretically analyzed to determine its sample complexity under a wide range of situations. This analysis can provide precise situations when the algorithm will succeed or fail.

3.6.3 How can Prior Knowledge be Transferred?

How knowledge can be transferred depends on the relationship between the source tasks and the target task and the kind of knowledge extracted from the source tasks. Because an agent learns prior knowledge from one or more source tasks, the acquired knowledge may initially be unsuitable or not meaningful in the target task. The knowledge may need to be translated into a form that is directly applicable to the target task. When the source task and target task share the same state-action space it may be possible to exploit this relationship to transfer knowledge.

A common problem is that the source task and the target task have different state-action spaces. Taylor and Stone [43] introduce the concept of an intertask mapping for relating two tasks with different state-action spaces. For example, given a source task $\Omega_{\text{SRC}} = \langle S_{\text{SRC}}, A_{\text{SRC}}, T_{\text{SRC}}, R_{\text{SRC}}, \gamma \rangle$ and a target task $\Omega_{\text{TRG}} = \langle S_{\text{TRG}}, A_{\text{TRG}}, T_{\text{TRG}}, R_{\text{TRG}}, \gamma \rangle$, an intertask mapping is a function

$$h : S_{\text{TRG}} \times A_{\text{TRG}} \rightarrow S_{\text{SRC}} \times A_{\text{SRC}}$$

relating the state-action pairs from the target task to state-action pairs in the source task. If a state-action pair from the target task $(s, a) \in S_{\text{TRG}} \times A_{\text{TRG}}$ maps to a state-action pair in the source task $(x, b) \in S_{\text{SRC}} \times A_{\text{SRC}}$, it signifies that these two state-action pairs are somehow similar. Most research using intertask mappings has assumed that a mapping is known *a priori*, but some progress has been made on learning intertask mappings from data [47, 48, 11].

3.6.4 How should we Evaluate a Transfer Learning System?

Possibly the most critical question we can ask about a transfer learning system is how to evaluate it. Even under the standard single task reinforcement learning

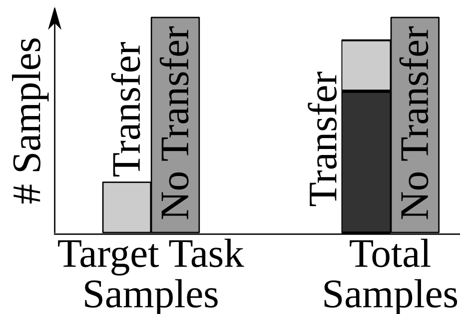


Figure 3.2: The sample complexity of transfer learning is more complex than sample complexity in a single task setting. The number of samples is distributed over the source and target task.

setting there are many criteria for evaluating and comparing algorithms. In the transfer learning setting there are even more options. For example, the number of samples is distributed over the source and target task (Figure 3.2). If acquiring samples from the source task is somehow less costly than acquiring samples from the target task we may be tempted to ignore the cost of learning in the source task. On the other hand we might count the total sample complexity of learning in both the source and the target task.

In this dissertation we take a two tiered approach to evaluation. We apply empirical evaluation and theoretical evaluation. We use theoretical analysis to motivate the development of new algorithms. We analyze the sample complexity of learning the knowledge to be transfered and we analyze how that transferred knowledge affects the sample complexity of learning in the target task. By analyzing both source and target task sample complexities, we have the potential to consider total sample complexity or target task sample complexity alone. The theoretical analysis also provides precise conditions for success and failure. However, the theoretical assumptions needed for analysis are often violated in real-world problems. Because of this problem, we also test our algorithms in complex environments.

3.7 Summary

Transfer learning (TL) is learning from source tasks and applying the learned knowledge to improve performance in a target task. Performance is usually a measure of how quickly the algorithm learns an acceptable policy in the target task [36, 43]. Transfer learning was first applied to supervised learning settings, which provided the foundation for applying transfer learning to RL. Transfer is more complex between RL tasks because RL tasks includes the notion of states and actions, and policies cannot be evaluated at a single timestep, whereas the error of a regression function or classifier can be measured for a single instance.

Another difference between applying TL to RL and supervised learning is that many different kinds of prior knowledge can be transferred between tasks. Previous research has considered transferring action-values, samples, policy information, transition models, etc. The kind of prior knowledge that is most useful depends on the relationship between the source tasks and target task.

When tasks do not share the same state-action space, they can be related using the concept of an intertask mapping [43]. Multitask RL assumes that tasks are drawn from a probability distribution.

The main gap in the literature on applying TL to RL is that there is no finite sample analysis describing how TL affects the sample complexity of RL. Throughout this dissertation, we will focus on the creation of TL+RL algorithms with provable bounds on sample complexity.

4. TARGETED EXPLORATION BY PRUNING STATES

Previous research has thoroughly established the importance of prior knowledge in machine learning. Prior knowledge is any kind of true additional information exploited by a learning algorithm, that the learning algorithm itself did not derive from data. Prior knowledge can range from partial solutions to low dimensional representations. The potential benefit depends on the kind of prior knowledge given.

In this section, we consider what kinds of prior knowledge can be used to speed up reinforcement learning (RL) algorithms.

4.1 Background

Previous research has investigated the application of several kinds of prior knowledge to RL.

One popular way to supply prior knowledge to value-based RL algorithms is to set the initial action-values [1]. If R_{MIN} is the minimum possible immediate reward and R_{MAX} is the maximum possible immediate reward, then the value function is constrained to the interval $\left[\frac{R_{\text{MIN}}}{1-\gamma}, \frac{R_{\text{MAX}}}{1-\gamma}\right]$. Setting all of the action-values to $\frac{R_{\text{MIN}}}{1-\gamma}$ causes an RL algorithm to avoid exploration unless some exploration strategy is build into the RL algorithm. If the action-values are set to $\frac{R_{\text{MAX}}}{1-\gamma}$ the RL algorithm will thoroughly explore the state-action space until settling on a policy. However, if more information is known about the action-values of the optimal policy they can be set to bias exploration so that the initial policy can be quickly discovered. Strehl et al. [6] introduce the concept of an admissible heuristic U, defined by

$$Q^*(s,a) \leq U(s,a) \leq \frac{R_{\text{MAX}}}{1-\gamma} \quad , \tag{4.1}$$

where Q^* is the optimal action-value function and prove that the R-MAX and Delayed Q-learning converge to near-optimal policies. Further, if the admissible heuristic is close to the optimal action-value function, it can greatly reduce the sample complexity of those algorithms.

Another way to supply prior knowledge is by modifying the reward function to speed up learning. This idea is known as reward shaping. Mataric [49] demonstrated that multiple reward functions can be used to speed up learning. The main idea is that reward functions can be used to specify subtasks that are necessary to accomplish the desired objective. Ng et al. [50] present a class of reward transformations that do not change the optimal policy and prove that all transformations that do not change the optimal policy belong to this class.

Hierarchy can provide important prior knowledge. Dietterich [3] introduce the hierarchical MAXQ algorithm. Given a hierarchical description of subtasks, the MAXQ algorithm decomposes the learning problem and is able to learn more efficiently than "flat" or non-hierarchical learners.

Additional structure can enable faster learning. Sherstov and Stone [51] introduced the concept of Relocatable Action Models (RAMs) for learning efficiently in environments with many actions. The RAM concept abstracts the result of a state transition from a particular state by introducing the notion of outcomes. The state space can then be partitioned into a set of classes where all of the states in the same class have the same probability distribution with respect to outcomes. Leffler et al. [52] were the first to use the phrase "Relocatable Action Model" to describe this structure and they proved that RAM-MDPs are equivalent to MDPs and introduced an algorithm RAM R-MAX that has sample complexity dependent on the number of classes and outcomes rather than the number of states and actions.

Dynamic Bayesian Networks also provide valuable structural information that

can be used to improve RL. Kearns and Koller [53] demonstrated that the sample complexity of a modified version of their E^3 algorithm can learn with sample complexity that is polynomial with respect to the number of conditional probability table parameters that need to be learned, which typically grows logarithmically with respect to the number of states.

Many researchers have applied function approximation to RL to approximate the action-value function. To use function approximation, the designer of the algorithm must make assumptions about the optimal action-value function. This can also be considered a form of prior knowledge. For example, the kernel or basis functions used with linear function approximators can be considered important knowledge for the success of a function approximator.

So far we have not defined what we mean by "speeding up" learning. One potential definition that has been used in many previous works is that the average cumulative reward of an algorithm with prior knowledge over a finite number of episodes is less than the average cumulative reward achieved by the base algorithm. Alternatively we can determine whether or not prior knowledge speeds up a base RL algorithm by comparing the sample complexity of a based algorithm with the sample complexity of the same algorithm given the prior knowledge. The advantage of considering sample complexity is that, as a theoretical result, its analysis holds over a broad range of tasks. In this section, we consider both of these definitions of speeding up learning. This gives us a better idea about how a particular kind of prior knowledge affects learning speed.

4.2 Algorithm: STAR-MAX

One kind of prior knowledge not mentioned in the previous section is specifying a subset of actions that are valid from each state or specifying a subset of states that

should be explored with priority. We describe an algorithm that can make use of this kind of prior knowledge.

State TArgeted R-MAX (STAR-MAX, Algorithm 6) takes as arguments:

- S : a state set
- A : an action set
- γ : a discount factor
- m : the number of visits to a state-action pair before it is considered known
- ξ : the set of states to actively explore ($\xi \subseteq S$)
- β : a recovery policy that returns the algorithm to a state ξ

STAR-MAX [54] is a generalization of the popular model-based R-MAX [15] algorithm. The main difference between STAR-MAX and R-MAX is that STAR-MAX is given an exploration envelope ξ and a recovery rule β. Together these additional pieces of information allow STAR-MAX to direct is exploration to the states in ξ, instead of exploring more exhaustively.

An exploration envelope $\xi \subseteq S$ is the set of states where exploration should be focused by the learning algorithm. When ξ is small, only a few states need to be explored and the majority of states can be ignored. This allows an RL algorithm to settle on a policy without exhaustively (or nearly exhaustively) explore the state space.

A recovery rule is a partial policy defined on the states $S \backslash \xi$ that "quickly" directs an RL algorithm back to a state in ξ. In practice, the recovery rule need only direct an RL algorithm back to a state in ξ in a small number of timesteps (in expectation). However, in the theoretical analysis the recovery rule can be handled in several ways.

Algorithm 6 State TArgeted R-MAX (STAR-MAX)

Require: $S, A, \gamma, m, \xi, \beta$

1: **for all** $(s, a) \in S \times A$ **do**
2: **if** $s \in \xi$ **then**
3: $Q(s, a) \leftarrow \frac{R_{\text{MAX}}}{1-\gamma}$
4: **else**
5: $Q(s, a) \leftarrow \frac{R_{\text{MIN}}}{1-\gamma}$
6: **end if**
7: $n(s, a) \leftarrow 0$ {# visits to (s, a)}
8: $r(s, a) \leftarrow 0$ {Cumulative reward at (s, a)}
9: **for all** $s' \in S$ **do**
10: $l(s, a, s') \leftarrow 0$ {# transitions $(s, a) \rightarrow s'$}
11: **end for**
12: **end for**
13: **for** $t = 1, 2, 3, \ldots$ **do**
14: Let s denote the state at time t.
15: **if** $s \in \xi$ **then**
16: Choose action $a := \arg\max_{b \in A} Q(s, b)$.
17: **else**
18: Choose action $a := \beta(s)$.
19: **end if**
20: Let r be the immediate reward and s' be the next state after executing action a from state s.
21: **if** $n(s, a) < m$ **then**
22: $n(s, a) \leftarrow n(s, a) + 1$
23: $r(s, a) \leftarrow r(s, a) + r$
24: $l(s, a, s') \leftarrow l(s, a, s') + 1$
25: **if** $s \in \xi$ and $n(s, a) = m$ **then**
26: Run Value Iteration on model (Algorithm 7).
27: **end if**
28: **end if**
29: **end for**

Algorithm 7 Estimate Model $(\hat{T}, \hat{R})$

Require: s, a, s', m, ξ

1: **if** $s \in \xi$ and $n(s,a) \geq m$ **then**
2:     **return** $\left(\frac{l(s,a,s')}{n(s,a)}, \frac{r(s,a)}{n(s,a)}\right)$ {Sufficiently explored}
3: **else**
4:     **if** $s \in \xi$ and $s = s'$ **then**
5:         **return** $(1, R_{\text{MAX}})$ {Under explored, in ξ}
6:     **else**
7:         **if** $s = s'$ **then**
8:             **return** $(1, R_{\text{MIN}})$ {Not in ξ}
9:         **else**
10:             **return** $(0, R_{\text{MIN}})$
11:         **end if**
12:     **end if**
13: **end if**

One way is to perform the analysis in the reset sampling model where every state is augmented with a special action that allows the algorithm to reset itself to an initial state, which could be thought of as a recovery rule. A second way to handle the recovery rule is to assume that for all states not in ξ, the optimal policy is given as the recovery rule. This would allow a straightforward analysis, because the algorithm can still converge to a near-optimal policy. The main point is that the recovery rule is mainly a practical consideration, rather than a theoretical one.

When STAR-MAX is initialized (lines 1 through 12), a data structure Q, used to estimate the optimal action-values, is initialized so that any state action pair with a state in ξ is initialized optimistically; otherwise it is initialized pessimistically. This encourages the agent to explore states in ξ and avoid other states. Data structures are also initialized to count $n(s,a)$ the number of times an agent has visited a particular state-action pair $(s,a) \in S \times A$, $l(s,a,s')$ the number of times an agent visited state-action pair $(s,a) \in S \times A$ and transitioned to $s' \in S$, and the cumulative reward $r(s,a)$ received when the agent visited state-action pair $(s,a) \in S \times A$.

After initialization, the learner enters a loop (line 13) interacting with its environment. It receives the current state s (line 14). Then if the current state is in the exploration envelope the agent will use the estimated Q function to greedily select what it believes to be the best action (line 16). On the other hand, if the state s is not in ξ, then the recovery rule selects the action (line 18). This way, if the agent leaves ξ, then the recovery rule is able to return the agent to a state in ξ. Finally, if the state-action pair (s, a) is unknown, then each of the counters is updated, and if a new state action pair becomes known (line 25), then Q is updated by running value iteration on the estimated model ($\hat{T}$ and $\hat{R}$). The estimated model, outlined in algorithm 7, returns a pair where the first element is the state transition probability and the second element is the immediate reward.

Now that we have an algorithm that can make use of prior knowledge about where to explore we would like to understand how prior knowledge affects the algorithms sample complexity. Because of some of the details of STAR-MAX it is not easy to provide a sample complexity analysis. However, because the algorithm is similar to R-MAX, we can provide a rough comparison of sample complexity. When only considering the number of state N and the number of actions K, the upper bound on the sample complexity of exploration of R-MAX is

$$\tilde{O}\left(N^2 K\right)$$

where $\tilde{O}$ suppresses log factors [30, 6]. Now the proof of this upper bound depends on the pidgin hole principle with respect to the number of state-action pairs that need to become "known" before there are no more possible uknown state-action pairs. However, STAR-MAX reduces the number of state-action pairs that need to be well

modeled by including ξ as prior knowledge. Thus, STAR-MAX only needs

$$\tilde{O}\left(N|\xi|K\right)$$

timesteps before all modeled state-action pairs become known. In other words, the savings in sample complexity is proportional to the number of states not in ξ. By eliminating more states from exploration we can learn much faster in the worst case MDP. However, this depends on the assumption that the given recovery rule β selects the same actions as the optimal policy. If the recovery rule is arbitrarily bad, then STAR-MAX may perform poorly. This is the price of targeted exploration. In the following experiments, we show that for several problems a reasonable recovery rule can be learned on-line.

4.3 Experiment: Simple Case

Our first question was whether or not the exploration envelope concept resulted in better empirical performance. To test this we constructed a series of stochastic gridworld tasks, where the agent always starts in the same state and finds a goal in the environment by moving up, down, left, or right. An example gridworld task with an exploration envelope shaded in gray can be seen in Figure 4.1a. The green square represents the initial state of the agent, the blue square represents the agent's current location, and the red square represents the goal state. Figure 4.1b shows the visitation table derived from running the STAR-MAX algorithm on the task in Figure 4.1a. The visitation table was constructed by recording the number of times that the learning algorithm visited each state in the gridworld. States that were visited many times are brighter than states that were visited infrequently. The important point to notice about Figure 4.1b is that STAR-MAX explores almost entirely within the states given by the exploration envelope.

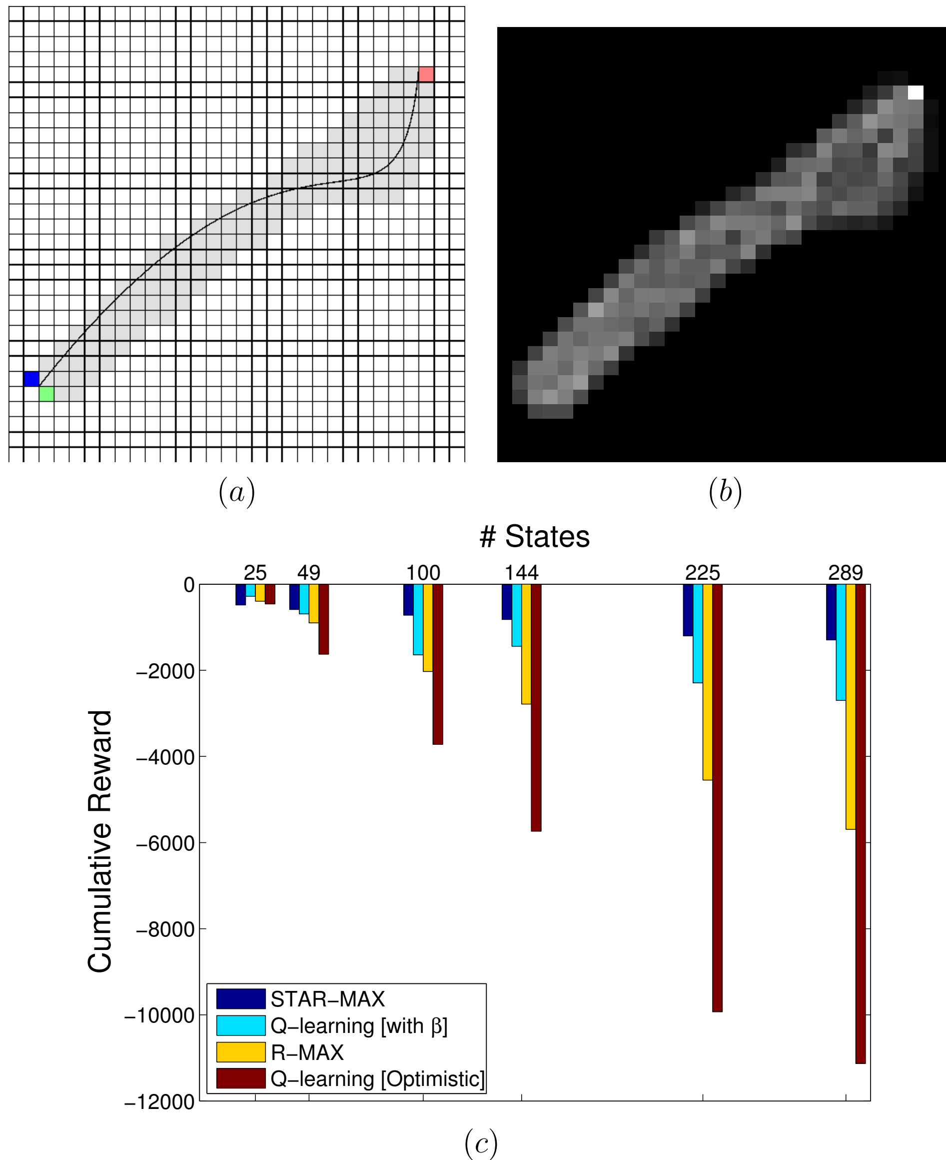


Figure 4.1: (a) Example gridworld with exploration envelope shaded in gray. The start state is denoted by a green square, the agent by a blue square, and the target state is denoted by a red square. (b) Example visitation table by the STAR-MAX algorithm (lighter cells were visited more frequently). (c) Comparison of cumulative reward between multiple RL algorithms as the number of states in the gridworld increases. Notice that the cumulative rewards are negative because the task gives negative rewards to encourage the algorithm to move to the target state as quickly as possible. STAR-MAX scales much better than Q-learning and R-MAX. Adapted from Mann and Choe [54].

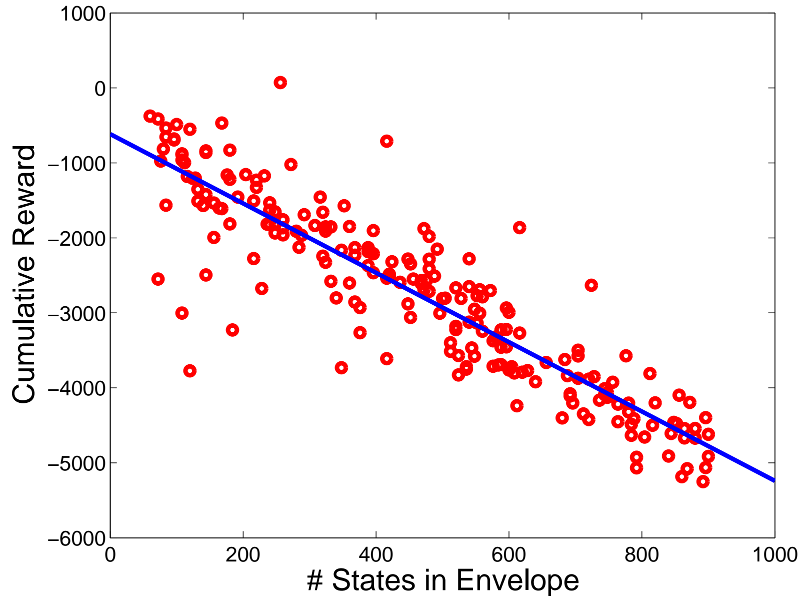


Figure 4.2: As the number of states in the STAR-MAX exploration envelope increase, the cumulative reward decreases linearly. Adapted from Mann and Choe [54].

To test how STAR-MAX scales compared to other common RL algorithms, we generated similar gridworld domains with 25, 49, 100, 144, 225, 289 states, with exploration envelopes similar to the one showed in Figure 4.1a but increased in number of states as necessary due to the increased size of the underlying gridworld. We compared STAR-MAX to the R-MAX algorithm, the popular Q-learning algorithm with ϵ-greedy exploration and optimistically initialized action-values, and Q-learning confined to the same exploration envelope as STAR-MAX. We recorded the cumulative reward scored by each of the algorithms. Figure 4.1c shows that STAR-MAX achieves higher cumulative reward compared to the other algorithms. Further the gap between STAR-MAX and the other algorithms increases as the number of states increases. Figure 4.2 shows that the cumulative reward decreases approximately linearly as the number of states in the exploration envelope increase.

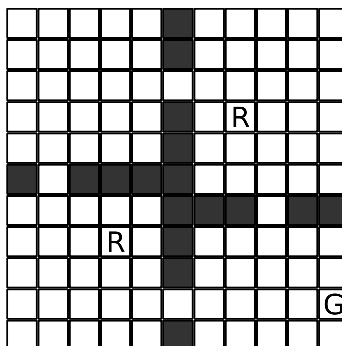


Figure 4.3: The red herring domain introduced by [2]. This gridworld domain contains two red herring states (denoted by R) with small rewards and one goal state (denoted by G) that gives a large reward. Learning algorithms that explore too little may settle on one of the suboptimal red herring states rather than finding the goal state.

4.4 Experiment: Dropping Arbitrary States

In many environments, the vast majority of states are irrelevant to the task. Thus, it makes sense that in those tasks, learning can be sped up by simply dropping random states. If only a few states are relevant, then the chances of dropping a relevant state may be very small. Therefore, we may be willing to accept the small probability of dropping a critical state if it will help to reduce the sample complexity of exploration.

To test this possibility, we conducted an experiment using the Red Herring domain (Figure 4.3) introduced by Hester et al. [2]. The Red Herring domain (see figure 4.3) is a gridworld instance introduced by Hester and Stone [55] to demonstrate a potential weakness of the RL-DT algorithm. The space is partitioned into four rooms. The initial state is selected randomly from one of the cells in the top left room. All states produce an immediate reward of -1 except for the two "red herring" states marked by "R", which provides a reward of 0 and terminates, and the goal state marked by "G", which gives a reward of $+25$ and terminates. For the Red Herring domain we used a value of $m = 10$ to match with experimental results from other

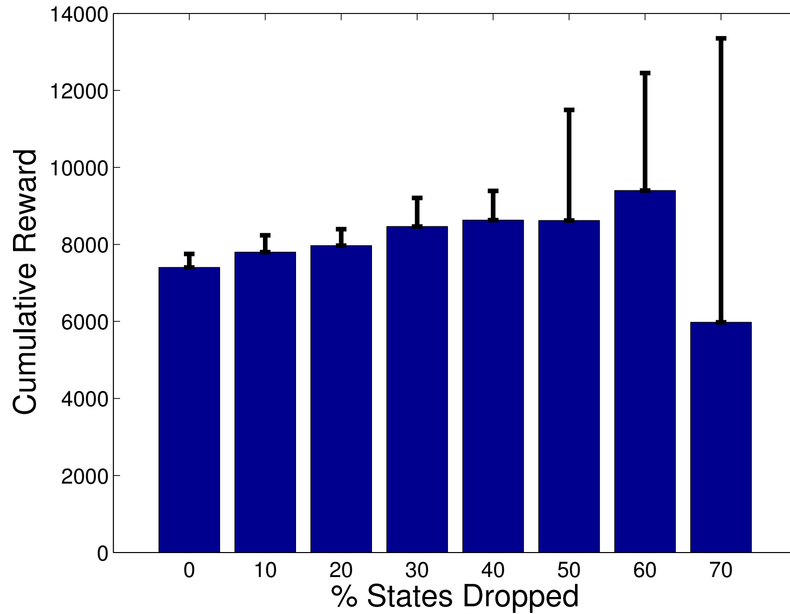


Figure 4.4: Cumulative reward increases simply by randomly dropping states (or state-action pairs) from the exploration envelope in the Red Herring domain [2] up to about 60% of the number of states. Adapted from Mann and Choe [54].

work. This domain is interesting because learning algorithms that do not explore enough of the environment may settle on one of the suboptimal red herring states. The domain is also interesting because many states are conceivably irrelevant. For example, the states in the upper right corner or the states in the lower right corner may (under usual circumstances) be completely avoided while traveling to the goal state.

Provided that ξ contains the goal state "G", STAR-MAX and R-MAX will both eventually explore the goal state in favor of the "red herring" states. The main difference is that STAR-MAX will have fewer states to explore before exploiting.

Figure 4.4 shows that envelopes with more states receive less cumulative reward, which is expected. What is surprising is that envelopes constructed by randomly dropping a certain percentage of states improves cumulative rewards (statistically significant for 10% - 60% with p-values less than 0.02, see figure 4.4). This provides

evidence that in practice STAR-MAX can learn more efficiently than R-MAX even when little is known about the environment.

4.5 Experiment: Learning from Demonstration

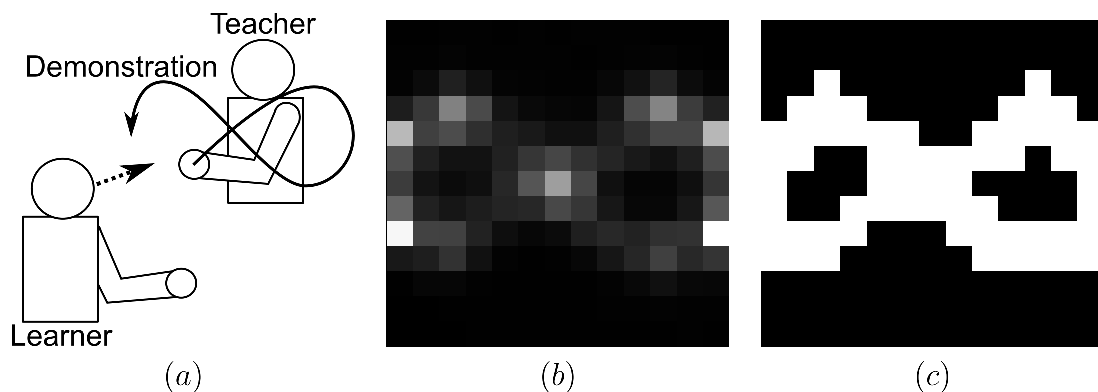
Exploration envelopes might be acquired by learning from demonstrations. When observing an expert perform a task, the expert emphasizes some regions of the state-action space rather than others. By focusing exploration on the regions of the state space that the expert emphasized (i.e., the exploration envelope) an RL algorithm can learn more quickly than focusing more broadly.

To demonstrate how this might work, we implemented a figure-eight tracing task. In the figure-eight tracing task, an agent must move around the environment in the shape of a figure-eight (Figure 4.5a). We first trained an imperfect expert using the Q-learning algorithm with ϵ-greedy exploration. While the expert was learning the figure-eight tracing task, we recorded its visitation table (Figure 4.5b). The visitation table shows that the expert visited some states much more frequently than others. Thus, we can conclude that some parts of the state space are irrelevant to the task.

To construct an exploration envelope (Figure 4.5c), we added the states to ξ that were visited very frequently by the expert, where very frequently was defined by the state being visited more than the 95-percentile state in the visitation table. This created a sparse exploration envelope for the STAR-MAX algorithm. Figure 4.5d shows that STAR-MAX was able to learn the figure-eight tracing task much more quickly than either R-MAX or Q-learning.

4.6 Discussion

Eliminating states from consideration is useful for speeding up RL. STAR-MAX demonstrates this point both in terms of sample complexity analysis and in experiments. It is not difficult to see why eliminating states from the search space can



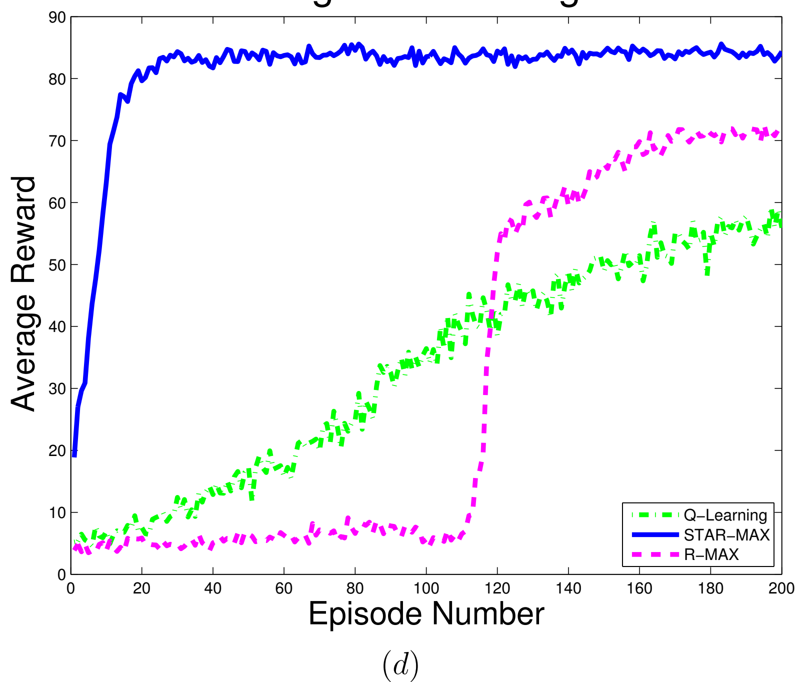


Figure 4.5: (*a*) Learning to perform a figure-eight from demonstrations by a teacher. (*b*) A state visitation table generated by a teacher. (*c*) The exploration envelope extracted from (*b*) by selecting the states visited more frequently than the 95th percentile. (*c*) Comparison of average reward at the figure-eight task by Q-learning, R-MAX, and STAR-MAX initialized with the exploration envelope in (*c*). Notice that STAR-MAX quickly achieves high average reward at the task, while R-MAX thoroughly explores before settling on a reasonable policy. Q-learning slowly improves its policy. Adapted from Mann and Choe [54].

speed up learning, and we argue that the fact that this concept is simple to understand is a great advantage compared to other methods of specifying prior knowledge. For example, Bayesian approaches specify prior knowledge in terms of a prior distribution, which may be difficult to understand or design. In contrast, our approach simply asks a designer to specify any states they know are probably not relevant to the task in advance. If the designer cannot eliminate any states, then the algorithm is equivalent to the R-MAX algorithm, which will solve the problem (albeit slower).

In practice, we can often specify *a priori* that some states are probably not relevant to solving a task. We can also often specify a basic policy to use as a recovery rule or learn a recovery rule using reasonable assumptions. The recovery rule is a significant drawback from a theoretical perspective because it is domain specific and may not exist or be difficult to learn in some domains. However, the recovery rule can be avoided in certain theoretical settings. For example, in the reset sampling model, it is assumed that every state is augmented with a special action a_{RESET} that returns the agent to a start state. In this case, the reset action can be used in stead of a domain specific recovery rule.

In this section, we have eliminated states from exploration. However, in many scenarios it may make more sense to eliminate some state-action pairs. That is because in stochastic environments it is often possible that an RL algorithm will end up in an undesirable state and still have to make intelligent decisions. If we know that taking a particular action in that state would make things even worse, we can save the RL algorithm from making a poor decision in advance by eliminating that state-action pair. This allows the RL algorithm to explore multiple actions in that state, but avoids taking an action that we already know is a mistake. Another advantage of eliminating state-action pairs instead of states is that as long as at least one action is not eliminated at each state, then there is no need for a recovery rule.

Therefore, in future sections we will consider elimination of state-action pairs rather than states.

4.7 Summary

Prior knowledge is information given to a learning algorithm that the learning algorithm itself does not need to derive from data. It is well established that prior knowledge can greatly speed up learning. However, this depends on the kind of prior knowledge used and what is meant by "speed up". We have considered pruning states and state-action pairs, and measured "speed up" by the difference between the sample complexity of the base algorithm and the algorithm exploiting prior knowledge. We found that the improvement in speed is roughly proportional to the number of state-action pairs that are eliminated from consideration. In the following sections, we will use this approach of eliminating state-action pairs to scale RL algorithms to handle large problems.

5. ANALYSIS OF PRUNING ACTIONS

In the previous section, we saw that pruning states can decrease the time needed to learn a "good" policy for several different tasks. However, pruning states resulted in an awkward problem. What should the policy do when the learning agent accidentally enters a pruned state? In stochastic environments this can happen by chance. State-action pairs that normally transition to another state in the exploration envelope may have some small probability of transitioning to a state outside of the exploration envelope. To deal with this issue, we introduced the concept of a recovery rule. However, we then had to face the problem of how a recovery rule could be known in advance or learned. If we change our focus from pruning states to pruning state-action pairs, then we can evade the issue of learning a recovery rule and still achieve many of the same benefits.

In this section, we analyze the sample complexity of pruning actions from an MDP in two ways: (1) we consider the sample complexity of exploration when some state-action pairs can be explicitly ignored, and (2) we consider the sample complexity of exploration when the RL algorithm is initialized with action-values that implicitly eliminate certain state-action pairs. Although the analysis techniques used in the section are similar to those used by Kearns and Singh [27], Kakade [30], Strehl et al. [16, 6], Szita and Szepesvári [29] our results emphasize the impact of prior knowledge on sample complexity rather than sample complexity in the absolute worst case. The results derived in this section show how prior knowledge can decrease sample complexity and are foundational for the remaining sections.

5.1 Background

Throughout this section, we will be concerned with the sample complexity of exploration of an RL algorithm (Inequality (2.16) introduced by Kakade [30]). Recall that the sample complexity of exploration of an RL algorithm is the maximum number of timesteps that the algorithm acts according to a sub-ϵ-optimal policy, with high probability, where ϵ is a small positive value. We will use the phrases "sample complexity" and "sample complexity of exploration" interchangeably, except when noted otherwise.

For some $\epsilon > 0$, an MDP Ω, and a timestep t, if an RL algorithm is at state s_t and following a policy π, we say that that algorithm is following an ϵ-optimal policy provided that $V_\Omega^\pi(s_t) \geq V_\Omega^*(s_t) - \epsilon$. We say that a collection of action-value estimates $\hat{Q}$ are ϵ-accurate with respect to the optimal action-values for Ω if

$$\left|\left|Q_\Omega^* - \hat{Q}\right|\right|_\infty \leq \epsilon$$

where $||x||_\infty = \max_x |x|$ is the max-norm operator.

Strehl et al. [6] developed a framework for analyzing PAC-MDP algorithms under the explore sampling model. Before stating any theorems that will help us with our analysis, we need to explain two important concepts. The first concept is the notion of a greedy policy. Suppose an RL algorithm estimates the optimal action-values for an MDP Ω with a structure $\hat{Q}$. The RL algorithm is said to be acting according to a greedy policy π, if the action selected by the RL algorithm at a state $s \in S$ is

$$\pi(s) = \arg\max_{a \in A_s} \hat{Q}(s, a) \tag{5.1}$$

where $A_s \subseteq A$ is the set of all valid actions at state s. The second important concept

is the notion of an induced MDP. The induced MDP concept was introduced by **?**] in their analysis of the E[3] algorithm. Here we use the definition introduced by Strehl et al. [6], which has been slightly modified to facilitate the analysis of both R-MAX and Delayed Q-learning.

Definition 5.1. *(Strehl et al. [6]) Let* $\Omega = \langle S, A, T, R, \gamma \rangle$ *be an MDP and* $\kappa \subseteq S \times A$. *The* **induced MDP** *(and for reasons that will be made clear below, also known as a known state-action MDP) with respect to* Ω *and* κ *is denoted by* $\Omega_\kappa = \langle S, A, T', R', \gamma \rangle$ *where*

$$T'(s'|s,a) = \begin{cases} T(s'|s,a) & \textit{if } (s,a) \in \kappa \\ 1 & \textit{if } (s,a) \notin \kappa \textit{ and } s' = s \\ 0 & \textit{otherwise} \end{cases}$$

and

$$R'(s,a) = \begin{cases} R(s,a) & \textit{if } (s,a) \in \kappa \\ (1-\gamma)\hat{Q}(s,a) & \textit{otherwise} \end{cases}$$

where T' *defines the transitions probabilities and* R' *defines the reward function of* Ω_κ.

Inside the set κ the induced MDP Ω_κ has the exact same transition probabilities and reward function as the MDP Ω. However, outside of κ, Ω_κ has overly optimistic values compared to Ω. While an RL algorithm is executing in an MDP, it will observe some state-action pairs more than others. When the RL algorithm has observed a state-action pair enough times, it has enough samples from that state-action pair to model it well or to "know" that state-action pair. A known state-action MDP is an idealized version of what the RL algorithm has learned about the MDP. When an RL algorithm is exploring an MDP, the value of its policy will be the same in Ω and Ω_κ as long as the RL algorithm remains within κ. However, if the algorithm

escapes from κ, then the value of the policy in Ω_κ will be greater than or equal to the value of the policy in Ω. This is due to the definition of Ω_κ. If escaping from κ is very difficult, then the near-optimal policy for Ω_κ is also near-optimal for Ω. If it is easy to escape from κ, then an escape event is likely to occur because the RL algorithms internal model will overestimate the value of state-action pairs outside of κ. However, because there are a finite number of state-action pairs, the number of times escape events occur is bounded. After these escape events occur, then the RL algorithm must be acting near-optimally in Ω with high probability.

Now we can introduce one of the most important theorems introduced by Strehl et al. [6], as it will be useful in our analysis of RL algorithms.

Theorem 5.2. *(Strehl et al. [56, Proposition 1] and Strehl et al. [6, Theorem 10]) Let $\epsilon > 0$, $\delta \in (0, 1]$, and $\mathcal{A}(\epsilon, \delta)$ be any value-based greedy learning algorithm such that, for every timestep t, $\mathcal{A}(\epsilon, \delta)$ maintains action-value estimates $\hat{Q}_t \leq \frac{1}{1-\gamma}$ and there exists a set κ_t of state-action pairs that depends on the agent's history up to timestep t. We denote $\max_{a \in A} \hat{Q}_t(s, a)$ by $\hat{V}_t(s)$ and assume that the set κ does not change unless an action-value is updated (i.e., $\kappa_t = \kappa_{t+1}$ unless, $\hat{Q}_t \neq \hat{Q}_{t+1}$). Let Ω_{κ_t} be the known state-action MDP with respect to MDP Ω and π_t be the current greedy policy. Suppose that with probability at least $1 - \delta$ the following conditions hold for all state-action pairs $s \in S$ and timesteps $t \geq 1$:*

Condition 1: $\hat{V}_t(s) \geq V^*_\Omega(s) - \epsilon$ *(optimism),*

Condition 2: $\hat{V}_t(s) - V^{\mathcal{A}_t}_{\Omega_{\kappa_t}}(s) \leq \epsilon$ *(accuracy), and*

Condition 3: *the total number of updates of action-value estimates plus the number of times the escape event from κ_t can occur is bounded by $\zeta(\epsilon, \delta)$ (learning complexity).*

If $\mathcal{A}(\epsilon, \delta)$ is executed on Ω it will follow a 4ϵ-optimal policy on all but

$$O\left(\frac{\zeta(\epsilon,\delta)}{\epsilon^2(1-\gamma)^2}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps t with probability at least $1-2\delta$.

This is a useful theorem for sample complexity analysis because it makes only minimal assumptions about the RL algorithm. Strehl et al. [6] used Theorem 5.2 to derive PAC-MDP bounds for R-MAX and Delayed Q-learning. To apply the theorem, we need to demonstrate that the RL algorithm being analyzed follows a greedy policy with respect to its action-value estimates $\hat{Q}$, show that it maintains approximately optimistic action-values, show that the action-values are an accurate approximation of the action-values for the known state MDP, and bound the number of times ζ that an escape events and attempted updates might occur, where ζ is called the learning complexity. Then we can plug in the learning complexity to the bound provided by Theorem 5.2. We will use Theorem 5.2 in the next section and most of the bounds throughout this dissertation will depend implicitly on the theorem.

The lowest known upper bound on sample complexity of exploration for the R-MAX algorithm is

$$O\left(\frac{NK}{\epsilon^3(1-\gamma)^6}\left(N+\ln\frac{NK}{\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

due to [6, Theorem 11], where ϵ is the allowable error of the learned policy and δ is the acceptable probability of failure. If we ignore log factors this upper bound simplifies to

$$\tilde{O}\left(\frac{N^2K}{\epsilon^3(1-\gamma)^6}\right) \quad ,$$

where $\tilde{O}$ denotes the suppression of log factors. The squared dependence on the

number of states (N^2) is due to the fact that R-MAX constructs an accurate model of the MDP, which has N^2K parameters, and solves this model for its policy.

Strehl et al. [16, Theorem 1] provides the following upper bound on the sample complexity of exploration for the Delayed Q-learning algorithm:

$$O\left(\frac{NK}{\epsilon^4(1-\gamma)^8}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\ln\frac{NK}{\delta\epsilon(1-\gamma)}\right)$$

where ϵ is the allowable error of the learned policy at the current state and δ is the acceptable probability of failure. Again, by ignoring log factors

$$\tilde{O}\left(\frac{NK}{\epsilon^4(1-\gamma)^8}\right)$$

we get a simplified version of the bound. This bound has a better dependence on the number of states and actions than R-MAX. However, it has a slightly worse dependence on parameters $\frac{1}{\epsilon}$ and $\frac{1}{1-\gamma}$ that control the policy accuracy and planning horizon. Interestingly, Szita and Szepesvári [29] have presented a model-based algorithm inspired by both R-MAX and Delayed Q-learning that has sample complexity that is tighter than both R-MAX and Delayed Q-learning. In this dissertation, we will analyze R-MAX and Delayed Q-learning because their analyses are more straightforward than that of Szita and Szepesvári [29].

The bounds on sample complexity provided by [6] and [16] primarily provide worst case sample complexity bounds over all MDPs. On the other hand, we are interested in how sample complexity may change as more prior knowledge is available. In the next two sections, we will derive bounds for R-MAX and Delayed Q-learning when structures that eliminate actions are given.

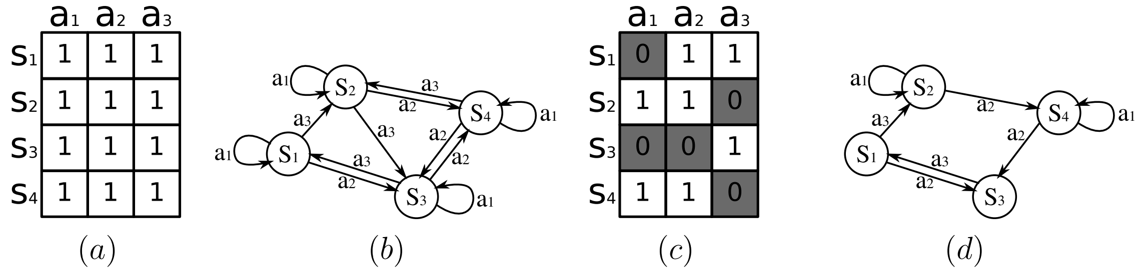


Figure 5.1: (a) A full exploration table with no state-action pruning. All state-action pairs may need to be explored to guarantee learning a near-optimal policy. (b) The 4-state, 3-action MDP described by the full exploration table. (c) A sparse exploration table with pruned state-action pairs. (d) The 4-state, 3-action MDP described by the sparse exploration table.

5.2 Explicit Action Pruning

Knowledge about which state-action pairs should be explored and which state-action pairs should not be explored can be provided in the form of an exploration table.

Definition 5.3. *Given a state set S and an action set A, a function $\xi : S \times A \to \{0, 1\}$ is called an* **exploration table** *(Figure 5.1) if (1) it encodes with* 1 *state-action pairs that may be explored by an RL algorithm and* 0 *state-action pairs that are not considered by an RL algorithm applied to some MDP over S and A and (2) at every state $s \in S$ there exists $a \in A$ such that $\xi(s, a) = 1$. Let $A_\xi(s) = \{a \in A | \xi(s, a) = 1\}$ denote the subset of valid actions that can be selected at state $s \in S$ and $K_s = |A_\xi(s)|$.*

An exploration table specifies a value of 1 for state-action pairs that may be explored by an RL algorithm and a value of 0 for state-action pairs that should never be visited. Using this notion of an exploration table, we can take an MDP Ω and transform it into a new MDP with fewer state-action pairs (Figure 5.1).

Definition 5.4. *Let $\Omega = \langle S, A, T, R, \gamma \rangle$ be an MDP and ξ be an exploration table. We denote by Ω_ξ the* **pruned Markov decision process (pruned MDP)** *with*

respect to Ω and ξ, such that from each state $s \in S$ only the actions $A_\xi(s)$ can be selected by a learning algorithm.

A pruned MDP is similar to the original MDP except that the pruned MDP has fewer state-action pairs. Unfortunately, this means that if the wrong state-action pairs are pruned, the optimal policy for the pruned MDP may be suboptimal for the original MDP. This leads us to the notion of optimality loss.

Definition 5.5. *Given MDP Ω and an exploration table ξ, the* **optimality loss** *between Ω and Ω_ξ is*

$$\mathcal{L}(\Omega, \Omega_\xi) = \max_{s \in S} \left(V^*_\Omega(s) - V^\pi_{\Omega_\xi}(s) \right) \tag{5.2}$$

where π is the optimal policy with respect to the pruned MDP Ω_ξ.

Optimality loss is a worst case measure defined by the maximum difference between the optimal value function for the original MDP Ω and the optimal value function for the pruned MDP Ω_ξ. Notice that any policy for Ω_ξ is also defined for Ω and has the same value in both MDPs. However, there could be policies for Ω that are not defined for Ω_ξ because they make use of some pruned state-action pairs. If the optimal policy for Ω is not defined for Ω_ξ, then this will result in non-zero optimality loss. The next lemma allows us to bound the worst case optimality loss based on which state-action pairs have been pruned by ξ.

Lemma 5.6. *Let $\alpha > 0$, ξ be an exploration table, and Ω be an MDP such that for every state $s \in S$ there exists an action $a \in A$ such that (1) $\xi(s, a) = 1$ and (2) $Q^*_\Omega(s, a) \geq V^*_\Omega(s) - \alpha$, then the optimality loss*

$$\mathcal{L}(\Omega, \Omega_\xi) \leq \frac{\alpha}{1 - \gamma}$$

where Ω_ξ is the pruned MDP with respect to Ω and ξ.

Proof. See the proof of Lemma 5.17 below. □

If the specified exploration table leaves behind actions that are α optimal at each state, then the optimality loss is bounded above by $\frac{\alpha}{1-\gamma}$. This suggests an objective for choosing exploration tables. We want to select an exploration table that eliminates as many state-action pairs as possible (sparsity) without eliminating state-action pairs that are close to optimal.

Next we will analyze how the sample complexity of exploration for R-MAX and Delayed Q-learning are influenced by pruning with an exploration table.

5.2.1 R-MAX

The following theorem states an upper bound on the sample complexity of exploration of R-MAX (Algorithm 3) in a pruned MDP, with appropriately chosen parameters m and ϵ_1.

Theorem 5.7. (R-MAX Exploration Table Bound) *Suppose that $\epsilon > 0$ and $\delta \in [0, 1)$, Ω_ξ is a pruned MDP with respect to an exploration table ξ and MDP $\Omega = \langle S, A, T, R, \gamma \rangle$. There exists inputs m and ϵ_1, satisfying $m = O\left(\frac{(\psi + \ln(\sum_{s\in S} K_s/\delta))}{\epsilon^2(1-\gamma)^4}\right)$ and $\frac{1}{\epsilon_1} = O\left(\frac{1}{\epsilon}\right)$, such that if R-MAX is executed on Ω_ξ with inputs m and ϵ_1, then the following holds. Let $\mathcal{A}_t$ denote R-MAX's policy at time t and s_t denote the state at time t. With probability at least $1 - \delta$, $V_\Omega^{\mathcal{A}_t}(s_t) \geq V_\Omega^*(s_t) - \epsilon$ is true for all but*

$$O\left(\frac{\sum_{s\in S} K_s}{\epsilon^3(1-\gamma)^6}\left(\psi + \ln\frac{\sum_{s\in S} K_s}{\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps t, where K_s is the number of valid actions for state $s \in S$ specified by the exploration table ξ and ψ is the maximum out-degree of any state-action pair.

Notice that even though we are using existing analysis techniques, this bound on the sample complexity of R-MAX has several improvements and modifications compared to the bound specified by Strehl et al. [6, Theorem 11].

First, the bound depends on the $\sum_{s \in S} K_s$ rather than NK. In the worst case, if $K_s = K$ for all $s \in S$, then the bound depends on NK. However, when the exploration table ξ is sparse, then the sum of state-action pairs can be very small compared to NK. This explains the key advantage of providing an exploration table. If the exploration table is sparse, then the sample complexity is decreased.

Second, the bound introduces and exploits the notion of the maximum out-degree ψ over all state-action pairs to achieve a tighter sample complexity bound in many MDPs. The out-degree of a state-action pair is the number of states with positive probability of being transitioned to. For example, if all state-action pairs are deterministic, then $\psi = 1$. If all state-action pair transition to 5 or fewer different states, then $\psi = 5$. If $\psi << N$, as it is in most previously studied problems (e.g., mountain car [1], taxi domain [3], Robocup Keepaway [57]), then the sample complexity of R-MAX is significantly smaller than the worst case where $\psi = N$. This adjustment to the theorem is only a minor change, but it helps to explain why R-MAX often finds near-optimal policies even when the m parameter is set to values that are much smaller than $O(N)$.

Our approach to the proof of Theorem 5.7 is to apply Theorem 5.2. However, this theorem has several conditions and concepts that need further explanation.

The first and possibly most important concept in the analysis of the R-MAX algorithm is the idea of “known” state-action pairs. The R-MAX algorithm takes a parameter m, which is the number of times it needs to visit a state-action pair before it is considered well-modeled.

Definition 5.8. *If an instance of the R-MAX algorithm initialized with $m > 0$ is executed on an MDP $\Omega = \langle S, A, T, R, \gamma \rangle$, then a state-action pair $(s, a) \in S \times A$ is considered m-**known** (or just known) if and only if (s, a) has been experienced at least m times.*

After visiting an "unknown" state-action pair m times, the R-MAX algorithm incorporates the reward and transition samples into its internal model and plans a new policy. We denote the set of m-known state-action pairs at timestep t by

$$\kappa_t = \{(s, a) \in S \times A \mid n(s, a) \geq m\} \tag{5.3}$$

where $n(s, a)$ is the number of times that the state-action pair (s, a) has been visited. If m is large enough then the internal model maintained by R-MAX will be accurate at the state-action pairs that are m-known. At all other state-action pairs, R-MAX will overestimate there values with $\frac{1}{1-\gamma}$, which is the highest possible value. This trick encourages R-MAX to systematically explore unknown regions of the state-action space. Furthermore, any time that the algorithm visits a state-action pair outside of κ, we call this an escape event.

The ideal internal model for R-MAX is the induced MDP (Definition 5.1) with respect to the known state-action pairs. The main significance of the induced MDP is that when κ is the set of known states, Ω_κ represents the ideal internal model, where every state-action pair transition probability distribution and reward is perfectly modeled if $(s, a) \in \kappa$ and results in a self-absorbing state with the highest possible immediate reward if $(s, a) \notin \kappa$.

Theorem 5.2 has several assumptions and conditions. The theorem assumes that the RL algorithm maintains a set of action-values $\hat{Q}_t$ bounded by $\left[0, \frac{1}{1-\gamma}\right]$ and that the algorithm follows a greedy policy $\pi_t(s) = \arg\max_{a \in A_s} \hat{Q}_t(s, a)$ at all timesteps

t. The conditions are (1) that at each state $s \in S$ the RL algorithm's maximum action-value estimate is not too much smaller than the optimal action-value for that state (i.e., $\max_{a \in A_s} \hat{Q}(s,a) \geq V^*_{\Omega_\xi}(s) - \epsilon$) (2) that the estimated action-values $\hat{Q}_t$ are always close to the action-values associated with running the greedy policy on the induced MDP $\Omega_{\xi,\kappa}$ (i.e., $\max_{a \in A_s} \hat{Q}_t(s,a) - V^\pi_{\Omega_{\xi,\kappa}}(s) \leq \epsilon$), and (3) that the number of escape events is bounded (since action-value updates can only occur when an escape event occurs).

The R-MAX algorithm always follows a greedy policy and for any finite $m > 0$ the number of times that an escape event can occur is bounded by $m \left(\sum_{s \in S} K_s\right)$. So the proof is mainly a matter of selecting values for the parameters m and ϵ_1 that satisfy the conditions 1 and 2 and plugging these values into the sample complexity bound given by Theorem 5.2.

Lemma 5.9. *If R-MAX is executed on an MDP Ω_ξ with parameters m and ϵ_1 where m satisfies*

$$m \geq \frac{C\left(\psi + \ln\left(\sum_{s \in S} K_s/\delta\right)\right)}{\epsilon_1^2(1-\gamma)^4} \quad ,$$

then $\left|V^\pi_{\Omega_{\xi,\kappa_t}}(s) - V^\pi_{\widehat{\Omega}}(s)\right| \leq \epsilon_1$ is true for all stationary policies π and timesteps t with probability at least $1 - \delta$, where Ω_{ξ,κ_t} is the induced MDP with respect to Ω_ξ and κ_t and $\widehat{\Omega}$ is the MDP specified by the internal model maintained by R-MAX.

Proof. The proof of this lemma follows from [6, Lemma 15] and the fact that our bound here only needs to hold over $\sum_{s \in S} K_s$ state-action pairs rather than NK state-action pairs. □

Now we are ready to prove Theorem 5.7. Our proof is similar to the proof of Strehl et al. [6, Theorem 11], however, we emphasize that there are several important differences.

Proof. (of Theorem 5.7) If we select m according to Lemma 5.9 then we have that $\left|V^{\pi}_{\Omega_{\xi,\kappa_t}}(s) - V^{\pi}_{\widehat{\Omega}}(s)\right| \leq \epsilon_1$ with probability at least $1-\delta$. If we choose $\epsilon_1 = \epsilon/2$, then with probability at least $1-\delta$ we have

$$\begin{aligned}
\max_{a\in A_s} \hat{Q}(s,a) &\geq V^*_{\widehat{\Omega}}(s) - \epsilon_1 \\
&\geq V^*_{\Omega_{\xi,\kappa_t}}(s) - 2\epsilon_1 \\
&\geq V^*_{\Omega_{\xi}}(s) - 2\epsilon_1 \\
&\geq V^*_{\Omega_{\xi}}(s) - \epsilon
\end{aligned}$$

where the first inequality is due to the fact that R-MAX computes ϵ_1-accurate action-values from its internal model $\widehat{\Omega}$. The second inequality is due to Lemma 5.9. The third inequality is due to the fact that the induced MDP Ω_{ξ,κ_t} always has a greater value (at every state) than Ω_ξ because the induced MDP has the same rewards and transition probabilities in ξ and maximum possible values at every other state-action pair. The last inequality is due to selecting $\epsilon_1 = \epsilon/2$. This satisfies condition 1 of Theorem 5.2 and condition 2 also follows from our choice of m and Lemma 5.9.

Now, we can obtain our result by setting $\epsilon_1 \leftarrow \epsilon/8$ and $\delta \leftarrow \delta/4$ and plugging into

$$\begin{aligned}
m &= \frac{C\left(\psi+\ln\left(\sum_{s\in S} K_s/\delta\right)\right)}{\epsilon_1^2(1-\gamma)^4} \\
&= \left(\frac{C\left(\psi+\ln\left(\sum_{s\in S} K_s/(\delta/4)\right)\right)}{(\epsilon/8)^2(1-\gamma)^4}\right) \\
&= \left(\frac{64C\left(\psi+\ln\left(4\sum_{s\in S} K_s/\delta\right)\right)}{\epsilon^2(1-\gamma)^4}\right) \\
&= O\left(\frac{\psi+\ln\left(\sum_{s\in S} K_s/\delta\right)}{\epsilon^2(1-\gamma)^4}\right)
\end{aligned}$$

and then the bound provided by Theorem 5.2, we have an ϵ-optimal policy with probability at least $1-\delta$ (by applying the union bound) on all but

$$O\left(\frac{\sum_{s\in S} K_s}{\epsilon^3(1-\gamma)^6}\left(\psi + \ln\frac{\sum_{s\in S} K_s}{\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps. $\square$

5.2.2 Delayed Q-learning

Now we turn our attention to the analysis of the sample complexity of exploration of Delayed Q-learning (Algorithm 5) with a pruned MDP. Our proof and the lemmas are similar to those found in Strehl et al. [6], however, several of the proofs require changes due to the fact that we are proving a slightly different result.

Theorem 5.10. (Delayed Q-learning Exploration Table Bound) *Suppose that $\epsilon > 0$ and $\delta \in [0, 1)$, Ω_ξ is a pruned MDP with respect to an exploration table ξ and MDP $\Omega = \langle S, A, T, R, \gamma \rangle$. There exists inputs m and ϵ_1, satisfying $m = O\left(\frac{1}{\epsilon_1(1-\gamma)^2} \ln\left(\frac{\sum_{s \in S} K_s}{\epsilon_1 \delta(1-\gamma)}\right)\right)$ and $\frac{1}{\epsilon_1} = O\left(\frac{1}{\epsilon(1-\gamma)}\right)$, such that if Delayed Q-learning is executed on Ω_ξ, then the following holds. Let $\mathcal{A}_t$ denote Delayed Q-learning's policy at time t and s_t denote the state at time t. With probability at least $1 - \delta$, $V_\Omega^{\mathcal{A}_t}(s_t) \geq V_\Omega^*(s_t) - \epsilon$ is true for all but*

$$O\left(\frac{\sum_{s \in S} K_s}{\epsilon^4(1-\gamma)^8} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)} \ln \frac{\sum_{s \in S} K_s}{\delta \epsilon(1-\gamma)}\right)$$

timesteps t, where K_s is the number of valid actions for state $s \in S$ specified by the exploration table ξ.

The primary importance of the Delayed Q-learning algorithm is that it is a model-free PAC-MDP algorithm and its sample complexity bound has a smaller dependence on the number of state-action pairs than the R-MAX algorithm. By model-free it is meant that the algorithm uses memory that depends linearly on the number of state-action pairs and has a per-timestep computational complexity that depends only on the number of actions at the current state. R-MAX, on the other hand, uses memory that is polynomial in the number of state-action pairs and has a per-timestep

computational complexity that is also polynomial in the number of state-action pairs. If we ignore logarithmic factors the bound for R-MAX (Theorem 5.7) is

$$\tilde{O}\left(\frac{\psi \sum_{s \in S} K_s}{\epsilon^3 (1-\gamma)^6}\right)$$

while the bound for Delayed Q-learning is

$$\tilde{O}\left(\frac{\sum_{s \in S} K_s}{\epsilon^4 (1-\gamma)^8}\right)$$

where $\tilde{O}$ indicates that we have suppressed logarithmic factors. Although R-MAX has better dependence on $\frac{1}{\epsilon}$ and $\frac{1}{(1-\gamma)}$ it additionally depends on ψ, the maximum out-degree over all state-action pairs, whereas Delayed Q-learning avoids this dependence. In practice, however, it is often observed that R-MAX outperforms Delayed Q-learning in terms of sample efficiency. This is probably due to the fact that R-MAX makes better use of samples by performing a computationally expensive planning step and for most interesting RL problems the maximum out-degree ψ is much smaller than the number of states N. In fact, $\psi \leq \ln(N)$ is common. Nevertheless, Delayed Q-learning may be more applicable to real-world problems because of its small per-timestep computational complexity.

The main difference between our analysis of Delayed Q-learning and the analysis of Delayed Q-learning by Strehl et al. [16] is that the exploration table explicitly excludes a number of state-action pairs. Thus, in our analysis the learning complexity depends on $\sum_{s \in S} K_s$ rather than NK. However, if the exploration table does not eliminate any state-action pairs, we recover the same bound derived by Strehl et al. [16].

The analysis of Delayed Q-learning mostly consists of finding appropriate values

for arguments $m > 0$ and $\epsilon_1 > 0$. Delayed Q-learning is called "delayed" because it updates action-value estimates in a series of batches of m samples from a state-action pair (s, a) before attempting to update (s, a)'s action-value estimate.

Definition 5.11. *A* **batch** *of m samples*

$$AU(s,a) = \frac{1}{m}\sum_{i=1}^{m}\left(R_{k_i}(s,a) + \gamma \max_{a' \in A_{s'}} \hat{Q}_{k_i}(s',a')\right) \tag{5.4}$$

occurring at timesteps $k_1 < k_2 < \ldots < k_m$ for a state-action pair (s, a) consists of a sequence of m visits to (s, a) where the first visit occurs at a timestep k_1 corresponding to either the first timestep that (s, a) is visited by the algorithm or during the most recent prior visit to (s, a) at timestep $k' < k_1$, $LEARN_{k'}(s, a) = true$ and $l_{k'}(s, a) = m$ was true. A batch of samples is said to be completed on the first timestep $t > k_1$ such that $LEARN_t(s, a) = true$ and $l_t(s, a) = m$.

Delayed Q-learning has three notions of update:

1. Attempted Updates : An attempted update occurs when a batch of m samples has just been completed.

2. Update (or Successful Updates) : A successful update to a state-action pair (s, a) occurs when a completed batch of m samples at timestep t causes a change to the action-value estimates so that $\hat{Q}_t(s, a) \neq \hat{Q}_{t+1}(s, a)$.

3. Unsuccessful Updates : An unsuccessful update occurs when a batch of m samples completes but no change occurs to the action-values.

An attempted update to a state-action pair (s, a) at timestep t is successful if

$$\hat{Q}_t(s,a) - AU_t(s,a) \geq 2\epsilon_1 \tag{5.5}$$

the completed batch of samples is significantly lower ($< 2\epsilon_1$) than the current action-value estimate, and a successful update assigns

$$\hat{Q}_{t+1}(s,a) = AU_t(s,a) + \epsilon_1 \tag{5.6}$$

the completed batch of samples plus a small constant to the new action-value estimate. These rules guarantee that whenever a successful update occurs, the action-value estimates decrease by at least ϵ_1.

First recall that we have assumed that the immediate rewards are bound to the interval $[0,1]$. The Delayed Q-learning algorithm works by initializing its action-values to $\frac{1}{1-\gamma}$ (the maximum possible action-value) and decreasing these estimates in a series of updates. Unlike R-MAX, which only uses a single batch of m samples at each state-action pair, Delayed Q-learning may collect many batches of m samples from each state-action pair. To know when to stop updating a state-action pair, Delayed Q-learning maintains a boolean value for each state-action pair. These boolean values are called the *LEARN* flags. Due to the fact that every successful update decreases an action-value by at least ϵ_1, a particular state-action pair $(s,a) \in S \times A$ cannot be updated more than $\frac{1}{\epsilon_1(1-\gamma)}$ times. Because there are at most $\sum_{s \in S} K_s$ state-action pairs that can be explored, then there can only be a total of $\frac{\sum_{s \in S} K_s}{\epsilon_1(1-\gamma)}$ successful updates in an execution of the Delayed Q-learning algorithm.

Lemma 5.12. *No more than $\sum_{s \in S} K_s \left(1 + \frac{\sum_{s \in S} K_s}{\epsilon_1(1-\gamma)}\right)$ attempted updates can occurs during an execution of Delayed Q-learning on Ω_ξ.*

Proof. When Delayed Q-learning first starts its execution the *LEARN* flags are true for every state-action pair. Therefore, an attempted update can occur at least once at every state-action pair. After an attempted update occurs at a state-action pair (s,a) at timestep t, another attempted update at (s,a) can only occur if either the

attempted update at timestep t was successful or a successful update occurred at another state-action pair after timestep t. Therefore there are at most $(1 + \frac{\sum_{s \in S} K_s}{\epsilon_1(1-\gamma)}$ attempted updates at a state-action pair, and because there are only $\sum_{s \in S} K_s$ state-action pairs that can be explored in Ω_ξ, the total number of attempted updates cannot exceed $\sum_{s \in S} K_s \left(1 + \frac{\sum_{s \in S} K_s}{\epsilon_1(1-\gamma)}\right)$. □

Lemma 5.12 is very similar to [6, Lemma 19] and is used to determine a value for m. However, our lemma depends on $\sum_{s \in S} K_s$ rather than NK, so when we choose a value for m, our value may be smaller than NK if the exploration table is sparse.

The set

$$\kappa_t = \left\{ (s,a) \in D \mid \hat{Q}_t(s,a) - \left(R(s,a) - \gamma \sum_{s' \in S} T(s'|s,a) \max_{a' \in A_{s'}} \hat{Q}_t(s',a') \right) \leq 3\epsilon_1 \right\} \tag{5.7}$$

where $D = \{(s,a) \in S \times A \mid \xi(s,a) = 1\}$ and t is the current timestep, consists of the set of state-action pairs with small Bellman residual. Because the state-action pairs in κ_t have low Bellman error, Szita and Szepesvári [29] has called Eq. (5.7) the "nice" set. The set κ_t is somewhat analogous to the known-state MDP used in the analysis of R-MAX. However, unlike R-MAX, the algorithm cannot determine which state-action pairs are actually in this set. It is used strictly for the purposes of analysis. Notice that we have defined κ_t only over the set D because we do not care about the accuracy of state-action pairs that are pruned by the exploration table.

Having low Bellman error does not mean that the estimate for a state-action pair is accurate in an absolute sense. Instead it means that if an attempted update occurs at a state-action pair in κ_t, then (if m is large enough) it is unlikely that a successful update will occur. On the other hand, if an attempted update occurs at a state-action pair that is not in κ_t, then (if m is large enough) it is very likely that

an attempted update will be successful. Furthermore, if all of the state-action pairs (or at least all of the likely visited state-action pairs) have low Bellman error, then we can derive bounds on the absolute accuracy of the action-values.

Let $\mathcal{X}$ denote the event that when Delayed Q-learning is executed on Ω_ξ, then every time k_1 when a new batch of samples for some state-action pair (s, a) begins, if $(s, a) \notin \kappa_{k_1}$ and the batch is completed at timestep k_m, then a successful update to (s, a) will occur at timestep k_m.

Lemma 5.13. *If Delayed Q-learning is executed on an MDP Ω_ξ with parameters m and ϵ_1 where m satisfies*

$$m \geq \frac{1}{2\epsilon_1^2(1-\gamma)^2} \ln\left(\frac{\sum_{s\in S} K_s}{\delta}\left(1 + \frac{\sum_{s\in S} K_s}{\epsilon_1(1-\gamma)}\right)\right)$$

then

1. *event $\mathcal{X}$ will occur, and*

2. *$\hat{Q}_t(s, a) \geq Q^*_{\Omega_\xi}(s, a)$ for all timesteps t,*

with probability at least $1 - 2\delta$.

Proof. First we prove claim 1 and then prove claim 2.

Suppose that Delayed Q-learning has just finished collecting a batch $AU(s, a)$ of m samples for some state-action pair (s, a) and is about to perform an attempted update. Due to the Markov property we know that the rewards and next states in the given batch were sampled independently. Therefore the probability of observing a sequence of m rewards and next states in the MDP is less than or equal to the probability of observing those rewards and next states from a m sequence of calls to a generative model at (s, a). The reason that the probability may be less is because

when exploring the MDP the algorithm may not complete the batch because (s,a) may not be experienced $m-1$ more times after k_1.

For convenience let $\hat{V}(s') = \max_{a' \in A_\xi(s)} \hat{Q}(s',a')$. By the Hoeffding inequality with $0 \leq R_{k_i}(s_{k_i}, a_{k_i}) + \gamma \hat{V}_{k_i}(s_{k_{i+1}}) \leq \frac{1}{1-\gamma}$ for each $i = 1, 2, \ldots, m$ and our choice of m we have

$$AU(s,a) - E\left[AU(s,a)\right] < \epsilon_1$$

with probability at least $1 - \delta / \left(\sum_{s \in S} K_s \left(1 + \frac{\sum_{s \in S} K_s}{\epsilon_1 (1-\gamma)}\right)\right)$.

Now if the accuracy holds, $(s,a) \notin \kappa_{k_1}$, and an attempted update occurs at timestep k_m, then

$$\hat{Q}_{k_m}(s,a) - AU(s,a) > \hat{Q}_{k_m}(s,a) - E\left[AU(s,a)\right] - \epsilon_1 > 3\epsilon_1 + \epsilon_1 > 2\epsilon_1$$

which implies that the attempted update will be successful by Eq. (5.5). Where the second step is due to the definition of the "nice" set and the fact that $\hat{V}_{k_1}(s') \geq \hat{V}_{k_i}(s')$ for all $s' \in S$ and $i = 1, 2, \ldots, m$. To prove claim 1, we simply take the union bound over all the number of attempted updates (see Lemma 5.12) so that the event $\mathcal{X}$ occurs with probability at least $1 - \delta$.

Now to prove the second claim, notice that for a batch of m samples

$$\begin{aligned}
Q^*_{\Omega_\xi}(s,a) - AU(s,a) &= Q^*_{\Omega_\xi}(s,a) - \tfrac{1}{m}\left(R_{k_i}(s,a) + \gamma \hat{V}_{k_i}(s')\right) \\
&\leq Q^*_{\Omega_\xi}(s,a) - \tfrac{1}{m}\left(R_{k_i}(s,a) + \gamma V^*_{\Omega_\xi}(s')\right) \\
&< Q^*_{\Omega_\xi}(s,a) - E\left[\tfrac{1}{m}\left(R_{k_i}(s,a) + \gamma V^*_{\Omega_\xi}(s')\right)\right] + \epsilon_1 \\
&= \epsilon_1
\end{aligned}$$

by our choice of m according to the Hoeffding inequality, with probability at least $1 - \delta / \left(\sum_{s \in S} K_s \left(1 + \frac{\sum_{s \in S} K_s}{\epsilon_1 (1-\gamma)}\right)\right)$. By the union bound this inequality holds over all

attempted updates with probability at least $1 - \delta/3$. Assuming that the previous inequality holds, we can prove the second claim by induction. First notice that $\frac{1}{1-\gamma} = \hat{Q}(s,a) \geq Q^*_{\Omega(s,a)_\xi}$ for all $(s,a) \in D$ where $D = \{(s,a) \in S \times A \mid \xi(s,a) = 1\}$. Now if claim 2 holds up until timestep t, $\hat{Q}_t(s,a) \geq Q_{\Omega_\xi}(s,a)$. If no update or an unsuccessful update occur, then the action-values will not change. However, if a successful update occurs, then by Eq. (5.6),

$$
\begin{aligned}
\hat{Q}_{t+1} &= AU(s,a) + \epsilon_1 \\
&\geq \tfrac{1}{m} \textstyle\sum_{i=1}^{m} \Big(R_{k_i}(s,a) + \gamma V^*_{\Omega_\xi}(s') \Big) + \epsilon_1 \\
&= Q^*_{\Omega_\xi}(s,a)
\end{aligned}
$$

which maintains the optimism of the estimated action-values at state-action pair (s,a). Thus by the principle of mathematical induction, claim 2 holds with probability at least $1 - \delta$. Since both claims hold with probability at least $1 - \delta$ we can apply the union bound to see that they will jointly hold with probability at least $1 - 2\delta$. □

The key importance of Lemma 5.13 is that it specifies a value of m that causes Delayed Q-learning to have successful updates when it completes a batch of samples for a state-action pair outside of the "nice" set, and it guarantees that the action-value estimates will be optimistic with high probability. These properties will be important for bounding the learning complexity ζ. We also need to determine conditions when a state-action pair enters the nice set.

Lemma 5.14. *([16, Lemma 3]) If event $\mathcal{X}$ occurs and an unsuccessful update at (s,a) occurs a timestep t and $LEARN_{t+1}(s,a) = false$, then $(s,a) \in \kappa_{t+1}$.*

Now we are ready to bound the learning complexity ζ.

Lemma 5.15. *If event $\mathcal{X}$ occurs and Delayed Q-learning is executed on Ω_ξ, then the learning complexity of the algorithm is at most $\zeta = 2m\left(\frac{\sum_{s\in S} K_s}{\epsilon_1(1-\gamma)}\right)$.*

Proof. In Ω_ξ there are a total of $\sum_{s\in S} K_s$ state-action pairs. These state-action pairs can be updated at most $\frac{1}{\epsilon_1(1-\gamma)}$ times, so the total number of successful updates (changes to the action-value estimates) is at most $\frac{\sum_{s\in S} K_s}{\epsilon_1(1-\gamma)}$.

Now we will show that if at a timestep t if $(s,a) \notin \kappa_t$ is experienced, then after at most $2m$ more experiences of (s,a) that state-action pair will belong to the "nice" set. There are two different cases where $(s,a) \notin \kappa_t$ might be experienced.

Case 1: $(s,a) \notin \kappa_t$ and $LEARN_t(s,a) = false$. This can happen if at some timestep $t' < t$, $(s,a) \in \kappa_{t'}$ and an unsuccessful update occurred at timestep t' (implying that $(s,a) \in \kappa_{t'+1}$ by Lemma 5.14) followed by a successful update at some other state-action pair between timesteps t' and t. In this case, the $LEARN$ flag for (s,a) will be set to *true* at timestep t by the rules of Delayed Q-learning. Since event $\mathcal{X}$ occurs, the next attempted update will be successful.

Case 2: $(s,a) \notin \kappa_t$ and $LEARN_t(s,a) = true$. An attempted update will occur after at most $m-1$ more experiences of (s,a). If (s,a) was not in the "nice" set at the beginning of collecting this batch of samples, then the next attempted update will be successful because we have assumed that event $\mathcal{X}$ holds. However, if (s,a) was in the "nice" set at the beginning of collecting this batch of samples, then a successful update must have occurred at some other state-action pair. In this case, $LEARN_{t+1}(s,a) = true$ and (s,a) will not be in the "nice" set when the batch completes. Again, by event $\mathcal{X}$, the next batch of samples from (s,a) will result in a successful update.

In both cases, at most $2m$ experiences of (s,a) will occur before a successful update occurs. Since the there are $\sum_{s\in S} K_s$ state-action pairs and each state-action

pair can only be updated $\frac{1}{\epsilon_1(1-\gamma)}$ times, the total number of experiences of some state-action pair not in the "nice" set is at most $2m\left(\frac{\sum_{s\in S} K_s}{\epsilon_1(1-\gamma)}\right)$. □

The significance of this learning complexity bound is that it is approximately linear in the number of state-action pairs in Ω_ξ because our chosen value for m only depends on the number of state-action pairs within log factors. Now we are ready to prove the sample complexity bound for Delayed Q-learning executed on Ω_ξ. This proof is essentially the same as that of Strehl et al. [6, Theorem 16] but it depends on the modified versions of the lemmas proved above.

Proof. (of Theorem 5.10) We proceed by applying Theorem 5.2 with $\epsilon_1 = \epsilon(1-\gamma)/3$ and m selected according to Lemma 5.13. The "nice" set does not change unless a change has occurred to some action-value. Now we assume that claims 1 and 2 of Lemma 5.13 hold, which occurs with probability at least $1 - 2\delta$. Claim 2 directly satisfies Condition 1 of Theorem 5.2 that the action-value estimates are optimistic at all state-action pairs and all timesteps t. Next, we will show that $\hat{V}_t(s) - V_{\Omega'}^{\pi_t}(s) \leq \frac{3\epsilon_1}{1-\gamma} = \epsilon$ where π_t is the greedy policy of the algorithm at timestep t and Ω' denotes Ω_{ξ,κ_t} the induced MDP with respect to Ω_ξ and κ_t, which satisfies Condition 2 of Theorem 5.2. Since $\hat{Q}_t$ is identical to $Q^*_{\Omega'}$ for all $(s,a) \notin \kappa_t$, the only error is introduced by state-action pairs that are within κ_t and by the definition of the "nice" set they only differ by at most $3\epsilon_1$. Thus the total difference between the two policies is at most $\frac{3\epsilon_1}{1-\gamma}$.

Condition 3 is satisfied by $\zeta = O\left(2m\frac{\sum_{s\in S} K_s}{\epsilon_1(1-\gamma)}\right)$. By plugging in our chosen values for m and ϵ_1, we obtain the desired sample complexity bound. To obtain an ϵ-optimal policy rather than a 4ϵ-optimal policy we can simply set $\epsilon \leftarrow \frac{\epsilon}{4}$ and similarly to obtain a probability of failure greater than δ we can substitute $\delta \leftarrow \frac{\delta}{4}$, both of which alter only constant factors in the bound. □

5.3 Implicit Action Pruning

We have seen that providing an exploration table for an MDP can decrease the sample complexity of exploration of both R-MAX and Delayed Q-learning. Exploration tables explicitly prune certain state-action pairs from the MDP. Alternatively it is possible to initialize the action-values in ways that implicitly eliminates state-action pairs from being explored. In this section, we investigate action-value initializations that eliminate state-action pairs from consideration during learning.

Strehl et al. [6] introduced the concept of an admissible heuristic

$$Q^*_\Omega(s, a) \leq U(s, a) \leq V_{\text{MAX}} \tag{5.8}$$

where U is an admissible heuristic for all $(s, a) \in S \times A$ and demonstrated that if either the Delayed Q-learning algorithm or R-MAX are initialized with an admissible heuristic the sample complexity of the algorithms can be greatly reduced. This idea is related to reward shaping [50], where additional rewards are given to the RL algorithm to decrease learning time.

An admissible heuristic U is a valuable tool for providing prior knowledge to an RL algorithm that uses the OFU exploration strategy. The most interesting thing about admissible heuristics is that they do not need to specify any exact information about the optimal action-values of the task. A range of values can still result in a valid admissible heuristic. This provides some leniency for guessing or, in our case, transferring action-values (see Chapter 6).

Admissible heuristics enable algorithms such as R-MAX and Delayed Q-learning to converge to a near-optimal policy with respect to their current state. However, it is possible to imagine many action-value initializations that are not admissible heuristics and yet somehow surreptitiously converge to an optimal policy. In this

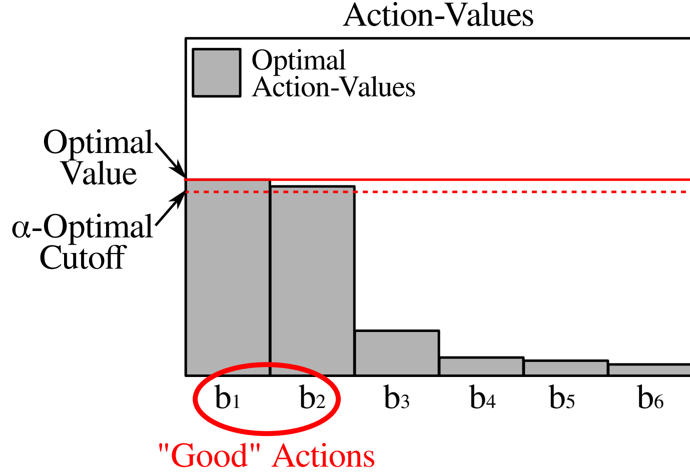


Figure 5.2: The "good" actions at a state are the actions with optimal action-values that are α-close to the optimal action-value at that state.

chapter, we introduce a much weaker condition on initial action-values where the learned policy is still near-optimal. This weaker condition will be useful for analyzing action-value transfer in the next chapter.

For one thing, it is not always necessary to select optimal actions. Instead it is often good enough to select α-optimal or "good" actions (Figure 5.2).

Definition 5.16. *Let $\alpha \geq 0$ and $\Omega = \langle S, A, T, R, \gamma \rangle$ be an MDP. An action $a \in A$ that satisfies*

$$Q^*_\Omega(s, a) \geq V^*_\Omega(s) - \alpha \tag{5.9}$$

*is called an α-**good action** (or simply **good action**) with respect to the state $s \in S$, and*

$$G^\alpha_\Omega(s) = \{a \in A \mid Q^*_\Omega(s, a) \geq V^*_\Omega(s) - \alpha\} \tag{5.10}$$

denotes the set of α-good actions at state $s \in S$.

So good actions at a state $s \in S$ are the actions that have action-values very close to $V^*_\Omega(s)$ if the policy is optimal at every other state. These good actions are very

important because if the policy always selects good actions, then that policy is at least $\left(\frac{\alpha}{1-\gamma}\right)$-optimal or better. Keep in mind, however, that a policy can occasionally select good actions and not be acting optimally because the optimality of a policy is evaluated over a sequence of actions. Nevertheless, the next lemma shows that good actions have an important relationship with near-optimal policies.

Lemma 5.17. *Let $\alpha \geq 0$, $\Omega = \langle S, A, T, R, \gamma \rangle$ be an MDP, and $\pi : S \rightarrow A$ be a policy satisfying $\pi(s) \in G^{\alpha}_{\Omega}(s)$, then π is an $\left(\frac{\alpha}{1-\gamma}\right)$-optimal policy for Ω.*

Proof. At any state $s \in S$, the policy $\pi(s) = \tilde{a}$ such that $Q^*_{\Omega}(s, \tilde{a}) \geq V^*_{\Omega}(s) - \alpha$. So

$$\begin{aligned} V^*_{\Omega}(s) - V^{\pi}_{\Omega}(s) &= V^*_{\Omega}(s) - Q^{\pi}_{\Omega}(s, \pi(s)) \\ &= V^*_{\Omega}(s) - Q^*_{\Omega}(s, \pi(s)) + Q^*_{\Omega}(s, \pi(s)) - Q^{\pi}_{\Omega}(s, \pi(s)) \\ &\leq \alpha + Q^*_{\Omega}(s, \pi(s)) - Q^{\pi}_{\Omega}(s, \pi(s)) \\ &= \alpha + \gamma E_{s' \sim T(\cdot|s,\pi(s))} \left[V^*_{\Omega}(s') - V^{\pi}_{\Omega}(s')\right] \end{aligned}$$

where the proof follows by recursing on this inequality and the linearity of the expectation operator E. □

This lemma captures the intuitive notion that a policy that makes good decisions at every state is nearly optimal in the long run. Notice that Lemma 5.17 does not make any assumptions about the action-value function estimates kept by an algorithm. The only thing that is required is that the policy itself takes good actions at each state. So if an RL algorithm follows a greedy policy, it may act near-optimally even if its action-value estimates are inaccurate. Figure 5.3 provides an example where the action-values estimates may be far from the optimal action-values at a state, yet a greedy policy will selects the optimal action. Thus it is possible and useful to consider even weaker sufficient conditions than those used by the admissible heuristic.

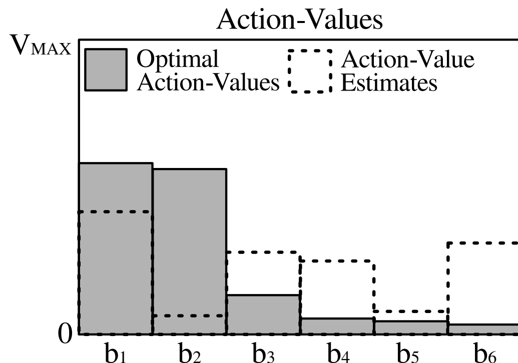


Figure 5.3: Example of poorly estimated action-values at a single state. Notice that a greedy policy would select the optimal action b_1 in this case despite the fact that the action-value estimates are extremely poor.

Definition 5.18. *A function $W : S \times A \to \mathbb{R}$ is an α-**weak admissible heuristic** (or just **weak admissible heuristic**) for MDP $\Omega = \langle S, A, T, R, \gamma \rangle$, if for each $s \in S$, there exists $\tilde{a} \in A$ such that*

$$V^*_\Omega(s) - \alpha \leq Q^*_\Omega(s, \tilde{a}) \leq W(s, \tilde{a}) \leq \frac{1}{1-\gamma}$$

where α is the smallest non-negative value satisfying this inequality.

In other words, a weak admissible heuristic W differs from the admissible heuristic U defined by Strehl et al. [6] in that the weak admissible heuristic only needs to be optimistic for a single good action $a \in G^\alpha_\Omega(s)$ at each state $s \in S$. For example, Figure 5.4 provides an example of a weak admissible heuristic where the optimal action b_1 is severely underestimated and the lowest valued action is severely overestimated. Despite the fact that these estimates are very far from the optimal action-values, a simple algorithm following the OFU admissible heuristic is still likely to converge on the good action b_2 (which is slightly over-estimated). Initially an OFU algorithm will select the action b_6, but after sampling b_6 several times the estimated action-value for b_6 will likely decrease dramatically. Once this happens, then b_2 will

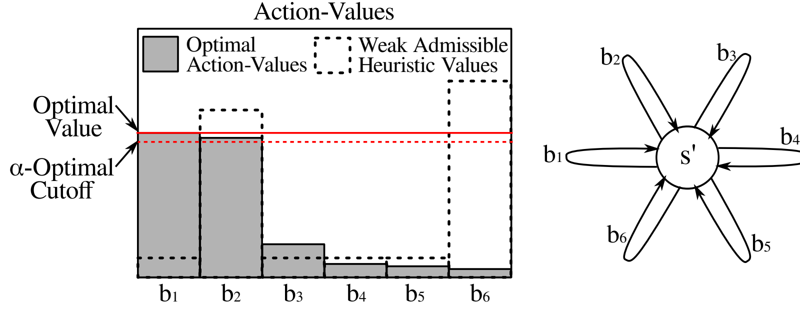


Figure 5.4: Weak admissible heuristic applied to a one state task with six actions. The weak admissible heuristic only needs to optimistically initialize the action-value for a single near-optimal action at each state.

have the highest estimated value. The estimated value of b_2 is unlikely to decrease very far because the optimal action-value for b_2 is not too far from its estimate. Thus this weak admissible heuristic causes convergence to a near-optimal policy after examining only one suboptimal policy. This is a much weaker assumption, and yet, we will demonstrate that various provably efficient RL algorithms that use the OFU exploration strategy will still converge to a near-optimal policy with respect to its current state.

The following theorem will help in our analysis of sample complexity with a weak admissible heuristic.

Theorem 5.19. *(State-action Pair Elimination) Let $\eta \geq 0$, W be an α-weak admissible heuristic for an MDP Ω, and $\mathcal{A}$ is a value-based RL algorithm with initial action-value estimates $\hat{Q}_0 = W$ such that for all timesteps $t \geq 1$,*

1. *$\mathcal{A}$ follows a greedy policy ($\mathcal{A}_t(s) = \arg\max_{a \in A} \hat{Q}_t(s,a)$),*

2. *Updates to $\hat{Q}_t(s,a)$ can only occur if (s,a) has been tried, and*

3. *if $W(s,a) \geq Q^*_\Omega(s,a)$, then $\hat{Q}_t(s,a) \geq Q^*_\Omega(s,a) - \eta$,*

then for all $(s,a) \in S \times A$ where $W(s,a) < V^*_\Omega(s) - (\alpha + \eta)$, $\mathcal{A}$ will never explore

(s,a) *($\mathcal{A}_t(s) \neq a$ at any timestep $t \geq 1$, where $\mathcal{A}_t$ denotes the policy of $\mathcal{A}$ at timestep t).*

Proof. Since $\mathcal{A}$ follows a greedy policy with respect to $\hat{Q}_t$, a state-action pair (s,a) will only be selected if $\hat{Q}_t(s,a) = \max_{a' \in A} \hat{Q}_t(s,a')$.

Suppose, without loss of generality, that $W(s,a) < V^*_\Omega(s) - (\alpha + \eta)$, then by the definition of W there exists $\tilde{a}$ such that $W(s,\tilde{a}) \geq Q^*_\Omega(s,\tilde{a}) \geq V^*_\Omega(s) - \alpha > W(s,a)$. Thus $a \neq \arg\max_{a' \in A} W(s,a') = \arg\max_{a' \in A} \hat{Q}_0(s,a')$.

By assumption 3 the action-value estimate for $\hat{Q}_t(s,\tilde{a}) \geq Q^*_\Omega(s,\tilde{a}) - \eta \geq (V^*_\Omega(s) - \alpha) - \eta = V^*_\Omega(s) - (\alpha + \eta) > W(s,a)$. Since (s,a) has not been tried yet no update to its action-value could have occurred (assumption 2). Therefore, $\hat{Q}_t(s,a) = W(s,a) < \hat{Q}_t(s,\tilde{a})$. Therefore, (s,a) will never be executed. □

Theorem 5.19 states that if a value-based RL algorithm with certain properties is initialized with a weak admissible heuristic, then the actions with very low estimates are never explored. In other words, a weak admissible heuristic implicitly eliminates certain state-action pairs. Supplying a weak admissible heuristic W is similar to supplying the exploration table

$$\xi(s,a) = \begin{cases} 1 & \text{if } W(s,a) \geq V^*_\Omega(s) - (\alpha + \eta) \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

for all $(s,a) \in S \times A$. However, we refer to the action pruning as implicit because the exploration table cannot be explicitly defined without knowing the optimal value function V^*_Ω, which is generally not known. However, we will use this relationship between weak admissible heuristics and exploration tables in our analysis of R-MAX and Delayed Q-learning.

5.3.1 R-MAX

The sample complexity of R-MAX is influenced by initializing its action-values with a weak admissible heuristic. The weak admissible heuristic causes R-MAX to never try certain state-action pairs (Theorem 5.19). However, this depends on the given weak admissible heuristic W. If $W(s,a) = \frac{1}{1-\gamma}$ at every state-action pair $(s,a) \in S \times A$, then the sample complexity is equivalent to the original R-MAX algorithm. However, if many state-action pairs can be eliminated from consideration, this can greatly improve the sample complexity of the algorithm.

Theorem 5.20. *Let $\epsilon > 0$, $\delta \in (0,1]$, $\alpha > 0$ and W be an α-weak admissible heuristic for the MDP $\Omega = \langle S, A, T, R, \gamma \rangle$. Let $\mathcal{A}$ be an instance of R-MAX with action-values $\hat{Q}_0$ initialized by W. There exists a values for ϵ_1 and m such that if $\mathcal{A}$ is executed on Ω, then $V_\Omega^{\mathcal{A}_t}(s) < V_\Omega^*(s) - (\epsilon + \frac{\alpha}{1-\gamma})$ on all but*

$$O\left(\frac{NK - X}{\epsilon^3(1-\gamma)^6}\left(\psi + \ln\frac{NK - X}{\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps t, with probability at least $1 - \delta$, where

$$X = |\{(s,a) \in S \times A \mid W(s,a) < V_\Omega^*(s) - (\epsilon/2 + \alpha)\}|$$

is the number of state-action pairs that are never explored and ψ is the maximum out-degree of any state-action pair.

Theorem 5.20 describes the sample complexity of R-MAX when a weak admissible heuristic is used to initialize the algorithm. The sample complexity depends on X, which is the number of state-action pairs that are implicitly pruned by initializing the action-values with W. If the given weak admissible heuristic is an admissible heuristic, then this bound reduces to the one given by Strehl et al. [6, Theorem

11] with $\alpha = 0$ (which is true for all admissible heuristics). However, our bound is more flexible because it allows us to understand the interaction between optimality loss and sample complexity even when the initial action-values do not optimistically valuate the best action in every state.

Proof. We will proceed by applying Theorem 5.19.

By its definition, the R-MAX algorithm always follows a greedy policy with respect to its action-value estimates $\hat{Q}_t$ (satisfying Condition 1 of Theorem 5.19) and never alters an action-value estimate unless the corresponding state-action pair has been visited (satisfying Condition 2 of Theorem 5.19). Thus in order to apply Theorem 5.19 we only need to ensure that R-MAX will maintain $\hat{Q}(s, a) \geq Q^*_\Omega(s, a) - \eta$ whenever $W(s, a) \geq Q^*_\Omega(s, a)$ for some η.

Suppose that Theorem 5.19 holds with $\eta = \epsilon_1 = \epsilon/2$. Then this is equivalent to running R-MAX on the exploration table defined by Eq. (5.11). By Theorem 5.7, we have that the sample complexity of exploration for the MDP Ω_ξ is bound by

$$O\left(\frac{NK - X}{\epsilon^3(1-\gamma)^6}\left(\psi + \ln\frac{NK - X}{\delta}\right)\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

where $X = |\{(s, a) \in S \times A \mid W(s, a) < V^*_\Omega(s) - (\epsilon/2 + \alpha)\}|$ with probability at least $1 - \delta$. Since $\xi(s, \tilde{a}) = 1$ for at least one action $\tilde{a} \in G^\alpha_\Omega(s)$ at every state $s \in S$, then the optimality loss between Ω and Ω_ξ is at most $\frac{\alpha}{1-\gamma}$, by Lemma 5.6. Thus the policy learned by R-MAX is at least $\left(\epsilon + \frac{\alpha}{1-\gamma}\right)$-optimal in Ω.

Now, we claim that $\eta = \epsilon_1 = \epsilon/2$. To prove Theorem 5.7, we selected m according to Lemma 5.9, which guarantees that $\left|\hat{Q}_t(s, a) - Q^*_{\Omega_{\xi,\kappa_t}}(s, a)\right| \leq \epsilon_1 = \epsilon/2$ for all timesteps t and $(s, a) \in D$ where $D = \{(s, a) \in S \times A \mid \xi(s, a) = 1\}$. Since $Q^*_{\Omega_{\xi,\kappa_t}}(s, a) \geq Q^*_{\Omega_\xi}(s, a)$ for all $(s, a) \in D$, then $\hat{Q}_t(s, a) \geq Q^*_{\Omega_\xi}(s, a) - \epsilon/2$. □

5.3.2 Delayed Q-learning

Now we will analyze the sample complexity of exploration for the Delayed Q-learning algorithm with action-values initialized by an α-weak admissible heuristic.

Theorem 5.21. *Let $\epsilon > 0$, $\delta \in (0, 1]$, $\alpha > 0$ and W is an α-weak admissible heuristic for the MDP $\Omega = \langle S, A, T, R, \gamma \rangle$. Let $\mathcal{A}_t$ denote the policy of an instance of Delayed Q-learning at timestep t with action-values $\hat{Q}_0$ initialized by W. There exists values for ϵ_1 and m such that if $\mathcal{A}$ is executed on Ω, then $V_\Omega^{\mathcal{A}_t}(s) < V_\Omega^*(s) - (\epsilon + \frac{\alpha}{1-\gamma})$ on all but*

$$O\left(\frac{\sum_{(s,a)\in D}\left[W(s,a) - (V_\Omega^*(s) - \alpha)\right]_+}{\epsilon^4(1-\gamma)^7} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{NK - X}{\delta\epsilon(1-\gamma)}\right)$$

timesteps t, with probability at least $1 - \delta$, where $[x]_+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$,

$$X = |\{(s,a) \in S \times A \mid W(s,a) < V_\Omega^*(s) - \alpha\}|$$

is the number of state-action pairs that are never explored, and

$$D = \{(s,a) \in S \times A \mid W(s,a) \geq V_\Omega^*(s) - \alpha\}$$

is set state-action pairs that may be explored.

This bound is somewhat different from the bound derived for R-MAX (Theorem 5.20) because the sample complexity depends on $\left[W(s,a) - (V_\Omega^*(s) - \alpha)\right]_+$. When W is an admissible heuristic, then the bound is essentially the same as the bound given by Strehl et al. [6, Theorem 16]. Again, our bound is more flexible though, because it allows us to understand the interaction between optimality loss and sample

complexity even when the initial action-values do not optimistically valuate the best action in every state.

Proof. We will proceed by applying Theorem 5.19.

By its definition, the Delayed Q-learning algorithm always follows a greedy policy with respect to its action-value estimates $\hat{Q}_t$ (satisfying Condition 1 of Theorem 5.19) and it never alters an action-value estimate unless the corresponding state-action pair has been visited (satisfying Condition 2 of Theorem 5.19). Thus in order to apply Theorem 5.19 we only need to ensure that Delayed Q-learning will maintain $\hat{Q}(s,a) \geq Q^*_\Omega(s,a) - \eta$, whenever $W(s,a) \geq Q^*_\Omega(s,a)$.

For now, we will assume that this holds with $\eta = 0$ (and later show that this is true with high probability given our choice of m). By Theorem 5.19 every state-action pair $(s,a) \in S \times A$ such that $W(s,a) < Q^*_\Omega(s,a) - \alpha$ is never explored by Delayed Q-learning. This is implicitly equivalent to executing Delayed Q-learning with the exploration table defined by Eq. (5.11). We select $m = O\left(\frac{1}{\epsilon_1^2(1-\gamma)^2} \ln\left(\frac{\sum_{s \in S} K_s}{\epsilon_1 \delta (1-\gamma)}\right)\right)$ according to Lemma 5.13 and $\epsilon_1 = \frac{\epsilon(1-\gamma)}{3}$ such that the sample complexity bound for Ω_ξ is

$$O\left(\frac{\sum_{s \in S} K_s}{\epsilon^4 (1-\gamma)^8} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)} \ln \frac{\sum_{s \in S} K_s}{\delta \epsilon (1-\gamma)}\right)$$

where $K_s = |\{a \in A \mid \xi(s,a) = 1\}| = |\{a \in A \mid W(s,a) \leq Q^*_\Omega(s,a) - \alpha\}|$, with probability at least $1 - \delta$. By the definition of the α-weak admissible heuristic, at every state $s \in S$ there exists an action $\tilde{a} \in G^\alpha_\Omega(s)$ such that $\xi(s,a) = 1$. Therefore the optimality loss between Ω and Ω_ξ is at most $\frac{\alpha}{1-\gamma}$, by Lemma 5.6. Thus, the policy learned by Delayed Q-learning is at least $\left(\epsilon + \frac{\alpha}{1-\gamma}\right)$-optimal or better in Ω.

Furthermore, the notice that at any state $s \in S$, any action whose value falls below $V^*_\Omega(s) - \alpha$ will never be explored again because there exists an action $\tilde{a} \in G^\alpha_\Omega(s)$ such that $W(s,\tilde{a}) \geq V^*_\Omega(s) - \alpha$. Thus the total number of times that a state-action pair

outside of the "nice" set can be visited is $2m\frac{\left[W(s,a)-(V_\Omega^*(s)-\alpha)\right]_+}{\epsilon_1}$ (rather than $2m\frac{\sum_{s\in S}K_s}{\epsilon_1(1-\gamma)}$ specified by Lemma 5.15). So Delayed Q-learning will follow an $\left(\epsilon+\frac{\alpha}{1-\gamma}\right)$-optimal policy in Ω on all but

$$O\left(\frac{\sum_{(s,a)\in D}\left[W(s,a)-(V_\Omega^*(s)-\alpha)\right]_+}{\epsilon^4(1-\gamma)^7}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\ln\frac{\sum_{s\in S}K_s}{\delta\epsilon(1-\gamma)}\right)$$

timesteps t, with probability at least $1-\delta$.

Finally, by Lemma 5.13, $\hat{Q}_t(s,a) \geq Q^*_{\Omega_{\xi,\kappa_t}}(s,a) \geq Q^*_\Omega(s,a)$ for all timesteps t and state-action pairs $(s,a)\in D$, satisfying Condition 3 of Theorem 5.19 with $\eta=0$. □

5.4 Summary

In this section, we have derived upper bounds on the sample complexity of exploration for R-MAX and Delayed Q-learning with two different prior knowledge structures that support action pruning. The first structure, exploration tables, explicitly specify which state-action pairs can be explored by an algorithm (as well as which state-action pairs cannot be explored). The second structure, α-weak admissible heuristics, implicitly specify state-action pairs that will never be explored. We have assumed that either an exploration table or an α-weak admissible heuristic is given and demonstrated how this structure impacts sample complexity of exploration with either R-MAX or Delayed Q-learning. However, we have not explained how an autonomous learning system might acquire either an exploration table or α-weak admissible heuristic. In the following sections, we will explore potential solutions to these shortcomings under transfer learning and multitask learning scenarios.

6. ANALYSIS OF ACTION-VALUE TRANSFER

In this section, we consider the transfer of action-values from a single source task MDP $\Omega_{\text{SRC}} = \langle S_{\text{SRC}}, A_{\text{SRC}}, T_{\text{SRC}}, R_{\text{SRC}}, \gamma \rangle$ to a single target task MDP $\Omega_{\text{TRG}} = \langle S_{\text{TRG}}, A_{\text{TRG}}, T_{\text{TRG}}, R_{\text{TRG}}, \gamma \rangle$ given an intertask mapping $h : S_{\text{TRG}} \times A_{\text{TRG}} \to S_{\text{SRC}} \times A_{\text{SRC}}$. The intertask mapping provides a relationship from state-action pairs in the target task to state-action pairs in the source task. Intuitively, action-value transfer should work, when the target task state-action pairs are mapped to source task state-action pairs with similar optimal action-values. Surprisingly, our analysis reveals that action-value transfer often works well, even (in some cases) where the optimal action-values mapped from the target to the source task are quite different.

Previous research on transferring action-values has primarily focused on directly initializing target task action-values with source task action-values [45, 58, 46, 7, 8]. While the proof of concept has been established, it is not clear when direct transfer of action-values reduces the number of timesteps that the base RL algorithm will act according to a suboptimal policy. In this section, we generalize the notion of sample complexity to a TL setting and analyze how transferred action-values influence sample complexity.

6.1 Action-value Transfer Framework

Because of the useful properties of admissible heuristics for the analysis of sample complexity, we introduced a simple generalization of the direct value transfer algorithm, called Biased Value Transfer (BVT).

BVT (Algorithm 8) adds a small value

$$\beta \left(\frac{R_{\text{MAX}} - R_{\text{MIN}}}{1 - \gamma} \right)$$

Algorithm 8 Biased Value Transfer (BVT)

Require: $\mathcal{A}, h, Q_{\text{SRC}}, \beta$

1: **for** $(s,a) \in S_{\text{TRG}} \times A_{\text{TRG}}$ **do**
2:     **if** (s,a) is in domain of h **then**
3:         $Q_{\text{TRG}}(s,a) \leftarrow \min\left(Q_{\text{SRC}}(h(s,a)) + \beta\left(\frac{R_{\text{MAX}}-R_{\text{MIN}}}{1-\gamma}\right), \frac{R_{\text{MAX}}}{1-\gamma}\right)$
4:     **else**
5:         $Q_{\text{TRG}}(s,a) \leftarrow \frac{R_{\text{MAX}}}{1-\gamma}$
6:     **end if**
7: **end for**
8: Initialize($\mathcal{A}, Q_{\text{TRG}}$)

to each source task action-value as it is assigned to a target task action-value, where $\beta \in [0,1]$ is a user defined constant. This small value biases the transferred action-values in an attempt to ensure that the initial action-values for the target task are optimistic. If the initial action-values are optimistic, then we can often guarantee that provably efficient reinforcement learning algorithms that use these action-values will converge to near-optimal policies with a small number of suboptimal actions.

The main problem comes from selecting β. If we set $\beta = 0$, then we recover the direct value transfer algorithm. In this case, if any of the target task action-values are pessimistically initialized the algorithm may not learn to act near-optimally. However, if β is too large, then there may not be any benefit of transfer. The designer must specify β so that it initializes the target task action-values optimistically, but choose β as small as possible to gain the most benefit from transfer.

6.2 Analysis of Action-value Transfer

In this section, we will analyze the sample complexity of the BVT action-value transfer approach, with value-based, PAC-MDP algorithms serving as the RL algorithms in the target task. First we will analyze the sample complexity of learning action-values from an MDP. This problem is equivalent to the one faced by the source

task algorithm. Next we analyze the sample complexity and potential loss caused by transferred action-values in the target task.

6.2.1 Analysis of Source Task Sample Complexity

We first turn our attention to the sample complexity of learning source task action-values. We report negative and positive results. The negative results show that learning action-values through exploration is, in general, infeasible. The positive results suggest that if we have a generative model of the source task, we can learn arbitrarily accurate action-values with only polynomially many samples in the number of states, actions, and other relevant parameters.

6.2.1.1 Negative Results

For the explore sampling model, we cannot hope to learn accurate action-values for all state-action pairs in general MDPs because many MDPs have a nonzero probability of transitioning to a cluster of states where some other states are not reachable.

An alternative to the explore sampling model is to assume the reset sampling model. In this scenario it is possible to visit all state-action pairs. Fiechter [59] considered this sampling model and developed a polynomial time algorithm that learns an ϵ-optimal policy with respect to the reset state s_0, but as we mentioned before we need ϵ-optimal action-values for all state-action pairs.

A suitable number of timesteps τ must be chosen for $\mathcal{A}_{\text{SRC}}$ to be run on the source task Ω_{SRC}. Unfortunately the worst case sample complexity for obtaining ϵ_{SRC}-accurate action-values is exponential in the number of states.

Theorem 6.1. *For all $N > 1$, there exists an MDP M with N states, such that the sample complexity of learning ϵ_{SRC}-accurate action-values for all state-action pairs, given access to a reset sampling model for M, is exponential in the number of states.*

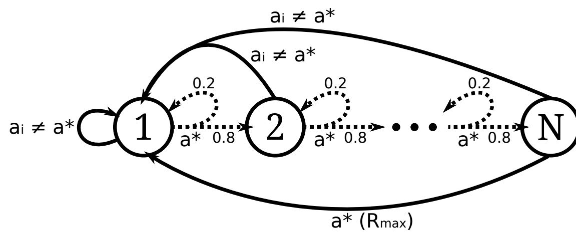


Figure 6.1: A reset MDP is a chain of N states. Action a^* transitions the agent to the next state (or state 1 if the current state is N) with probability 0.8 and stays in the current state with probability 0.2, while all other actions reset the agent to state 1. The agent only receives a reward for taking action a^* in state N.

Proof. Since no prior information is assumed to be given about the state-action pairs of the MDP M, each state-action pair must be visited at least once to learn an ϵ_{SRC}-accurate action-value for that state-action pair.

Consider the family of combination lock MDPs (e.g. Figure 6.1). If $0 < \eta < 1$ is the probability of transitioning to the next state when action a^* is taken, reaching the final state even once can require $\Omega\left(\left(\frac{1}{\eta}\right)^N\right)$ timesteps, which is exponential w.r.t. N, where Ω represents the notation for a lower bound (and not an MDP). □

In other words, Theorem 6.1 demonstrates that in some MDPs, an RL algorithm may require an exponential number of timesteps in the number of states to learn accurate action-values. This seems like a serious problem for action-value transfer, because without accurate action-values, what good will transfer be? However, later we will see that requiring all source task action-values to be accurate is too strict in many practical cases.

6.2.1.2 Positive Results

If we give up on exploring the source task to learn accurate action-values, and instead, are given a generative model of the source task, then we can learn accurate action-values with polynomially many samples with respect to the number of states and actions.

Algorithm 9 Phased Q-Learning [34]

Require: S, A, γ, m, and H

1: Set $\hat{V}_0$ and $\hat{Q}_0$ to 0 for all $s \in S$ and $(s,a) \in S \times A$, respectively
2: **for** $t = 1, 2, \ldots, H$ **do**
3: Sample every state-action pair (s,a), m times, denoting the ith reward and next state samples by $r_{t,i}(s,a)$ and $s'_{t,i}(s,a)$, respectively
4: $\hat{Q}_t(s,a) \leftarrow \frac{1}{m}\left(\sum_{i=1}^{m} r_{t,i}(s,a) + \gamma \hat{V}_{t-1}(s'_{t,i}(s,a))\right)$
5: $\hat{V}_t(s) \leftarrow \max_{a \in A} \hat{Q}(s,a)$
6: **end for**

Observation 6.2. *For any $\epsilon \geq 0$ and $\delta \in (0,1]$, and an MDP $\Omega = \langle S, A, T, R, \gamma \rangle$ with access to Ω via a generative model, there exists an algorithm that can estimate optimistic, ϵ-accurate action-values $\hat{Q}$, with probability at least $1-\delta$, and use at most*

$$O\left(\frac{NK}{\epsilon^2(1-\gamma)^3}\ln^3\left(\frac{4}{\epsilon}\right)\ln\left(\frac{NK\ln\left(\frac{4}{\epsilon}\right)}{\delta(1-\gamma)}\right)\right)$$

calls to the generative model.

Proof. By choosing $m = \frac{32H^2}{\epsilon^2}\ln\left(\frac{2NKH}{\delta}\right)$ we have that

$$\left|E_{s' \sim \Pr(\cdot|s,a)}\left[\hat{V}_t(s')\right] - \frac{1}{m}\sum_{i=1}^{m}\hat{V}_t(s'_{t,i}(s,a))\right| \leq \frac{\epsilon}{8H} \tag{6.1}$$

and

$$\left|E\left[r(s,a)\right] - \frac{1}{m}\sum_{i=1}^{m} r_{t,i}(s,a)\right| \leq \frac{\epsilon}{8H}$$

is true for all states $s \in S$, actions $a \in A$, and times $t \in \{1, 2, \ldots, H\}$, with probability at least $1-\delta$, through application of the Chernoff-Hoeffding inequality and union bound.

It follows that for all $(s,a) \in S \times A$

$$\begin{aligned}
\left|Q_t^*(s,a) - \hat{Q}_t(s,a)\right| &\leq \left|E_{s' \sim \Pr(\cdot|s,a)}\left[V_{t-1}^*(s')\right] - \tfrac{1}{m}\textstyle\sum_{i=1}^m \hat{V}_{t-1}(s'_{t,i}(s,a))\right| + \tfrac{\epsilon}{8H} \\
&\leq \left|E_{s' \sim \Pr(\cdot|s,a)}\left[V_{t-1}^*(s')\right] - E_{s' \sim \Pr(\cdot|s,a)}\left[\hat{V}_{t-1}(s')\right]\right| + \tfrac{\epsilon}{8H} + \tfrac{\epsilon}{8H} \\
&\leq \max_{s \in S}\left|V_{t-1}^*(s) - \hat{V}_{t-1}(s)\right| + \tfrac{\epsilon}{4H} \\
&\leq \max_{(s,a) \in S \times A}\left|Q_{t-1}^*(s,a) - \hat{Q}_{t-1}(s,a)\right| + \tfrac{\epsilon}{4H}
\end{aligned}$$

where the first step is due to the fact that the error introduced by the reward estimate is less then or equal to $\frac{\epsilon}{8H}$ and the definition of the action-value and action-value estimates, the second step is due to (6.1), the third step simply replaces the expectation with the state that maximizes the difference between the optimal value function and the estimate, and the final step states the previous step in terms of action values.

By recursing on $\left|Q_t^*(s,a) - \hat{Q}_t(s,a)\right| \leq \max_{(s,a) \in S \times A}\left|Q_{t-1}^*(s,a) - \hat{Q}_{t-1}(s,a)\right| + \frac{\epsilon}{4H}$ and remembering that the base case $\left|Q_0^*(s,a) - \hat{Q}_0(s,a)\right| = 0$, we have $\left|Q_H^*(s,a) - \hat{Q}_H(s,a)\right| \leq H \cdot \frac{\epsilon}{4H} = \frac{\epsilon}{4}$.

Clearly $|Q_H^*(s,a) - Q^*(s,a)| \leq \gamma^H \left(\frac{1}{1-\gamma}\right)$. Therefore by choosing $H = \log_\gamma\left(\frac{\epsilon(1-\gamma)}{4}\right)$, we have $\left|Q^*(s,a) - \hat{Q}_H(s,a)\right| \leq \frac{\epsilon}{4} + \gamma^{\log_\gamma\left(\frac{\epsilon(1-\gamma)}{4}\right)}\left(\frac{1}{1-\gamma}\right) = \frac{\epsilon}{4} + \frac{\epsilon}{4} = \frac{\epsilon}{2}$.

To ensure that the action value estimates optimistic we add $\frac{\epsilon}{2}$ to each action value estimate to get

$$\left|Q^*(s,a) - \left(\hat{Q}_H(s,a) + \frac{\epsilon}{2}\right)\right| \leq \frac{\epsilon}{2} + \frac{\epsilon}{2}$$

Finally, the bound on sample complexity

$$O\left(mNKH\right)$$

of the Phased Q-learning algorithm depends on our choice of H and m. By plugging

in our choices for H and m, we get

$$O\left(\frac{NK}{\epsilon^2(1-\gamma)^3}\ln^3\left(\frac{4}{\epsilon}\right)\ln\left(\frac{NK\ln\left(\frac{4}{\epsilon}\right)}{\delta(1-\gamma)}\right)\right) \quad ,$$

which concludes the proof. □

The significance of Observation 6.2 is that we can learn arbitrarily accurate action-values for for any MDP with only polynomial number of samples with respect to the number of states and actions, given a generative model for that MDP.

6.2.2 Analysis of Target Task Sample Complexity

Transfer learning often results in faster learning in the target task, but this faster learning often comes at a price. In some cases, the transferred knowledge causes the target task RL algorithm to converge to a suboptimal policy. We refer to this situation as optimality loss. Optimality loss occurs because the transferred knowledge implicitly transforms the original target task into a different task. This new task can never have a higher valued policy than the original task because it is embedded in the original task. So we say that the transformed task induces optimality loss if the value of its optimal policy is less at some states than the value of the optimal policy in the original task (Definition 5.5).

The goal of TL in the target task is to decrease sample complexity of exploration without incurring much optimality loss. As we have seen previously, weak admissible heuristics are an effective mechanism for decreasing sample complexity and when α is kept small, they do not introduce much error into the learned policy. Therefore our objective is to determine when transferred action-values will satisfy an α-weak admissible heuristic.

We use the concept of a weak admissible heuristic to analyze action-value transfer

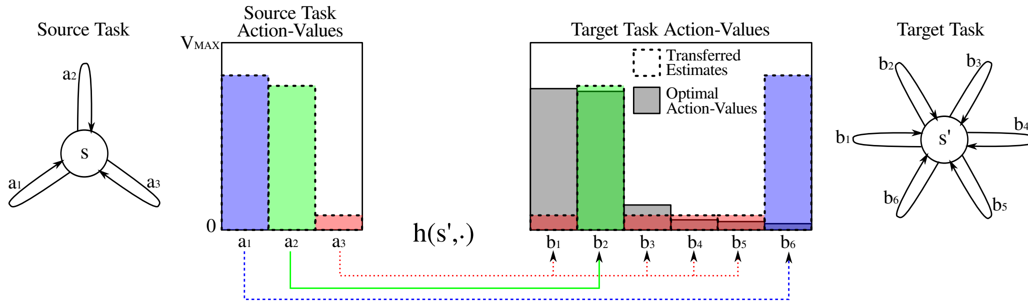


Figure 6.2: Transfer from a one-state source task with three actions to a one-state target task with six actions. Despite the transferred action-values severely underestimating the optimal action b_1 and severely overestimating the lowest valued action b_6,, and OFU exploration strategy can still converge to a near-optimal policy (i.e., b_2).

with the objective of learning to act near-optimally in the target task with sample complexity of exploration (Eq. 2.16) that is smaller than R-MAX or Delayed Q-learning without transferred knowledge. We denote the source task/MDP by Ω_{SRC} and the target task/MDP by Ω_{TRG}. Consider the situation in Figure 6.2. First the agent learns action-values for the source task. Next, because the source task and the target task have a different number of actions, a function h called an intertask mapping (defined below) is used to relate action-values from the source task to the target task. Finally, notice that in Figure 6.2 the transferred action-values satisfy a weak admissible heuristic. In this section, we explore assumptions about the intertask mapping needed to ensure that the transferred action-values satisfy a weak admissible heuristic and how transfer influences sample complexity of exploration in the target task.

Table 6.1: Transfer Outcomes

	Sample Complexity (compared to baseline)		
	Lower	No Change	Higher
$V_{\text{TRG}}^{\mathcal{A}_t}(s_t) \approx V_{\text{TRG}}^{*}(s_t)$	+	0	−
Optimality Loss	−	−	−

Positive Transfer	+
Neutral Transfer	0
Negative Transfer	−

There are two factors that affect positive transfer: (1) sample complexity and (2) optimality loss. Table 6.1 outlines when positive, negative, and neutral transfer occur. Positive transfer occurs when the sample complexity of exploration in the target task is lower than the sample complexity of the base RL algorithm and no optimality loss has occurred. Optimality loss occurs when transferred knowledge causes an RL algorithm to converge to a suboptimal policy along its current trajectory.

Typically, access to samples of the source task is less "expensive" than access to samples from the target task. For the purposes of this section, we assume unrestricted access to a generative model for Ω_{SRC}. As we have seen in the previous section, this assumption enables us to learn arbitrarily accurate source task action-values with arbitrarily high confidence with polynomially many samples. Therefore, we will assume that the estimated source task action-values $\hat{Q}_{\text{SRC}}$ are ϵ_{SRC}-accurate.

If Ω_{SRC} and Ω_{TRG} have different state-action spaces, then an intertask mapping $h : D \to S_{\text{SRC}} \times A_{\text{SRC}}$ is needed, where $D \subseteq S_{\text{TRG}} \times A_{\text{TRG}}$, to relate a subset of state-action pairs from the target task to state-action pairs in the source task. We assume that if $(s, a) \in D$, either there exists $(s, \tilde{a}) \in D$ such that

$$V^*_{\text{TRG}}(s) - \alpha \leq Q^*_{\text{TRG}}(s, \tilde{a}) \leq Q^*_{\text{SRC}}(h(s, \tilde{a})) \leq \frac{1}{1-\gamma} \ , \tag{6.2}$$

which is analogous to our assumption made for weak admissible heuristics with $W(s, a) = Q^*_{\text{SRC}}(h(s, a))$ or there exists $(s, \tilde{a}) \notin D$ such that

$$V^*_{\text{TRG}}(s) - \alpha \leq Q^*_{\text{TRG}}(s, \tilde{a}) \tag{6.3}$$

in which case we can assign the value $W(s, \tilde{a}) = \frac{1}{1-\gamma}$. To transfer action-values we use Algorithm 8, which is used to set initial action-value estimates given an

intertask mapping h, and ϵ_{SRC}-accurate source task action-value estimates $\hat{Q}_{\text{SRC}}$. In other words, at every state at least one nearly (α-)optimal action is mapped to an action-value which overestimates the true action-value or not mapped at all. If a state-action pair is not in the domain D, then we simply assign the maximum possible value to ensure it is optimistically initialized. Under these assumptions the transferred action-values are an α-weak admissible heuristic.

Theorem 6.3. *Let $\epsilon > 0$, $\epsilon_{\text{SRC}} > 0$, $\delta \in (0, 1]$, $h : D \to S_{\text{SRC}} \times A_{\text{SRC}}$ be an intertask mapping from a subset of state-action pairs in Ω_{TRG} to Ω_{SRC} satisfying (6.2) and (6.3), and $\hat{Q}_{\text{SRC}}$ be ϵ_{SRC}-accurate action-value estimates for Ω_{SRC}. If an instance $\mathcal{A}$ of the R-MAX algorithm, with action-value estimates initialized by Algorithm 8, is executed on Ω_{TRG} with appropriate parameters, then $V^{\mathcal{A}_t}_{\text{TRG}}(s) < V^*_{\text{TRG}}(s) - (\epsilon + \frac{\alpha}{1-\gamma})$ occurs on at most*

$$O\left(\frac{NK - Y}{\epsilon^4(1-\gamma)^8} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{NK - Y}{\delta\epsilon(1-\gamma)}\right)$$

timesteps t, with probability at least $1 - \delta$, where

$$Y = \left|\left\{(s, a) \in D \mid \hat{Q}_{\text{SRC}}(h(s, a)) < V^*_{\text{TRG}}(s) - (\alpha + \epsilon_{\text{SRC}})\right\}\right|$$

is the number of state-action pairs that are never explored.

Proof. The result follows from Theorem 5.20 and the fact that the transferred action-values satisfy an α-admissible heuristic. □

Theorem 6.4. *Let $\epsilon > 0$, $\epsilon_{\text{SRC}} > 0$, $\delta \in (0, 1]$, $h : D \to S_{\text{SRC}} \times A_{\text{SRC}}$ be an intertask mapping from a subset of state-action pairs in Ω_{TRG} to Ω_{SRC} satisfying (6.2) and (6.3), and $\hat{Q}_{\text{SRC}}$ be ϵ_{SRC}-accurate action-value estimates for Ω_{SRC}. If an instance $\mathcal{A}$ of the Delayed Q-learning algorithm, with action-value estimates W initialized*

by Algorithm 8, is executed on Ω_{TRG} with appropriate parameters, then $V^{\mathcal{A}_t}_{\text{TRG}}(s) < V^*_{\text{TRG}}(s) - (\epsilon + \frac{\alpha}{1-\gamma})$ occurs on at most

$$O\left(\frac{\sum_{(s,a)\in S_{\text{TRG}}\times A_{\text{TRG}}}[W(s,a) \geq V^*_{\text{TRG}}(s) - \alpha]_+}{\epsilon^4(1-\gamma)^7} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{NK - Y}{\delta\epsilon(1-\gamma)}\right)$$

timesteps t, with probability at least $1-\delta$, where

$$Y = \left|\left\{(s,a) \in D \mid \hat{Q}_{\text{SRC}}(h(s,a)) < V^*_{\text{TRG}}(s) - (\alpha + \epsilon_{\text{SRC}})\right\}\right|$$

is the number of state-action pairs that are never explored.

Proof. The result follows from Theorem 5.21 and the fact that the transferred action-values satisfy an α-admissible heuristic. □

The main importance of Theorems 6.3 and 6.4 is that we have reduced the analysis of action-value transfer in the target task to the analysis of learning with an α-weak admissible heuristic. Here, α controls optimality loss, and we can think of α (or $\alpha/(1-\gamma)$) as the error introduced by the intertask mapping h. If $\alpha \approx 0$ compared to ϵ, then there is little or no optimality loss compared to learning from scratch. In many, cases $\alpha = 0$ can be achieved. For example, if W turns out to be an admissible heuristic [6]. However, if α is large, then the result of TL is likely to be poor in the worst case. Similar to X in Theorems 5.20 and 5.21, when Y is large the sample complexity of exploration in the target task decreases significantly compared to learning from scratch with the base RL algorithm. Notice, however, that the sample complexity of exploration is never worse than learning from scratch. Thus, positive transfer is characterized by optimality loss and sample complexity, and Theorems 6.3 and 6.4 help to clarifies this relationship.

Table 6.2: Algorithm Conditions

Abbr.	Description
QL	Q-learning (ε-greedy exploration) without Transferred Knowledge
DQL	Delayed Q-learning without Transferred Knowledge
RMAX	R-MAX without Transferred Knowledge
TL(QL)	Q-learning initialized with Transferred Action-Values
TL(DQL)	Delayed Q-learning initialized with Transferred Action-Values
TL(RMAX)	R-MAX initialized with Transferred Action-Values

Table 6.3: (Source Task, Target Task) Pairs

Source Task	Src. Desc.	Target Task	Trg. Desc.
Three-Arm Bandit	One state task with three actions (Figure 6.2).	Six-Arm Bandit	One state task with six actions (Figure 6.2).
Reset	A chain of states with a single reward at the end of the chain (Figure 6.1).	Double Reset	Two chains of states the end of one chain has a higher reward than the other (Figure 6.3).
Red Herring	Gridworld with two "red herring" states and one "goal" state (Figure 6.4a)	Taxi	Transport a passenger two and from one of four pickup and drop-off locations (Figure 6.4b)
Two-joint Reaching	Reach for various target points with a two-joint arm (Figure 6.5).	Three-joint Reaching	Reach for various target points with a three-joint arm (Figure 6.5).

6.3 Experiments & Results

In this section, we describe and report the results for several experiments meant to emphasize several aspects of our analysis. Throughout this section we will refer to algorithmic conditions by abbreviations documented in Table 6.2.

We use several different (source task, target task) pairs in our experiments to emphasize different aspects of TL. Table 6.3 summarizes the (source task, target task) pairs from our experiments.

Table 6.4: One-State Transfer Expected Rewards

Source Task			**Target Task**		
$R(s, a_1)$	=	0.9	$R(s', b_1)$	=	0.8
$R(s, a_2)$	=	0.82	$R(s', b_2)$	=	0.78
$R(s, a_3)$	=	0.12	$R(s', b_3)$	=	0.15
			$R(s', b_4)$	=	0.12
			$R(s', b_5)$	=	0.1
			$R(s', b_6)$	=	0.08

Our first (source task, target task) pair was chosen for its simplicity. Figure 6.2 shows a simple transfer scenario between a target task with one state and six actions and a source task with one state and three actions. In both the source and the target task we assume that the discount factor $\gamma = 0$ so that the value of each action is equivalent to its expected reward. Each actions reward distribution is a Bernouli distribution, assigning a value of 1 with probability equal to the expected reward μ and a value of 0 with probability $1 - \mu$. Table 6.4 shows the expected rewards assigned to each action in the source task and the target task. We chose this simple domain to enable comparison between various intertask mappings in simplest tasks possible.

Our second (source task, target task) pair was designed to demonstrate a flaw with transferring action-values below the corresponding optimal target task action-values. In other words, if we simply transfer action-values without adding the β term this sometimes results in choosing a suboptimal policy. In this scenario, the source task is the difficult reset task (Figure 6.1), which consists of a single chain of $N_{\text{SRC}} = 15$ states. From each state, except for the final state in the chain, one action transitions to the next state in the chain with probability 0.8 and remains at the same state with probability 0.2. All other actions reset the agent to the first state in the chain. At the final state in the chain all actions reset the agent to the first state in the chain. One state-action pair at the final state gives a reward of 1 while all

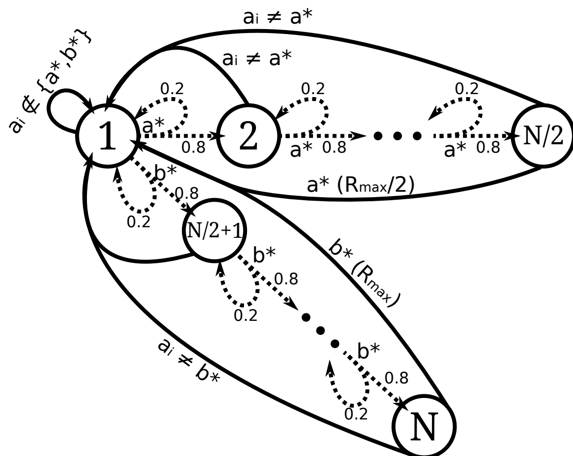
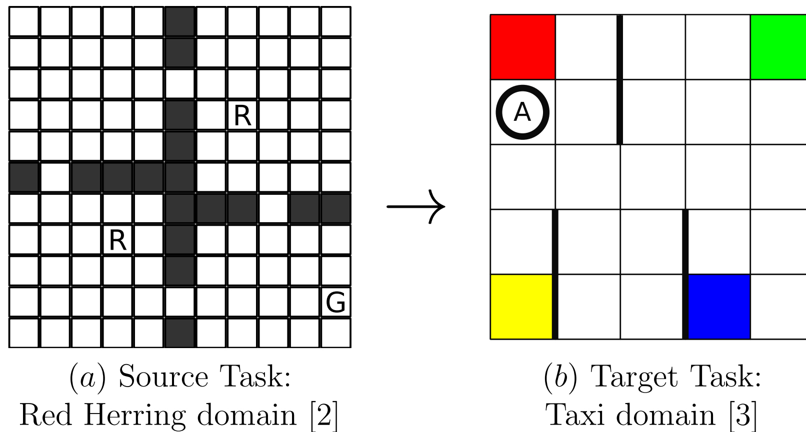


Figure 6.3: A double reset MDP is similar to a reset MDP (Figure 6.1), except that it has two chains of states. The final state of the first chain ($N/2$) gives reward $R_{\text{MAX}}/2$ when action a^* is executed, while the final state of the second chain N gives reward R_{MAX} when b^* is executed.

other state-action pairs give no reward (i.e., a reward of 0). The target task, on the other hand, is the double reset task (Figure 6.3) with $N_{\text{TRG}} = 25$ states. The double reset task consists of two chains of states. Both chains consist of 13 states and have similar dynamics to the reset task. The difference between the two chains is that the end of one chain gives a reward of 1 while the end of the other chain gives a reward of 0.5. If the optimal chain is pessimistically represented, the agent may select the suboptimal chain.

Our third (source task, target task) pair tests transfer between two gridworld tasks. The source task is the Red Herring task introduced by Hester et al. [2] with 104 states, and the target task is the Taxi task introduced by Dietterich [3] with 500 states. These tasks were chosen because they have far more states than the previous transfer scenarios, but not so many states, that we cannot execute R-MAX on the tasks in a reasonable amount of time. The Red Herring domain (Figure 6.4a) is an 11×11 gridworld domain where the agent is initially placed in one of the 25 cells of the upper left room. The agent can attempt to move in one of four possible



(a) Source Task:
Red Herring domain [2]

(b) Target Task:
Taxi domain [3]

Figure 6.4: The Red Herring task (a) [2] contains a goal state G that gives a reward +20 and two red herring states R that give a reward 0, while all other states give the reward -1. The Taxi task (b) [3] requires the agent to pick up a passenger at one of four colored locations and drop the passenger off at its desired location.

directions: up, down, left, and right. However, there is a small probability 0.2 that the agent will move diagonally instead of its desired direction. There are two "red herring" states, denoted by the capital letter 'R', which give a reward of 0, a goal state, denoted by 'G', which gives a reward of +25, and all other states give a reward of -1. An episode ends when the agent enters either red herring state or the goal state. The significance of the red herring states is that they are much easier to find than the goal state. Thus if the task's state space is not sufficiently explored, then the agent may settle on one of the red herring states rather than the goal state. The Taxi task has a smaller number of cell locations but a larger overall state-action space. Four of the cells are marked with different colors: red, green, yellow, and blue. As in the Red Herring task, the agent can attempt to move up, down, left, and right. Again there is a probability 0.2 that the agent will move in a diagonal direction instead of the desired direction. In addition, the agent can attempt to pickup and drop off a passenger. At the beginning of each episode the agent is initialized to a random cell in the environment. A passenger is placed at one of the colored cells

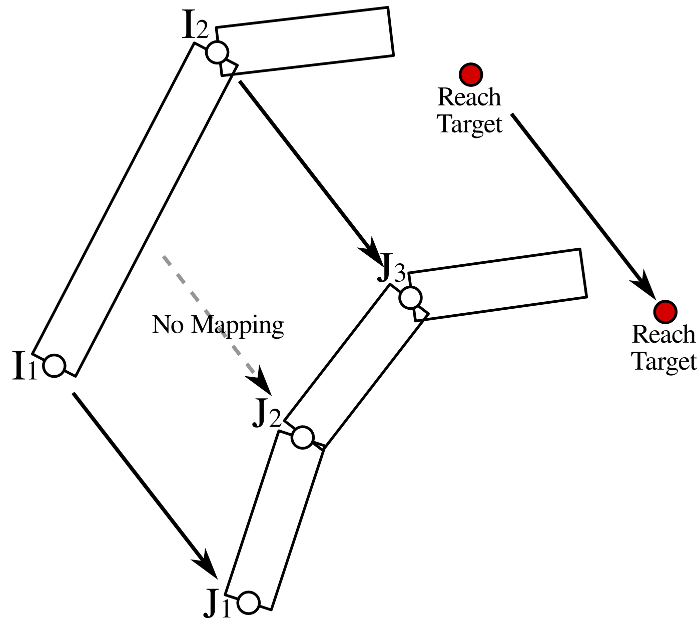


Figure 6.5: Transfer from a two-joint arm (source task) to a three-joint arm (target task).

and has a desired destination at one of the other colored cells. The agent received a reward of -1 for moving and a reward of $+20$ for picking the passenger up and successfully dropping him off at his desired location. An episode ends when the agent successfully picks up the passenger and drops him off at his desired location. If the agent attempts to pickup or drop off the agent at any cell other than one of the four colored cells, the agent receives a reward of -10. The significance of the Taxi task is that it has a large state-space and presents an interesting benchmark problem for RL algorithms. Although both the Red Herring task and the Taxi task are gridworld tasks the objectives are quite different and it is interesting to see if positive transfer can occur between such different tasks.

Our last (source task, target task) pair tests transfer between two inverse kinematic problems (Figure 6.5). These tasks have a large number of states compared to the previous transfer scenarios and therefore test how action-value transfer scales as

the number of state-action pairs grows large. The state encoding for the two-joint control task was defined by

$$\langle I_1, I_2, TI \rangle$$

where I_1 is one of seven different joint angles for the first joint in the arm, I_2 is one of seven different joint angles for the second joint in the arm, and TI is the index of one of four different target points.The source task has four possible targets and each joint can take on one of seven different angle so that the total number of states is 512. The state encoding for the three-joint control task was defined by

$$\langle J_1, J_2, J_3, TI \rangle$$

where J_1 is one of seven different joint angles for the first joint in the arm, J_2 is one of seven different joint angles for the second joint in the arm, J_3 is one of seven different joint angles for the third joint in the arm, and TI is the index of one of four different target points. The target task has 8748 states.

6.3.1 Experiment: One State Transfer

We compared three different intertask mappings (Figure 6.6) in the one state transfer scenario to show how intertask mappings can impact sample complexity of exploration in the target task. The first intertask mapping (Figure 6.6a), denoted by BAD, represents a poor intertask mapping because the transferred action-values underestimate the near-optimal actions. The only actions that are overestimated have much lower expected reward than the optimal action. The second intertask mapping (Figure 6.6b), denoted by AH, induces an admissible heuristic, because the transferred action-values are overestimates at every state-action pair. Notice however, that several of the action-values are well below the optimal action-value

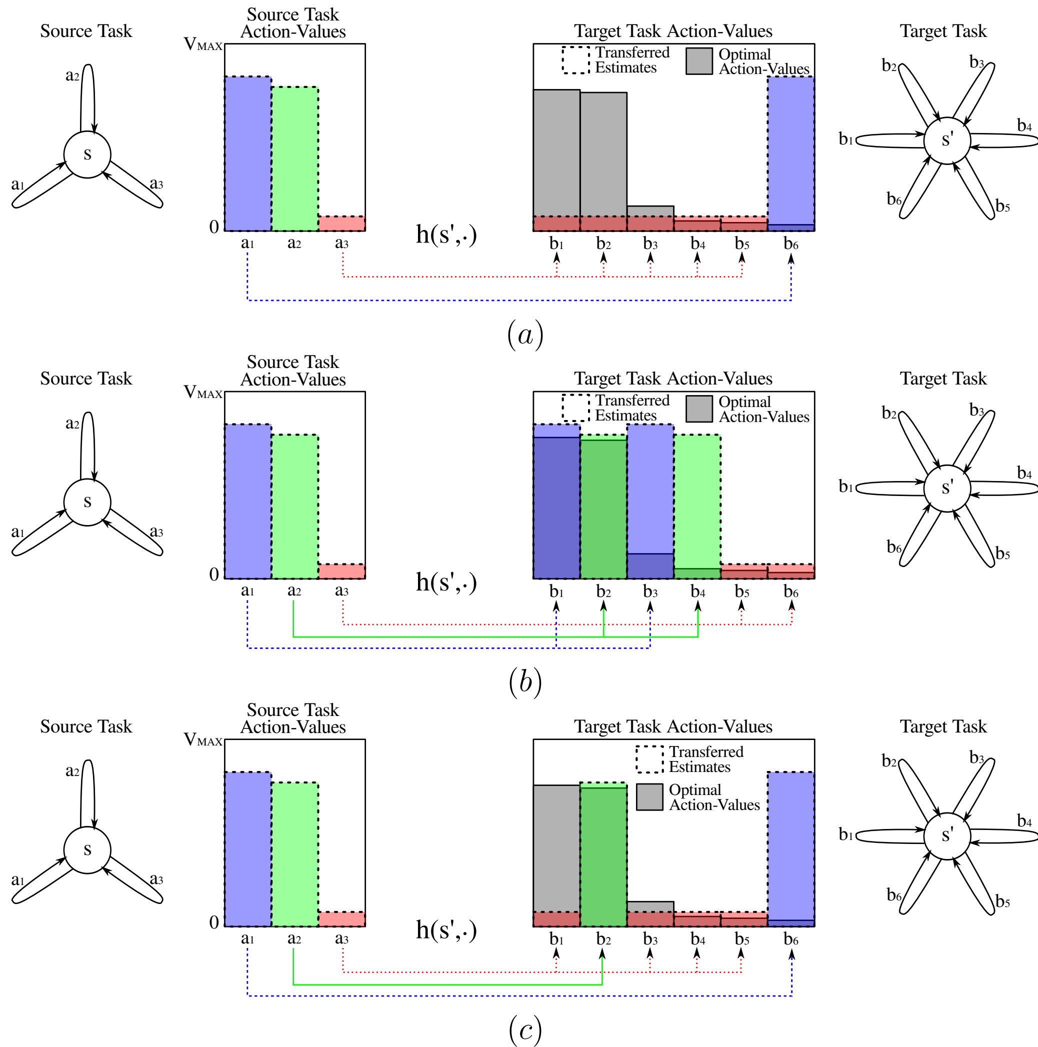


Figure 6.6: Comparison between three possible intertask mappings in the one state transfer scenario. (a) A poor intertask mapping, denoted BAD. The transferred action-values underestimate the near-optimal actions and overestimate the worst action b_6. (b) An intertask mapping that induces an admissible heuristic, denoted AH. The transferred action-values are all overestimated. (c) An intertask mapping that induces a weak admissible heuristic, denoted WAH. The transferred action-values overestimate one (but not both) near-optimal action.

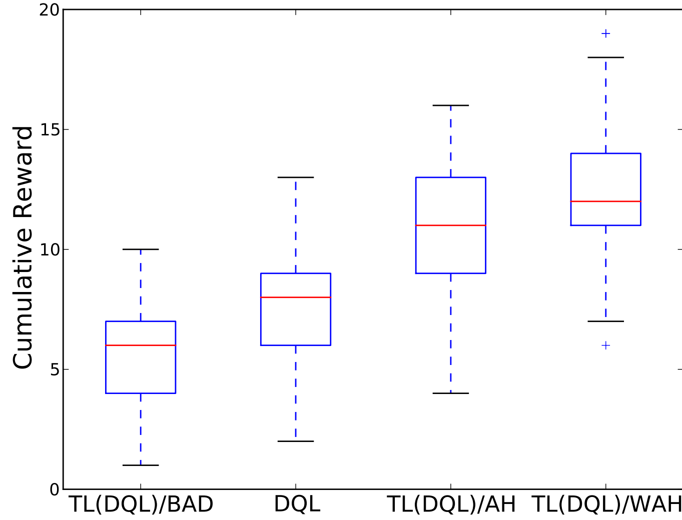


Figure 6.7: Comparison of cumulative reward for 100 different runs of each Delayed Q-learning transfer condition under the one state transfer scenario. Each algorithm was run in the target task for only 45 timesteps. Whiskers indicate 1.5 times the interquartile range.

and may therefore never be executed. The third intertask mapping (Figure 6.6c), denoted by WAH, induces an α-weak admissible heuristic in the target task, with $\alpha = 0.02$. This is because the action b_2, which has expected reward that is 0.02 smaller than the optimal action b_1 is overestimated.

Figure 6.7 shows the cumulative reward achieved in the one state transfer scenario's target task by each of the three intertask mappings and the baseline Delayed Q-learning algorithm over 20 timesteps. Using a poor intertask mapping TL(DQL)/BAD results in the lowest cumulative reward. This is lower than if we had applied the baseline algorithm Delayed Q-learning (DQL) without transferred action-values. However, the conditions with intertask mappings that induce an admissible heuristic TL(DQL)/AH or induce a weak admissible heuristic TL(DQL)/WAH both achieve higher cumulative reward than the baseline DQL.

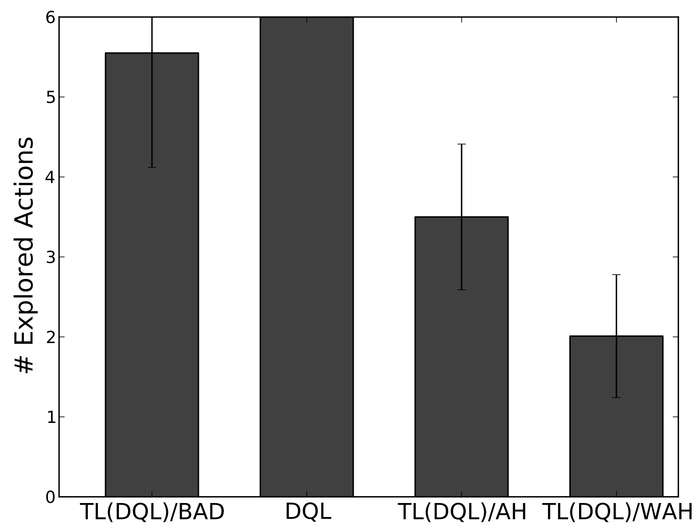


Figure 6.8: Having a reasonable intertask mapping helps to eliminate certain state-action pairs from consideration. In the target task, TL(DQL)/BAD and DQL typically explore all six actions, while TL(DQL)/AH and TL(DQL)/WAH typically explore about half (or fewer) of the actions. Error bars indicate ±1 standard deviation.

Figure 6.8 shows the number of different actions explored in the target task by each of the different learning conditions. The two transfer learning conditions TL(DQL)/AH and TL(DQL)/WAH with reasonable intertask mappings typically explore half of the action space, while TL(DQL)/BAD and DQL explore all six actions. This demonstrates how transferred action-values implicitly eliminate state-action pairs.

Keep in mind that all admissible heuristics as defined by Strehl et al. [6] are also weak admissible heuristics with $\alpha = 0$. The critical point is that when the transferred action-values satisfy the conditions of an α-weak admissible heuristic, positive transfer is likely to occur. This provides a simple example of positive transfer.

If we take a closer look at the three intertask mappings presented in Figure 6.6, we can see that all of the intertask mappings are actually weak admissible heuristics with different α values. The BAD intertask mapping transfers action-values that are optimistic for action b_4. Since its true value is 0.12 and the optimal action-value is 0.8 the value of $\alpha = 0.8 - 0.12 = 0.68$. This large value of α explains why the BAD intertask mapping performs so poorly. The AH intertask mapping has an α value of 0 because the value of the optimal action b_1 is overestimated, and as mentioned above the WAH intertask mapping is a weak admissible heuristic with $\alpha = 0.02$. So we can see that restricting the intertask mappings to the set of intertask mappings that induce weak admissible heuristics allows us to explain TL performance for a very general set of intertask mappings.

We also investigated using Q-learning as the target task algorithm, instead of Delayed Q-learning. Figure 6.9 shows the cumulative reward achieved in the one state transfer scenario's target task by each of the three intertask mappings and the baseline Q-learning algorithm with randomly initialized action-values. The initial action-values made little difference, with respect to achieved cumulative reward. This

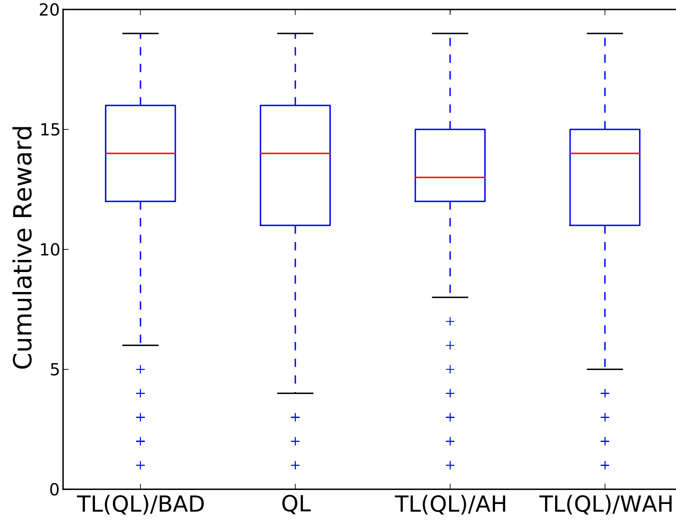


Figure 6.9: Comparison of cumulative reward for 100 different runs of each Q-learning transfer condition under the one state transfer scenario. Each algorithm was run in the target task for only 45 timesteps. Whiskers indicate 1.5 times the interquartile range.

is in contrast to what we saw for Delayed Q-learning (Figure 6.7) where the AH and WAH intertask mappings induced higher cumulative reward than the baseline. The reason for this difference is probably due to the fact that the action-value estimates maintained by Delayed Q-learning either decrease or stay the same (i.e., action-value estimates never increase) and therefore Delayed Q-learning is better able to utilize overestimated action-values than Q-learning.

6.3.2 Experiment: Variable β

Algorithm 8 requires a parameter β that biases the transferred action-values. The importance of the parameter β is that when β is large enough it can transform an intertask mapping that does not induce a weak admissible heuristic into an intertask mapping that does.

Figure 6.10 demonstrates the impact of the choice of β on cumulative reward when

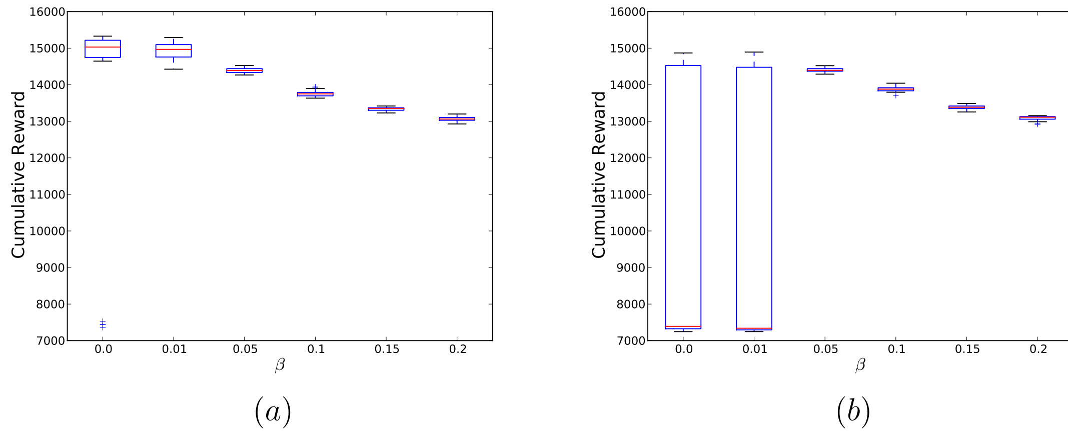


Figure 6.10: Varying β influences the cumulative reward achieved at the (reset, double reset) transfer scenario. Whiskers indicate 1.5 times the interquartile range. (a) When a good intertask mapping is used to transfer action-values, as β increases the cumulative reward decreases. (b) If a poor intertask mapping is used, adding a small positive value improves the cumulative reward. However, adding to large a value causes the cumulative reward to decrease. Notice that the penalty for selecting β too small is much worse than selecting a value that is too large.

transferring from an instance of the reset task to an instance of the double reset task with two different intertask mappings. Figure 6.10a shows the impact of β when the chosen intertask mapping closely matches action-values in the source and target task. Under this "good" intertask mapping the cumulative reward decreases as β increases. This is due to the fact that the Delayed Q-learning algorithm potentially requires more updates to converge to a near-optimal policy when the transferred action-values are greater. However, Figure 6.10b shows that always selecting $\beta = 0$ can have strong negative consequences. In this scenario, transfer of action-values was performed with a "poor" intertask mapping that arbitrarily mapped state-action pairs from the target task to the source task. When β is too small (< 0.05) the transferred action-values often result in the TL agent selecting the suboptimal path in the double reset task. However, if β is large enough, the TL agent almost always converges to the optimal path in the double reset task.

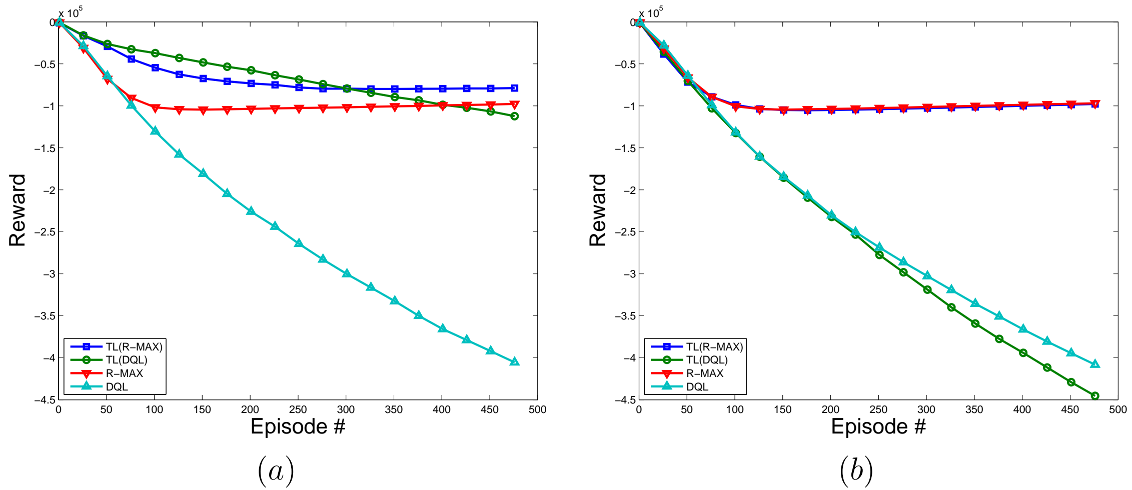


Figure 6.11: (a) With a nearly optimal intertask mapping, transferring action-values from the Red Herring domain to the Taxi domain results in higher cumulative reward for both R-MAX and Delayed Q-learning than their respective base algorithms without transfer. (b) With an arbitrarily assigned intertask mapping, the transferred action-values do not result in much loss for either R-MAX or Delayed Q-learning compared to their respective base algorithms without transfer.

We want to select β as small as possible, so that the resulting transferred action-values are small, but, on the other hand, we want to ensure that the transferred action-values satisfy the constraints for a weak admissible heuristic with small α.

6.3.3 Experiment: Scaling Up in a Gridworld Tasks

In this experiment, we compared the use of R-MAX and Delayed Q-learning as target task learning algorithms in the Red Herring/Taxi transfer scenario. Again we compared results using two different intertask mappings. The first intertask mapping was a "good" intertask mapping chosen by pairing state-action pairs from the target task to state-action pairs from the source task with similar optimal action-values. The second intertask mapping was a "poor" intertask mapping, with state-action pairs mapped arbitrarily.

Figure 6.11a compares cumulative reward when transferring action-values with the "good" intertask mapping. Transferring action-values improves both DQL and

RMAX. Notice that there is a large difference between the performance of the base RL algorithm RMAX and DQL. RMAX without action-value transfer learns a good solution to the Taxi task in far fewer episodes than the DQL. On the other hand, the improvement caused by transferring action-values compared to the base RL algorithm is much larger for DQL than RMAX. This is an important finding because RMAX uses a very computationally expensive planning phase, which may make learning with RMAX infeasible in problems with large state-action spaces.

Figure 6.11b compares cumulative reward when transferring action-values with the "poor" intertask mapping. In this case, transferring action-values does not improve performance. However, it is important to notice that TL(RMAX) does not perform worse than RMAX and TL(DQL) performs only marginally worse than DQL. This is an interesting finding because we would expect to see significant negative transfer when arbitrary action-values are transferred.

6.3.4 Experiment: Scaling Up in an Inverse Kinematic Task

In this experiment, we attempted to decrease the time needed to learn to control the end-effector of a three-joint mechanical arm by transferring control knowledge learned for a two-joint mechanical arm. Unlike the previous experiments, we hand coded two different intertask mappings. For both intertask mappings, we defined h by

$$h(\langle J_1, J_2, J_3, TI\rangle, \langle a_1, a_2, a_3\rangle) = (\langle J_1, J_3, TI\rangle, \langle a_1, a_3\rangle) \tag{6.4}$$

where $\langle J_1, J_3, TI\rangle$ is the state in the source task and $\langle a_1, a_3\rangle$ is the action in the source task. The difference between the two intertask mappings is that the conservative intertask mapping had a smaller domain than the more aggressive intertask mapping. The conservative intertask mapping only mapped state-action pairs when the second joint J_2 was straight and the second component of the action $a_2 = 0$ did not change

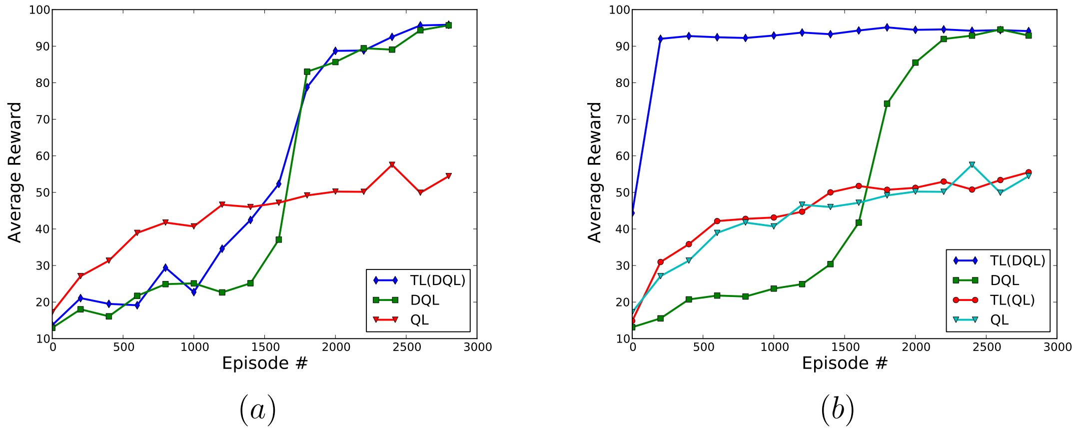


Figure 6.12: (a) Average reward achieved with a conservative intertask mapping. (b) Average reward achieved with an aggressive intertask mapping.

the J_2. The aggressive intertask mapping was defined over the entire state-action space of the target task.

Figure 6.12 shows the average reward received per episode with the conservative and aggressive intertask mappings. Notice that both intertask mappings provide some advantage compared to learning from scratch. However, the improvement due to transfer with the aggressive intertask mapping is much more dramatic than using the conservative intertask mapping. Notice that transferred knowledge does not seem to help QL. This is most likely because the transferred action-values are not accurate approximations of the target task's optimal action-value function Q^*_{TRG}.

We further investigated transfer with the aggressive intertask mapping to better understand the factors influencing transfer. To help with our investigation we generated the optimal action-values for the source task and the optimal action-values for the target task. This enabled us to look at the final action-values learned by DQL and TL(DQL) to determine the proportion of states where the greedy policy selects α-good actions. Figure 6.13 shows the proportion of α-good actions for the final learned policies of DQL and TL(DQL), where we selected $\alpha = 0.1$. Other values of

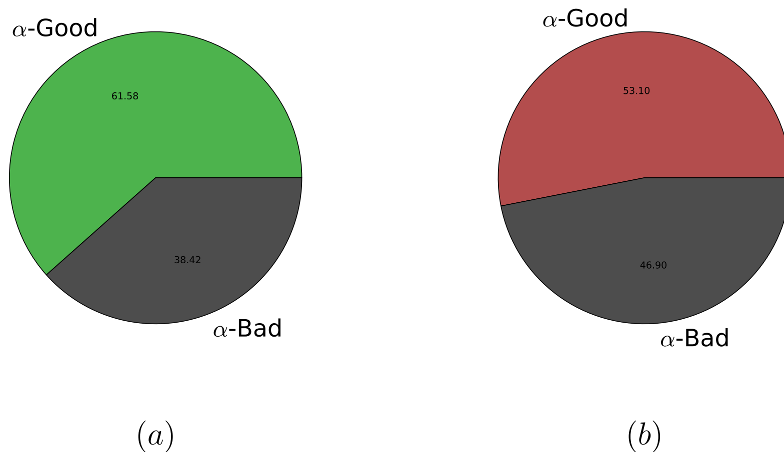


Figure 6.13: Proportion of α-good actions (with $\alpha = 0.1$) in the final learned policy for (a) DQL and (b) TL(DQL) and $\beta = 0$.

α around 0.1 provided similar results. Notice that neither policy comes anywhere near the theoretical demand that the policy select α-good actions at every state, and what may be more surprising is that TL(DQL) has even fewer α-good actions despite performing just as well as the policy learned by DQL in terms of average reward.

Even more confusing is the fact that the transferred action-values only satisfied the α-weak admissible heuristic conditions at a few states (Figure 6.14). This seems to suggest that weak admissible heuristics have little to do with the efficacy of action-value transfer. However, it turns out that the learned policies only need to be good at a few states.

To help understand why the policy learned by TL(DQL) performs as well as DQL even though it uses less α-good actions we looked at how frequently different states were visited. Figure 6.15 shows that both DQL and TL(DQL) spend the vast majority of timesteps in a small number of the total number of states. However, TL(DQL) is even more concentrated on a few states than DQL. Figure 6.15 demonstrates the important ability of action-value transfer to focus exploration on fewer but poten-

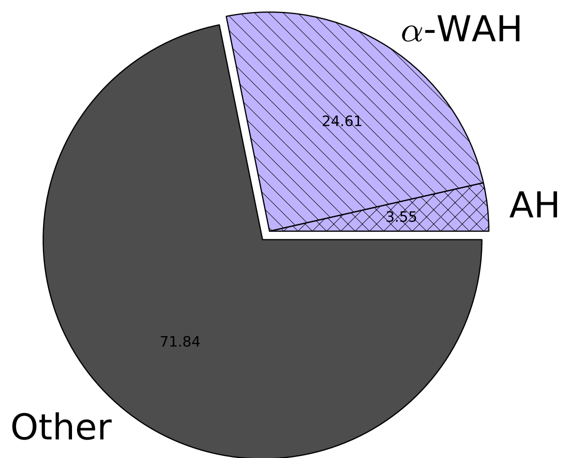


Figure 6.14: Average proportion of states with transferred action-values ($\beta = 0$) that optimistically initialize an α-good action satisfying the weak admissible heuristic (WAH) criterion and the number of states with transferred action-values that are optimistic for every action satisfying the admissible heuristic criterion (which also belongs to WAH).

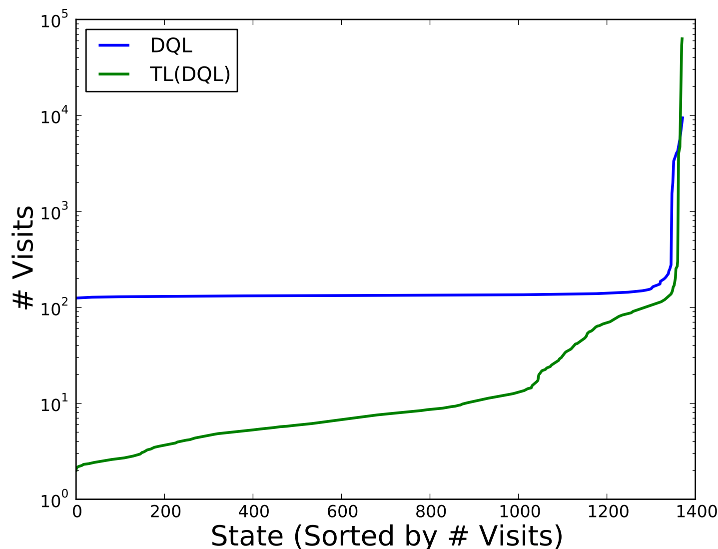


Figure 6.15: Comparison between the number of visits made to each state by DQL (the baseline algorithm) and TL(DQL) with $\beta = 0$ averaged over 100 runs. Both DQL and TL(DQL) visit a small fraction of states far more frequently than all of the other states. However, TL(DQL) is more concentrated on a few states than DQL.

tially more important state-action pairs. This figures is also important because it suggests that the target task has a small number of critical states, where accuracy of the policy is most important.

If we only consider the set of highly visited states (in this case states that were visited more than mK times), then Figure 6.16 shows a very different story for the final policy learned by TL(DQL). In this case, the policy selects an α-good action at almost all of the highly visited states.

Figure 6.17 shows that a large proportion of the highly visited states satisfy the α-weak admissible heuristic condition (although not all hightly visited states). This suggests that the α-weak admissible heuristic concept may significantly contribute to the efficacy of action-value transfer after all. To further confirm the importance of α-weak admissible heuristic, we looked at the proportion of highly visited states

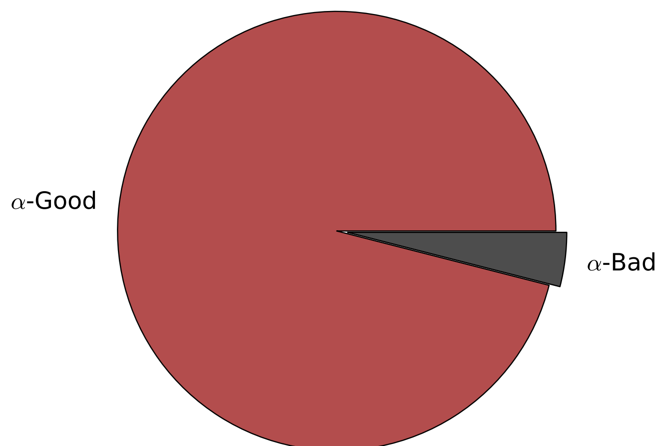


Figure 6.16: Proportion of α-good actions selected by the final learned policy of TL(DQL) over the set of highly visited states.

where the transferred action-values satisfied the admissible heuristic conditions (i.e., all action-value estimates are optimistic for that state) and where the transferred action-values satisfied the α-weak admissible heuristic conditions (i.e., at least one α-good action has an optimistic action-value estimate for that state). We measured the proportion of highly visited states where the learned policy at that state selects an α-good action. Figure 6.18 shows the results of this test. Practically every time the transferred action-values for a state conformed to the conditions of a weak admissible heuristic, the learned policy selects an α-good action at that state. This confirms that the weak admissible heuristic concept is important for the efficacy of action-value transfer.

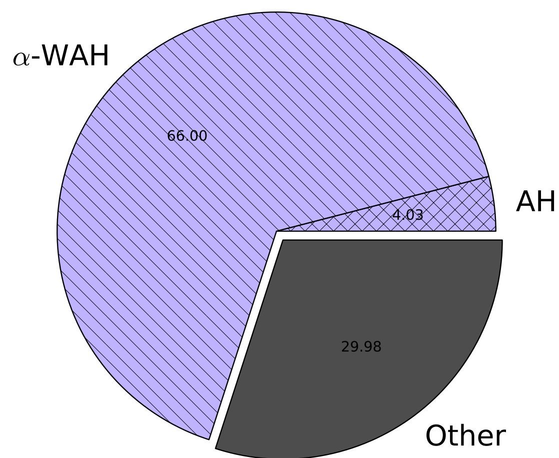


Figure 6.17: Average proportion of highly visited states with transferred action-values ($\beta = 0$) that optimistically initialize an α-good action satisfying the weak admissible heuristic (WAH) criterion and the number of states with transferred action-values that are optimistic for every action satisfying the admissible heuristic criterion (which also belongs to WAH).

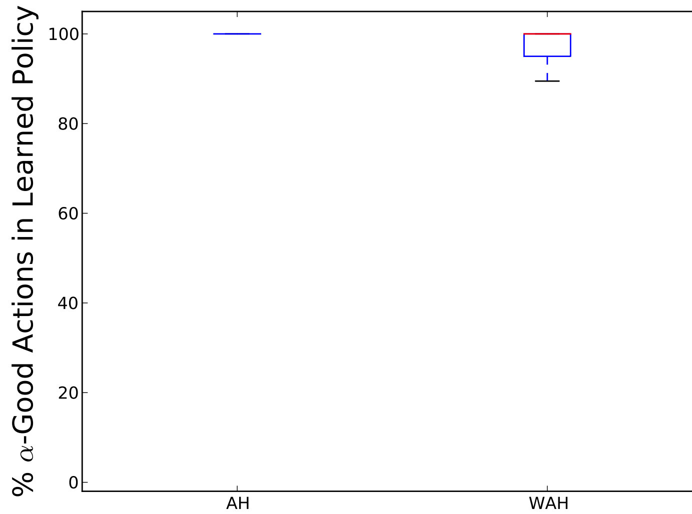


Figure 6.18: The percentage of highly visited states conforming to the admissible heuristic or α-weak admissible heuristic conditions where the learned policy at that state selects an α-good action. Whiskers indicate 1.5 times the interquartile range.

6.4 Discussion

Because action-value transfer depends on learning accurate action-values for the source task, we can only realistically expect to apply action-value transfer in scenarios where the dynamics and reward function of the source task are well-known. If we don't have a model or a generative model for the source task, we may not be able to estimate accurate action-values for the source task. However, in practice, if we use Delayed Q-learning or Modified R-MAX, these algorithms guarantee (with appropriate parameter values) that their action-value estimates are optimistic. Therefore if the intertask mapping induces an α-weak admissible heuristic with ϵ_{SRC}-accurate action-values, then the if the source task action-values are optimistic, the resulting transferred action-values will still induce an α-weak admissible heuristic. Thus we will not incur additional optimality loss under by applying transfer with these potentially less accurate action-values.

However, we cannot expect to apply action-value transfer in a cascade, treating the target task from a previous application of action-value transfer as a source task for a second action-value transfer. In this case, many of the action-values may have been pessimistically mapped and therefore may result in significant optimality loss in the next target task.

Another issue with the application of action-value transfer is obtaining an intertask mapping. As we have seen, the intertask mapping plays an important role in learning performance in the target task. There are two potential options. The first is to expect a domain expert to provide an intertask mapping. The second is that we can attempt to learn an intertask mapping. In this section, we have assumed that intertask mappings were provided by a domain expert. There has, on the other hand, been some work towards learning an intertask mapping [47, 11]. Learning intertask mappings would go a long way toward deploying TL autonomously. However, the best intertask mapping cannot in general be resolved without knowing the optimal action-values in the source and target task. To solve this problem, it may be possible to consider learning algorithms with regularization to limit the choice of intertask mappings.

Finally, action-value transfer often seems to improve performance even in some situations where the transferred action-values do not satisfy an α-weak admissible heuristic. We have observed in several experiments that when some critical action-values are initially underestimated by the target task RL algorithm, the algorithm will still converge to a near-optimal policy because the estimated action-values capture the overall structure of the optimal action-values, but they are lower than the optimal action-values. This suggests that, although we have a good idea of when action-value transfer will successfully result in positive transfer in a broad sense there are many additional idiosyncratic circumstances where action-value transfer may still

work.

6.5 Summary

Previous research has established the potential value of transferring action-values [45, 58, 46, 7, 8], however, these works have been primarily heuristic. We approached the study of action-value transfer from a theoretical perspective. First, we noted that some single task RL algorithms are provably efficient in terms of sample complexity. Then we developed a structure called a weak admissible heuristic that influences the sample complexity of exploration without introducing much optimality loss. We developed a method for transferring action-value transfer that uses weak admissible heuristics, and analyzed the sample complexity of learning accurate action-values from a source task. Unfortunately, learning accurate action-values is a difficult problem and can only be done in a provably sample-efficient manner when it is possible to fully explore the source task. Then we analyzed the sample complexity of exploration in the target task with the assumption that the given intertask mapping can be manipulated to produce action-values that satisfy a weak admissible heuristic. Our experiments provide further support that combining action-value transfer with directed exploration has important benefits. We believe that these findings provide strong support for pairing action-value transfer with directed exploration.

7. ANALYSIS OF MULTITASK LEARNING

General purpose RL algorithms such as R-MAX and Delayed Q-learning achieve polynomial sample complexity bounds over all MDPs. In a restricted set of MDPs, it may be possible to find much more sample efficient, domain specific RL algorithms. For example, controlling the same robot arm to manipulate different objects can be thought of as a set of different but related tasks. After solving a few control tasks, it may be possible to discover sequences of actions that are almost never useful. By eliminating these sequences of actions from consideration, solutions for new tasks in the same domain can be found more quickly.

Multitask RL (MTRL) is a special case of TL, where the learning system interacts with a sequence of tasks that are distributed by a probability distribution. As the learning system interacts with tasks it ought to be able to improve its ability to learn within the domain. By making this assumption it may be easier to prove benefits of TL than in the more general TL setting. In this section, we consider the following questions:

1. How can the notion of sample complexity of exploration be generalized to the MTRL setting?
2. Given a domain of tasks, how much better might a domain specific learning algorithm be compared to a base RL algorithm?
3. How can a domain specific RL algorithm be learned automatically?

In this section, we extend the notion of sample complexity of exploration to the MTRL setting by considering the sample complexity of the RL algorithm in the next task drawn from the domain.

To answer the second question, we develop several notions of the complexity of a domain based on the number of state-action pairs that need to be explored to achieve our generalized sample complexity criterion. All of the measures of complexity of a domain are valid, but each notion take advantage of different properties of the domain of tasks.

To answer the third question, we continue our general theme of trying to eliminate state-action pairs from consideration while still preserving optimality guarantees. As we saw with Theorems 5.7 and 5.10 having an exploration table can significantly decrease sample complexity of exploration, but a domain specific exploration table must also attempt to minimize optimality loss. We show how an exploration table can be learned by sampling tasks from a domain.

7.1 Background

A domain of MDPs $\mathcal{D}$ is a probability distribution over a finite or infinite set D of MDPs. Figure 7.1 depicts the general MTRL setting, where a learning agent faces a sequence of tasks over time. Each time a task is sampled, the agent constructs a domain specific RL algorithm using its (initially empty) library of knowledge and interacts with the current task. When finished learning in the current task, the agent's library is updated with the statistics acquired by the learning algorithm. Although not strictly necessary, we assume that all of the MDPs in a domain share the same state-action space but not necessarily the same transition probabilities or reward functions. Throughout this section we make the assumption that every MDP $\Omega \in D$ has state set S with $N = |S|$ states and action set A with $K = |A|$ actions.

Tanaka and Yamamura [41] considered MTRL and introduced algorithms that learn statistics about the the action-values, such as the mean and standard deviation. They used these statistics to speed up learning in new tasks sampled from

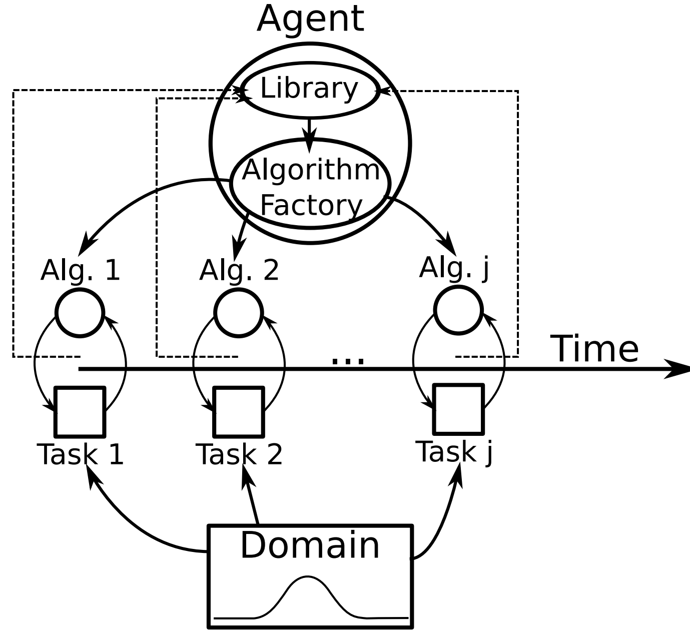


Figure 7.1: Under multitask RL an agent, consisting of a library and algorithm factory, is confronted with a sequence of tasks drawn from the same domain (or distribution over tasks). For each new task, the agent constructs an algorithm by combining knowledge from its library with its algorithm factory to construct a domain specific algorithm. The domain library is updated with each experienced task.

Algorithm 10 Multitask Q-learning (MTQL)

Require: $\mathcal{D}, S, A, \gamma, \alpha, \varepsilon, \tau$

1: Initialize Q_0 for all $(s,a) \in S \times A$ with values from $\left[\frac{R_{\text{MIN}}}{1-\gamma}, \frac{R_{\text{MAX}}}{1-\gamma}\right]$
2: **for** $n = 1, 2, \ldots$ **do**
3: $\quad \Omega_n \sim \mathcal{D}$ {Sample a new task.}
4: $\quad$ **if** $n \leq \tau$ **then** {Training Phase}
5: $\quad\quad$ Learn action-value estimates $\hat{Q}_n$ for Ω_n
6: $\quad\quad$ $Q_0 \leftarrow Q_0 + \hat{Q}_n$
7: $\quad$ **else** {Domain Specific Learning}
8: $\quad\quad$ $\mathcal{A} \leftarrow$ Q-learning$(S, A, \alpha, \gamma, \varepsilon, \frac{Q_0}{\tau})$
9: $\quad\quad$ Execute $\mathcal{A}$ on Ω_n
10: $\quad$ **end if**
11: **end for**

the distribution. Algorithm 10 is a simple algorithm that uses the learned mean action-value estimates to initialize the Q-learning algorithm before learning in each new task. If most tasks in the domain have similar action-values, then this approach can considerably improve learning speed in novel tasks. However, if there is high variance in the action-values, then this approach may lead Q-learning to settle on a suboptimal policy. Tanaka and Yamamura [41] used information about the standard deviation to add exploration bonuses to state-action pairs where the action-values tend to deviate significantly from the average. Tanaka and Yamamura [41] assumed that the action-values of each task are distributed independently.

Definition 7.1. *The* **Independent Action-Value (IAV) Assumption** *hypothesizes that during construction of each task in a domain the action-value for some state-action pair* $(s, a) \in S \times A$ *is generated independently of every other action-value.*

This is a useful assumption because it allows us to learn about the distribution used to generate the action-value at each state-action pair with every sampled task. If the action-values were dependent in an unknown way, then we could only learn about a single state-action pair from each sampled task. The main limitation with the IAV assumption is that it is restrictive and unrealistic, since action-values depend on each other. Alternatively, we can make a slightly less restrictive assumption.

Definition 7.2. *The* **Independent State (IS) Assumption** *hypothesizes that during construction of each task in a domain the set of* α*-good actions at each state are chosen independently of the set of* α*-good actions at every other state.*

This assumption is more flexible than the IAV assumption, because every distribution that satisfies the IAV assumption also satisfies the IS assumption. Furthermore, the IS assumption allows for more flexible ways of generating tasks. For example,

consider a state s with two actions a and b, in one task Ω the value $Q^*_\Omega(s,a) = 1$ and $Q^*_\Omega(s,b) = 0.9$ while in another task Ω' the value $Q^*_{\Omega'}(s,a) = 0.2$ and $Q^*_{\Omega'}(s,b) = 0.05$. Despite the fact that the values are quite different in the two tasks, the action a is optimal in both. This situation can be accounted for by the IS assumption but is difficult to account for with the IAV assumption. In our analysis, we will make the IS assumption about the domain.

Wilson et al. [60] and Lazaric and Ghavamzadeh [61] consider MTRL from a Bayesian perspective and attempt to identify hierarchies of tasks based on task similarities. Modeling Bayesian hierarchies of tasks is potentially useful but beyond the scope of this section. Wilson et al. [60] considers tasks to be related if they share similar reward functions and transition probabilities. Lazaric and Ghavamzadeh [61], on the other hand, assume that two tasks are related if they have similar action-values. Our approach measures the similarity between tasks by the number of α-good actions that two tasks share in common. This has the advantage that two tasks can have arbitrarily different transition probabilities and rewards but, in some cases, share the same α-good actions.

To the authors knowledge, no prior work, has analyzed the sample complexity of exploration with respect to a domain of tasks. In this section, we extend the concept of sample complexity to the MTRL setting and propose and analyze algorithms for learning domain specific RL algorithms.

7.2 Learning Objective and Approach

To analyze learning algorithms for MTRL, we need to define a learning objective. The single task RL objectives used by [27, 30, 6] cannot be straightforwardly applied to MTRL because, like the TL setting, in MTRL the agent interacts with multiple tasks. In MTRL, we are interested in learning a domain specific RL algorithm. Thus

we need a notion of sample complexity that takes the domain into consideration.

Definition 7.3. *Let $\epsilon > 0$, $\delta \in (0, 1]$, and $\mathcal{D}$ is a probability distribution over a set of MDPs D. If Ω is drawn from $\mathcal{D}$ and an instance of an RL algorithm $\mathcal{A}$ is executed on Ω, then the* **domain sample complexity of exploration** *(or DSCE) is the number of timesteps $t \geq 1$ such that*

$$V_{\Omega}^{\mathcal{A}_t}(s_t) < V_{\Omega}^{*}(s_t) - \epsilon$$

with probability at least $1 - \delta$.

Definition 7.3 defines sample complexity of exploration with respect to a probability distribution over a set of MDPs rather than the entire set of MDPs with N states and K actions. It is important to notice that the algorithm $\mathcal{A}$ may perform very poorly on some tasks in D, if their probability of being drawn from the domain is sufficiently small. This small change in the definition of sample complexity allows us to examine the sample complexity of an RL algorithm in a particular domain rather than the set of all MDPs. For many domains, existing lower bounds on sample complexity no longer apply because the tricky MDPs used to prove these lower bounds are either not in the set of tasks D or have an extremely small probability associated with them (and can effectively be ignored). Thus there is a potential to develop domain specific RL algorithms that have lower sample complexity of exploration with respect to the domain $\mathcal{D}$ than is possible to develop over the set of all MDPs.

In the MTRL setting, our algorithm initially starts with a full exploration table (Figure 5.1a). After sampling some tasks used for training, the multitask algorithm determines which state-action pairs are safe to prune resulting in a more sparse domain specific exploration table (Figure 5.1b). For very sparse exploration tables,

the sample complexity of the resulting PAC-MDP algorithm may be much smaller than a general purpose PAC-MDP algorithm. However, if the exploration table is too sparse, this may result in optimality loss (Definition 5.5). Therefore we are faced with a trade-off. We would like to learn as sparse an exploration table as possible, while at the same time, not pruning state-action pairs that are likely to prevent a PAC-MDP algorithm from learning to act ϵ-optimally on the next task sampled from the domain $\mathcal{D}$.

7.3 Complexity of a Domain of Tasks

In supervised learning the complexity of the hypothesis class plays an important role in finite sample analysis. More complex classes require more samples to achieve low error. However, the advantage of more complex classes is that they can accurately describe more problem settings. The same general principle is true for MTRL. Some multitask domains are simpler than others. In the extreme case, a domain with a single task is very simple because only one policy needs to be learned, whereas some domains may contain an infinite number of tasks that require radically different policies. To create a meaningful finite sample analysis of MTRL we need to develop a notion of the complexity of a domain of tasks.

One notion of complexity that is already included in RL sample complexity bounds is the number of states N and the number of actions K. As N and K increase the sample complexity grows at least linearly. However, there is a wide range of tasks that have N states and K actions. Some collections of tasks may be very simple to learn a good policy for even though they have a large number of states and actions. For example, there is always the degenerate MDP where every action in every state provides a maximum reward. Nevertheless, we believe that the number of state-action pairs that need to be explored is a critical measure of the

Table 7.1: Domain Complexity

Deterministic		Stochastic	
$\mathbb{C}_1(\mathcal{D})$	# of tasks in D times the number of states	$\mathbb{C}_4(\mathcal{D}, \omega)$	# of tasks in D with probability mass $1 - \omega$ times the number of states
$\mathbb{C}_2(\mathcal{D}, \alpha)$	Size of the union of α-"good" state-action pairs	$\mathbb{C}_5(\mathcal{D}, \alpha, \omega)$	Size of the union of α-"good" state-action pairs over tasks with probability mass $1 - \omega$
$\mathbb{C}_3(\mathcal{D}, \alpha)$	# of α-"good" state-action pairs in the minimal hitting set of each state over all tasks	$\mathbb{C}_6(\mathcal{D}, \alpha, \omega)$	# of α-"good" state-action pairs in the minimal hitting set of each state over tasks with probability mass $1 - \omega$

complexity of a set of MDPs. As we develop measures of complexity, we will express complexity based on the number of state-action pairs that may need to be explored. Framing complexity in terms of a number of state-action pairs that may need to be explored allows us to directly compare different notions of complexity and provides a straightforward interpretation of a complexity measure's meaning.

These measures of complexity help to answer the question: Given a domain of tasks, how much better might a domain specific learning algorithm be compared to a base RL algorithm? If the complexity of a domain is small, then a domain specific RL algorithm can learn to act near-optimally on the next sampled task with high probability after exploring only a small number state-action pairs. If the complexity of a domain is denoted by $\mathbb{C}$ and we knew an exploration table $\xi_{\mathbb{C}}$ that selected actions appropriately, then the domain sample complexity of exploration for R-MAX initialized with $\xi_{\mathbb{C}}$ would be

$$\tilde{O}\left(\frac{\psi\mathbb{C}}{\epsilon^3(1-\gamma)^3}\right)$$

by Theorem 5.7, where $\tilde{O}$ suppresses log factors, ψ is the maximum out-degree over all state-action pairs, ϵ controls the acceptable sub-optimality of the learned policy. The corresponding domain sample complexity of exploration for Delayed Q-learning initialized with $\xi_{\mathbb{C}}$ would be

$$\tilde{O}\left(\frac{\mathbb{C}}{\epsilon^4(1-\gamma)^8}\right)$$

by Theorem 5.10. So learning a domain specific exploration table can significantly improve domain sample complexity of exploration.

We explore two kinds of measures of complexity: (1) deterministic, and (2) stochastic. Deterministic measures of complexity do not take into consideration any information about the probability distribution over tasks in the domain, while stochastic measures of complexity weight the influence of different tasks based on their probability to achieve a tighter fit of complexity. Table 7.1 provides an overview of the different measures of complexity considered throughout this section.

7.3.1 Deterministic Measures of Domain Complexity

Possibly the simplest measure of complexity is related to the number of tasks in a domain $|D|$, where D is the set of tasks in a domain. Intuitively, if there are only a few tasks in a domain, then at most a few policies need to be considered. If there are fewer tasks in the domain than the number of actions, then there cannot be more than

$$\mathbb{C}_1(\mathcal{D}) = N \min\left(|D|, K\right) \tag{7.1}$$

state-action pairs that are ever optimal across all of the tasks. This is because, in the worst case, each task may have a different optimal action at each of the N states.

Although $\mathbb{C}_1$ is a reasonable first attempt at measuring the complexity of domains, it is possible to define a more flexible measure of domain complexity. One way

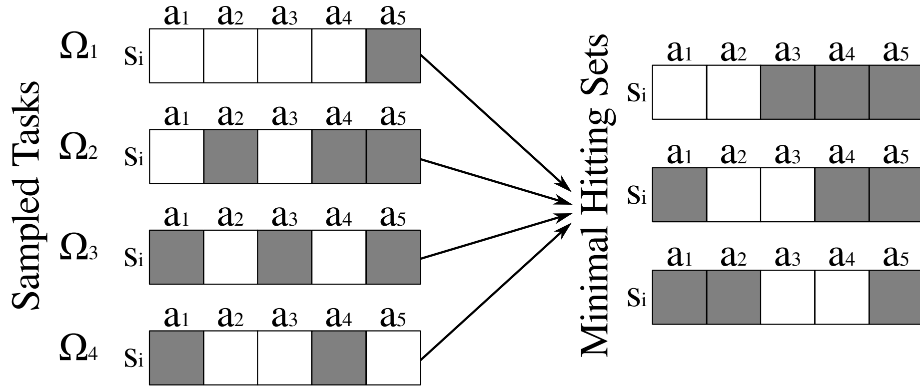


Figure 7.2: Depicts the minimal hitting set problem for a state s_i over four MDPs. White cells depict α-good actions, while gray depicts actions that are not in $G^{\alpha}_{\Omega_j}(s_i)$ for $j = 1, 2, 3, 4$. Notice that there are multiple minimal hitting sets.

of improving on $\mathbb{C}_1$ is to consider the properties of tasks within the domain. For example, at a state $s \in S$, each task Ω in the domain has a set of α-"good" actions $G^{\alpha}_{\Omega}(s)$. If the union of these sets is smaller than the total number of actions, then the actions that are never α-"good" can be pruned without any optimality loss. This leads to our next measure of domain complexity defined by

$$\mathbb{C}_2(\mathcal{D}, \alpha) = \sum_{s \in S} |\cup_{\Omega \in D} G^{\alpha}_{\Omega}(s)| \tag{7.2}$$

where $\alpha \geq 0$ determines how suboptimal the "good" actions are allowed to be (see Definition 5.16).

However, the complexity measure Eq. (7.2) may, in some cases, be too conservative. To understand why consider a collection of two MBPs with 6 actions. In Ω_1 the actions $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ are all optimal (all give the same expected reward) and in Ω_2 only the action a_1 is optimal. In this case, the best response would be to always select action a_1 regardless of which task is selected because a_1 is always optimal. However, Eq. (7.2) labels this simple domain with maximal complexity.

To avoid this problem, we can sum up the size of minimal hitting sets (Figure 7.2)

instead of the size of the union

$$\mathbb{C}_3(\mathcal{D}, \alpha) = \sum_{s \in S} |H(s, D, \alpha)| \tag{7.3}$$

where $H(s, D, \alpha)$ finds a minimum hitting set over all of the sets of "good" actions for state s over all tasks in D.

Definition 7.4. *(Reiter [62]) Given a set $A = \{1, 2, \ldots, K\}$ called the universe and a collection of subsets $B = \{\eta_1, \eta_2, \ldots, \eta_\tau\}$ such that $\eta_i \subseteq A$ for all $i = 1, 2, \ldots, \tau$, a* **hitting set** *H is a subset of A and $H \cap \eta_i \neq \emptyset$ for all $i = 1, 2, \ldots, \tau$. For all hitting sets H a* **minimal hitting set** *H^* has the additional property that $|H^*| \leq |H|$.*

Notice that under Eq. (7.6) the complexity of a set containing Ω_1 and Ω_2 is 1, which is as small as possible. This is because the minimal hitting set picks out the one action a_1 from both tasks instead of selecting the union.

Algorithm 11 Find a Minimal Hitting Set

```
Require: A, B = {η_1, η_2, ..., η_τ}
1: K ← |A|
2: for k = 1, 2, ..., K do
3:    for H ∈ enum(A, k) do {For all sets of size k}
4:       if H is a hitting set wrt B (Algorithm 12) then
5:          return H
6:       end if
7:    end for
8: end for
```

Algorithm 12 takes a set A called the universe, a collection of subsets $B = \{\eta_1, \eta_2, \ldots, \eta_\tau\}$, and $H \subseteq A$. This algorithm determines whether or not H is a hitting set with respect to A and B. Algorithm 11 enumerates all possible subsets of A and returns a minimal hitting set. Unfortunately, finding minimal hitting sets is known to be NP-complete as it is reducible to the vertex set cover problem ([63]

Algorithm 12 Is H a Hitting Set?

Require: $A, B = \{\eta_1, \eta_2, \ldots, \eta_\tau\}, H$

```
1: for i = 1, 2, ..., τ do
2:     if H ∩ η_i = ∅ then
3:         return false
4:     end if
5: end for
6: return true
```

citing [64]) and therefore there is no known polynomial time solution. However, when the number of actions is small, Algorithm 11 is computationally feasible. Even when the number of actions is large there exist approximate algorithms that can be used in practice to find nearly minimal hitting sets [63].

Next we will introduce several measures of complexity that take into account the probability mass assigned to different tasks in the domain.

7.3.2 Stochastic Measures of Domain Complexity

Stochastic measures of domain complexity enable further improvement over deterministic measures by taking the probability mass assigned to each task in the domain into consideration. This can affect the complexity of a task compared to deterministic measures if some tasks have extremely small probability of being drawn from the domain. For example, if $\mathcal{D}$ is a domain with 500 tasks each having one state and 500 actions (i.e., $N = 1$ and $K = 500$), then the deterministic complexity measure $\mathbb{C}_1(\mathcal{D}) = 500$. However, it may be the case that 300 of the tasks in $\mathcal{D}$ have a combined probability mass of 0.00001. In this case, it seems reasonable to judge $\mathcal{D}$'s complexity based on the 200 tasks that are much more likely to occur. This suggests our first stochastic measure of domain complexity

$$\mathbb{C}_4(\mathcal{D}, \omega) = N \min\left(\left|\mathcal{X}(\omega)\right|, K\right) \tag{7.4}$$

where $\mathcal{X}(\omega)$ is the smallest subset of D such that $\sum_{\Omega \in \mathcal{X}} \Pr_{\mathcal{D}}[\Omega] \geq 1 - \omega$. The parameter $\omega \in (0, 1)$ allows control over the amount of probability mass to that can be ignored. The main difference between $\mathbb{C}_1$ and $\mathbb{C}_4$ is that $\mathbb{C}_4$ does not count tasks with very small probability of being drawn. When ω is selected appropriately $\mathbb{C}_4$ can be a more accurate measure of a domain's complexity because it ignores tasks that rarely occur.

The main problem with $\mathbb{C}_4$ is that the number of tasks is not very descriptive of a domains complexity. Our second stochastic measure of domain complexity is based on $\mathbb{C}_2$. We define

$$\mathbb{C}_5(\mathcal{D}, \alpha, \omega) = \sum_{s \in S} \left| \cup_{\Omega \in \mathcal{X}(\omega)} G_{\Omega}^{\alpha}(s) \right| \tag{7.5}$$

where α determines which actions are considered "good" and ω determines the amount of probability mass we are willing to ignore. Again $\mathcal{X}(\omega)$ is the smallest subset of D such that $\sum_{\Omega \in \mathcal{X}} \Pr_{\mathcal{D}}[\Omega] \geq 1 - \omega$.

As with the deterministic measures of complexity, we can push the stochastic measures of complexity even further by considering the minimal hitting set concept. We define this measure of domain complexity by

$$\mathbb{C}_6(\mathcal{D}, \alpha) = \sum_{s \in S} |H(s, \mathcal{X}(\omega), \alpha)| \tag{7.6}$$

where $H(s, \mathcal{X}, \alpha)$ finds a minimum hitting set over all of the sets of "good" actions for state s over all tasks in $\mathcal{X}(\omega)$.

These stochastic measures of complexity add additional probability of failure to the domain specific RL algorithm. In a typical, sample complexity bound we select $\delta \in (0, 1]$ and produce algorithms that succeed with a probability of at least $1 - \delta$. Given an exploration table ξ tuned for a stochastic measure of complexity, the

additional probability of failure introduced is at most ω. So the total probability of failure is bound by $(\delta + \omega)$.

So far, we have introduced six measures of domain complexity. If we are given the task set D and the probability mass for each task in D, then we can compute exploration tables for a domain that exactly matches their complexity. However, we are more interested in the case where we do not know the tasks probabilities of each task or even which tasks are in D. Next, we will present two different approaches for learning exploration tables under these conditions.

7.4 Algorithm: Evolving Exploration Tables (EET)

One method for MTRL is to evolve a structure that encodes which state-action pairs to explore and those state-action pairs that can be ignored without jeopardizing optimality guarantees. Algorithm 13 outlines the pseudo-code for Evolving Exploration Tables (EET). This algorithm is a simple genetic algorithm that searches through the space of exploration tables to find tables that when paired with a PAC-MDP RL algorithm achieve high cumulative reward on the next n tasks sampled from $\mathcal{D}$.

The genome of EET is an array of $\{0, 1\}^{NK}$, which encodes a 0 or a 1 for each state-action pair. However, in tasks with large state spaces, we have found that partitioning table so that a set of K entries in the genome control the resulting exploration table entries for multiple states.

7.5 Algorithm: Learning Maximum Values (LMV)

The main disadvantage of the EET algorithm is that evolution is extremely slow. More precisely, the number of tasks that need to be sampled to evaluate even one generation of exploration tables seems large. This problem increases dramatically as the complexity of the genome increases (i.e., the number of states and actions

Algorithm 13 Evolving Exploration Tables (EET)

1: Generate a random population of exploration tables P
2: **while** Termination criteria hasn't been achieved **do**
3: **for** $p \in P$ **do**
4: $F(p) \leftarrow 0$
5: **for** $i = 1, 2, ..., n$ **do**
6: $\Omega_i \sim \mathcal{D}$
7: Record sum of cumulative reward c for p wrt Ω_i
8: $F(p) \leftarrow F(p) + c$ {Update fitness of p}
9: **end for**
10: **end for**
11: Generate new generation P' using mutation and crossover
12: $P \leftarrow P'$
13: **end while**

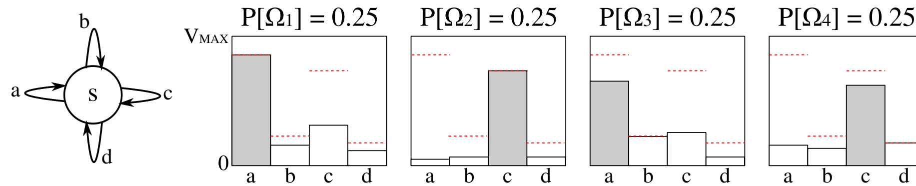


Figure 7.3: A one-state domain with four actions and the optimal action-values for each task. The red dashes indicate the maximum action-values across all tasks. Notice that maximum action-values implicitly eliminate actions b and d from each task because either action a or c has a higher value than the maximum value of b and d.

increase).

Alternatively, we could attempt to learn a weak admissible heuristic, like we did in the previous section, to decrease the sample complexity of exploration in the next task. Learning average action-values, as is done by Algorithm 10, does not ensure that the learned action-values are a weak admissible heuristic for the next sampled task. One way to ensure that a set of action-values are a weak admissible heuristic is to ensure that every action-value is greater than the true optimal action-value for every task. This leads to the Learning Maximum Values (LMV) algorithm outlined in Algorithm 14.

Algorithm 14 Learning Maximum Values (LMV)

Require: S, A, and τ

1: **for** $(s, a) \in S \times A$ **do** {Initialize}
2: $\quad W(s, a) \leftarrow 0$ {Max. observed action-value for (s, a).}
3: **end for**
4: **for** $n = 1, 2, \ldots$ **do**
5: $\quad \Omega_n \sim \mathcal{D}$ {Sample a new task.}
6: $\quad$ **if** $n \leq \tau$ **then** {Training Phase}
7: $\quad\quad$ Learn accurate action-values $\hat{Q}^*$
8: $\quad\quad$ **for** $(s, a) \in S \times A$ **do** {Update Library}
9: $\quad\quad\quad W(s, a) \leftarrow \max\left(W(s, a), \hat{Q}^*(s, a)\right)$
10: $\quad\quad$ **end for**
11: $\quad$ **else** {Domain Specific Learning}
12: $\quad\quad$ Initialize a compatible PACMDP algorithm $\mathcal{A}$ with action-values W
13: $\quad\quad$ Execute $\mathcal{A}$ on Ω_n
14: $\quad$ **end if**
15: **end for**

The LMV algorithm has a training phase and a domain specific learning phase. During the training phase, LMV samples $\tau \geq 1$ tasks from a domain $\mathcal{D}$ and records the maximum observed action-value for each state-action pair (s, a). During the domain specific phase, the LVM algorithm initializes a compatible PACMDP algorithm (such as R-MAX or Delayed Q-learning) using the action-values learned during the training phase. If the learned action-values are all $\frac{1}{1-\gamma}$, then the sample complexity of exploration is no better than the base RL algorithm. However, if some of the action-values are consistently smaller than others, then their corresponding state-action pairs can be eliminated. For example, Figure 7.3 shows a simple one-state domain with four actions where using the maximum action-values implicitly eliminates two of the four actions (b and d).

The main problem with LMV is that it can easily be tricked resulting in better performance with only a small probability. For example, consider the domain speci-

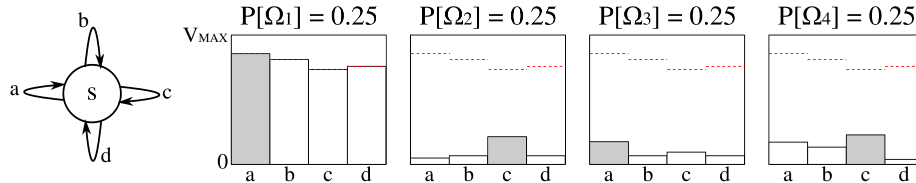


Figure 7.4: A one-state domain with four actions and the optimal action-values for each task. The red dashes indicate the maximum action-values across all tasks. Task Ω_1 has probability mass 0.25 and much higher action-values than the action-values for every other task. The learned maximum action-values are only helpful in task Ω_1. No actions can be pruned in tasks Ω_2, Ω_3, or Ω_4, even though only actions a and c are ever optimal.

fied by Figure 7.4. In this domain, one task Ω_1 has much higher action-values than all of the other tasks. These maximum action-values do not allow any actions to be pruned from the other tasks, even though the complexity $\mathbb{C}_2(\mathcal{D}, \alpha = 0) = 2$. In other words, even if we new the maximum action-values over all tasks, LMV may not improve performance in domains where $\mathbb{C}_2(\mathcal{D}, \alpha) \leq NK$. This is a serious problem for LVM because we would like to develop a learning algorithm that tightly fits the domains complexity according to $\mathbb{C}_2$ or $\mathbb{C}_3$.

7.6 Algorithm: Learning Exploration Tables (LET)

Instead of learning good initial action-values, we could attempt to learn an exploration table by sampling and analyzing a small number of tasks. The main idea is to record the α-good actions at each state for a collection of sampled tasks. If enough tasks are sampled, we can accurately determine, which state-action pairs can be ignored with only a small probability of incurring significant optimality loss.

Pseudo-code for the Learning Exploration Tables 1 (LET1) algorithm is shown in Algorithm 15. This algorithm samples and explores exhaustively τ tasks, which it uses to construct an exploration table. For all following tasks LET1 constructs a domain specific RL algorithm using the learned exploration table.

Algorithm 15 Learning Exploration Tables 1 (LET1)

Require: S, A, τ, and α

1: **for** $(s, a) \in S \times A$ **do** {Initialize}
2: $c(s, a) \leftarrow 0$ {Counter for important (s, a).}
3: **end for**
4: **for** $n = 1, 2, \ldots$ **do**
5: $\Omega_n \sim \mathcal{D}$ {Sample a new task.}
6: **if** $n \leq \tau$ **then** {Training Phase}
7: Learn accurate action-values $\hat{Q}^*$
8: **for** $(s, a) \in S \times A$ **do** {Update Library}
9: **if** $\hat{Q}^*(s, a) \geq \hat{V}^*_{\Omega_n}(s) - \alpha$ **then**
10: $c(s, a) \leftarrow c(s, a) + 1$
11: **end if**
12: **end for**
13: **else** {Domain Specific Learning}
14: **for** $(s, a) \in S \times A$ **do**
15: **if** $c(s, a) \geq 1$ **then**
16: $\xi(s, a) = 1$
17: **else**
18: $\xi(s, a) = 0$
19: **end if**
20: **end for**
21: Initialize a compatible PACMDP algorithm $\mathcal{A}$ with exploration table ξ
22: Execute $\mathcal{A}$ on Ω_n
23: **end if**
24: **end for**

The exploration table learned by LET1 is defined by

$$\xi(s,a) = \begin{cases} 1 & \text{if } c(s,a) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.7}$$

where $c(s,a)$ is the number of sampled tasks where $Q^*(s,a) > V^*(s) - \alpha$. In other words, LET1 attempts to learn the union of α-good actions in the domain at each state. This corresponds with learning the optimal exploration table for domain complexity measure $\mathbb{C}_2$. The following theorem specifies the number of tasks that need to be observed by LET1 to learn an exploration table, assuming that the optimal action-values of each sampled task during the training phase are given.

Theorem 7.5. *Let $\alpha \geq 0$, $\omega \in (0,1]$, and $\mathcal{D}$ be a domain of tasks. Assuming that we are given access to the optimal action-values Q^*_Ω for each task Ω that we sample from $\mathcal{D}$, there exists $\tau = O\left(\frac{NK}{\omega}\ln\left(\frac{N}{\omega}\right)\right)$ such that after sampling τ tasks from $\mathcal{D}$ with complexity $\mathbb{C}_2(\mathcal{D},\alpha)$, LET1 (Algorithm 15) produces an exploration table ξ that satisfies*

1. $\sum_{(s,a)\in S\times A} \xi(s,a) \leq \mathbb{C}_2(\mathcal{D},\alpha)$ *and*
2. $\mathcal{L}(\Omega, \Omega_\xi) \leq \frac{\alpha}{1-\gamma}$

with probability at least $1-\omega$.

Theorem 7.5 formalizes the number of tasks that need to be observed before an acceptable exploration table is learned with high probability. This theorem holds for the IS assumption, which is more general than the IAV assumption. The dependence is approximately linear with respect to the number of states and actions. This is somewhat disappointing because it will not be reasonable to sample such a large number of tasks when the state-action space is large. However, this bound makes

sense because we have assumed that each state is generated according to a separate distribution. Keep in mind that this is a worst-case upper bound on the number of tasks that need to be sampled. In many special cases the number of tasks needed may be much smaller without sacrificing optimality. The main significance here is that combined with Theorems 5.7 and 5.10, Theorem 7.5 provides a complete solution for analyzing the sample complexity of exploration on the next sampled task. This allows us to compare the sample complexity of domain specific RL algorithms with general RL algorithms, such as R-MAX and Delayed Q-learning without prior knowledge.

To prove this theorem it will be useful to refer to a well-known lemma.

Lemma 7.6. *(Strehl et al. [6, Lemma 8]) Let* $\delta \in (0,1]$ *and* $X_1, X_2, X_3, \ldots, X_\tau$ *be a sequence of Bernoulli random variables with* $\Pr[X_i = 1] \geq p$ *of a success for* $i = 1, 2, 3, \ldots, \tau$*, then there exists* $\tau = O\left(\frac{k}{p}\ln\left(\frac{1}{\delta}\right)\right)$ *such that after observing* τ *experiments, we will observe at least* k *successes with probability at least* $1 - \delta$*.*

Proof. (of Theorem 7.5) The first claim is true due to the fact that LET1 only assigns $\xi(s,a) = 1$ if the algorithm has observed some task from the domain where (s,a) is α-good and $\mathbb{C}_2(\mathcal{D}, \alpha)$ is the sum of all state-action pairs that are α-good in any task in $\mathcal{D}$.

Now we will argue that the second claim is also true. If ξ contains at least one α-good action per state, then by Lemma 5.6 the optimality loss $\mathcal{L}(\Omega, \Omega_\xi) \leq \frac{\alpha}{1-\gamma}$. We want to show that for some $\tau = O\left(\frac{NK}{\omega}\ln\left(\frac{N}{\delta}\right)\right)$, the exploration table learned by LET1 includes at least one α-good action at every state of the next sampled task Ω with probability at least $1 - \omega$.

Because the domain is stochastic, we cannot expect LET1 to observe a collection of tasks during the training phase that reveals every state-action pair that is α-good in some task in the domain. Instead, our strategy is to select τ large enough so that

LET1 samples enough tasks to guarantee that at every state $s \in S$ all actions that are α-good with probability at least ω_1 are observed, with probability at least $1-\omega_2$. This strategy suggests two distinct ways that LET1 can fail:

- Failure Event A: the probability that (s, a) is α-good is greater than or equal to ω_1 but LET1 does not observe any task during the training phase where (s, a) is α-good, or
- Failure Event B: the probability that (s, a) is α-good is less than ω_1 but (s, a) is α-good in the next sampled task during the domain specific phase.

The value ω_1 represents the probability of failure event B for a single state s, while the value ω_2 represents the probability of failure event A for a single state s. If failure event A and failure event B do not occur at a state $s \in S$, then $\xi(s, a) = 1$ for some α-good state-action pair (s, a). Furthermore if failure event A and failure event B do not occur at any state, then $\mathcal{L}(\Omega, \Omega_\xi) \leq \frac{\alpha}{1-\gamma}$ where Ω is the next task sampled from the domain during the domain specific phase.

Since there are K actions, for a single state $s \in S$, after sampling $\tau = O\left(\frac{K}{\omega_1} \ln\left(\frac{1}{\omega_2}\right)\right)$ tasks from the domain we will observe every action a at s where the probability of being α-good is greater than or equal to ω_1, with probability at least $1 - \omega_2$, by Lemma 7.6. If we select $\omega_2 \leftarrow \frac{\omega}{2N}$, then the probability of failure event A for the state s is at most $\frac{\omega}{2N}$. By the union bound (Theorem 2.2) over all N states the probability of failure event A is at most $N\omega_2 = N\frac{\omega}{2N} = \frac{\omega}{2}$.

Now, if we set the probability of failure event B to $\omega_1 \leftarrow \frac{\omega}{2N}$ for each state, then the probability of failure event B over all N states is at most $N\omega_1 = N\frac{\omega}{2N} = \frac{\omega}{2}$. Therefore the total probability that LET1 will fail is at most $N(\omega_1+\omega_2) = \frac{\omega}{2}+\frac{\omega}{2} = \omega$. By plugging in our values for ω_1 and ω_2 into τ we obtain our result. □

The main limitation of LET1 is that it takes the union over the sets of α-good actions at each state. This results in learning an exploration table that approximately matches $\mathbb{C}_2(\mathcal{D}, \alpha)$, which in some cases is larger than $\mathbb{C}_3(\mathcal{D}, \alpha)$. To learn an exploration that approximately matches the tighter domain complexity of $\mathbb{C}_3(\mathcal{D}, \alpha)$ we need to select a minimal hitting set over the sets of α-good actions at each state. The algorithm LET2 (Algorithm 16) does precisely that.

Similar to the LET1 algorithm, LET2 has two phases: (1) a training phase, and (2) a domain specific phase. For the first τ sampled tasks, LET2 updates its library of state-action pairs that have been observed to be α-good. For each state, LET2 builds up a collection of α-good action sets. At the end of the training phase, minimal hitting sets are found for each state and only the actions in the minimal hitting set are included in the exploration table. During the domain specific phase a compatible PACMDP algorithm is initialized with the learned exploration table and executed on the next sampled task. The only real difference between LET1 and LET2 is that LET2 eliminates state-action pairs from the learned exploration table more aggressively. However, this raises the question: does LET2 need to sample more tasks than LET1 to learn an acceptable exploration table? The following theorem suggests that LET1 and LET2 need to sample approximately the same number of tasks.

Theorem 7.7. *Let $\alpha \geq 0$, $\omega \in (0, 1]$, and $\mathcal{D}$ be a domain of tasks. Assuming that we are given access to the optimal action-values Q^*_Ω for each task Ω that we sample from $\mathcal{D}$, there exists $\tau = O\left(\frac{NK}{\omega} \ln\left(\frac{N}{\omega}\right)\right)$ such that after sampling τ tasks from $\mathcal{D}$ with complexity $\mathbb{C}_3(\mathcal{D}, \alpha)$, LET2 (Algorithm 16) produces an exploration table ξ that satisfies*

1. *$\sum_{(s,a) \in S \times A} \xi(s, a) \leq \mathbb{C}_3(\mathcal{D}, \alpha)$ and*

Algorithm 16 Learning Exploration Tables 2 (LET2)

Require: S, A, τ, and α

1: **for** $s \in S$ **do** {Initialize}
2:     $B(s) \leftarrow \emptyset$
3:     **for** $a \in A$ **do**
4:         $\xi(s, a) = 0$
5:     **end for**
6: **end for**
7: **for** $n = 1, 2, \ldots$ **do**
8:     $\Omega_n \sim \mathcal{D}$ {Sample a new task.}
9:     **if** $n \leq \tau$ **then** {Training Phase}
10:         Learn accurate action-values $\hat{Q}^*$
11:         **for** $s \in S$ **do** {Update Library}
12:             $G \leftarrow \emptyset$
13:             **for** $a \in A$ **do**
14:                 **if** $\hat{Q}^*(s, a) \geq \hat{V}^*_{\Omega_n}(s) - \alpha$ **then**
15:                     $G \leftarrow G \cup \{a\}$ {Add α-good actions to G.}
16:                 **end if**
17:             **end for**
18:             Append the set G to $B(s)$
19:         **end for**
20:         **if** $n = \tau$ **then** {End of Training Phase}
21:             **for** $s \in S$ **do**
22:                 $h_s \leftarrow H(s, B(s))$ {Compute a minimal hitting set.}
23:                 **for** $a \in h_s$ **do**
24:                     $\xi(s, a) = 1$
25:                 **end for**
26:             **end for**
27:         **end if**
28:     **else** {Domain Specific Learning}
29:         Initialize a compatible PACMDP algorithm $\mathcal{A}$ with exploration table ξ
30:         Execute $\mathcal{A}$ on Ω_n
31:     **end if**
32: **end for**

2. $\mathcal{L}(\Omega, \Omega_\xi) \leq \frac{\alpha}{1-\gamma}$

with probability at least $1 - \omega$*.*

Proof. The first claim is true due to the fact that LET2 selects a minimal hitting set of α-good actions at each state for a subset of the total task space.

By an argument similar to the one used to prove claim 2 of Theorem 7.5, if $\tau = O\left(\frac{NK}{\omega} \ln\left(\frac{N}{\omega}\right)\right)$, we have for each state a collection of sets of α-good actions with probability mass greater $1 - \omega/2$, with probability at least $1 - \omega/2$ after the training phase is complete. A minimal hitting set can only eliminate an action if that action always occurs in a set with some other α-good action that is included in the hitting set. Thus the probability mass of the hitting sets is still greater than $1-\omega/2$. By the union bound (Theorem 2.2), claim 2 occurs with probability at least $1 - (\omega/2 + \omega/2) = 1 - \omega$. □

The significance of Theorem 7.7 is that it shows that LET2 can learn an acceptable exploration table with approximately the same number of samples as LET1, even though the exploration table learned by LET2 is potentially sparser than the exploration table learned by LET1. The trade-off is that LET2 employs the computationally expensive step of finding a minimal hitting set for each state at the end of the training phase. If the number of actions K is small, then this may not be much of a penalty. However, for domains with large action spaces, LET2 may be computationally intractable.

In practice, LET1 and LET2 are practically equivalent when $\alpha = 0$ because the set of α-good actions will typically only contain a single action. Therefore, in our experiments, we will assume $\alpha = 0$ and simply refer to LET rather than LET1 and LET2.

Table 7.2: $\mathcal{D}_1$: Multiarmed Bandit Domain with Four Actions.

	Ω_1	Ω_2	Ω_3	Ω_4
Task Probability	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$E[R(a_1)]$	0.9	0.9	0.9	0.9
$E[R(a_2)]$	0.1	0.4	0.6	0.7
$E[R(a_3)]$	0.1	0.4	0.6	0.4
$E[R(a_4)]$	0.1	0.4	0.6	0.1

7.7 Experiments & Results

In this section, most of our experiments will be with multiarmed bandit domains. This is because many of the important aspects of MTRL can be highlighted without some of the additional complexities of exploring an MDP with many states. We created six of multiarmed bandit domains that highlight different issues in MTRL.

Our first domain $\mathcal{D}_1$ (Table 7.2) contains a collection of four equally probable tasks. Each task has four actions. In $\mathcal{D}_1$ the action a_1 has the highest expected reward in every task in the domain. The purpose of $\mathcal{D}$ is to determine whether learning algorithms are able to learn exploration tables that eliminate all but action a_1.

Domain $\mathcal{D}_2$ (Table 7.3) contains a collection of four equally probable tasks. Each task has four actions. In $\mathcal{D}_2$ the action a_1 is optimal in half of the tasks while the action a_2 is optimal in the other tasks. The purpose of $\mathcal{D}_2$ is to determine whether learning algorithms are able to learn exploration tables that eliminate actions a_3 and a_4 but keep the optimal actions a_1 and a_2. Domain $\mathcal{D}_3$ (Table 7.4) is similar to $\mathcal{D}_2$ except that actions $\{a_1, a_2, a_3\}$ are optimal in different tasks in the domain.

Domain $\mathcal{D}_4$ (Table 7.5) contains four equally probable tasks. However, each task has a different optimal action. The purpose of $\mathcal{D}_4$ is to determine whether learning algorithms eliminate any actions in the exploration table in this case. When a new task is sampled the algorithm must explore every action to determine, which action

Table 7.3: $\mathcal{D}_2$: Multiarmed Bandit Domain with Four Actions.

	Ω_1	Ω_2	Ω_3	Ω_4
Task Probability	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$E[R(a_1)]$	0.9	0.9	0.6	0.7
$E[R(a_2)]$	0.1	0.4	0.9	0.9
$E[R(a_3)]$	0.1	0.4	0.6	0.4
$E[R(a_4)]$	0.1	0.4	0.6	0.1

Table 7.4: $\mathcal{D}_3$: Multiarmed Bandit Domain with Four Actions.

	Ω_1	Ω_2	Ω_3	Ω_4
Task Probability	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$E[R(a_1)]$	0.9	0.9	0.6	0.7
$E[R(a_2)]$	0.1	0.4	0.9	0.4
$E[R(a_3)]$	0.1	0.4	0.6	0.9
$E[R(a_4)]$	0.1	0.4	0.6	0.1

gives the highest expected reward. A reasonable domain specific learning algorithm should recognize that there is no better domain specific RL algorithm in a domain like $\mathcal{D}_4$.

Domain $\mathcal{D}_5$ (Table 7.6) contains four equally probable tasks. In Ω_3 every action is optimal, but only actions a_1 or a_2 are optimal in all the other tasks. The minimal hitting set notion is useful in this domain because a domain specific learning algorithm only needs to explore actions a_1 and a_2 to find an optimal policy. Taking the union of actions that are optimal in some task results in a set containing all four actions.

Domain $\mathcal{D}_6$ (Table 7.7) contains seven tasks. The first four tasks divide up most

Table 7.5: $\mathcal{D}_4$: Multiarmed Bandit Domain with Four Actions.

	Ω_1	Ω_2	Ω_3	Ω_4
Task Probability	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$E[R(a_1)]$	0.9	0.4	0.6	0.7
$E[R(a_2)]$	0.1	0.9	0.6	0.4
$E[R(a_3)]$	0.1	0.4	0.9	0.1
$E[R(a_4)]$	0.1	0.4	0.6	0.9

Table 7.6: $\mathcal{D}_5$: Multiarmed Bandit Domain with Four Actions. In Task Ω_3 All Actions are Optimal.

	Ω_1	Ω_2	Ω_3	Ω_4
Task Probability	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
$E[R(a_1)]$	0.9	0.4	0.9	0.9
$E[R(a_2)]$	0.1	0.9	0.9	0.7
$E[R(a_3)]$	0.1	0.4	0.9	0.4
$E[R(a_4)]$	0.1	0.4	0.9	0.1

Table 7.7: $\mathcal{D}_6$: Multiarmed Bandit Domain with Four Actions.

	Ω_1	Ω_2	Ω_3	Ω_4	Ω_5	Ω_6	Ω_7
Task Probability	$\frac{25}{100}$	$\frac{25}{100}$	$\frac{25}{100}$	$\frac{22}{100}$	$\frac{1}{100}$	$\frac{1}{100}$	$\frac{1}{100}$
$E[R(a_1)]$	0.9	0.4	0.9	0.9	0.0	0.1	0.1
$E[R(a_2)]$	0.1	0.9	0.9	0.7	0.3	0.9	0.1
$E[R(a_3)]$	0.1	0.4	0.9	0.4	0.9	0.1	0.1
$E[R(a_4)]$	0.1	0.4	0.9	0.1	0.1	0.1	0.9

of the probability mass, while the last three tasks each have probability $\frac{1}{100}$. This domain explores what happens when a few tasks have most of the probability mass. Notice that in the first four tasks either action a_1 is optimal or action a_2, while in the remaining three tasks one of a_2, a_3, or a_4 are optimal. The probability that action a_3 or a_4 is only $\frac{2}{100}$, so if the learned exploration table did not contain those actions, the domain specific RL algorithm would still be able to find an optimal policy almost all the time.

Finally, we introduce four domains over Markov decision processes called reset domains. In each of the reset domains all of the tasks have a similar structure (Figure 7.5) as a chain of states. The only way that the tasks differ is which actions propel the agent further up the chain of states, and which actions reset the agent to state 1. In reset domain $\mathcal{R}_1$, the action a_1 is always the optimal action in every task. In reset domain $\mathcal{R}_2$, either action a_1 or action a_2 (but not both at the same time) is the optimal action in sampled tasks. In reset domain $\mathcal{R}_3$, either action a_1, a_2, or a_3 (but only one at a time) is the optimal action in sampled tasks. In reset domain

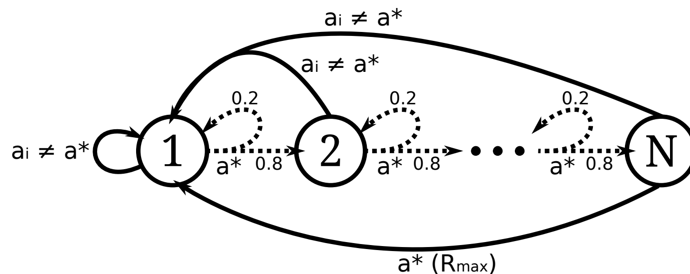


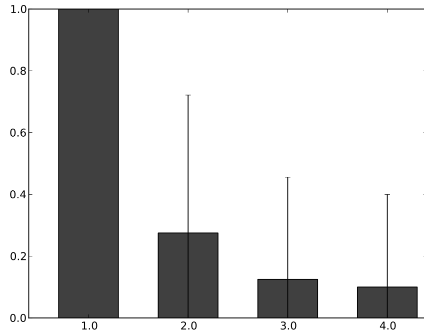
Figure 7.5: Example of one task sampled from a reset domain.

$\mathcal{R}_4$, any action (but only one at a time) is the optimal action in sampled tasks. The significance of the reset domains is that each task is a difficult RL problem, but the results of learning are easy to understand because the same action is optimal in every state (although the RL algorithm does not know that).

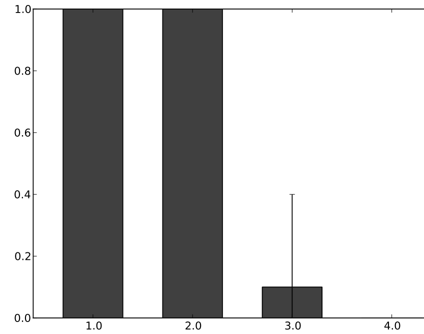
7.7.1 Experiment: Evolving an Exploration Table

We ran EET on several domains to determine whether or not EET learns an appropriately domain specific exploration table. Figure 7.6 shows the average of the best genomes evolved by EET for domains 1 through 6. In the first domain $\mathcal{D}_1$, we can see that most of the genomes learned that the first action a_1 is optimal and that the other actions are never optimal. The results for $\mathcal{D}_2$ show similarly that EET was able to determine that actions a_1 and a_2 are worth exploring, while actions a_3 and a_4 are not. In domains $\mathcal{D}_3$ and $\mathcal{D}_4$, the results are less impressive. This is most likely because the evaluation window was too short to make it worth while to explore three or four actions before settling on an optimal action. However, the average of the best genomes match the number of potentially optimal actions in domains 1 through 4.

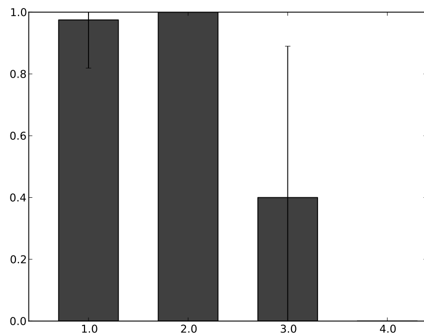
The results for $\mathcal{D}_5$ (Figure 7.6e) show that despite the fact that every action is optimal in Ω_3 the best exploration table only explored the first two actions. This makes sense because one of the first two actions is always optimal in every task in the domain.



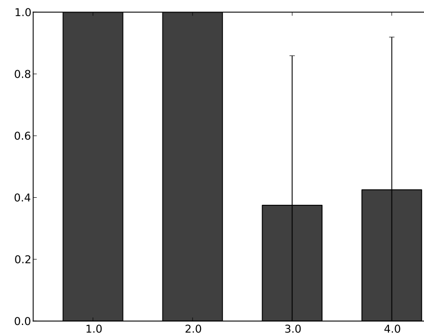
(a) $\mathcal{D}_1$



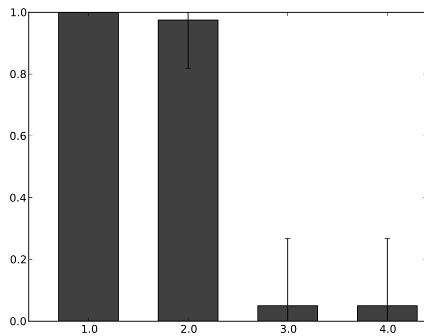
(b) $\mathcal{D}_2$



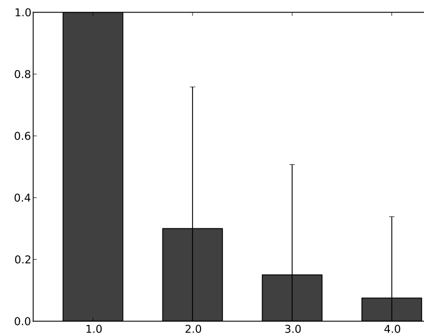
(c) $\mathcal{D}_3$



(d) $\mathcal{D}_4$



(e) $\mathcal{D}_5$



(f) $\mathcal{D}_6$

Figure 7.6: Average of best genomes (with error bars indicating ±1 standard deviation) learned over 40 runs EET on domains (a) $\mathcal{D}_1$, (b) $\mathcal{D}_2$, (c) $\mathcal{D}_3$, (d) $\mathcal{D}_4$, (e) $\mathcal{D}_5$, and (f) $\mathcal{D}_6$.

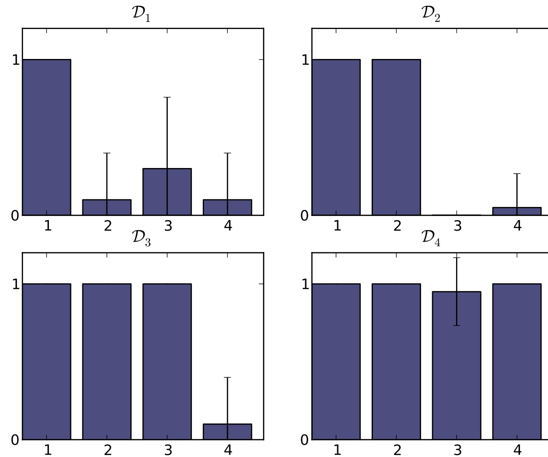


Figure 7.7: Average learned exploration tables (with error bars indicating ±1 standard deviation) by the LET algorithm on domains $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$. The learned exploration tables accurately determine the minimum number of actions that can be explored depending on the domain.

In $\mathcal{D}_6$ (Figure 7.6f) the action a_1 was always included in the evolved exploration table because it had a 72% chance of being optimal. Action a_2 was included surprisingly infrequently. However, this may have been due to evaluating genomes on only 10 sampled tasks.

What we found from these experiments is that EET can learn domain specific exploration tables that tightly fit the domains complexity. However, the approach is impractical because every genome needs to be evaluated on a large number of tasks. Too few tasks results in noisy evaluation signals that make it difficult to determine the best exploration tables. Next we will consider learning an exploration table.

7.7.2 Experiment: Learning an Exploration Table

We executed LET on domains $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$. Figure 7.7 shows that LET is able to learn good exploration tables for each of the four domains. If we expand domains $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$ to have 10 actions, the problem may be more challenging.

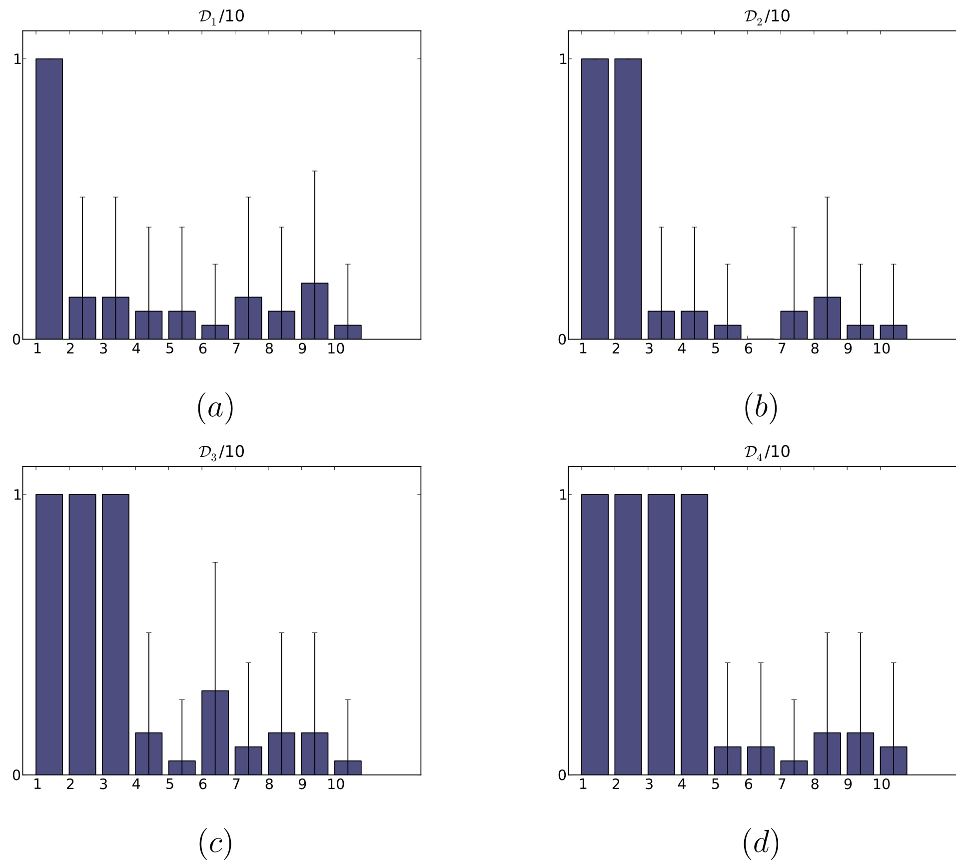


Figure 7.8: Average learned exploration tables (with error bars indicating ±1 standard deviation) by the LET algorithm on domains (a) $\mathcal{D}_1$, (b) $\mathcal{D}_2$, (c) $\mathcal{D}_3$, and (d) $\mathcal{D}_4$ expanded to have ten actions instead of four. The learned exploration tables accurately determine the minimum number of actions that can be explored depending on the domain.

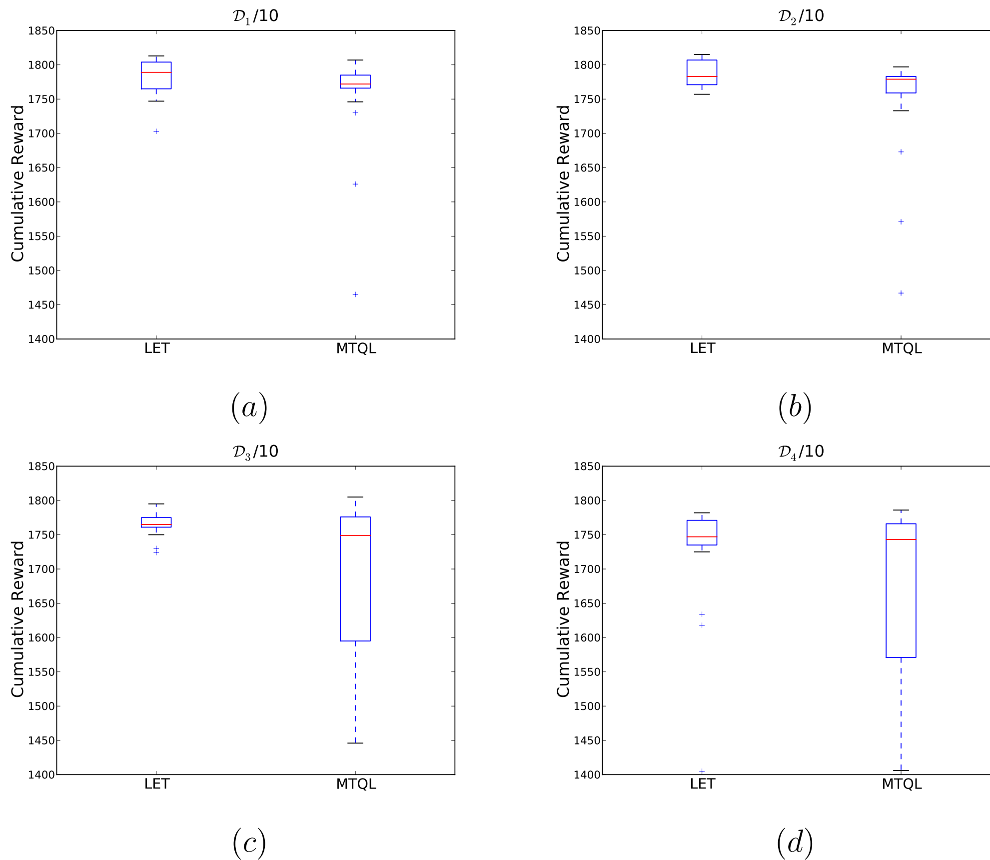


Figure 7.9: Comparison between the cumulative reward earned by LET and MTQL in $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$ extended to 10 actions rather than 4. Whiskers indicate 1.5 times the interquartile range.

Figure 7.8 shows the learned exploration tables for this scenario. Again we see that LET is able to learn reasonable exploration tables. We compared the performance of LET with MTQL (Algorithm 10) on these four domains. As the number of potentially optimal actions increases, LET performs better than MTQL because MTQL relies on undirected exploration to decide when to try different actions (Figure 7.9). LET, on the other hand, tries each of the candidate actions m times and then selects the action with highest empirical reward.

We also tried LET on the reset domains. Figure 7.10 shows that as the number of potentially optimal actions increases, LET finds exploration tables that contain

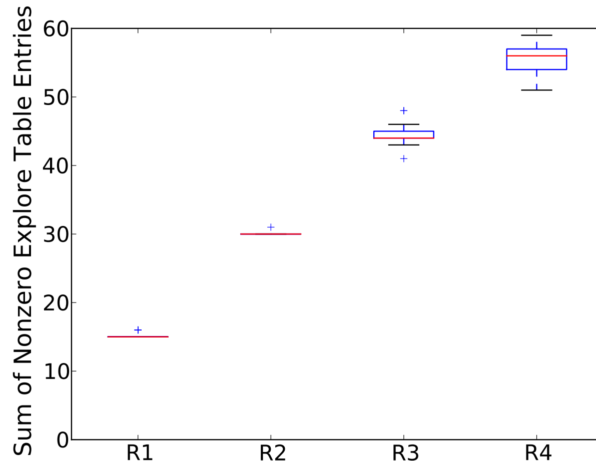


Figure 7.10: The number of entries in exploration tables learned by LET for domains $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$. As the number of tasks with different optimal actions increases. The number of nonzero entries in the learned exploration tables increase. Whiskers indicate 1.5 times the interquartile range.

more entries. Figure 7.11 compares the average reward achieved on the final sampled reset task between RMAX, LET (using RMAX with learned exploration table) and MTQL. MTQL is unable to solve the difficult reset task in a reasonable amount of time due to the fact that it uses undirected exploration. LET learns more quickly than RMAX in domains $\mathcal{R}_1$ and $\mathcal{R}_2$. However, as the number of potentially optimal actions increases, LET is more likely to eliminate these actions. This problem can be solved in two ways. One way is to sample more tasks before applying the domain specific exploration table. This will give LET more opportunities to observe actions that are α-optimal in only a few states. Another possibility is to partition the states so that if an action is observed to be α-optimal at any state in a partition it is included in the exploration table for all states in that partition.

The results of our experiments suggest that LET is able to find reasonable exploration tables with fewer samples than EET. This is especially important for domains with many states and actions. Our results also suggest that LET has some advan-

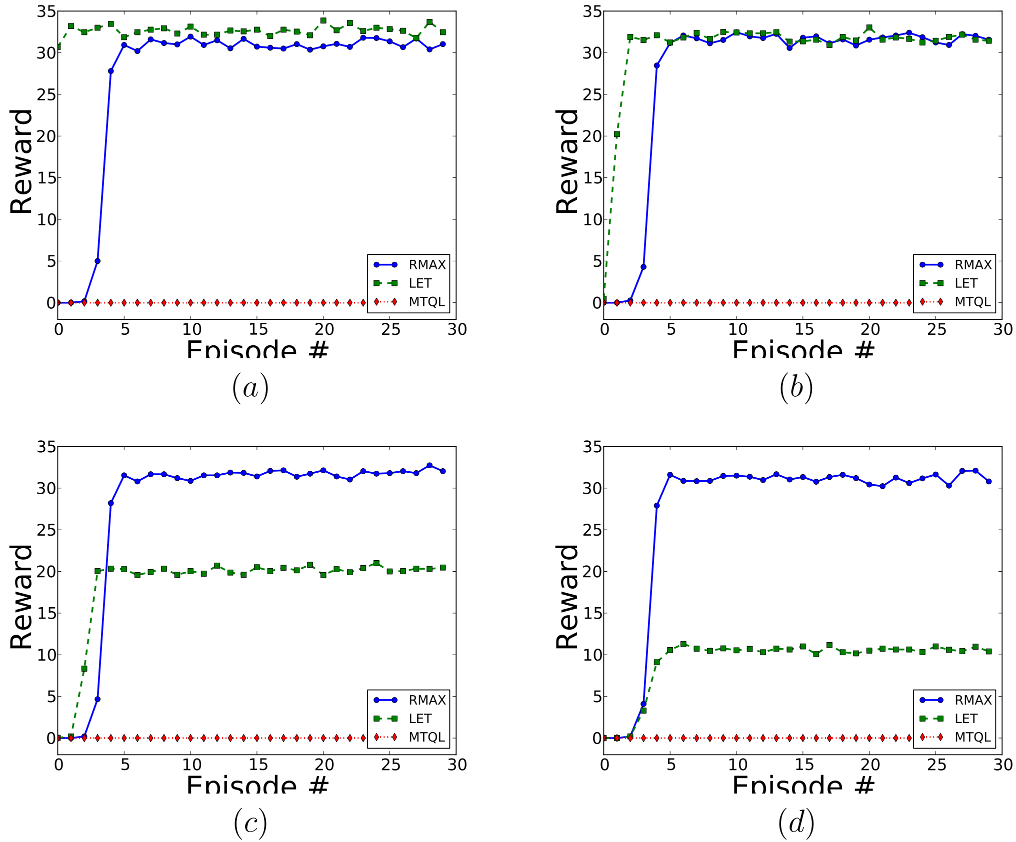


Figure 7.11: Comparison between the cumulative reward earned by LET and MTQL in $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$. MTQL is unable to solve the task in a reasonable amount of time due to relying on undirected exploration. LET performs well compared to RMAX when there are few potentially optimal actions, but it begins to drop important actions as the task complexity increases.

tages over MTQL, which does not perform as well as LET (using RMAX) when the number of potentially optimal actions is large (but only a small number are optimal at any one time). MTQL was also unable to solve reset tasks in a reasonable amount of time because of its use of undirected exploration.

7.8 Discussion

Using EET to evolve exploration tables necessarily samples a large number of tasks, because each table needs to be evaluated. LET partially alleviates this problem by learning an exploration table based on information from a set of sampled tasks.

Unfortunately, the analysis for the LET algorithm depends on the ability to access the tasks optimal action-values or at least generative models for sampled tasks. The generative models are necessary to allow the algorithm to learn accurate action-values so that the sets of "good" state-action pairs are accurate. In practice, we observed that for tasks with large state-action spaces it may be difficult to learn accurate action-values for all state-action pairs. This can often result in poor exploration tables. One potential remedy for this problem is to provide a partition on the state space so that if an action is observed to be α-good at one state in a class, then the resulting exploration table will include that action at every state in the equivalence class. This would allow a learning algorithm to include α-good state-action pairs from difficult to reach states in the exploration table.

7.9 Summary

In this section, we have introduced and analyzed algorithms for learning domain specific exploration strategies. We developed deterministic and stochastic measures of the complexity of a domain of MDPs. Using an evolutionary algorithm, we were able to demonstrate how tightly the various measures fit a domain's difficulty. We presented an algorithm for learning an exploration table that matches our determin-

istic measures of domain complexity with a bounded number of tasks sampled from the domain. This algorithm is able to learn a good exploration table for the domain with high probability. After the initial training phase, the Delayed Q-learning algorithm paired with the learned exploration table has a high probability of learning a near-optimal policy for any task sampled from the domain after exploring far fewer state-action pairs. By framing these complexity measures in terms of the number of state-action pairs that may need to be explored to learn a near-optimal policy on the next sample task, we were able to reuse our analysis of exploration tables to analyze sample complexity and compare the various complexity measures. To the author's knowledge these results are the first attempt to analyze multitask RL from a sample complexity perspective.

8. DISCUSSION

In this section, we discuss the findings, main contributions, and limitations of the results developed throughout this dissertation. First, we discuss action pruning, followed by action-value transfer, and finally multitask learning.

8.1 Action Pruning

Our theoretical analysis of action pruning was foundational for our analysis of action-value transfer and multitask learning. We introduced two structures for pruning actions. Exploration tables enabled explicit action pruning, while weak admissible heuristics implicitly pruned actions.

8.1.1 Findings & Contributions

The main findings related action pruning are Theorems 5.7, 5.10, 5.20, and 5.21, which establish sample complexity bounds for R-MAX and Delayed Q-learning when initialized with an exploration table or weak admissible heuristic. These theorems formalized the intuition that constraining the action space leads to more sample-efficient reinforcement learning. We also introduced the notion of optimality loss and provided a lemma for bounding that loss (i.e., Lemma 5.6).

The main contribution of Section 5 is the development of key theorems that are used for analyzing action-value transfer and multitask learning. These theorems establish that sample complexity decreases proportionally to the number of state-action pairs that can be ignored while learning.

8.1.2 Limitations

The main limitation of this analysis is that exploration tables and weak admissible heuristics that decrease sample complexity but also result in small optimality loss

may be difficult to acquire. The section on action-value transfer and the section on multitask learning show how these transfer learning mechanisms could be used to learn weak admissible heuristics or exploration tables that result in small optimality loss.

8.1.3 Future Work & Open Questions

The most promising directions of future work are to continue exploring various structures that explicitly or implicitly prune state-action pairs. Weak admissible heuristics are more general than the admissible heuristics presented by [6], however, weak admissible heuristics only provide sufficient conditions for converging to a near-optimal policy (when α is very small). One promising direction of future work is to consider ways of further weakening the assumptions placed on weak admissible heuristics. This would enable us to understand even more conditions where transfer learning succeeds with high probability.

8.2 Action-value Transfer

Action-value transfer has been found to work well in many experiments [45, 58, 8, 12]. We have provided a deeper theoretical and experimental analysis of action-value transfer using PAC-MDP algorithms in the target task.

8.2.1 Findings & Contributions

First, we found that learning action-values through exploration is a difficult problem. It can require a number of samples that is exponential with respect to the number of states. However, if a generative model for an MDP is known, then it is possible to obtain arbitrarily accurate action-values with polynomially many samples. This is an important finding because learning action-values is a critical step for generating source task knowledge. This suggests that when applying action-value transfer the

source task should be well known or easy to explore the most significant regions of the state-action space.

Second, we found that when the intertask mapping transfers action-values that satisfy a weak admissible heuristic (with small α) for the target task, then negative transfer will not occur, with high probability. Furthermore, if the transferred weak admissible heuristic eliminates some state-action pairs, then positive transfer will occur, with high probability.

8.2.2 Limitations

The main limitations of action-value transfer are obtaining action-values from the source task and an intertask mapping that relates action-values from the target task to action-values from the source task. In practice, learning action-values from the source task tends not to be much of a problem. Furthermore, if the Delayed Q-learning algorithm is used to acquire action-values from the source task, its estimates will be optimistic and therefore even if they are inaccurate, they will not cause optimality loss in the target task.

Acquiring an intertask mapping relating target task state-action pairs to source task state-action pairs may be more problematic. In many settings, where the source and target task share the same state-action space, the identity intertask mapping can be used. In our experiments, we either supplied intertask mappings designed by hand or generated intertask mappings using perfect knowledge of the source and target task for experimental purposes. In practice, it is often possible for human engineers to design reasonable intertask mappings between two domains that share important structure. Liu and Stone [47] and Taylor et al. [11] have considered the problem of learning intertask mappings. However, it is an open question whether or not intertask mappings can be autonomously learned in practice.

8.2.3 Future Work & Open Questions

The main open question is how reasonable intertask mappings can be acquired autonomously. Liu and Stone [47] consider selecting from a small set of useful intertask mappings to relate different types of games. Their method requires knowledge of the high-level structure shared between tasks. Taylor et al. [11] introduce the MASTER algorithm, which uses samples from the source and target task to learn an intertask mapping through an exhaustive search. This approach is a good first step, but it is probably too computationally expensive to apply to tasks with large state-action spaces. Furthermore, the number of samples needed to select the best intertask mapping is probably enough to select a near-optimal policy for the target task. Future work should consider selecting an intertask mapping under heavy regularization. This may allow a learning system to select a reasonable intertask mapping with fewer samples and lower computational costs.

8.3 Multitask Learning

We considered learning a domain specific RL algorithm for a domain of tasks. The multitask scenario matches the intuition that intelligent agents are often faced with related but not identical tasks.

8.3.1 Findings & Contributions

Our main contribution is our attempt to develop a measure of the complexity of a domain of tasks. We developed six different domain complexity measures, all based on the number of state-action pairs that need to be explored to learn a near-optimal policy with high probability. Our first three measures do not take into account the probability mass assigned to tasks in the domain, while our last three use this information to describe the domain's complexity more tightly.

Evolutionary algorithms allowed us to find exploration tables that closely matched the tightest measures of domain complexity, however, these algorithms sample a large number of tasks in order to evaluate different exploration tables. In response, we developed a more sensible algorithm for learning domain specific exploration tables. Although the learned exploration tables do not match the tightest measure of domain complexity, they provide a good trade-off.

8.3.2 Limitations

Similar to the problem pointed out for action-value transfer. Learning any meaningful statistics about sampled tasks depends on the learning algorithms ability to explore the tasks. If the tasks are highly stochastic and the algorithm is given only a single chain of experience through the task, then it would be unreasonable to guarantee the algorithm will learn any statistics of interest about the task. However, in practice this tends to be less of a concern. Typically, useful information is learned from each task sampled from the domain. Unfortunately, we do not have a good way of quantifying this learned information. In our analysis, we ignored this problem assuming that the learning system had complete knowledge of sampled tasks.

The main limitation of our multitask approach is that our algorithm for learning an exploration table does not learn the most aggressive exploration table suggested by our more aggressive notions of domain complexity $\mathbb{C}_3$ and $\mathbb{C}_6$. However, we argued that learning exploration tables that select a minimal hitting set at each state would require sampling an unreasonable number of tasks from the domain.

8.3.3 Future Work & Open Questions

We have only considered learning exploration tables. It may be possible to learn about other relationships between tasks in a domain that constrain the state-action space. Future work should explore other structures and relationships that are useful

for multitask learning.

Another important area of future work is to scale up the methods explored here to multitask domains with larger state spaces. This will almost certainly require partitioning the state space to decrease the number of tasks sampled before discovering an improved domain specific exploration strategy.

9. CONCLUSIONS

The goal of this dissertation was to understand how transfer learning (TL) can be used to scale up reinforcement learning (RL) algorithms to solve problems with large state-action spaces. Although many works have demonstrated experimentally that TL can be successfully applied to speed up RL, there has been a lack of theoretical analysis explaining when TL will succeed in speeding up RL and when it will fail. The main contributions of this dissertation are (1) the development of new algorithms for using prior knowledge, (2) the development of new algorithms and structures for transfer and multitask learning, and (3) the theoretical analysis of transfer and multitask learning from a sample complexity perspective.

We introduced the STAR-MAX algorithm that is able to use prior knowledge about which states of a task are irrelevant to constrain the search space. This method suggested the alternative idea of constraining state-action pairs. We introduced two structures, (1) exploration tables and (2) weak admissible heuristics, as forms of prior knowledge that both turn out to constrain the number of state-action pairs. We analyzed sample complexity and optimality loss when R-MAX and Delayed Q-learning are paired with either exploration tables or weak admissible heuristics. Using the weak admissible heuristic concept we were able to analyze action-value transfer from a sample complexity perspective, finding that positive transfer can occur under a general range of circumstances. Experiments with action-value transfer suggested that action-value transfer is even more robust in practice. We analyzed multitask learning from the perspective of learning an exploration table by sampling tasks from a domain. We developed measures of domain complexity that describe how many state-action pairs can be eliminated from exploration given a perfect exploration

table and developed a learning algorithm that learns an exploration table efficiently. Together, these results demonstrate the importance of viewing transfer learning as a mechanism for intelligently constraining the exploration space.

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[2] T. Hester, M. Quinlan, and P. Stone, "Generalized model learning for reinforcement learning on a humanoid robot," in *IEEE International Conference on Robotics and Automation*, 2010.

[3] T. G. Dietterich, "The MAXQ method for hierarchical reinforcement learning," in *International Conference on Machine Learning*, 1998, pp. 118–126.

[4] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and Autonomous Systems*, vol. 15, pp. 25–46, 1995.

[5] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. 1, pp. 1633–1685, 2009.

[6] A. L. Strehl, L. Li, and M. Littman, "Reinforcement learning in finite MDPs: PAC analysis," *Journal of Machine Learning Research*, vol. 10, pp. 2413–2444, 2009.

[7] M. E. Taylor and P. Stone, "Behavior transfer for value-function-based reinforcement learning," in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '05. New York, NY, USA: ACM, 2005, pp. 53–59. [Online]. Available: http://doi.acm.org/10.1145/1082473.1082482

[8] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, pp. 2125–2167, 2007.

[9] M. E. Taylor and P. Stone, "Cross-domain transfer for reinforcement learning," in *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: ACM, 2007, pp. 879–886. [Online]. Available: http://doi.acm.org/10.1145/1273496.1273607

[10] M. E. Taylor, N. K. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Artificial Intelligence, vol. 5212, September 2008, pp. 488–505.

[11] M. E. Taylor, G. Kuhlmann, and P. Stone, "Autonomous transfer for reinforcement learning," in *The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2008.

[12] T. A. Mann and Y. Choe, "Prenatal to postnatal transfer of motor skills through motor-compatible sensory representations," in *IEEE Ninth International Conference on Development and Learning (ICDL)*, aug. 2010, pp. 185 –190.

[13] S. Zoia, L. Blason, G. D'Ottavio, M. Bulgheroni, E. Pezzetta, A. Scabar, and U. Castiello, "Evidence of early development of action planning in the human foetus: a kinematic study," *Experimental Brain Research*, vol. 176, pp. 217–226, 2007.

[14] T. A. Mann, Y. Park, S. Jeong, M. Lee, and Y. Choe, "Autonomously improving binocular depth estimation," in *Proceedings of the Japanese Neural Networks Society*, 2011.

[15] R. I. Brafman and M. Tennenholtz, "R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2002.

[16] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC model-free reinforcement learning," in *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-06)*, 2006.

[17] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin American Mathematical Society*, vol. 55, pp. 527–535, 1952.

[18] E. Even-Dar, S. Mannor, and Y. Mansour, "PAC bounds for multi-armed bandit and markov decision processes," in *Fifteenth Annual Conference on Computational Learning Theory (COLT)*, 2002, pp. 255–270.

[19] ——, "Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems," *Journal of Machine Learning Research*, vol. 7, pp. 1079–1105, 2006.

[20] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multi-armed bandit problem," *Machine Learning*, vol. 47, pp. 235–256, 2002.

[21] L. G. Valiant, "A theory of the learnable," in *STOC '84: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing.* New York, NY, USA: ACM, 1984, pp. 436–445.

[22] S. Mannor and J. N. Tsitsiklis, "The sample complexity of exploration in the multi-armed bandit problem," *Journal of Machine Learning Research*, vol. 5, pp. 623–648, 2004.

[23] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. pp. 13–30, 1963. [Online]. Available: http://www.jstor.org/stable/2282952

[24] M. L. Puterman, *Markov Decision Processes - Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[25] R. E. Bellman, "The theory of dynamic programming," Rand Corp., Tech. Rep., 1954.

[26] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving markov decision problems," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, ser. UAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 394–402. [Online]. Available: http://dl.acm.org/citation.cfm?id=2074158.2074203

[27] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, vol. 49, pp. 209–232, 2002, 10.1023/A:1017984413808. [Online]. Available: http://dx.doi.org/10.1023/A:1017984413808

[28] A. L. Strehl and M. L. Littman, "A theoretical analysis of model-based interval estimation," in *Proceedings of the Twenty-Second International Conference on Machine Learning*, 2005.

[29] I. Szita and C. Szepesvári, "Model-based reinforcement learning with nearly tight exploration complexity bounds," in *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, 2010.

[30] S. M. Kakade, "On the sample complexity of reinforcement learning," Ph.D. dissertation, University College London, March 2003.

[31] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[32] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *Journal of Machine Learning Research*, vol. 11, pp. 1563–1600, Aug. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1756006.1859902

[33] A. Lazaric, M. Ghavamzadeh, and R. Munos, "Finite-sample analysis of lstd," in *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML-10)*, 2010, pp. 615–622.

[34] M. J. Kearns and S. P. Singh, "Finite-sample convergence rates for q-learning and indirect algorithms," in *Advances in Neural Information Processing Systems 11*. The MIT Press, 1999, pp. 996–1002.

[35] J. Baxter, "A model of inductive bias learning," *Journal of Artificial Intelligence Research*, vol. 12, pp. 149–198, 2000.

[36] A. Lazaric, "Knowledge transfer in reinforcement learning," Ph.D. dissertation, Politecnico Di Milano, 2008.

[37] G. Konidaris, I. Scheidwasser, and A. Barto, "Transfer in reinforcement learning via shared features," *Journal of Machine Learning Research*, vol. 13, pp. 1333–1371, 2012.

[38] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, July 1997.

[39] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and

the vapnik-chervonenkis dimension," *Journal of the Association for Computing Machinery*, vol. 36, pp. 929–965, 1989.

[40] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine Learning*, vol. 79, pp. 151–175, 2010.

[41] F. Tanaka and M. Yamamura, "Multitask reinforcement learning on the distribution of MDPs," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, July 2003.

[42] F. Fernández, J. Garcá, and M. Veloso, "Probabilistic policy reuse for inter-task transfer learning," *Robotics and Autonomous Systems*, vol. 58, pp. 866–871, 2010.

[43] M. E. Taylor and P. Stone, "An introduction to inter-task transfer for reinforcement learning," *AI Magazine*, vol. 32, no. 1, pp. 15–34, 2011.

[44] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '06. New York, NY, USA: ACM, 2006, pp. 720–727. [Online]. Available: http://doi.acm.org/10.1145/1160633.1160762

[45] O. G. Selfridge, R. Sutton, and A. Barto, "Training and tracking in robotics," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985, pp. 670–672.

[46] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia, "Generalizing plans to

new environments in relational mdps," in *International Joint Conference on Artificial Intelligence*, 2003.

[47] Y. Liu and P. Stone, "Value-function-based transfer for reinforcement learning using structure," in *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 2006.

[48] G. Kuhlmann and P. Stone, "Graph-based domain mapping for transfer learning in general games," in *Proceedings of the European Conference on Machine Learning*, 2007.

[49] M. Mataric, "Reward functions for accelerated learning," in *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

[50] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.

[51] A. A. Sherstov and P. Stone, "Improving action selection in MDP's via knowledge transfer," in *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[52] B. R. Leffler, M. L. Littman, and T. Edmunds, "Efficient reinforcement learning with relocatable action models," in *Proceedings of the Twenty-Second National Conference on Artificial Intelligence - Volume 1*. AAAI Press, 2007, pp. 572–577. [Online]. Available: http://portal.acm.org/citation.cfm?id=1619645.1619737

[53] M. Kearns and D. Koller, "Efficient reinforcement learning in factored mdps,"

in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999, pp. 740–747.

[54] T. A. Mann and Y. Choe, "Scaling up reinforcement learning through targeted exploration," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[55] T. Hester and P. Stone, "Generalized model learning for reinforcement learning in factored domains," in *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2009.

[56] A. L. Strehl, L. Li, and M. L. Littman, "Incremental model-based learners with formal learning-time guarantees," in *Uncertainty in Artificial Intelligence*, 2006.

[57] P. Stone, G. Kuhlmann, M. E. Taylor, and Y. Liu, *RoboCup-2005: Robot Soccer World Cup IX*. Springer Verlag, 2006, ch. Keepaway Soccer: From Machine Learning Testbed to Benchmark, pp. 93–105.

[58] S. P. Singh, "Transfer of learning by composing solutions of elemental sequential tasks," *Machine Learning*, vol. 8, pp. 323–339, 1992.

[59] C.-N. Fiechter, "Efficient reinforcement learning," in *Proceedings of the Seventh Annual Conference on Computational Learning Theory*, ser. COLT '94. New York, NY, USA: ACM, 1994, pp. 88–97. [Online]. Available: http://doi.acm.org/10.1145/180139.181019

[60] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: A hierarchical bayesian approach," in *Proceedings of the* 24th *International Conference on Machine Learning*, 2007.

[61] A. Lazaric and M. Ghavamzadeh, "Bayesian multi-task reinforcement learning," in *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, 2010.

[62] R. Reiter, "A theory of diagnosis from first principles," *Artificial Intelligence*, vol. 32, pp. 57–95, 1987.

[63] S. Vinterbo and A. Øhrn, "Minimal approximate hitting sets and rule templates," *International Journal of Approximate Reasoning*, vol. 25, pp. 123–143, 2000.

[64] R. Karp, *Reducibility Among Combinatorial Problems*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972.