LEARNING IN THE PRESENCE OF UNAWARENESS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Nan Rong

May 2016

LEARNING IN THE PRESENCE OF UNAWARENESS

Nan Rong, Ph.D.

Cornell University 2016

Markov decision processes (MDPs) are widely used for modeling decision-making problems in robotics, automated control, and economics. Traditional MDPs assume that the decision maker (DM) knows all states and actions. However, this may not be true in many situations of interest. We define a new framework, *MDPs with unawareness* (MDPUs), which allows for the possibility that a DM may not be aware of all possible actions. We provide a complete characterization of when a DM can learn to play near-optimally in an MDPU, and give an algorithm that learns to play near-optimally when it is possible to do so, as efficiently as possible. In particular, we characterize when a near-optimal solution can be found in polynomial time.

We formalize decision-making problems in robotics and automated control using continuous MDPs and actions that take place over continuous time intervals. We then approximate the continuous MDP using finer and finer discretizations. Doing this results in a family of systems, each of which has an extremely large action space, although only a few actions are "interesting". This can be modeled using MDPUs, where the action space is much smaller. As we show, MDPUs can be used as a general framework for learning tasks in robotic problems. We prove results on the difficulty of learning a near-optimal policy in an an MDPU for a continuous task. We apply these ideas to the problem of having a humanoid robot learn on its own how to walk.

Finally, we consider the scenario in which the DM has a limited budget for solving the problem on top of unawareness. In order to deal with such problems, we define a model called MDPUs with a prior and a budget (MDPUBs) that considers both unawareness and a limited budget. We also consider the problem of learning to play approximately optimally in a subclass

of MDPUBs called budgeted learning problems with unawareness (BLPUs), which are multi-armed bandit problems in which there may be arms that the DM is unaware of. We provide a policy that is $0.25c$-optimal for BLPUs, where c is a constant determined by the probability of discovering a new arm and the probability of there being undiscovered arms, that can be computed in polynomial time.

BIOGRAPHICAL SKETCH

Born in Wuhan, Nan Rong is the second child of her family. She went to Wuhan Foreign Language School in 1994 for middle school and high school. She obtained her Bachelor of Computing (w. first-class honors) in Computer Engineering from the National University of Singapore (NUS) in 2005, and obtained her Master of Science in Computer Science from the Singapore-MIT Alliance in 2006. She worked as a research engineer at the Department of Electrical and Computer Engineering in NUS for one year in 2007. She is currently a PhD candidate in Computer Science at Cornell under the supervision of Professor Joseph Halpern. Her research interests include artificial intelligence, algorithms, robotics, and game theory. She is passionate in theorem proving and working with robots.

To my parents.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor, Joe Halpern, for his enormous patience, encouragements and support. His knowledge, foresight, and sharpness have guided me through all the time of my research, and has made me a better researcher. The methodology he uses to examine models and to break down workable solutions to retain only their essential elements is impressive, and have influenced me in many ways of viewing problems. I have learned many things from him, among which the property of never give up. I would also like to thank Ashutosh Saxena for his ideas and passions for robotic research. Without his participation, we could not have initiated the robotic experiments.

I would like to express my sincere gratitude to Bart Selman for inspiring suggestions and comments on the research, to Bobby Kleinberg for valuable discussions on budgeted learning and the continuous-time MDP model, and to Larry Blume for his great classes on Microeconomics and questions on unawareness. I would also like to thank David Hsu, Lee Wee Sun, Ian Kash, Bao-Toan Nguyen, Andrew Perrault, and Jonathan Diamond.

Finally, I want to thank my parents for their love and support, to Mr. Qin Mingming for the novels he wrote, his encouragements and supports, and to everyone who cares about me.

This work was supported in part by NSF grants IIS-0534064, IIS-0812045, and IIS-0911036, by AFOSR grants FA9550-08-1-0438 and FA9550-09-1-0266, and by ARO grant W911NF-09-1-0281.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

People have motives and thoughts of which they are unaware. –Albert Ellis

In decision making, standard models such as Markov decision processes (MDPs) [3; 26; 36] often assume that the decision maker (DM) knows the complete set of states and actions. Unfortunately, in many cases, the decision maker (DM) does not know the state space, and is unaware of some possible actions she can perform. For example, before evidence was found on September 28th, 2015, scientists were unaware that there is liquid water on Mars [5], which means the Mars rover was unaware of certain states and actions on the planet; also, someone buying insurance may not be aware of all possible contingencies; a mathematician trying to prove a math problem may not be aware of all possible proof techniques. This leads us to the problem this thesis aims to solve – how to use mathematical models to capture unawareness in decision making problems and learn to play optimally when there is unawareness.

Two types of unawareness are relevant here - unawareness in states (e.g., in the first and second events above) and unawareness in actions (e.g., in the third event). The fact that the DM may not be aware of all states does not cause major problems. If an action leads to a new state and the set of possible actions is known, we can use standard techniques (discussed below) to decide what to do next. The more interesting issue comes in dealing with actions that the DM may not be aware of. If the DM is not aware of her lack of awareness then it is clear how to proceed—we can simply ignore these actions. We are interested in a situation where the DM realizes that there are actions (and states) that she is not aware of, and thus will want to explore the MDP. In this thesis, we propose a model that captures this by a special *explore* action. As a result of playing this action, the DM might become aware of more actions, whose effect she

can then try to understand.

We have been deliberately vague about what it means for a DM to be "unaware" on an action. We have in mind a setting where there is a (possibly large) space A^* of *potential actions*. For example, in a video game, the space of potential actions may consist of all possible inputs from all input devices combined (e.g., all combinations of mouse movements, presses of keys on the keyboard, and eye movements in front of the webcam); if a DM is trying to prove a theorem, at least in principle, all possible proof techniques can be described in English, so the space of potential actions can be viewed as a subset of the set of English texts. The space A of *actual actions* is the (typically small) subset of A^* that are the "useful actions". For example, in a video game, these would be the combinations of arrow presses (and perhaps head movements) that have an appreciable effect on the game. Of course, A^* may not describe how the DM conceives of the potential acts. For example, a first-time video-game player may consider the action space to include only presses of the arrow keys, and be completely unaware that eye movement is an action. Similarly, a mathematician trying to find a proof probably does not think of herself as searching in a space of English texts; she is more likely to be exploring the space of "proof techniques". A sophisticated mathematician or video game player will have a better understanding of the space that she views herself as exploring. Moreover, the space of potential actions may change over time, as the DM becomes more sophisticated. Thus, we do not explicitly describe A^* in our formal model, and abstract the process of exploration by just having an *explore* action. (It actually may make sense to have several different *explore* actions, with different properties, although we do no consider that possibility in this paper.)

This type of exploration occurs all the time. In video games, first-time players often try to learn the game by exploring the space of moves, without reading the instructions (and thus, without being aware of all the moves they can make). Indeed, in many games, there may

not be instructions at all (even though players can often learn what moves are available by checking various sites on the web). Mathematicians trying to generate new approaches to proving a theorem can be viewed as exploring the space of proof techniques. More practically, in robotics, if we take an action to be a "useful" sequence of basic moves, the space of potential actions is often huge. For instance, most humanoid robots (such as the Honda Asimo robot [48]) have more than 20 degrees of freedom; in such a large space, while robot designers can hand-program a few basic actions (e.g., standing up, sitting down, or walking with a fixed gait), it is practically impossible to do so for other general scenarios (e.g., learning various gaits). Conceptually, it is useful to think of the designer as not being aware of the actions that can be performed. Exploration is almost surely necessary to discover new actions necessary to enable the robot to perform the new tasks.

Note that there are two senses of unawareness here. In the first sense, a DM is unaware of an action if it is not in his potential action space; roughly speaking, it is not on his radar screen. For example, suppose that there is a proof that $P \neq NP$. In that case, leading computer scientists and mathematicians are not aware of it, in this sense of awareness. Similarly, The second sense of unawareness is prehaps closer to "aware but unaware it is relevant here" type; the action is in the DM's potential action space, but the DM has no idea whether the action is useful. This is the case in the robotics example above: all the "useful" sequence of moves are in the robot's action space, but the robot does not know which of them are useful. Note that the difference between the two depends in part on how we model the problem. If we take the potential action space for proofs to be all finite length English texts, then a computer scientist would not be aware of the proof that $P \neq NP$ in the second sense; if we take the potential action space to consist only of proofs that use current proof techniques (e.g., diagonalization) and the actual proof uses a novel technique, then the computer scientist would not be aware of the proof in the first sense. Our model, which we call *MDPs with unawareness* (MDPUs), is

useful for dealing with both senses of unawareness.

Given the prevalence of MDPUs, the problem of learning to play well in an MDPU becomes of interest. There has already been a great deal of work on learning to play optimally in an MDP. Kearns and Singh [31] gave an algorithm called E^3 that converges to near-optimal play in polynomial time. Brafman and Tennenholtz [4] later gave an elegant algorithm they called RMAX that converges to near-optimal play in polynomial time not just in MDPs, but in a number of adversarial settings. Can we learn to play near-optimally in an MDPU? (By "near-optimal play", we mean near-optimal play in the actual MDP.) In the earlier work, near-optimal play involved learning the effects of actions (that is, the transition probabilities induced by the action). In our setting, the DM still has to learn the transition probabilities, but also has to learn what actions are available.

Perhaps not surprisingly, we show that how effectively the DM can learn optimal play in an MDPU depends on the probability of discovering new actions. For example, if it is too low, then we can never learn to play near-optimally. If it is a little higher, then the DM can learn to play near-optimally, but it may take exponential time. If it is sufficiently high, then the DM can learn to play near-optimally in polynomial time. We give an expression whose value, under minimal assumptions, completely characterizes when the DM can learn to play optimally, and how long it takes. Moreover, we show that a modification of the RMAX algorithm (that we call URMAX) can learn to play near-optimally if it is possible to do so. Note that, for the settings of parameters for which URMAX learns to play near-optimally in polynomial time, the policy that it learns to play is quite simple; it is in fact linear in the relevant parameters.

There is a subtlety here. Not only might the DM not be aware of what actions can be performed in a given state, she may be unaware of *how many* actions can be performed. Thus, for example, in a state where she has discovered five actions, she may not know whether she

has discovered all the actions (in which case she should not explore further) or there are more actions to be found (in which case she should). All our lower bounds and impossibility results hold even when the DM knows that there is only one action to be discovered. (For example, if the action to be discovered is a proof that P $\neq$ NP, the DM may know that the action has a high reward; she just does not know what that action is.) On the other hand, URMAX works even if the DM does not know how many actions there are to be discovered.

Having defined MDPUs and analyzed the complexity of learning to play near-optimally in MDPUs, we would like to show that the model can be applied to real problems. In order to do so, we apply MDPUs to robotics and automated control problems. One difficulty of doing so lies in the fact that the state space and action space of such problems usually being continuous. To find appropriate policies, we typically discretize both states and actions. However, we do not know in advance what level of discretization is good enough for getting a good policy. Moreover, in the discretized space, the set of actions is huge. However, relatively few of the actions are "interesting". For example, when flying a robotic helicopter, only a small set of actions lead to useful flying techniques; an autonomous helicopter must learn these techniques. Similarly, a humanoid robot needs to learn various maneuvers (e.g., walking or running) that enable it to move around, but the space of potential actions that it must search to find a successful gait is huge, while most actions result in the robot losing control and falling down. In these applications, we can think of the DM (e.g., a humanoid robot) as being unaware of which actions are the useful actions, and thus can model what is going on using an MDPU.

We model such problems using a new model of continuous MDPs where actions are performed over a continuous duration of time. Although many problems fit naturally in our continuous MDP framework, and there has been a great deal of work on continuous-time MDPs, our approach seems new, and of independent interest. (See the discussion in Section 3.4.) It

is hard to find near-optimal policies in continuous MDPs. A standard approach is to use discretization. We use discretization as well, but our discrete models are MDPUs, rather than MDPs, which allows us both to use relatively few actions (the "interesting actions"), while taking into account the possibility of there being interesting actions that the DM is unaware of. We would like to find a discretization level for which the optimal policy in the MDP underlying the approximating MDPU provides a good approximation to the optimal policy in the continuous MDP that accurately describes the problem, and then find a near-optimal policy in that discretized MDPU.

We extend and generalize the above results in the complexity of learning to play near-optimally, so as to apply them to the continuous problems. We characterize when brute-force exploration can be used to find a near-optimal policy in our setting, and show that a variant of the URMAX algorithm can find a near-optimal policy. We also characterize the complexity of learning to play near-optimally in continuous problems, when more "guided" exploration is used.

We apply MDPUs to solve a real robotic problem: to enable a humanoid robot to learn walking on its own using simulations. In our experiment, the robot learned various gaits at multiple discretization levels, including both forward and backward gaits; both efficient and inefficient gaits; and both gaits that resemble human walking, and those that do not.

Having defined the MDPU model and applied it to a real robotic problem, we examine the problem of learning to play near-optimally in MDPUs more closely. When we analyzed the complexity of learning above, we assumed that the DM has unlimited budget (i.e., the DM can make unlimited number of moves). This may not hold in practice where there is usually a limited budget in time. What if the DM can make only a fixed number of moves? More specifically, even if the DM can learn to play optimally in polynomial time, this is not helpful

if the DM can make only 10 moves. Notice that all the examples above remain of interest if we are given a budget (i.e., a bound on the number of moves that can be made). Indeed, they are arguably of even more practical interest. The Mars rover has limited time to explore the Mars and to carry out its tasks; the insurance buyer has limited time to make the decision; and the mathematician has limited time to prove the math problem.

There has been work on budgeted learning without unawareness (i.e., where all actions are known in advance) [15; 18; 37; 38]. Madani et al. [38] first defined the *budgeted learning problem*: the problem of learning to play nearly optimally, given a budget. Much of the work on this problem has been done in the context of multi-armed bandits. But now the meaning of "near-optimal" is somewhat different than in the context of above. Rather than there being an underlying "true" multi-armed bandit problem (with a probability of success for each arm), in which case the goal would be to learn an arm with the highest expected reward (to the extent possible, given the budget), it is assumed that the DM has a prior probability on the expected reward of each arm. Moreover, it is assumed that each arm pays off either 1 ("success") or 0 ("failure"), so the expected reward is just the expected success probability. If the budget is limited, it is clearly unreasonable to expect optimal performance. Thus, given a budget h, the goal is to find a policy whose expected reward is the best among all policies that use only h steps, where the expectation is taken with respect to the DM's beliefs. Madani et al. [38] proved that the problem of finding an optimal policy is NP-hard. Guha and Munagala [18] and Goel et al. [15] each gave a polynomial-time algorithm for the budgeted learning problem that, given a budgeted learning problem B, returns an *approximately optimal policy* for B, that is, a policy whose expected reward is within a constant factor of that of the optimal policy for B.

As we observed above, in many cases of interest, not all the relevant actions are known in advance. In this paper, we consider the budgeted learning problem in the presence of un-

awareness. We define this formally by considering a variant of MDPUs. There are two key differences: we add a budget and, rather than assuming that there is a "true" underlying MDP, we assume a prior distribution over possible MDPs to get what we call *MDPs with unawareness, a prior, and a budget (MDPUBs)*. We now take an optimal policy in an MDPUB with budget h to be a policy that gives the highest expected reward among all policies that run in h steps, where the expected reward of a policy π is taken over the probability on the possible outcomes of running π for h steps (see Section 4.3 for a discussion).

There are a number of subtleties involved with making this precise. For example, what exactly does it mean for a DM to put a positive probability on an MDP that involves actions that the DM is not aware of? The DM can consider an MDP that includes such actions possible, but cannot play such actions. Because we allow such actions, the notion of optimal policy used above is different from that used here, even in the special case of an MDPUB M' that places probability 1 on a single MDP M (which we can think of as the "true" MDP) and has an infinite budget.

In this thesis, like the earlier literature on budgeted learning, we focus on multi-armed bandits. Moreover, like the earlier literature, we assume that the DM's beliefs about the success probability of each arm is given by an (α, β) prior (also known as a *beta density*) [11] and that the success probability of the arms are independent. Without unawareness, given a policy π, this is enough to determine the probability on the outcomes of π, and hence the expected reward of π. In our setting, we need more information. Specifically, we need to know (a) the probability of there being a new arm; (b) the success probability of new arms, if a new arm is discovered; and (c) the terms $D(1, t)$ described above, which give the probability of discovering a new arm if there is a new arm to be discovered after having played explore t times since an arm was last discovered. We call this restricted class of MDPUBs *budgeted learning problems*

with unawareness (BLPUs). We provide an algorithm that, given a BLPU B returns a policy π that is approximately optimal for B in time polynomial in the number of arms initially known in B and the budget.

The rest of the thesis is organized as follows. In Chapter 2, we define MDPUs, analyze the complexity of learning to play near-optimally in MDPUs, and present the URMAX algorithm that computes a near-optimal policy for MDPUs in polynomial time whenever such a policy can be computed in polynomial time. (Chapter 2 is taken from [20] and [21], joint with Joseph Y. Halpern and Ashutosh Saxena.) In Chapter 3, we show that MDPUs can be used to solve real problems by applying them to robotic problems. We describe in detail how robotic problems can be modeled as MDPUs, and then solved by the URMAX algorithm. We give an example of such applications, in which our approach enabled a DARwIn-OP robot to learn to walk on its own. (Chapter 3 is taken from [46], joint with Joseph Y. Halpern and Ashutosh Saxena.) In Chapter 4, we define the models of MDPUBs and BLPUs for problems where there are both unawareness and a limited budget. We provide an algorithm that computes an approximately-optimal policy for BLPUs in polynomial time. (Chapter 4 is taken from [45], joint with Joseph Y. Halpern.)

CHAPTER 2

MDPS WITH UNAWARENESS

Nobody is bored when he [or she] is trying to make something that is beautiful, or to discover something that is true. –Willian Ralph Inge

2.1 Introduction

Markov decision processes (MDPs) [3; 26; 36] have been used in a wide variety of settings to model decision making. The description of an MDP includes a set S of possible states and a set A of actions. Unfortunately, in many decision problems of interest, the decision maker (DM) does not know the state space, and is unaware of possible actions she can perform. For example, someone buying insurance may not be aware of all possible contingencies; someone playing a video game may not be aware of all the actions she is allowed to perform nor of all states in the game.

The fact that the DM may not be aware of all states does not cause major problems. If an action leads to a new state and the set of possible actions is known, we can use standard techniques (discussed below) to decide what to do next. The more interesting issue comes in dealing with actions that the DM may not be aware of. If the DM is not aware of her lack of awareness then it is clear how to proceed—we can simply ignore these actions. We are interested in a situation where the DM realizes that there are actions (and states) that she is not aware of, and thus will want to explore the MDP. We model this by using a special *explore* action. As a result of playing this action, the DM might become aware of more actions, whose effect she can then try to understand.

We have been deliberately vague about what it means for a DM to be "unaware" on an action. We have in mind a setting where there is a (possibly large) space A^* of *potential actions*. For example, in a video game, the space of potential actions may consist of all possible inputs from all input devices combined (e.g., all combinations of mouse movements, presses of keys on the keyboard, and eye movements in front of the webcam); if a DM is trying to prove a theorem, at least in principle, all possible proof techniques can be described in English, so the space of potential actions can be viewed as a subset of the set of English texts. The space A of *actual actions* is the (typically small) subset of A^* that are the "useful actions". For example, in a video game, these would be the combinations of arrow presses (and perhaps head movements) that have an appreciable effect on the game. Of course, A^* may not describe how the DM conceives of the potential acts. For example, a first-time video-game player may consider the action space to include only presses of the arrow keys, and be completely unaware that eye movement is an action. Similarly, a mathematician trying to find a proof probably does not think of herself as searching in a space of English texts; she is more likely to be exploring the space of "proof techniques". A sophisticated mathematician or video game player will have a better understanding of the space that she views herself as exploring. Moreover, the space of potential actions may change over time, as the DM becomes more sophisticated. Thus, we do not explicitly describe A^* in our formal model, and abstract the process of exploration by just having an *explore* action. (It actually may make sense to have several different *explore* actions, with different properties, although we do no consider that possibility in this paper.)

This type of exploration occurs all the time. In video games, first-time players often try to learn the game by exploring the space of moves, without reading the instructions (and thus, without being aware of all the moves they can make). Indeed, in many games, there may not be instructions at all (even though players can often learn what moves are available by checking various sites on the web). Mathematicians trying to generate new approaches to

proving a theorem can be viewed as exploring the space of proof techniques. More practically, in robotics, if we take an action to be a "useful" sequence of basic moves, the space of potential actions is often huge. For instance, most humanoid robots (such as the Honda Asimo robot [48]) have more than 20 degrees of freedom; in such a large space, while robot designers can hand-program a few basic actions (e.g., standing up, sitting down, or walking with a fixed gait), it is practically impossible to do so for other general scenarios (e.g., learning various gaits). Conceptually, it is useful to think of the designer as not being aware of the actions that can be performed. Exploration is almost surely necessary to discover new actions necessary to enable the robot to perform the new tasks.

Note that there are two senses of unawareness here. In the first sense, a DM is unaware of an action if it is not in his potential action space; roughly speaking, it is not on his radar screen. For example, suppose that there is a proof that $P \neq NP$. In that case, leading computer scientists and mathematicians are not aware of it, in this sense of awareness. Similarly, The second sense of unawareness is prehaps closer to "aware but unaware it is relevant here" type; the action is in the DM's potential action space, but the DM has no idea whether the action is useful. This is the case in the robotics example above: all the "useful" sequence of moves are in the robot's action space, but the robot does not know which of them are useful. Note that the difference between the two depends in part on how we model the problem. If we take the potential action space for proofs to be all finite length English texts, then a computer scientist would not be aware of the proof that $P \neq NP$ in the second sense; if we take the potential action space to consist only of proofs that use current proof techniques (e.g., diagonalization) and the actual proof uses a novel technique, then the computer scientist would not be aware of the proof in the first sense. Our model is useful for dealing with both senses of unawareness.

Given the prevalence of MDPUs—*MDPs with unawareness*, the problem of learning to

play well in an MDPU becomes of interest. There has already been a great deal of work on learning to play optimally in an MDP. Kearns and Singh [31] gave an algorithm called E^3 that converges to near-optimal play in polynomial time. Brafman and Tennenholtz [4] later gave an elegant algorithm they called RMAX that converges to near-optimal play in polynomial time not just in MDPs, but in a number of adversarial settings. Can we learn to play near-optimally in an MDPU? (By "near-optimal play", we mean near-optimal play in the actual MDP.) In the earlier work, near-optimal play involved learning the effects of actions (that is, the transition probabilities induced by the action). In our setting, the DM still has to learn the transition probabilities, but also has to learn what actions are available.

Perhaps not surprisingly, we show that how effectively the DM can learn optimal play in an MDPU depends on the probability of discovering new actions. For example, if it is too low, then we can never learn to play near-optimally. If it is a little higher, then the DM can learn to play near-optimally, but it may take exponential time. If it is sufficiently high, then the DM can learn to play near-optimally in polynomial time. We give an expression whose value, under minimal assumptions, completely characterizes when the DM can learn to play optimally, and how long it takes. Moreover, we show that a modification of the RMAX algorithm (that we call URMAX) can learn to play near-optimally if it is possible to do so. Note that, for the settings of parameters for which URMAX learns to play near-optimally in polynomial time, the policy that it learns to play is quite simple; it is in fact linear in the relevant parameters.

There is a subtlety here. Not only might the DM not be aware of what actions can be performed in a given state, she may be unaware of *how many* actions can be performed. Thus, for example, in a state where she has discovered five actions, she may not know whether she has discovered all the actions (in which case she should not explore further) or there are more actions to be found (in which case she should). All our lower bounds and impossibility results

hold even when the DM knows that there is only one action to be discovered. (For example, if the action to be discovered is a proof that P ≠ NP, the DM may know that the action has a high reward; she just does not know what that action is.) On the other hand, URMAX works even if the DM does not know how many actions there are to be discovered.

There has been a great deal of recent work on awareness in the game theory literature (see, for example, [12; 13; 19; 24]). There has also been work on MDPs with a large action space (see, for example [9; 23]), and on finding new actions once exploration is initiated [1]. None of these papers, however, considers the problem of learning in the presence of lack of awareness; we believe that we are the first to do so.

The rest of the paper is organized as follows. In Section 2.2, we review the work on learning to play optimally in MDPs. In Section 2.3, we describe our model of MDPUs. We give our impossibility results and lower bounds in Section 2.4. In Section 2.5, we present a general learning algorithm (adapted from R-MAX) for MDPU problems, and give upper bounds. We conclude in Section 2.7.

2.2 Preliminaries

In this section, we review the work on learning to play optimally in MDPs and, specifically, Brafman and Tennenholtz's R-MAX algorithm [4].

2.2.1 MDPs

An MDP is a tuple $M = (S, A, g_A, P, R)$, where S is a finite set of states; A is a finite set of actions; $g_A : S \to 2^A$, where $g_A(s)$ is the set of actions that can be performed at state s; $P : \cup_{s \in S}(\{s\} \times S \times g_A(s)) \to [0, 1]$ is the transition probability function, where $P(s, s', a)$ is the probability of going to state s' given that the DM is in state s and action a is used; and $R : \cup_{s \in S}(\{s\} \times S \times g_A(s)) \to \boldsymbol{R}$ is the reward function, where $R(s, s', a)$ gives the reward for playing action a at state s and transition to s'. Since P is a (conditional) probability function, we have $\sum_{s' \in S} P(s, s', a) = 1$ for all $s \in S$ and $a \in A$.

The expected average reward: A *policy* for an MDP (S, A, g_A, P, R) is a function from histories to actions in A. (Thus, we are implicitly restricting to *deterministic* policies here.) Given an MDP $M = (S, A, g_A, P, R)$, let $U_M(s, \pi, T)$ denote the expected T-step undiscounted average reward of policy π started in state s—that is, the expected total reward of running π for T steps, divided by T. Policy π is an *optimal T-step policy starting at s* if $U_M(s, \pi, T) \geq U_M(s, \pi', T)$ for all policies π'.

We would like to define $U_M(s, \pi)$ as $\lim_{T \to \infty} U_M(s, \pi, T)$; however, for an arbitrary policy π, this limit may not exist. However, note that $U_M(s, \pi, T)$ is bounded from below (by $\min_{s,s',a} R(s, s', a)$). Thus, although $\lim_{T \to \infty} U_M(s, \pi, T)$ may not exist,

$$\liminf_{T \to \infty} U_M(s, \pi, T) = \lim_{T \to \infty} \inf_{T' > T} U_M(s, \pi, T')$$

does; thus, following Brafman and Tennenholtz [4], we define $U_M(s, \pi) = \liminf_{T \to \infty} U_M(s, \pi, T)$. (Of course, $\liminf_{T \to \infty} U_M(s, \pi, T) = \lim_{T \to \infty} U_M(s, \pi, T)$ if the limit exists.) Finally, let $U_M(\pi) = \min_{s \in S} U_M(s, \pi)$.

The mixing time: For a policy π such that $U_M(\pi) = \alpha$, it may take a long time for π to get an expected reward of α. For example, if getting a high reward involves reaching a particular state s^*, and the probability of reaching s^* from some state s is low, then the time to get the high reward will be high. To deal with this, Kearns and Singh [31] argue that the running time of a learning algorithm should be compared to the time that an algorithm with full information takes to get a comparable reward. Define the *ϵ-return mixing time of policy π* to be the smallest value of T such that π guarantees an expected reward of at least $U_M(\pi) - \epsilon$; that is, it is the least T such that $U_M(s, \pi, t) \geq U_M(\pi) - \epsilon$ for all states s and times $t \geq T$. Let $\Pi(\epsilon, T)$ consist of all policies whose ϵ-mixing time is at most T. Let $\text{Opt}(M, \epsilon, T) = \max_{\pi \in \Pi(\epsilon,T)}(\min_{s \in S} U_M(s, \pi, T))$. Thus, $\text{Opt}(M, \epsilon, T)$ is the best average reward over T steps that can be guaranteed by a policy whose ϵ-mixing time is T, no matter which state it starts in.

Paths: Let $M = (S, A, g_A, P, R)$ be an MDP, and let π be a policy for M. A *path* in M is a sequence ρ of state-action pairs followed by a final state:

$$\rho = (s_1, a_1), (s_2, a_2), \ldots, (s_T, a_T), s_{T+1}.$$

The probability that ρ is traversed in M when starting at state s and executing policy π is

$$\Pr_M^{\pi,s}(\rho) = \delta_{s_1}(s)\Pi_{k=1}^{T} P(s_k, s_{k+1}, a_k)\delta_{\pi(s_k)}(a_k),$$

where $\delta_x(y) = 1$ if $x = y$, and 0 otherwise. The undiscounted average reward of path ρ in M is

$$U_M(\rho) = \frac{1}{T}\sum_{k=1}^{T} R(s_k, s_{k+1}, a_k).$$

Define a *T-path* to be a path of length T, that is, a sequence of T state-action pairs followed by a final state.

Definition 2.2.1: A policy π is (ϵ, T)-*optimal* for an MDP M if $U_M(\pi) \geq \text{Opt}(M, \epsilon, T) - \epsilon$, and a reward r is (ϵ, T)-*optimal* for M if $r \geq \text{Opt}(M, \epsilon, T) - \epsilon$. □

2.2.2 The RMAX algorithm

RMAX [4] is a model-based near-optimal polynomial-time reinforcement learning algorithm for zero-sum stochastic games, which also directly applies to standard MDPs. RMAX assumes that the DM knows all the actions that can be played in the game, but needs to learn the transition probabilities and reward function associated with each action. It does not assume that the DM knows all states; new states might be discovered when playing actions at known states. On the other hand, it does assume that the DM knows the number of states. RMAX does not assume that the DM knows the ϵ-mixing time T. However, following Brafman and Tennenholtz [4], we first present the algorithm under the assumption that T is known, and then show how to remove this assumption.

RMAX uses an "explore or exploit" approach that is biased towards exploration. Given an MDP M, let $R^*_{\max}$ be the maximum possible reward in M, let $R_{\max}$ be an upper bound on $R^*_{\max}$, and let T be an estimate of the ϵ-return mixing time. RMAX gets T and $R_{\max}$ as inputs, as well as parameters ϵ, δ, $|S|$ (the total number of states in the MDP), and s_0 (the start state).

RMAX is described formally below. We briefly discuss some of its key features here; we refer the reader to [4] for details:

- **The initial approximation:** RMAX proceeds iteratively. At each step, t gets a better and better approximation to the true MDP M and the optimal policy π. The variable M' is used to keep track of the current approximation to M, and the variable π' is used to keep

track of the current approximation to π. M' is initialized to M^0, where M^0 has a dummy state called s_d, and its transition and reward functions are trivial: when an action a is taken in any state s (including the dummy state s_d), with probability 1 the DM transits to s_d and gets reward $R_{\max}$. The algorithm then sets π' to an optimal T-step policy for M' starting at the DM's current state. For an MDP with N states and k actions, such a policy can be computed using standard techniques (*value iteration*) in time $O(N^2kT)$ [47].

- **Updating the approximation:** Each iteration of RMAX goes through the outer loop in the RMAX algorithm once (see the psuecode in Figure 2.1). In each iteration, RMAX plays π', the current approximation to π, for T steps or until some state-action pair (s, a) becomes *known*, whichever happens first. A state-action pair becomes known if it is played sufficiently many times, as explained below. As π' is played, statistics are collected. Specifically, for each state s that is reached, the algorithm keeps track of the number of times that a transition from s is taken, and the reward associated with that transition. If the iteration terminates because the state-action pair (s, a) has just become known, then the approximation M' is updated by replacing the reward of (s, a) with the observed reward.That is, if (s, a) has just become known, then we take $R(s, s', a)$ to be the observed reward for playing a in state s and transitioning to s', and take $P(s, s', a)$ to be the fraction of times that a transition from s to s' was observed when playing a in state s.

  At the end of each iteration, RMAX sets π' to be a T-step optimal policy for the updated M', and continues to the next iteration.

- **Becoming known:** A state-action pair (s, a) becomes known when it has been played enough time to learn a good approximation to the transition probabilities for action a in state s, given that T is the ϵ-mixing time and $R_{\max} \geq R^*_{\max}$, where "good" means with probability at least $1 - \frac{\delta}{3Nk^2}$, all the estimated transition probabilities are within $\frac{\epsilon}{2NTR_{\max}}$

of the actual transition probabilities. Brafman and Tennenholtz show that the required number of times to do this is polynomial in T, $1/\epsilon$, and $1/\delta$. This number is computed in the algorithm as $H_1(T)$. Thus, a state-action pair becomes known when it has been played $H_1(T)$ times.

- **Termination:** The algorithm terminates when it has run sufficiently many iterations to attain an expected reward $\geq Opt(M, \epsilon, T) - 2\epsilon$ with probability at least $1 - \delta$. The number of iterations required is denoted $H_4(T)$ in the algorithm; again, it is polynomial in T, $1/\delta$, and $1/\epsilon$. $H_4(T)$ is computed in terms of two other quantities, $H_2(T)$ and H_3. To understand $H_2(T)$ and H_3, it is useful to partition the iterations of RMAX into *exploration iterations* and *exploitation iterations*. If the policy π'_i being played in iteration i is (ϵ, T)-optimal for M, then i is an exploitation iteration (since the DM gets a near-optimal expected reward in such iterations); otherwise, i is an exploration iteration. Brafman and Tennenholtz [4] show that, in exploration iterations, RMAX explores (i.e., visits) an unknown state-action pair with probability at least $\epsilon/R_{\max}$. Of course, the DM does not know in general which are the exploration iterations and which are the exploitation iterations. $H_2(T)$ is the number of exploration iterations required so that, with high probability, RMAX learns an (ϵ, T)-optimal policy; H_3 is the number of exploitation iterations required so that, with high probability, the average reward of these H_3 iterations is at least $\text{Opt}(M, \epsilon, T) - 3\epsilon/2$. During the $H_2(T)$ exploration iterations, the DM may be getting a low reward. Thus, extra exploitation iterations are required to make up for this period of low reward. Brafman and Tennenholtz [4] showed that for $0 < \theta < 1$, in order to make up the potential loss during the $H_2(T)$ exploration iterations, so as to get the average expected reward to within θ of the average reward that is available during the exploitation iteration, we need an extra $H_2(T)R_{\max}/\theta$ exploitation iterations. In our case, $\theta = \epsilon/2$, so that the average reward is at least $\text{Opt}(M, \epsilon, T_M) - 3\epsilon/2 - \epsilon/2 = \text{Opt}(M, \epsilon, T_M) - 2\epsilon$. Thus,

we need $H_2(T)(1 + \frac{R_{\max}}{\epsilon/2}) + H_3$ iterations altogether.

RMAX($|S|, |A|, R_{\max}, T, \epsilon, \delta, s_0$):

$s_c := s_0$
$H_1(T) := \max((\lceil \frac{4|S|\,T\,R_{\max}}{\epsilon} \rceil)^3, \lceil -6 \ln^3(\frac{\delta}{6|S|\,|A|^2}) \rceil) + 1$
$H_4(T) = H_2(T)(1 + \frac{R_{\max}}{\epsilon/2}) + H_3$, where
  $H_2(T) = \max((k^2 N(H_1(T))^3, 6(\ln \frac{3}{\delta})^3)$, $H_3 = \max((\frac{2R_{\max}}{\epsilon})^3, 6(\ln \frac{3}{\delta})^3)$
$M' := M^0$ (the initial approximation described in the main text)
Compute an optimal T-step policy π' for M' starting at s_c
Repeat $H_4(T)$ times:
  Play π' starting at s_c for T steps, or until some (s, a) has just become *known* and record the transitions and rewards
  if (s, a) has just become *known* (i.e., has been played $H_1(T)$ times)
    then update M' so that the transition probabilities for (s, a) are the observed frequencies, and the rewards for playing (s, a) are those that have been observed (as described in the main text)
  s_c := the DM's current state
  Compute an optimal T-step policy π' for M' starting at s_c
Return π'

Figure 2.1: The RMAX algorithm.

Brafman and Tennenholtz [4] show that RMAX($|S|, |A|, R_{\max}, T, \epsilon, \delta, s_0$) achieves an expected reward of at least $\text{Opt}(M, \epsilon, T) - 2\epsilon$ with probability greater than $1 - \delta$, no matter what state s_0 the DM starts at, in time polynomial in $|S|$, $|A|$, $R_{\max}$, T, $1/\delta$, and $1/\epsilon$. What makes RMAX work is that, in each iteration, it either uses a near-optimal policy with respect to the real model (i.e., it exploits), or it visits an unknown state-action pair with constant probability (i.e., it explores). Brafman and Tennenholtz [4] showed that by running RMAX for $H_2(T)$ exploration iterations, with probability $1 - 2\delta/3$, RMAX learns an (ϵ, T)-optimal policy. In addition, as discussed above, it takes RMAX at most H_3 exploitation iterations until, with probability $1 - \delta/3$, the average reward of these H_3 exploitation iterations is at least $\text{Opt}(M, \epsilon, T) - 3\epsilon/2$; and it takes a further $\frac{H_2(T)R_{\max}}{\epsilon/2}$ iterations until the algorithm attains an overall average reward of at least $\text{Opt}(M, \epsilon, T) - 2\epsilon$. Thus, after $H_4(T)$ iterations, with probability $1 - \delta$, RMAX attains

an expected reward of at least $\text{Opt}(M, \epsilon, T) - 2\epsilon$.

All this assumes that the ϵ-return mixing time T is known; it is an argument to the algorithm. If the ϵ-return mixing time T is not known, we run the algorithm with $T = 1$, then $T = 2$, $T = 3$, and so on. Sooner or later, we hit the right value of T; from then on, RMAX computes a policy that is near-optimal. We expand on this point in Section 2.5.

2.3 MDPs with Unawareness

Intuitively, an MDPU is like a standard MDP except that the player is initially aware of only a subset of the complete set of states and actions. To reflect the fact that new states and actions may be learned during the game, the model provides a special *explore* action. By playing this action, the DM may become aware of actions that she was previously unaware of. The model includes a *discovery probability function* characterizing the likelihood that a new action is discovered. At any moment in game, the DM can perform only actions that she is currently aware of.

Definition 2.3.1: An MDPU is a tuple $M = (S, A, S_0, a_0, g_A, g_0, D, P, R, G_0)$, where

- S is the set of states in the underlying MDP;
- A is the set of actions in the underlying MDP;
- $S_0 \subseteq S$ is the set of states that the DM is initially aware of;
- $a_0 \notin A$ is the *explore* action;
- $g_A : S \rightarrow 2^A$, where $g_A(s)$ is the set of actions that can be performed at s other than a_0 (we assume that a_0 can be performed in every state);

- $g_0 : S_0 \to 2^A$, where $g_0(s) \subseteq g_A(s)$ is the set of actions that the DM is initially aware of (i.e., that the DM can perform) at state s other than a_0 (we assume that the DM is always aware of a_0);
- $D : \boldsymbol{N} \times \boldsymbol{N} \times S \to [0, 1]$ is the discovery probability function. $D(j, t, s)$ gives the probability of discovering a new action by playing a_0 in state $s \in S$ given that there are $j > 0$ actions that have not yet been discovered at state s and a_0 has already been played $t - 1$ times in s since the last new action was discovered, or since the beginning, if no new action has yet been discovered (see below for further discussion);
- $P : \cup_{s \in S}(\{s\} \times S \times g_A(s)) \to [0, 1]$ is the transition probability function (as usual, we require that $\sum_{s' \in S} P(s, s', a) = 1$ if $a \in g_A(s)$);
- $R : \cup_{s \in S}(\{s\} \times S \times g_A(s)) \to \boldsymbol{R}^+$ is the reward function. We assume without loss of generality that all rewards are non-negative. If not, then we can increase all rewards by a constant so that all rewards become non-negative.[1]
- $G_0 \subseteq \cup_{s \in S_0}(\{s\} \times S_0 \times g_0(s))$. Intuitively, G_0 is the set of tuples for which the DM initially knows the transition probability and reward function. (For simplicity, we assume that the DM knows that transition probability and reward function for the same tuples.) We allow G_0 to be the empty set; this just says that the DM does not initially know the trasition probability or reward for any transitions.

Let $M^u = (S, A, g_A, P, R)$ be the MDP *underlying* the MDPU M. □

Given S_0 and g_0, we define $A_0 = \cup_{s \in S_0} g_0(s)$; that is, A_0 is the set of actions that the DM is initially aware of.

[1] Our definition of MDPU in the earlier version included functions R^+ and R^-, the rewards for performing a_0 and discovering (resp., not discovering) a new action. For ease of exposition, we omit these values here; our theorems do not depend on them in any way.

Just like a standard MDP, an MDPU has a state space S, action space A, transition probability function P, and reward function R.[2] Note that we do not give the transition function for the explore action a_0 above; since we assume that a_0 does not result in a state change (although new actions might be discovered when a_0 is played), for each state $s \in S$, we have $P(s, s, a_0) = 1$. The new features here involve dealing with a_0. We need to quantify how hard it is to discover a new action. Intuitively, this should in general depend on how many actions there are to be discovered, and how long the DM has been trying to find a new action. For example, if the DM has in fact found all the actions, then this probability is clearly 0.

As the notation suggests, while we are allowing the discovery probability $D(j, t, s)$ to depend on the current state s, the number t of times a_0 has been played, and the number j of actions there are to be discovered, we are assuming that it does not depend on anything else. Among other things, this means that we are assuming that it does not depend on what actions are played other than a_0, and the order in which actions have been discovered. For example, suppose that there is a state s where, initially, there are two undiscovered actions, say a_1 and a_2. Suppose that, intuitively, a_1 is easy to discover, and a_2 is hard to discover. In this case, if a_1 has been discovered and not a_2, then we might expect that $D(1, t, s)$ to be low, since a_2 is hard to discover, while if a_2 has been discovered and not a_1, then we might expect $D(1, t, s)$ to be relatively high. We cannot capture this in our present framework; thus, we implicitly assume that all actions are equally hard to discover.

Similarly, it might in principle be the case that it is easier to discover an action a_1 after playing a_3. Again, we are assuming that this is not the case. We are assuming that all that affects the probability of discovering a_1 is how many times the explore action a_0 has been played, and not what other actions are played, or the order in which they are played. While

[2] It is often assumed that the same actions can be performed in all states. Here we allow slightly more generality by assuming that the actions that can be performed is state-dependent, where the dependence is given by g.

all these assumptions are admittedly quite strong, they are reasonable in many settings, and help us gain an understanding of the impact of unawareness. Without such assumptions, even defining the likelihood of discovering a new action seems to be quite difficult.

In order to explain the D function, we need to consider *runs* of the MDPU. Intuitively, a run represents a possible execution of the MDPU, and records the set of states and actions that the DM is aware of at every time step.

Definition 2.3.2: An *extended state* in MDPU $M = (S, A, S_0, a_0, g_A, g_0, D, P, R, G_0)$ is a tuple (S', h, s, a), where (a) $S_0 \subseteq S' \subseteq S$, (b) $h : S' \to 2^A$ is a function such that $g_0(s') \subseteq h(s')$ for $s' \in S_0$ and $h(s') \subseteq g_A(s')$ for all $s' \in S$, (c) $s \in S'$, and (d) $a \in h(s) \cup \{a_0\}$. An *MDPU run* r is a function from time (which we take to range over the natural numbers) to extended states. We require that $r(0) = (S_0, g_0, s, a)$ for some s and a, denote $r(t)$ as (S_t, h_t, s_t, a_t), and require that $h_t(s') = h_{t+1}(s')$ for $s' \neq s_t$, $h_t(s_t) \subseteq h_{t+1}(s_t)$, $S_{t+1} = S_t \cup \{s_{t+1}\}$, and $h_{t+1}(s_t) = h_t(s_t)$ if $a_t \neq a_0$. □

Intuitively, an extended state records what happens at a specific time step, including the action taken by the DM (this is the a component), the state that the DM is in (this is the s component), and the DM's awareness, specifically, the set of states and actions that the DM is aware of at that time step (this is captured by the S' component and the h component; note that h describes what actions that DM is aware of at each state of S'). A run is a sequence of extended states. Initially, the DM is aware of only the states in S_0 and the only actions that the DM is aware of at a state $s' \in S_0$ are the ones in $g_0(s')$. That is why $r(0)$ has the form (S_0, g_0, s, a) for some state s and action a. During a run, the DM's awareness can change only if the DM performs the explore action; moreover, if the explore action is performed at state s at time t, then at time $t + 1$, only the DM's awareness of actions at state s can change. Thus, we require that $h_t(s') = h_{t+1}(s')$ if $s \neq s'$ and that $h_t(s) = h_{t+1}(s)$ if $a \neq a_0$. Finally, the DM becomes aware of a new state only if he reaches it, so $S_{t+1} = S_t \cup \{s_t\}$.

Note that each extended state includes a state-action pair, so a sequence of extended states determines a sequence of state-action pairs. Thus, given an MDPU run r, every segment $r(n_1), \ldots, r(n_2)$ of r determines a path $\rho = (s_{n_1}, a_{n_1}), (s_{n_1+1}, a_{n_1+1}), \ldots, (s_{n_2-1}, a_{n_2-1}), s_{n_2}$ in the MDP underlying the MDPU.

We can understand the D function in terms of a probability on runs. Note that such a probability distribution must incorporate assumptions about how likely it is that a new action will be discovered, given that a_0 has been played. To define this probability, we need some preliminary definitions.

Let $s \in S$, and let $m' \geq m \geq 0$ be integers. For a sequence $\vec{n} = (n_1, n_2, \ldots, n_k)$ of integers with $k \geq 0$, $0 < n_1 < n_2 < \ldots < n_k$, and $0 \leq k \leq |g_A(s) - g_0(s)|$,[3] let $\Gamma^{\vec{n}}_{s,m,m'}$ be the subset of MDPU runs where the jth new action at state s is discovered the n_jth time that a_0 is played at s, for $j = 1, \ldots, k$, a_0 is played at state s at least $n_k + m'$ times, and the $(k+1)$st new action is not discovered on or before the (n_k+m)th time that a_0 is played at s. Note that by $0 < n_1 < \ldots < n_k$, we are implicitly assuming that the DM can discover at most one new action at any given time.[4] Also note that if $k = 0$, then $\vec{n}$ is just the empty sequence ( ); $\Gamma^{()}_{s,m,m'}$ is the subset of MDPU runs where a_0 is played at s at least m' times, and no new action is discovered on or before the mth time that a_0 is played at s.

For $\vec{n} = (n_1, n_2, \ldots, n_k)$, let $\Gamma^{\vec{n}}_{s,-1}$ consist of the runs where the ith new action at state s is discovered the n_ith time that a_0 is played at s, for $i = 1, \ldots, k$ (with no constraints on how many times a_0 is played at s after the kth new action is discovered, or on when the $(k+1)$st action

[3] In this paper, we follow the convention of using $|A|$ to denote the size of set A. Recall that both $g_A(s)$ and $g_0(s)$ are sets, so $|g_A(s) - g_0(s)|$ is the size of $g_A(s) - g_0(s)$.

[4] By changing the constraint of $0 < n_1 < \ldots < n_k$ to $0 \leq n_1 \leq \ldots \leq n_k$, $D(j, t, s)$ allows multiple actions to be discovered at a single step. All of our results continue to hold even if multiple action discovery is allowed at a single step. However, since this add no insight, and makes the notation more clustered, we focus on our current setting where at most one new action can be discovered at each time step.

is discovered). Let $\Gamma^{\vec{n}}_{s,-1,m'}$ be the set of runs in $\Gamma^{\vec{n}}_{s,-1}$ where a_0 is played at least $n_k + m'$ times at s (with no constraints on when the $(k+1)$st new action is discovered). Let $\Gamma^{()}_{s,-1,m'}$ be the set of all MDPU runs where a_0 is played at least m' times. Note that $\Gamma^{()}_{s,-1,m'} = \Gamma^{()}_{s,0,m'}$. Moreover, if $k = 0$, then $\Gamma^{()}_{s,-1}$ is the set of runs where no new actions at s are discovered before the DM plays the explore action for the first time; thus, $\Gamma^{()}_{s,-1}$ consists of all MDPU runs.

Given a state $s_0 \in S$, let L_{s_0} be the set of MDPU runs starting at state s_0. A probability Pr on L_{s_0} is *compatible with D* if, for all integers j with $0 \leq j \leq |g_A(s) - g_0(s)|$, all integers u with $u \geq 0$, all integers t with $t \geq 0$, and all increasing sequences $\vec{n} = (n_1, n_2, \ldots, n_k)$ where $k = |g_A(s) - g_0(s)| - j$, $n_i \in \boldsymbol{N}$, and $\Pr(\Gamma^{\vec{n}}_{s,t-1,t+u} \cap L_{s_0}) > 0$, we have

$$D(j,t,s) = \Pr(\Gamma^{\vec{n}\cdot(n_k+t)}_{s,-1,u} \mid \Gamma^{\vec{n}}_{s,t-1,t+u} \cap L_{s_0}), \tag{2.1}$$

where $\vec{n} \cdot \vec{n}'$ is the result of appending $\vec{n}'$ to the end of $\vec{n}$ (so that $\vec{n} \cdot (n_k + t)$ is the sequence $(n_1, \ldots, n_k, n_k + t)$). Note that if $\vec{n} = (\,)$, then $\vec{n} \cdot \vec{n}' = \vec{n}'$.

Thus, $D(j,t,s)$ is the probability of discovering the $(k+1)$st new action at s, conditional on starting at s_0, having found the ith new action at s the n_ith time that a_0 is played, for $i = 1, \ldots, k$, and a_0 being played at least $n_k + t + u$ times. For (2.1) to hold, it must be the case that this conditional probability is independent of the sequence $\vec{n}$, the state s_0, and the integer u. In the rest of this paper, we consider only distributions Pr on L_{s_0} that are compatible with D, so we are assuming that these independence assumptions do in fact hold. These independence assumptions are already implicit in the notation $D(j,t,s)$. If the probability of discovering a new action depends only on the state s, the number j of remaining undiscovered actions, and the number t of times a_0 has been played since the last new action was discovered, then, among other things, it cannot depend on when the previous actions were discovered, or depend on the initial state that the DM starts in, or depend on future events such as how many times a_0 is played in the future.

These assumptions are reasonable in many applications of interest. For example, given a fixed potential action space, if we use an exploration strategy that explores the potential action space uniformly at random, then the probability of discovering a new action at the tth time that a_0 is played after the ith new action is discovered is indeed independent of when the previous actions were discovered, the start state, and how many times a_0 will be played in the future, no matter what policy is used. (Note that a policy decides when to explore, but cannot determine the likelihood of the outcomes of exploration.)

However, the assumptions are not always reasonable. For example, when playing a video game, compare the situation where the DM discovers the first three actions the first, second, and third times that she plays the explore action to one where she discovers the first three actions the 100th, 3,000th, and 10,000th times that she plays the explore action. In the first situation, the DM is likely to feel optimistic that she will discover a fourth new action soon, while in the second, the DM is likely to become discouraged, and consider it highly unlikely that she will discover the fourth new action any time soon after the third. The history of when previous actions were discovered may psychologically affect a DM's discovery probability. If the DM is insensitive to history (for example, if the DM is a robot), then the assumption that $D(j, t, s)$ is independent of when previous actions were discovered seems more reasonable.

To summarize the discussion thus far, to make sense of $D(j, t, s)$, we need to assume a probability Pr on L_{s_0}, the runs of the MDPU starting at state s_0. But rather than giving Pr explicitly when describing the MDPU, we just define $D(j, t, s)$. The D function can be viewed as putting constraints on Pr. There are two other sources of constraints on Pr: the policy used, and the transition probability function. To make the first constraint precise, recall that a policy maps histories to actions. Up to now, we have not defined a history in an MDPU formally. We now do so.

Definition 2.3.3: A *history x of length T* in an MDPU M is a prefix of a run in M; formally, it is a sequence of T extended states followed by a tuple (S'_T, h_T, s_T) such that there exists a run l of M such that $x(i) = l(i)$ for all $i \in [0, T-1]$, and $l(T) = (S'_T, h_T, s_T, a)$ for some action $a \in h_T(s_T)$. □

For a history x of length t in an MDPU M, let $C^{x,a}$ be the set of MDPU runs r with prefix x such that a is the action in $r(t)$ (i.e., a is the action played at time t in r). A probability Pr on L_{s_0} is *compatible with a policy π for M* if $\Pr(C^{x,a} \cap L_{s_0}) = 0$ for all histories x such that $\pi(x) \neq a$.

The second source of constraints on Pr is the transition probability function P of the MDP underlying the MDPU M. Given a history x, let $\Pr(x)$ be the probability of the set of runs extending x. A probability Pr on L_{s_0} is *compatible with a transition probability function P* if, for all $t > 0$, all histories x of length t, and all actions $a \neq a_0$, if $x(t) = (S', h, s)$, then $\Pr(y \mid x) = P(s, s', a)$ where y is the history of length $t+1$ such that $y(i) = x(i)$ for all $i = 0, \ldots, t-1$, $y(t) = (S', h, s, a)$, and $y(t+1) = (S' \cup \{s'\}, h, s')$. The constraint says that Pr is compatible with P if the probability of transitioning between extended states if an action $a \neq a_0$ is played is determined by the transition probability described by P. This constraint does not consider transitions between extended states induced by a_0; that is described by D. Given a state s_0, the discovery probability function D and the transition probability function P of an MDPU M, and a policy π for M, it is easy to see that there is a unique probability distribution Pr on L_{s_0} compatible with D, π, and P. Thus, we talk sometimes about *the* probability on runs of an MDPU M determined by M and a policy π.

Now that we have explained the D function, it is worth considering what we should assume about it. The only assumption that we make is that $D(j, t, s)$ is nondecreasing as a function of j: with more actions available, it is easier to find a new one. This seems like a natural assumption for all our applications. How $D(j, t, s)$ varies with t depends on the problem. For

example, if the DM is searching for the on/off button on her new iPhone, which is guaranteed to be found in a limited surface area, then $D(j, t, s)$ should increase as a function of t. The more possibilities have been eliminated, the more likely it is that the DM finds the button when the next possibility is tested. On the other hand, if the DM is searching for a proof, then the longer she searches without finding one, the more discouraged she gets; she will believe that it is more likely that no proof exists. In this case, we would expect $D(j, t, s)$ to decrease as a function of t. Finally, if we think of the *explore* action as doing a random test in some space of potential actions, the probability of finding a new action is a constant, independent of t.

In stating our results, we need to be clear about what the inputs to an algorithm for near-optimal play are. We assume that S_0, g_0 and G_0 are always part of the input to the algorithm. The reward function R is not part of the input; rather, it is part of what is learned. (We could equally well assume that the values of R for all the actions and states that the DM is aware of are included in the input; this assumption would have no impact on our results.) Brafman and Tennenholtz [4] assume that the DM is given a bound on the maximum reward, but later show that this information is not needed to learn to play near-optimally in their setting. Our algorithm URMAX does not need to be given a bound on the reward either. Perhaps the most interesting question is what the DM knows about A and S. Our lower bounds and impossibility result hold even if the DM knows $|S|$ and $|g_A(s)|$ for all $s \in S$. On the other hand, URMAX requires neither $|S|$ nor $|g_A(s)|$ for $s \in S$. That is, when something cannot be done, knowing the size of the set of states and actions does not help; but when something can be done, it can be done without knowing the size of the set of states and actions.

Formally, we can view the DM's knowledge as the input to the learning algorithm. An MDP M is *compatible with the DM's knowledge* if all the parameters of M agree with the corresponding parameters that the DM knows about. If the DM knows only S_0, g_0, and G_0

(we assume that the DM always knows at least this), then every MDP (S', A', g', P', R') where $S_0 \subseteq S'$, $g_0(s) \subseteq A'(s)$, $P(s, s', a) = P'(s, s', a)$ and $R(s, s', a) = R'(s, s', a)$ for all $(s, s', a) \in G_0$ is compatible with the DM's knowledge. If the DM also knows $|S|$, then we must have $|S'| = |S|$; if the DM knows that $S = S_0$, then we must have $S' = S_0$. We use $R^*_{\max}$ to denote the maximum possible reward. Thus, if the DM knows $R^*_{\max}$, then in a compatible MDP, we have $R(s, s', a) \leq R^*_{\max}$ for all $s, s' \in S$ and $a \in g_A(s)$, with equality holding for some transition. (The DM may just know a bound on $R^*_{\max}$, or not know $R^*_{\max}$ at all.) Brafman and Tennenholtz essentially assume that the DM knows $|A|$, $|S|$, and an upper bound on $R^*_{\max}$. They mentioned that they believe the assumption that the DM knows an upper bound of $R^*_{\max}$ can be removed. It follows from our results that, in fact, the DM does not need to know any of $|A|$, $|S|$, or $R^*_{\max}$, or even a bound on these quantities.

Our theorems talk about whether there is an algorithm for a DM to learn to play near-optimally given some knowledge. We define "near-optimal play" by extending the definitions of Brafman and Tennenholtz [4] and Kearns and Singh [31] to deal with unawareness. In an MDPU, a policy is again a function from histories to actions, but now the action must be one that the DM is aware of at the last state in the history. The DM *can learn to play near-optimally given a state space S_0 and some other knowledge* if, for all $\epsilon > 0$, $\delta > 0$, T, and $s \in S_0$, the DM can learn a policy $\pi_{\epsilon,\delta,T,s}$ such that, for all MDPs M compatible with the DM's knowledge, there exists a time $n_{M,\epsilon,\delta,T}$ such that, with probability at least $1 - \delta$, $U_M(s, \pi_{\epsilon,\delta,T,s}, t) \geq \mathrm{Opt}(M, \epsilon, T) - \epsilon$ for all $t \geq n_{M,\epsilon,\delta,T}$.[5] The DM *can learn to play near-optimally given some knowledge in polynomial (resp., exponential) time* if, there exists a polynomial (resp., exponential) function f of six arguments such that we can take $n_{M,\epsilon,\delta,T} = f(T, |S|, |A|, R^*_{\max}, 1/\epsilon, 1/\delta)$.

[5]Note that we allow the policy to depend on the start state. However, it must have an expected reward that is close to that obtained by M no matter what state M is started in.

We close this section with two detailed examples of how MDPUs can be used to model real-world situation.

Example 2.3.4: Consider a video game, where the game area is in the shape of a box. Roughly speaking, playing the game involves firing colored balls into the game area and clearing them from the game. Balls come in three colors: red, yellow, and green. When the game starts, the box has a red ball at the center of the box (as shown in Figure 2.2).

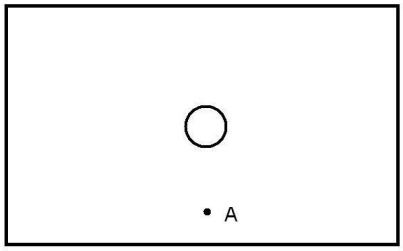

Figure 2.2: Video game scene.

The user controls the game using a mouse. When the user left-clicks the mouse and the cursor is inside the game area (call such actions *slc* actions, for single left-click), a colored ball is fired from the point marked *A* in the figure in the direction of the cursor; if the cursor is outside the game area, then a left-click has no effect. Balls and walls are sticky, so if a ball touches another ball, it sticks to that ball, and if a ball runs into a wall, it sticks to the wall. However, if two balls of the same color touch, then they both disappear from the game. Once a ball sticks to another ball or a wall, it stays there unless/until it is cleared from the game as a result of being touched by a ball of the same color. The color of the next ball to be fired is chosen at random. The user knows the color, so he can plan where to fire the next ball. The user wins the game if removes all balls from the game area; he loses if the game area fills up with balls.

There is one more possible action, which the user is initially not aware of: if the user double right-clicks the mouse (call this action *drc*), the color of the next ball changes to a random different color; double left-clicks and single right-clicks of the mouse have no effect. *Unaware* here means "not on the user's radar screen"; that is, initially, it does not occur to the user that *drc* is a possible move in the game. (If it seems implausible that it does not occur to the user that a double right-click could be a move in the game, consider a triple right-click instead.) Clearly, the easiest way to win the game is to change the color of first ball to red using double right-clicks, then to fire the ball at the red ball in the center of the game area, so that both balls disappear. In order to do so, the player needs to discover the *drc* action.

The game can be formally described as an MDPU. The state space, action space, and transition probabilities are almost immediate from the informal description. Thus, we focus here on elements that are new/different from the ones in the MDP model. We think of the explore action a_0 as an abstraction of trying out an action a^* and deciding whether a^* counts as a real action in the game. For example, the user may try right-clicking the mouse in various locations, or simultaneously pressing the left and right buttons of the mouse in various locations to see if they have any effect. The user may also try double right-clicking the mouse and not notice that this results in a change of color; that does not count as discovering a new action. The user adds *drc* to the set of actions only when he realizes that it actually has an effect.[6]

Note that in this game it is not even clear what the the potential action space A^* should be. An arbitrary sequence of mouse clicks (left or right) together with an arbitrary cursor movement could be a potential action. Moreover, if there is a video sensor, a head movement or a facial expression could also be a potential action. Although, in principle, we could describe the problem as an MDP with some large action space A^*, such a description is unlikely to be

[6]We are implicitly assuming that the user does not make mistakes here; for example, he does not mistakenly view double left-clicking as an action. if he did make such a mistake though, it would not be a major problem. He could add double left-clicking to the set of actions, and then discover that it has a trivial transition function.

useful, particularly since the exact choice of A^* is hard to motivate. It is not obvious why one choice for A^* is better than another, and the appropriate choice may be user dependent. (Certainly different users might try different actions.) Moreover, since a reasonable A^* could be huge, standard techniques for finding optimal policies for MDPs, which run in time polynomial in the number of actions, are likely to be infeasible. MDPUs provide arguably a more useful model, both because we are spared the need of deciding what A^* should be, and because (as we shall show), there are techniques for finding optimal policies that run in time polynomial in the number of actual actions, rather than the potential actions.

Here is a formal description of the game as an MDPU:

- $S = \{s : s = (color, \{(color, x, y), \ldots\})\}$. That is, the state of the game is described by the color of the next ball and a set of tuples describing the color and the position of the balls currently in the game area. For example, when the game starts, if the color of the next ball is yellow, then the state of the game is $(yellow, \{(red, 0, 0)\})$, where, for convenience, we take $(0, 0)$ to be the coordinates of the center of the game area. Given a state s, we use $s[1]$ to denote the color of the next ball, and $s[2]$ the game-area description, that is, the location and color of all the balls in the game area.
- $A = A_0 \cup \{drc\}$ where A_0 consists of all actions of the form $slc(x, y)$, which is a single left-click with the cursor in position (x, y), and drc is a double right-click of the mouse.
- $S_0 = S$.
- $g_A(s) = A$ for all $s \in S$.
- $g_0(s) = A_0$ for all $s \in S_0$. That is, the user is initially unaware of the double right-click action, but is aware of all the single left-click actions.
- a_0: The process of trying out a potential action, and deciding whether it is a real action in the game.

- $P(s, s', slc(x, y)) = 1/3$ if $s'[2]$ is the unique game-area position that results from $s[2]$ and action $slc(x, y)$ given the rules above (the $1/3$ arises because the next color is chosen randomly); otherwise, $P(s, s', slc(x, y)) = 0$.
- $P(s, s', drc) = 1/2$ if $s'[2] = s[2]$ and $s'[1] \neq s[1]$; otherwise $P(s, s', drc) = 0$.
- There is some flexibility in how we define the reward function R. For simplicity, we assume that it considers only whether a position is a winning or a losing position, although we could take a more refined choice. Thus, we take $R(s, s', a) = 10$ if $s'[2]$ is a winning position (i.e., there are no balls in the game area), we take $R(s, s', a) = -10$ if $s'[2]$ is a losing position (i.e., the game area is full of balls), and otherwise, we take $R(s, s', a) = 0$.
- Recall that $D(j, t, s)$ is the probability that the user discovers a new action the tth time that a_0 is performed, given that $j > 0$ actions have not yet been discovered and the user has not discovered a new action the previous $t - 1$ times that a_0 was performed. Since there is only one action to be discovered in this game, namely, drc, we care only about $D(1, t, s)$, that is, the likelihood of the user discovering the double right-click action if he has not discovered it in his first $t - 1$ exploration moves. Clearly this depends on the user (and perhaps on how often the user has played the game before), although we might expect that, for large values of t, $D(1, t, s)$ is likely to be close to 0, since if the user has not discovered the double right-click action after exploring for a large number of moves, it is likely that the game area is full, and the user has lost the game.

  Experiments could be carried out to determine plausible values for $D(1, t, s)$. A video game designer might well be interested in this probability, since the interest in the game depends on it. If the probability is too low, perhaps a double left-click action might be used instead, since it might be easier to discover; if it is too high, a triple right-click might be required.

Example 2.3.5: When building a robotic remote control (RC) car, we need to learn a policy that enables the car to drive on a desired path successfully. (See Figure 2.3 for an example path.) Some parts of these paths are smooth; the car can follow them using simple familiar actions, such as smoothly turning the steering wheel. Other parts of the path involve sharp turns. The best way to deal with them is often by *drifting*, a driving technique frequently used in car races to make fast sharp turns via slipping. Drifting requires taking a series of precisely-timed actions.

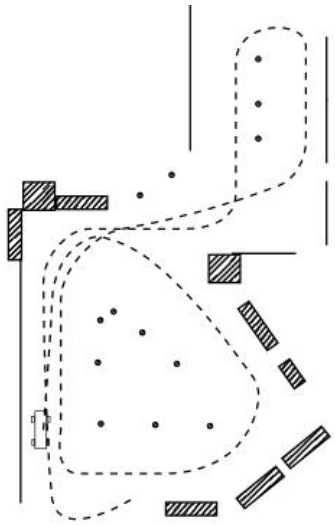

Figure 2.3: An example path for RC car driving.

Suppose that we preprogram the RC car with a few basic standard actions, but expect it to learn how to perform drifting actions. Then the problem of driving through a predefined path can be viewed as an MDPU, where the car is unaware of the drifting actions. The actions that the robotic car is aware of (the actions that are preprogrammed) include going straight forward or backward at a certain speed, or turning at a fixed slow speed at various angles (which results in turns with a large radius, rather than sharp turns). The state of the car includes its position,

orientation, velocity, and angular velocity. The goal is to let the car navigate the path as quickly as possible, while staying on the path (there are penalties for going off path). Since all paths contain sharp turns that are impossible to complete while staying in path without learning drifting actions, the car gets a low reward if only basic actions are used.

In this game, the space A^* of potential actions is somewhat clearer than in the videogame. Roughly speaking, it could consist of arbitrary car-control sequence over a certain interval of time. There are issues of the level of granularity at which to model actions, and how long an interval of time we should consider, but these can be dealt with. Once we have fixed A^*, the problem can be modeled as an MDP using A^* as the action set. But again, A^* is huge. A^* does have some structure that can be exploited—we can view the actions in A^* as high-level actions that are composed of a sequence of basic actions. There has been some work that deals with such structured sets of actions; see, for example, [52]. Thinking in terms of MDPUs gives us an alternative approach. We take the space of in the MDPU to be the "interesting" actions; intuitively, these are the ones that are useful for the driving task. The set of useful actions is much smaller than A^*. Thus, the algorithms we use will be much more efficient than comparable MDP algorithms. Of course, we cannot explicitly write down the action space of the MDPU; we do not know in advance what the useful actions are. This does not prevent us from using the framework (or the URMAX algorithm that we present later). More importantly, by modeling things in terms of MDPUs, we may get a deeper insight into why certain problems are harder to solve than others; specifically, the form of $D(j,t)$ may help explain why learning an optimal policy requires polynomial, exponential, or infinite time.

We now consider some elements of the formal model. As in the video game example, we think of the explore action a_0 here as an abstraction of trying out a potential action a^* and deciding whether a^* counts as a useful action for car control. For example, the car may try

spinning itself with maximum throttle, or speeding up and down very quickly, and find that these lead it to lose control of the drive, and are thus not useful. On the other hand, drifiting actions that allow the car to quickly turn around sharp corners are useful. We are implicitly assuming that when the car tries a potential action, it is able to detect whether the action is useful or not. The robotic car could use criteria predefined by the robot designer to do this; for example, it could take an action to be useful if it allows the car to complete certain path sections quickly. Alternatively, a human could supervise the exploration process, and tell the robot which actions are useful.

Here $D(j, t, s)$ is the probability for the car to discover a new useful action at the tth time that a_0 is performed, given that $j > 0$ actions have not yet been discovered, and the car has not discovered a new action in the previous $t - 1$ times that a_0 was performed. The actual value of $D(j, t, s)$ depends on the exploration method the car uses to select the next potential action to be explored in a_0; different exploration methods result in different values. For example, if the robot explores by just choosing a new action to try at random from the space of potential actions, then the probability will be essentially a (small) constant, whose value depends on the ratio of interesting actions to total actions. However, if the choice of actions is guided by a domain expert, then the discovery probability will be significantly higher.

Again, experiments could be carried out to determine the actual values for $D(j, t, s)$. A robot designer could use the values of this function to decide whether random experimentation suffices or a domain expert is needed.

Here is a formal description of the problem as an MDPU:

- $S = \{s : s = (x, y, \theta, v, \dot{\theta})\}$. That is, a state of the problem is described by the current position (x, y) of the car, the velocity v of the car, its current orientation θ, and its current

angular velocity $\dot{\theta}$.

- A: The car is preprogrammed with the following actions: going straight forward/backward at velocity v, making a non-drifting turn of angle α. Other than that, drifting turns at angles α are also actions in A.
- $S_0 = S$.
- a_0: The process of trying out a potentially useful action, and deciding whether the action is useful; if it is, then add the action into A.
- $g_A = A$.
- g_0: This includes all preprogrammed actions: going straight forward/backward at velocity v, making a non-drifting turn of angle α. That is, the car is aware of all actions in A except the drifting actions.
- $P(s, s', a)$: the probability of traversing from state s to state s' via action a.
- D: the value of the discovery probability depends on the exploration methods being used (that decides the order of the potentially useful action to be experimented). For example, if there are n potential actions, and the car experiments these potential actions uniformly at random, while there are j possible drifting actions to be discovered, then we have $D(j, t, s) = \frac{j}{n}$.
- R: given $s, s' \in S$, if the car is at the end of the track at s', then $R(s, s', a) = 1000 - 100t_1 - t_2$, where t_1 is the amount of time that the car is off track while performing a, and t_2 is the amount of time required for performing action a; otherwise $R(s, s', a) = 100t_1 - t_2$. Therefore, the car is rewarded for finishing the track, is penalized for being off track, and is encouraged to finish the track in a timely manner so as to avoid time penalty from t_2.

2.4 Impossibility Results and Lower Bounds

The ability to estimate in which cases the DM can learn to play optimally is crucial in many situations. For example, in robotics, if the probability of discovering new actions is so low that it would require an exponential time to learn to play near-optimally, then the designer of the robot must have human engineers design the actions and not rely on automatic discovery. We begin by trying to understand when it is feasible to learn to play optimally, and then consider how to do so.

We first show that, for some problems, there are no algorithms that can guarantee near-optimal play; in other cases, there are algorithms that learn to play near-optimally, but require at least exponential time to do so. These results hold even for problems where the DM knows that there are two actions, already knows one of them, and knows the reward of the other.

Example 2.4.1: Let $M = (S, A, S_0, a_0, g_A, g_0, D, P, R, G_0)$ be an MDPU where $A = \{a_1, a_2\}$, and the DM knows that $S = S_0 = \{s_1\}$, $g_0(s_1) = \{a_1\}$, $|A| = 2$, $P(s_1, s_1, a) = 1$ for both actions $a \in A$, $R(s_1, s_1, a_1) = r_1$, $D(j, t, s_1) = \frac{1}{(t+1)^2}$, and the reward for the optimal policy for the true MDP is r_2, where $r_2 > r_1$. Since the DM knows that there is only one state and two actions, the DM knows that in the true MDP, there is an action a_2 that she is not aware of such that $R(s_1, s_1, a_2) = r_2$. That is, she knows everything about the true MDP but the action a_2. We now show that, despite all this knowledge, the DM cannot learn to play optimally in M.

Clearly, in the true MDP, the optimal policy is to always play a_2. However, to play a_2, the DM must learn about a_2. As we now show, no algorithm can learn about a_2 with probability greater than $1/2$, and thus no algorithm can attain an expected reward $\geq (r_1 + r_2)/2 = r_2 - (r_2 - r_1)/2$.

Recall that $\Gamma^{()}_{s,t,t'}$ is the event that a_0 is played at least t' times at s, and no new action is discovered on or before the tth time that a_0 is played at state s_1, and $\Gamma^{()}_{s,-1,t}$ is the event that a_0 is played at least t times at s. Let $s_0 \in S$. Given a policy π, let Pr be the probability on the runs in L_{s_0} determined by M and π. We first show that, for all $t > 0$, and all $t' \geq t$, if $\Pr(\Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) > 0$, then

$$\Pr(\Gamma^{()}_{s_1,t,t'} \mid \Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) = \prod_{n=1}^{t}(1 - D(1, n, s_1)). \tag{2.2}$$

It should be intuitively clear why (2.2) is true. Since, by assumption there is exactly one new action to be discovered at s_1, the probability that no new action is discovered the first t times that a_0 is played at s_1 is just the probability that, for each j with $1 \leq j \leq t$, no new action is discovered the jth time that a_0 is played, given that no new action has yet been discovered. $D(1, j, t)$ is the probability that a new action is discovered the jth time that a_0 is played given that no new action has yet been discovered, so $1 - D(1, j, t)$ is the probability a new action is not discovered then. We prove (2.2) formally by induction on t.

Base case: For $t = 1$, note that since $\Gamma^{(1)}_{s_1,-1,t'-1}$ is the set of all runs where a_0 is played at least t' times at s_1 and a new action is discovered the first time that a_0 is played, and $\Gamma^{()}_{s_1,1,t'}$ is the set of runs where a_0 is played at least t' times at s, and a new action is not discovered the first time that a_0 is played at s, these two sets form a disjoint partition of $\Gamma_{s_1,-1,t'}$. Moreover, since $\Gamma^{()}_{s_1,-1,t'} = \Gamma^{()}_{s_1,0,t'}$, we must have

$$\begin{aligned}
\Pr(\Gamma^{()}_{s_1,1,t'} \mid \Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) &= \Pr(\Gamma^{()}_{s_1,1,t'} \mid \Gamma^{()}_{s_1,0,t'} \cap L_{s_0}) \\
&= 1 - \Pr(\Gamma^{(1)}_{s_1,-1,t'-1} \mid \Gamma^{()}_{s_1,0,t'} \cap L_{s_0}) \\
&= 1 - D(1, 1, s_1).
\end{aligned}$$

Induction step: Suppose that (2.2) holds for t. We now show that it holds for $t + 1$; that is,

for all $t' \geq t+1$, the following holds:

$$\Pr(\Gamma^{()}_{s_1,t+1,t'} \mid \Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) = \prod_{n=1}^{t+1}(1 - D(1,n,s_1)). \tag{2.3}$$

For $t \geq 0$, note that

$$\begin{aligned}
& \Pr(\Gamma^{()}_{s_1,t+1,t'} \cap L_{s_0}) \\
= \; & \Pr(\Gamma^{()}_{s_1,t+1,t'} \mid \Gamma^{()}_{s_1,t,t'} \cap L_{s_0}) \Pr(\Gamma^{()}_{s_1,t,t'} \cap L_{s_0}) \quad [\text{since } \Gamma^{()}_{s_1,t+1,t'} \subseteq \Gamma^{()}_{s_1,t,t'}] \\
= \; & \Pr(\Gamma^{()}_{s_1,t+1,t'} \mid \Gamma^{()}_{s_1,t,t'} \cap L_{s_0}) \Pr(\Gamma^{()}_{s_1,t,t'} \mid \Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) \Pr(\Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) \\
& [\text{since } \Gamma^{()}_{s_1,t,t'} \subseteq \Gamma^{()}_{s_1,-1,t'}]
\end{aligned} \tag{2.4}$$

As above, it is easy to see that $\Gamma^{(t+1)}_{s_1,-1,t'-t-1}$ and $\Gamma^{()}_{s_1,t+1,t'}$ form a disjoint partition of $\Gamma^{()}_{s_1,t,t'}$. Moreover, it follows from the definition of $D(1,t+1,s_1)$ that $D(1,t+1,s_1) = \Pr(\Gamma^{(t+1)}_{s_1,-1,t'-t-1} \mid \Gamma^{()}_{s_1,t,t'} \cap L_{s_0})$. Thus,

$$\Pr(\Gamma^{()}_{s_1,t+1,t'} \mid \Gamma^{()}_{s_1,t,t'} \cap L_{s_0}) = 1 - \Pr(\Gamma^{(t+1)}_{s_1,-1,t'-t-1} \mid \Gamma^{()}_{s_1,t,t'} \cap L_{s_0}) = 1 - D(1,t+1,s_1). \tag{2.5}$$

Plugging (2.5) into (2.4) and applying the induction hypothesis, we get

$$\begin{aligned}
& \Pr(\Gamma^{()}_{s_1,t+1,t'} \cap L_{s_0}) \\
= \; & (1 - D(1,t+1,s_1)) \Pr(\Gamma^{()}_{s_1,t,t'} \mid \Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) \Pr(\Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) \\
= \; & (1 - D(1,t+1,s_1)) \textstyle\prod_{n=1}^{t}(1 - D(1,n,s_1)) \Pr(\Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) \\
= \; & \textstyle\prod_{n=1}^{t+1}(1 - D(1,n,s_1)) \Pr(\Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}).
\end{aligned} \tag{2.6}$$

By assumption, $\Pr(\Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) > 0$. Thus, dividing both sides of (2.6) by $\Pr(\Gamma^{()}_{s_1,-1,t'} \cap L_{s_0})$, we get (2.3), as desired. This completes the proof of the induction hypothesis.

Thus, for all $t > 0$ and $t' \geq t$,

$$\begin{aligned}
\Pr(\Gamma^{()}_{s_1,t,t'} \mid \Gamma^{()}_{s_1,-1,t'} \cap L_{s_0}) &= \textstyle\prod_{n=1}^{t}(1 - D(1,n,s_1)) \\
&= \textstyle\prod_{n=1}^{t}\left(1 - \frac{1}{(n+1)^2}\right) \\
&= \frac{t+2}{2(t+1)} \quad [\text{see below}] \\
&> \frac{1}{2}.
\end{aligned} \tag{2.7}$$

For the third equality, note that $1 - \frac{1}{(n+1)^2} = (1 - \frac{1}{n+1}) \times (1 + \frac{1}{n+1})$; it follows that

$$\prod_{n=1}^{t}\left(1 - \frac{1}{(n+1)^2}\right) = \left(\frac{1}{2} \times \frac{3}{2}\right) \times \left(\frac{2}{3} \times \frac{4}{3}\right) \times \cdots \times \left(\frac{t}{t+1} \times \frac{t+2}{t+1}\right).$$

All terms but the first and last cancel out. Thus, the product is $\frac{t+2}{2(t+1)}$, as desired.

Since (2.7) holds for all $t > 0$ and $t' \geq t$, taking $t' = t$, we get that $\Pr(\Gamma^{()}_{s_1,t,t} \mid \Gamma^{()}_{s_1,-1,t} \cap L_{s_0})$ is always strictly greater than 1/2 for all t, which means the DM cannot discover the better action a_2 with probability greater than $1/2$ no matter how many times a_0 is played, and no matter what policy is used. It easily follows that the expected reward of any policy is at most $(r_1 + r_2)/2$. Thus, there is no policy that learns to play near-optimally. ∎

The problem in Example 2.4.1 is that the discovery probability is so low that there is a probability bounded away from 0 that some action will not be discovered, no matter how many times a_0 is played. The following theorem generalizes Example 2.4.1, giving a sufficient condition on the failure probability (which we later show is also necessary) that captures the precise sense in which the discovery probability is too low. Intuitively, the theorem says that if the DM is unaware of some acts that can improve her expected reward, and the discovery probability is sufficiently low (where "sufficiently low" means that, for some state s, $D(1, t, s) < 1$ for all t, and $\sum_{t=1}^{\infty} D(1, t, s) < \infty$), then the DM cannot learn to play near-optimally. To make the theorem as strong as possible, we show that the lower bound holds even if the DM has quite a bit of extra information, as characterized in the following definition.

Definition 2.4.2: Define a DM to be *quite knowledgeable* if (in addition to S_0, g_0) she knows D, $S = S_0$, $|A|$, the transition function P_0, the reward function R_0 for all states in S_0 and actions in A_0 (i.e., $G_0 = \cup_{s \in S_0}(s \times S_0 \times g_0(s))$), and $R^*_{\max}$. □

We can now state our theorem. It turns out that there are slightly different conditions on the lower bound depending on whether $|S_0| \geq 2$ or $|S_0| = 1$.

Theorem 2.4.3: *Using the notation of Definition 2.4.2, if there exists a state $s_1 \in S$ such that $D(1, t, s_1) < 1$ for all t and $\sum_{t=1}^{\infty} D(1, t, s_1) < \infty$, then there exists a constant $c > 0$ and an MDP M compatible with what the DM knows such that no algorithm can obtain within c of the optimal reward for M, even if the DM is quite knowledgeable, provided that $|S_0| \geq 2$, $|A| > |A_0|$, and $R^*_{\max}$ is greater than the reward of the optimal policy for the MDP $(S_0, A_0, g_0, P_0, R_0)$. If $|S_0| = 1$, the same result holds if there exists s_1 such that $\sum_{t=1}^{\infty} D(j, t, s_1) < \infty$, where $j = |A| - |A_0|$.*

Proof: Suppose that $|S_0| \geq 2$ and the hypotheses of the theorem hold for state s_1. We construct an MDP $M'' = (S, A'', g'', P'', R'')$ that is compatible with what the DM knows, such that no algorithm can obtain within a constant c of the optimal reward in M''. The construction is similar in spirit to that of Example 2.4.1. Let $j = |A| - |A_0|$, let $A'' = A_0 \cup \{a_1, \ldots, a_j\}$, where $a_1, \ldots, a_j$ are fresh actions not in A_0, and let g'' be such that $g''(s_1) = g_0(s_1) \cup \{a_1\}$, $g''(s) = A''$ for $s \neq s_1$ (such states s exist, since $|S| \geq 2$). That is, there is only one action that the DM is not aware of in state s_1, while in all other states, she is unaware of all actions in $A - A_0$. Let $P''(s_1, s_1, a_1) = P''(s, s_1, a) = 1$ for all $a \in A'' - A_0$ and $s \in S$ (note that P'' is determined by P_0 in all other cases), and define $R''(s_1, s', a_1) = R^*_{\max}$ and $R''(s, s', a) = R^*_{\max} - 1$ for all $s' \in S$ and $s \neq s_1$ and $a \in A - A_0$ (R'' is determined by R_0 in all other cases). It is easy to check that M'' is compatible with what the DM knows, even if the DM knows that $S = S_0$, knows $|A|$, and knows $R^*_{\max}$. By assumption, the reward of the optimal policy for $(S_0, A_0, g_0, P_0, R_0)$ is less than $R^*_{\max}$, so the optimal policy is clearly to get to state s_1 and then to play a_1 (giving an average reward of $R^*_{\max}$ per time unit). Of course, doing this requires learning a_1.

As in Example 2.4.1, we first prove that for M'' there exists a constant $d > 0$ such that, with probability d, no algorithm discovers action a_1 in state s_1.

Again, for a policy π, if Pr is the probability on L_{s_0} determined by M and π, we have

$$\Pr(\Gamma^{()}_{s_1,t,t} \mid \Gamma^{()}_{s_1,-1,t} \cap L_{s_0}) \quad = \quad \prod_{t'=1}^{t}(1 - D(1, t', s_1)).$$

Since $\sum_{t=1}^{\infty} D(1, t, s_1) < \infty$, we must have that $\lim_{t\to\infty} D(1, t, s_1) = 0$. Since $D(1, t, s_1) < 1$ for all t, there must exist a constant q with $0 < q < 1$ such that $D(1, t, s_1) < q$ for all t. We show below that $1 - D(1, t', s_1) \geq (1 - q)^{D(1,t',s_1)/q}$. Thus, we get that

$$\begin{aligned}\Pr(\Gamma^{()}_{s_1,t,t} \mid \Gamma^{()}_{s_1,-1,t} \cap L_{s_0}) &\geq \prod_{t'=1}^{t}(1 - q)^{D(1,t',s_1)/q} \\ &\geq (1 - q)^{\sum_{t'=1}^{t} D(1,t',s_1)/q}.\end{aligned}$$

Since, by assumption, $\sum_{t'=1}^{\infty} D(1, t', s_1) < \infty$, we can take $d = (1 - q)^{\sum_{t'=1}^{\infty} D(1,t',s_1)/q}$.

It remains to show that $1 - D(1, t', s_1) \geq (1 - q)^{D(1,t',s_1)/q}$. Since $0 \leq D(1, t', s_1) < q < 1$, it suffices to show that $1 - x \geq (1 - b)^{x/b} = e^{(x/b)\ln(1-b)}$ for $0 \leq x < b < 1$. Let $h(x) = 1 - x - e^{(x/b)\ln(1-b)}$. We want to show that $h(x) \geq 0$ for $0 \leq x < b < 1$. An easy substitution shows that $h(0) = h(b) = 0$. Differentiating h, we get that $h'(x) = -1 - \frac{\ln(1-b)}{b} e^{(x/b)\ln(1-b)}$, and $h''(x) = -\frac{\ln(1-b)^2}{b^2} e^{(x/b)\ln(1-b)} < 0$. Since $h(0) = h(b) = 0$ and h is concave, we must have $h(x) = h((1 - x/b)0 + (x/b)b) \geq (1 - x/b)h(0) + (x/b)h(b) = 0$ for $x \in [0, b]$, as desired.

As we have observed, the optimal policy for M'' has expected reward $R^*_{\max}$. Moreover, the optimal policy for $(S, A'' - \{a_1\}, g'', P''|_{A-\{a_1\}}, R''|_{A-\{a_1\}})$ has expected reward $R^*_0 < R^*_{\max}$. With probability at least d, no algorithm discovers a_1, so the DM will know at most the actions in $A'' - \{a_1\}$, and cannot get a reward higher than R^*_0. Thus, no algorithm can give the DM an expected reward higher than $(1 - d)R^*_{\max} + dR^*_0$, so we can take $c = d(R^*_{\max} - R^*_0)$.

If $|S_0| = 1$, essentially the same argument holds. We again construct an MDP $M'' = (S_0, A'', g'', P'', R'')$. Since $|S_0| = 1$, all components of M'' are determined except for R''. We

take $R''(s_1, s_1, a_1) = R^*_{\max}$ for all s', and $R''(s_1, s_1, , a) = R^*_{\max} - 1$, for $a \in A'' - (A_0 \cup \{a_1\})$. Again, the unique optimal policy is to play a_1 at all times, so the problem reduces to learning a_1. Without further assumptions, all we can say is that this probability of learning a_1 after t steps of exploration is at most $D(j, t, s_1)$, so we must replace $D(1, t, s_1)$ by $D(j, t, s_1)$ in the argument above.[7] ∎

Note that Example 2.4.1 is a special case of Theorem 2.4.3, since $\sum_{t=1}^{\infty} \frac{1}{(t+1)^2} < \int_{t=1}^{\infty} \frac{1}{t^2} dt = 1$.

In the next section, we show that if $\sum_{t=1}^{\infty} D(1, t, s) = \infty$ for all $s \in S$, then there is an algorithm that learns near-optimal play (although the algorithm may not be efficient). Thus, $\sum_{t=1}^{\infty} D(1, t, s)$ determines whether or not there is an algorithm that learns near-optimal play. We can say even more. If $\sum_{t=1}^{\infty} D(1, t, s) = \infty$ for all s, then the efficiency of the best algorithm for determining near-optimal play depends on how quickly $\sum_{t=1}^{\infty} D(1, t, s)$ diverges. Specifically, the following theorem shows that if for some $s \in S$ we have $\sum_{t=1}^{T} D(1, t, s) \leq f(T)$, where $f : [1, \infty) \to \boldsymbol{R}$ is an increasing function whose co-domain includes $(0, \infty)$ (so that $f^{-1}(t)$ is well defined for $t \in (0, \infty)$) and $D(1, t, s) < q < 1$ for all t, then the DM cannot learn to play near-optimally with probability $\geq 1 - \delta$ in time less than $f^{-1}(q \ln(\delta)/\ln(1 - q))$. It follows, for example, that if $f(T) = m_1 \log(T) + m_2$, then it requires time polynomial in $1/\delta$ to learn to play near-optimally with probability greater than $1 - \delta$. For if $f(T) = m_1 \log(T) + m_2$, then $f^{-1}(t) = e^{(t-m_2)/m_1}$, so $f^{-1}(q \ln(\delta)/\ln(1-q)) = f^{-1}(q \ln(1/\delta)/\ln(1/(1-q)))$ has the form $a(1/\delta)^b$ for constants $a, b > 0$. A similar argument shows that if $f(T) = m_1 \ln(\ln(T) + 1) + m_2$, then $f^{-1}(q \ln(1/\delta)/\ln(1/(1-q)))$ has the form $ae^{(1/\delta)^b}$ for constants $a, b > 0$; that is, the running time is exponential in $1/\delta$.

[7] We remark that we can still use $D(1, t, s_1)$ if the DM does not know that $S = S_0$, but does know $|S|$, and $|S| \geq 2$. We can also use $D(1, t, s_1)$ if the probability of learning the specific action a_1 after t steps of exploration is $D(1, t, s_1)$.

Theorem 2.4.4: *If* $|S_0| \geq 2$, $|A| > |A_0|$, $R^*_{\max}$ *is greater than the reward of the optimal policy for the MDP* $(S_0, A_0, g_0, P_0, R_0)$, $\sum_{t=1}^{\infty} D(1, t, s) = \infty$ *for all* $s \in S$, *and there exists a constant* $q < 1$ *and a state* $s_1 \in S$ *such that* $D(1, t, s_1) < q$ *for all t, and an increasing function* $f : [1, \infty) \to \boldsymbol{R}$ *such that the co-domain of* f *includes* $(0, \infty)$ *and* $\sum_{t=1}^{T} D(1, t, s_1) \leq f(T)$, *then for all* δ *with* $0 < \delta < 1$, *there exists a constant* $c > 0$ *and an MDP* M *compatible with what the DM knows such that no algorithm that runs in time less than* $f^{-1}(q \ln(\delta)/\ln(1-q))$ *can obtain within c of the optimal reward for* M *with probability* $\geq 1 - \delta$, *even if the DM is quite knowledgeable. If* $|S_0| = 1$, *the same result holds if* $\sum_{t=1}^{T} D(j, t, s_1) \leq f(T)$, *where* $j = |A| - |A_0|$.

Proof: Consider the MDP M'' constructed in the proof of Theorem 2.4.3. As we observed, M'' is compatible with what the DM knows (even if the DM knows $S = S_0$, $|A|$, and the maximum possible reward $R^*_{\max}$). Note that, for all $\epsilon > 0$, the ϵ-return mixing time of M'' is 1.

By assumption, there exists a constant q with $0 < q < 1$ such that $D(1, t, s_1) < q$ for all t. We now prove for all $d > 0$, all algorithms require at least time $f^{-1}(q \ln(d)/\ln(1-q))$ to discover a_1 in M'' with probability $\geq 1 - d$. The same argument as in Theorem 2.4.3 shows that $\Pr(\Gamma^{()}_{s_1,t,t} \mid \Gamma^{()}_{s_1,-1,t} \cap L_{s_0}) \geq (1-q)^{\sum_{t'=1}^{t} D(1,t',s_1)/q}$. Since $\sum_{t'=1}^{t} D(1, t', s_1) \leq f(t)$, it follows that $\Pr(\Gamma^{()}_{s_1,t,t} \mid \Gamma^{()}_{s_1,-1,t} \cap L_{s_0}) \geq (1-q)^{f(t)/q}$. Note that for the probability of discovering a_1 to be at least $1 - d$ at state s_1, we must have $\Pr(\Gamma^{()}_{s_1,t,t} \mid \Gamma^{()}_{s_1,-1,t} \cap L_{s_0}) \leq d$, which in turn requires that $(1-q)^{f(t)/q} \leq d$. Taking logs of both sides and rearranging terms, we must have $f(t) \geq q \ln(d)/\ln(1-q)$, so $t \geq f^{-1}(q \ln(d)/\ln(1-q))$, since f is increasing. (Note that since $0 < d < 1$ and $0 < q < 1$, both $\ln(1-q)$ and $\ln(d)$ are negative, so $\ln(d)/\ln(1-q) > 0$, and $f^{-1}(q \ln(d)/\ln(1-q))$ is well defined.) Thus, it requires at least time $f^{-1}(q \ln(d)/\ln(1-q))$ to discover a_1 with probability $\geq 1 - d$.

As before, the expected reward of the optimal policy for M'' is $R^*_{\max}$. Again, let $R^*_0 < R^*_{\max}$ be the expected reward of the optimal policy for $(S, A'', g'', P''|_{A''-\{a_1\}}, R''|_{A''-\{a_1\}})$. If a_1

is not discovered, the DM will know only the actions in $A'' - \{a_1\}$, and cannot get a reward higher than R_0^*. It follows that no algorithm can give the DM an expected reward greater than $(1-d)R_0^* + dR_{\max}^*$ in time less than $f^{-1}(q\ln(d)/\ln(1-q))$, so we can again take $c = d(R_{\max}^* - R_0^*)$. ∎

In the next section, we prove that the lower bound of Theorem 2.4.4 is essentially tight: if $\sum_{t=1}^{T} D(1,t,s) \geq f(T)$ for all $s \in S$, then the DM can learn to play near-optimally in time polynomial in $f^{-1}(\ln(4N/\delta))$ and all the other parameters of interest. In particular, if $f(t) \geq m_1 \ln(t) + m_2$ for some constants m_1 and m_2, then the DM can learn to play near-optimally in time polynomial in the relevant parameters.

2.5 Learning to Play Near-Optimally

In this section, we show that a DM can learn to play near-optimally in an MDPU where $\sum_{t=1}^{\infty} D(1,t,s) = \infty$ for all $s \in S$. Moreover, we show that when $\sum_{t=1}^{\infty} D(1,t,s) = \infty$ for all $s \in S$, the speed at which $D(1,t,s)$ decreases determines how quickly the DM can learn to play near-optimally. Specifically, if for all $s \in S$ we have $\sum_{t=1}^{T} D(1,t,s) \geq m_1 f(\ln T) + m_2$ for all $T > 0$, some constants $m_1 > 0$ and m_2, and some invertible function f, then the DM can learn to play near-optimally in time polynomial in $f^{-1}(1/\delta)$. In particular, if f is the identity (so that $\sum_{t=1}^{T} D(1,t,s) \geq m_1 \ln T + m_2$), then the DM can learn in time polynomial in $1/\delta$ (and all other parameters of interest). We call the learning algorithm URMAX, since it is an extension of RMAX to MDPUs. While the condition $\sum_{t=1}^{\infty} D(1,t,s) = \infty$ may seem rather special, it arises in many applications of interest. For example, when learning to fly a helicopter [1; 50], the space of potential actions in which the exploration takes place, while four-dimensional (resulting from the four degrees of freedom of the helicopter), can be discretized and taken to

be finite. Thus, if we explore by examining the potential actions uniformly at random, then for each state s, $D(1, t, s)$ is constant for all t, and hence $\sum_{t=1}^{\infty} D(1, t, s) = \infty$. Indeed, in this case $\sum_{t=1}^{T} D(1, t, s)$ is $O(T)$, so it follows from Corollary 2.5.5 below that we can learn to fly the helicopter near-optimally in time polynomial in the size of the state and action space. The same is true in any situation where the action space can be methodically explored.

We assume throughout this section that $\sum_{t=1}^{\infty} D(1, t, s) = \infty$ for all $s \in S$. We would like to use an RMAX-like algorithm to learn to play near-optimally in our setting too, but there are two major problems in doing so. The first is that we do not want to assume that the DM knows $|S|$, $|A|$, or $R^*_{\max}$. We deal with the fact that $|S|$ and $|A|$ are unknown by using essentially the same idea as Kearns and Singh use for dealing with the fact that the true ϵ-mixing time T is unknown: we start with an estimate of the value of $|S|$ and $|A|$, and keep increasing the estimate. Eventually, we get to the right values, and we can compensate for the fact that the reward may have been too low up to that point by playing the policy sufficiently often. The idea for dealing with the fact that $R^*_{\max}$ is not known is similar. (We remark that our approach for dealing with the case where $R_{\max}$ is unknown can also be applied to RMAX.) The second problem is more serious: we need to deal with the fact that not all actions are known, and that we have a special *explore* action. Specifically, we need to come up with an analogue of $H_1(T)$ that describes how many times we should play the explore action a_0 in a state s, with a goal of discovering all the actions in s. Clearly this value will depend on the discovery probability (it turns out that it depends only on $D(1, t, s)$ for all t and s) in addition to all the parameters that $H_1(T)$ depends on.

We now describe the algorithm URMAX$(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$. Intuitively, in this algorithm, N is an upper bound on the size of the state space S, k is an upper bound on the size of the action space A, $R_{\max}$ is an upper bound on the maximum reward $R^*_{\max}$, T is an up-

per bound on the ϵ-return mixing time, and K_0 is such that $\sum_{t=1}^{K_0} D(1,t,s) \geq \ln(4Nk/\delta)$ for all s. While URMAX$(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ is defined for all values of the parameters, it does the right thing only if the parameters have the "right" values. We later show how to define URMAX(ϵ, δ, s_0), a variant where these parameter values are not needed. In particular, an agent running URMAX(ϵ, δ, s_0) does not need to know $D(1,t,s)$, or bounds on $|S|$, $|A|$, or $R^*_{\max}$. Roughly speaking, in URMAX(ϵ, δ, s_0), all the parameter values are tried until the right one is found.

Define

- $K_1(T) = \max(\lceil(\frac{4NTR_{\max}}{\epsilon})^3\rceil, \lceil\frac{1}{8}(\ln\frac{8N^2k}{\delta})^3\rceil) + 1$;
- $K_2(T, K_0) = \max(\frac{16R_{\max}}{\epsilon}Nk(K_1(T) + K_0), \frac{1}{2}(\frac{16R_{\max}}{\epsilon})^2 \ln\frac{4}{\delta})$;
- $K_3 = \max((\frac{2R_{\max}}{\epsilon})^3, 8(\ln\frac{4}{\delta})^3)$;
- $K_4(T, K_0) = \max((\frac{2R_{\max}}{\epsilon})K_2(T, K_0), K_2(T, K_0) + K_3)$.

Just as $H_1(T)$ in RMAX, $K_1(T)$ is a bound on how many times a state-action pair (s, a) has to be played in order to learn a good estimate of the transition probabilities for action a at state s, given that T is the ϵ-mixing time. More precisely, it is the number of times a state-action pair must be played so that, with probability at least $1 - \frac{\delta}{4Nk}$, the estimated transition probabilities are within $\frac{\epsilon}{4NTR_{max}}$ of the actual transition probabilities. ($K_1(T)$ differs slightly from $H_1(T)$: it has a coefficient of 8 in the second argument to max, rather than 6; the difference turns out to be needed to allow for the fact that we do not know all the actions.)

Like RMAX, URMAX proceeds in iterations. Each iteration of URMAX goes through the outer loop in Figure 2.4 once. Just as for RMAX, we can divide URMAX iterations into *exploration iterations* and *exploitation iterations*. If the policy π'_i played in the ith iteration is

(ϵ, T)-optimal for the underlying MDP M, then i is an exploitation iteration, otherwise, i is an exploration iteration. Of course, the DM does not in general know whether an iteration is an exploration or exploitation, since she doesn't know the underlying MDP.

We now explain K_2, K_3, and K_4. Just as $H_2(T)$ in RMAX, $K_2(T, K_0)$ is the number of exploration iterations required so that, with probability at least $1 - \delta/4$ (instead of $1 - \delta/3$ in RMAX; we need a tighter bound here because the DM may not know all the actions), after $K_2(T, K_0)$ exploration iterations, all remaining iterations become exploitation iterations; just as H_3 in RMAX, K_3 is the minimum number of exploitation iterations required in a run so that, with probability at least $1 - \delta/4$ (instead of $1 - \delta/3$ in RMAX), the average reward of all exploitation iterations is at least $\text{Opt}(M, \epsilon, T) - 3\epsilon/2$; like H_4 in RMAX, K_4 is the total number of iterations required to ensure that, with probability at least $1 - \delta$, the expected reward of a URMAX run is within 2ϵ of optimal. We take a pair (s, a) for $a \neq a_0$ to be *known* if it is played $K_1(T)$ times; we take a pair (s, a_0) to be *known* if it is played kK_0 times.

URMAX$(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ is just like RMAX$(N, k, R_{\max}, T, \epsilon, \delta, s_0)$, except for the following modifications:

- **The initial approximation:** In M^0, the initial approximation to the actual MDP M, the state space S^0 has $N + 1$ states, one of which is the dummy state s_d. Recall that N is an upper bound on the number of states in M, so the state space of M^0 is a superset of the state space of M. We take $P^0(s, s', a) = P(s, s', a)$ and $R^0(s, s', a) = R(s, s', a)$ if $(s, s', a) \in G_0$. For $(s, s', a) \notin G_0$, take $R(s, s', a) = R_{\max}$. The remaining transition probabilities are a little complicated to define, because we must have $\sum_{s''} P^0(s, s'', a) = 1$, but it may be possible that $(s, s_1, a) \in G_0$ and $(s, s_2, a) \notin G_0$. For example, the DM may know that playing action a at s transitions to s_1 with probability 0.4, but is uncertain of what happens for the remaining 0.6 cases. Let $p_{s,a} = \sum_{\{s'':(s,s'',a)\in G_0\}} P(s, s'', a)$; that is,

$p_{s,a}$ is the sum of the known transition probabilities. In the example, $p_{s,a} = 0.4$. Now take $P^0(s, s_d, a) = 1 - p_{s,a}$, and take $P^0(s, s', a) = 0$ for all $(s, s', a) \notin G_0$ and $s' \neq s_d$. In the example, s_1 transitions to the default state s_d with the remaining 0.6 probability, and transitions to all other states with probability 0. As usual, we have $P^0(s, s, a_0) = 1$ for all $s \in S^0$; we take $R^0(s, s, a_0) = R_{\max}$.

- **Updating the approximation:** If (s, a_0) has just become known, then we set the reward for playing a_0 in state s to be $-R_{\max}$. (This ensures that a_0 does not affect the reward of the optimal policy, since 0 is the lowest reward. In fact, the choice of reward here does not matter as long as it is negative.)

  If a new action a is discovered at state s, then a new state-action pair (s, a) is created, with $P(s, s_d, a) = 1$ and $R(s, s', a) = R_{\max}$ for all $s' \in S$. (Intuitively, if a is discovered at state s, then the DM gets high reward for playing a at s, in order to learn the right transition probability and actual reward.)

  (Please see the RMAX algorithm in Section 2.2 for the case of updating when (s, a) with $a \neq a_0$ becomes known.)

- **Becoming known:** A state-action pair (s, a) with $a \neq a_0$ becomes known when it has been played $K_1(T)$ times, and a state-action pair (s, a_0) becomes known when it has been played kK_0 times.

- **Termination:** The algorithm terminates after at most $K_4(T, K_0)$ iterations (whereas RMAX terminates after exactly $H_4(T)$ iterations).

  The algorithm also terminates if it discovers a reward greater than $R_{\max}$, more than k actions, or more than N states (N, k, and $R_{\max}$ can be viewed as the current guesses for these values; if the guess is discovered to be incorrect, the algorithm is restarted with better guesses).

We say that *an inconsistency is discovered* if the algorithm discovers a reward greater than $R_{\max}$, more than k actions, or more than N states; a state-action pair (s, a) with $a \neq a_0$ becomes known if it is played $K_1(T)$ times; a state-action pair (s, a_0) becomes known if it is played kK_0 times.

URMAX$(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$:

$s_c := s_0$
$M' := M^0$ (the initial approximation described in the main text)
Compute an optimal T-step policy π' for M' starting at s_c
Repeat $K_4(T, K_0)$ times or until an inconsistency is discovered:
  Play π' starting at s_c for T steps
  $s_c :=$ the DM's current state
  for each (s, a) that becomes known during the T steps such that $a \neq a_0$
    update M' so that the transition probabilities for (s, a) are the observed frequencies and the rewards for (s, a) are the observed rewards
  for each (s, a_0) that becomes known during the T steps
    update M' so that $P(s, s, a_0) = 1$, and $R(s, s', a_0) = -R_{\max}$ for all $s' \in S$
  for each new action a discovered at some state s in the T steps,
    create a new state-action pair (s, a) in M', with $P(s, s_d, a) = 1$ and $R(s, s', a) = R_{\max}$ for all $s' \in S$
  Compute an optimal T-step policy π' for M' starting at s_c
Return π'

Figure 2.4: The URMAX algorithm.

Note that an execution of the URMAX algorithm generates an MDPU run, since each transition made is a transition of the underlying MDP. Define a *URMAX run* to be an MDPU run that is generated by URMAX.

We now prove the correctness of URMAX. Note that in URMAX, $K_4(T, K_0)$ iterations are executed if no inconsistency is found. Instead of proving the correctness of URMAX directly, we prove a more general version of the result. We show that if the algorithm executes *at least* $K_4(T, K_0)$ iterations (instead of exactly $K_4(T, K_0)$ iterations) and no inconsistency is found, then it achieves a near-optimal expected reward in polynomial time. (We use the more gen-

eral result in our proof of Theorem 2.5.3 below.) Define $\text{URMAX}_K(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ to be exactly the same algorithm as $\text{URMAX}(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$, except that it runs for K iterations instead of $K_4(T, K_0)$ iterations if no inconsistency is found. A *$URMAX_K$ run* is an MDPU run that is generated by URMAX_K. Note that $\text{URMAX}(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0) = \text{URMAX}_{K_4(T,K_0)}(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$.

The next lemma shows that if $s \in S$, $0 < \delta < 1$, $K_0 \geq \min\{H : \sum_{t=1}^{H} D(1, t, s) \geq \ln(4Nk/\delta)\}$, $u \geq 0$, $\vec{n} = (n_1, n_2, \ldots, n_i)$ is a strictly increasing sequence of integers where $i \geq 0$,[8] and initially more than i actions have not been discovered at s, then conditional on the set of runs starting at s_0 where the jth new action at state s is discovered the n_jth time that a_0 is played at s for $j = 1, \ldots, i$, and a_0 is played at s at least $n_i + u$ times at s, with probability at least $1 - \frac{\delta}{4Nk}$, one of the following happens: either (1) $l \in L_{s_0} - \Gamma^{()}_{s,-1,n_i+K_0}$ (which means a_0 is played fewer than $n_i + K_0$ times at s in l), or (2) $l \in \bigcup_{j=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+j)}_{s,-1,u}$ (which means a new action at s is discovered on or before the K_0th time that a_0 is played at s after the ith new action is discovered). In the special case where $i = 0$, so that $\vec{n} = ()$, we Recall that in the special case where $i = 0$, so that $\vec{n} = ()$, we take $\Gamma^{\vec{n}\cdot(n_i+j)}_{s,-1}$ to be $\Gamma^{(j)}_{s,-1}$. This convention allows us to treat the case $i = 0$ and $i > 0$ in a uniform way, and simplifies the exposition. The probability Pr here (and in all the later results) is with respect to the probability distribution on the runs in L_{s_0} that is determined by the MDPU and the policy π being used. To further simplify the exposition, for the remainder of this section, we take $\Pr(A \mid B)$ to be 1 if $\Pr(B) = 0$ (rather than being undefined).

Lemma 2.5.1: *Let $M = (S, A, S_0, a_0, g_A, g_0, D, P, R, G_0)$ be an MDPU where $|S| = N$ and $|A| = k$. Fix $s, s_0 \in S$, and let* Pr *be a probability on L_{s_0} determined by M and a policy π. If $0 < \delta < 1$, $K_0 \geq \min\{H : \sum_{t=1}^{H} D(1, t, s) \geq \ln(4Nk/\delta)\}$, and $u \geq 0$, then for all strictly increasing sequence of integers $\vec{n} = (n_1, n_2, \ldots, n_i)$ where $0 \leq i < |g_A(s) - g_0(s)|$, $\Pr(\bigcup_{t'=1}^{K_0} \Gamma^{(\vec{n}\cdot(n_i+t')}_{s,-1} \cup (L_{s_0} -$*

[8] A sequence $(n_1, n_2, \ldots, n_i)$ is *strictly increasing* if $0 < n_1 < n_2 < \ldots < n_i$.

$\Gamma^{()}_{s,-1,n_i+K_0}) \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \geq 1 - \frac{\delta}{4Nk}$. *(In the special case that $i = 0$, so that $\vec{n} = ()$, we take $\Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1}$ to be $\Gamma^{(t')}_{s,-1}$, and $n_i = 0$, so* $\Pr(\bigcup_{t'=1}^{K_0} \Gamma^{(t')}_{s,-1} \cup (L_{s_0} - \Gamma^{()}_{s,-1,K_0}) \mid \Gamma^{()}_{s,-1,u} \cap L_{s_0}) \geq 1 - \frac{\delta}{4Nk}$.*)*

Proof: Let $E = \bigcup_{t'=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1} \cup (L_{s_0} - \Gamma^{()}_{s,-1,n_i+K_0})$. We want to show that $\Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \geq 1 - \frac{\delta}{4Nk}$.

We partition L_{s_0} into two disjoint subsets: $L_1 = L_{s_0} - \Gamma^{()}_{s,-1,n_i+K_0}$ (so that, in a run in L_1, a_0 is played fewer than $n_i + K_0$ times at s); and $L_2 = L_{s_0} \cap \Gamma^{()}_{s,-1,n_i+K_0}$ (so that, in a run in L_2, a_0 is played at least $n_i + K_0$ times at s). Clearly, $\Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_1) = 1 \geq 1 - \delta/4Nk$. It thus suffices to prove that $\Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_2) \geq 1 - \delta/4Nk$. Let $j = |g_A(s) - g_0(s)| - i$ (so that after discovering i new actions at s, there are j more to discover), and let $v = \max(u, K_0)$,

$$
\begin{aligned}
& \Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_2) \\
\geq\ & \Pr(\textstyle\bigcup_{t'=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1} \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_2) && \text{[by the definition of } E\text{]} \\
=\ & \Pr(\textstyle\bigcup_{t'=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1} \mid \Gamma^{\vec{n}}_{s,-1,u} \cap \Gamma^{()}_{s,-1,n_i+K_0} \cap L_{s_0}) && \text{[by the definition of } L_2\text{]} \\
=\ & \Pr(\textstyle\bigcup_{t'=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1} \mid \Gamma^{\vec{n}}_{s,-1,\max(K_0,u)} \cap L_{s_0}) \\
=\ & \Pr(\textstyle\bigcup_{t'=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1} \mid \Gamma^{\vec{n}}_{s,-1,v} \cap L_{s_0}) \\
=\ & \Pr(\textstyle\bigcup_{t'=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1,v-t'} \mid \Gamma^{\vec{n}}_{s,-1,v} \cap L_{s_0}) \\
& \qquad \text{[since } \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1} \cap \Gamma^{\vec{n}}_{s,-1,v} = \Gamma^{\vec{n}\cdot(n_i+t')}_{s,-1,v-t'}\text{]} \\
=\ & 1 - \Pr(\Gamma^{\vec{n}}_{s,K_0,v} \mid \Gamma^{\vec{n}}_{s,-1,v} \cap L_{s_0}) && \text{[see below]} \\
=\ & 1 - \textstyle\prod_{t'=1}^{K_0}(1 - D(j, t', s)) && \text{[see below]} \\
\geq\ & 1 - \textstyle\prod_{t'=1}^{K_0}(1 - D(1, t', s)).
\end{aligned}
$$

The third last line holds since $\Gamma^{\vec{n}}_{s,K_0,v}$ is the subset of runs in $\Gamma^{\vec{n}}_{s,-1,v}$ where the $(i+1)$st new action is not discovered the first K_0 times that a_0 is played at s after the ith new action is discovered at s, and $\bigcup_{n=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1,v-n}$ is the subset of runs in $\Gamma^{\vec{n}}_{s,-1,v}$ where the $(i+1)$st new action is discovered the first K_0 times that a_0 is played at s after the ith new action is discovered at s, so these two

sets form a disjoint partition of $\Gamma^{\vec{n}}_{s,-1,v}$. The second last line holds since

$$\Pr(\Gamma^{\vec{n}}_{s,K_0,v} \mid \Gamma^{\vec{n}}_{s,-1,v} \cap L_{s_0}) = \prod_{n=1}^{K_0}(1 - D(j, n, s)).$$

This can be proved using similar techniques that we used to prove (2.2) in Example 2.4.1; we leave the details to the reader.[9] The last inequality follows since we have assumed that $D(j, t, s) \geq D(1, t, s)$ for $j \geq 1$; with more actions available, it is easier to find a new one.

We show below that $1 - D(1, t', s) \leq e^{-D(1,t',s)}$. It follows that

$$\begin{aligned} \Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_2) &\geq 1 - \textstyle\prod_{t'=1}^{K_0} e^{-D(1,t',s)} \\ &\geq 1 - e^{-\sum_{t'=1}^{K_0} D(1,t',s)}. \end{aligned}$$

The choice of K_0 guarantees that $\sum_{t=1}^{K_0} D(1, t, s) \geq \ln(4Nk/\delta)$. Thus,

$$\Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_2) \geq 1 - e^{-\ln(4Nk/\delta)} = \tfrac{\delta}{4Nk}.$$

It remains to show that $1 - D(1, t', s) \leq e^{-D(1,t',s)}$.

Since $D(1, t', s) \geq 0$, it suffices to show that $1 - x \leq e^{-x}$ for $x \geq 0$. Let $g(x) = 1 - x - e^{-x}$. We want to show that $g(x) \leq 0$ for $x \geq 0$. An easy substitution shows that $g(0) = 0$. Differentiating g, we get that $g'(x) = -1 + e^{-x} \leq 0$ when $x \geq 0$. Since $g(0) = 0$ and g is nonincreasing when $x \geq 0$, we must have $g(x) \leq 0$ for $x \geq 0$, as desired. ∎

The next theorem shows that for all $K \geq K_4(T, K_0)$, $\text{URMAX}_K(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ obtains an expected reward that is near-optimal with high probability if the parameter values are sufficiently large.

Theorem 2.5.2: *Let* $M' = (S', A', S_0, a_0, g'_A, g_0, D, P', R', G_0)$ *be an MDPU where* $|S'| = N$, $|A'| = k$, *and* $\max(R'(s, s', a) : s, s' \in S', a \in A') = R_{\max}$. *If* $0 < \delta < 1$, $\epsilon > 0$, $K_0 \geq \min\{H :$

[9] Note that since we have assumed that $|g_A(s) - g_0(s)| > i$, there are still new actions to be discovered in runs in $\Gamma^{\vec{n}}_{s,K_0,v}$, that is, after the first i new actions have been discovered at s. This assumption was critical in the argument in Example 2.4.1.

$\sum_{t=1}^{H} D(1,t,s) \geq \ln(4Nk/\delta)\}$, *and* $K \geq K_4(T, K_0)$, *then for all MDPs* $M = (S_M, A_M, g_M, P_M, R_M)$ *compatible with* S_0, g_0, G_0, N, k, $R_{\max}$, *and* T *(i.e.,* $S_M \supseteq S_0$, $g_M(s) \supseteq g_0(s)$ *for all* $s \in S_0$, $|S_M| \leq N$, $|A_M| \leq k$, $R_M(s,s',a) \leq R_{\max}$ *for all* $s, s' \in S_M$ *and* $a \in A_M$, *and the* ϵ*-return mixing time* T_M *of* M *is* $\leq T$*), and all states* $s_0 \in S_0$, *with probability at least* $1-\delta$, $\text{URMAX}_K(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ *running on* M *obtains an expected average reward that is at least* $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$ *in time polynomial in* N, k, T, *and* K.

Proof: See Appendix A. ∎

Since the proof of this theorem involves several steps, and the is similar to the proof in [4], we defer it to appendix.

Note that the running time of URMAX is polynomial in N, k, T, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $R_{\max}$, and K_0, since $\text{URMAX}(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0) = \text{URMAX}_{K_4(T,K_0)}(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ and $K_4(T, K_0)$ is polynomial in N, k, T, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $R_{\max}$, and K_0.

We get $\text{URMAX}(\epsilon, \delta, s_0)$ by running $\text{URMAX}(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ using larger and larger values for N, k, $R_{\max}$, T and K_0.

$\text{URMAX}(\epsilon, \delta, s_0)$:

$T := 1$
Repeat forever:
  $\text{URMAX}_{K_5(T)}(T, |S_0| + T - 1, |A_0| + T - 1, T, T, \epsilon, \delta, s_0)$, where
    $K_5(T) = K_4(T+1, T+1)^{\frac{2(T+1)}{\epsilon}}$
  $T := T + 1$

Algorithm 3

Eventually the parameters in $\text{URMAX}(\epsilon, \delta, s_0)$ become at least as large as their actual values (i.e., $|S_0| + T - 1 \geq |S|$, $|A_0| + T - 1 \geq |A|$, $T \geq R^*_{\max}$, $T \geq T_M$, the ϵ-return mixing time

of the underlying MDP, and $T \geq \min_{n_1}\{n_1 : \sum_{t=1}^{n_1} D(1,t,s) \geq \ln(4Nk/\delta)\}$ for all s). Once that happens, by the proof of Theorem 2.5.2, with high probability, URMAX$(T, |S_0| + T - 1, |A_0| + T - 1, T, T, \epsilon, \delta, s_0)$ obtains an expected reward of at least $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$ in all later iterations. However, since we do not know when that will happen, we need to continue running URMAX(ϵ, δ, s_0). We must thus ensure that, if the parameters have become sufficiently large (which they will be, eventually), then the average reward stays within 3ϵ of optimal while we are testing higher values of these parameters.

For example, suppose that the actual values of these parameters are all 100, and $|S_0| = |A_0| = 1$. Then, with high probability, the expected reward of URMAX$(100, 100, 100, 100, 100, \epsilon, \delta, s_0)$ is at least $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$. Nevertheless, URMAX$(\epsilon, \delta, s_0)$ then sets these parameters to 101 and runs URMAX$(101, 101, 101, 101, 101, \epsilon, \delta, s_0)$. This requires a recomputation of the optimal policy. While this recomputation is going on, the DM may get a low reward. We need to ensure that this period of low rewards does not affect the average reward significantly. This is ensured by running URMAX$(100, 100, 100, 100, 100, \epsilon, \delta, s_0)$ for a longer time after it has obtained a near-optimal expected reward with high probability.

In the Tth iteration of URMAX(ϵ, δ, s_0), URMAX$(T, |S_0| + T - 1, |A_0| + T - 1, T, T, \epsilon, \delta, s_0)$ is run; this algorithm computes an optimal policy π' for the current approximation to the MDP (and runs π' for T steps) $K_4(T, T)$ times if no inconsistency is found. For sufficiently large values of the parameters, no inconsistency will be found. In URMAX(ϵ, δ, s_0), this is done $K_5(T) = K_4(T + 1, T + 1)\frac{2(T+1)}{\epsilon}$ times, instead of $K_4(T, T)$ times. The extra $K_5(T) - K_4(T, T)$ times compensate for the low reward obtained in the next iteration of URMAX(ϵ, δ, s_0) while the optimal policy is being recomputed.

The following theorem shows that URMAX(ϵ, δ, s_0) has the required properties.

Theorem 2.5.3: *For all MDPs $M = (S, A, g, P, R)$ compatible with S_0, g_0 and G_0, if $0 < \delta < 1$, $\epsilon > 0$, the maximum possible reward in M is $R^*_{\max}$, and the ϵ-return mixing time of M is T_M, then for all states $s_0 \in S_0$, there exists a time t^* polynomial in $|S|$, $|A|$, T_M, $R^*_{\max}$, $1/\epsilon$, $1/\delta$, and $K^*_0 = \max\{\min\{n_1 : \sum_{t=1}^{n_1} D(1, t, s) \geq \ln(4|S||A|/\delta)\} : s \in S\}$, such that, for all $t \geq t^*$, the expected average reward of running* URMAX(ϵ, δ, s_0) *on M for t steps is at least $(1 - \delta)\text{Opt}(M, \epsilon, T_M) - 3\epsilon$.*

Proof: See Appendix B. ∎

Therefore, if $\sum_{t=1}^{\infty} D(1, t, s) = \infty$ for all s, the DM can learn to play near-optimally. We now get running time estimates that essentially match the lower bounds of Theorem 2.4.4.

Proposition 2.5.4: *If, for all $s \in S$, $\sum_{t=1}^{T} D(1, t, s) \geq f(T)$, where $f : [1, \infty) \to \boldsymbol{R}$ is an increasing function whose co-domain includes $(0, \infty)$, then $K^*_0 \leq f^{-1}(\ln(4|S||A|/\delta))$, and the running time of* URMAX(ϵ, δ, s_0) *is polynomial in $f^{-1}(\ln(4|S||A|/\delta))$, $|S|$, $|A|$, T_M, $1/\epsilon$, $1/\delta$ and $R^*_{\max}$.*

Proof: Immediate from Theorem 2.5.3 and the definition of K^*_0. ∎

Recall from Theorem 2.4.4 that if there exists s such that $\sum_{t=1}^{T} D(1, t, s) \leq f(T)$, then no algorithm that learns near-optimally can run in time less than $f^{-1}(c' \ln(1/\delta))$ (where $c' = c/\ln(1/(1 - c))$), so we have proved an upper bound that essentially matches the lower bound of Theorem 2.5.2.

Corollary 2.5.5: *If $\sum_{t=1}^{T} D(1, t, s) \geq m_1 \ln(T) + m_2$ (resp., $\sum_{t=1}^{T} D(1, t, s) \geq m_1 \ln(\ln(T) + 1) + m_2$) for all $s \in S$ for some constants $m_1 > 0$ and m_2, then the DM can learn to play near-optimally in polynomial time (resp., exponential time) in $|S|$, $|A|$, T_M, $1/\epsilon$, $1/\delta$ and $R^*_{\max}$.*

Proof: If $\sum_{t=1}^{T} D(1,t,s) \geq m_1 \ln(T) + m_2$, then we can take $f(t) = m_1 \ln(T) + m_2$ in Proposition 2.5.4. As we have observed, $f^{-1}(t) = e^{(t-m_2)/m_1}$, so $f^{-1}(\ln(4|S||A|/\delta)) = e^{-m_2/m_1}(4|S||A|)^{1/m_1} \cdot (1/\delta)^{1/m_1}$. Thus, $f^{-1}(\ln(4|S||A|/\delta))$ has the form $a(|S||A|/\delta)^{1/m_1}$ where a is a constant, and is polynomial in $|S|$, $|A|$, and $1/\delta$. The result now follows from Proposition 2.5.4. The argument is similar if for all s we have $\sum_{t=1}^{T} D(1,t,s) \geq m_1 \ln(\ln(T) + 1) + m_2$; we leave details to the reader. ∎

2.6 An Application: Learning Bipedal Walking Using MDPUs

In order to test the applicability of the MDPU model on real problems, we applied it to a humanoid robot walking problem, in which we require the robot to move from the center of an arena to its boundary; we take any reasonable motion to be "walking". This is describe in detail in Chapter 3.

2.7 Conclusion

We have defined an extension of MDPs that we call MDPUs, MDPs with unawareness, to deal with the possibility that a DM may not be aware of all possible actions. We provided a complete characterization of when a DM can learn to play near-optimally in an MDPU, and an algorithm that learns to play near-optimally when it is possible to do so, as efficiently as possible. Our methods and results thus provide principles for guiding the design of complex systems.

As we hope that our examples make clear, MDPUs should be widely applicable. In cases

that the agent is unaware of certain actions (and unsure of the potential action space), we cannot easily model the problem as an MDP, while modeling it as an MDPU is relatively straightforward. We applied MDPUs to the humanoid robot walking problem; using URMAX, the robot was able to learn a number of useful walking gaits . This shows that MDPUs can also be quite useful when the potential action space is known but it is large, and only a small subset of potential actions are actually useful. In such cases, thinking in terms of MDPUs can lead to much faster algorithms for finding near-optimal policies. We will discuss this in detail in Chapter 3.

We have also shown that there are situations when an agent cannot hope to learn to play near-optimally. In this case, an obvious question to ask is what the agent should do. Work on budgeted learning has been done in the MDP setting [15; 18; 38]; we extend this to MDPUs in Chapter 4.

CHAPTER 3

MDPS WITH UNAWARENESS IN ROBOTICS

"Can a robot write a symphony? Can a robot turn a canvas into a beautiful masterpiece?" –(I, Robot)

3.1 Introduction

Markov decision processes (MDPs) are widely used for modeling decision making problems in robotics and automated control. Traditional MDPs assume that the decision maker (DM) knows all states and actions. However, in many robotics applications, the space of states and actions is continuous. To find appropriate policies, we typically discretize both states and actions. However, we do not know in advance what level of discretization is good enough for getting a good policy. Moreover, in the discretized space, the set of actions is huge. However, relatively few of the actions are "interesting". For example, when flying a robotic helicopter, only a small set of actions lead to useful flying techniques; an autonomous helicopter must learn these techniques. Similarly, a humanoid robot needs to learn various maneuvers (e.g., walking or running) that enable it to move around, but the space of potential actions that it must search to find a successful gait is huge, while most actions result in the robot losing control and falling down.

In Chapter 2, we defined *MDPs with unawareness (MDPUs)*, where a decision-maker (DM) can be unaware of the actions in an MDP. In the robotics applications in which we are interested, we can think of the DM (e.g., a humanoid robot) as being unaware of which actions are the useful actions, and thus can model what is going on using an MDPU.

In this paper, we apply MDPUs to continuous problems. We model such problems using *continuous MDPs*, where actions are performed over a continuous duration of time. Although many problems fit naturally in our continuous MDP framework, and there has been a great deal of work on continuous-time MDPs, our approach seems new, and of independent interest. (See the discussion in Section 3.4.) It is hard to find near-optimal policies in continuous MDPs. A standard approach is to use discretization. We use discretization as well, but our discrete models are MDPUs, rather than MDPs, which allows us both to use relatively few actions (the "interesting actions"), while taking into account the possibility of there being interesting actions that the DM has not yet discovered. We would like to find a discretization level for which the optimal policy in the MDP underlying the approximating MDPU provides a good approximation to the optimal policy in the continuous MDP that accurately describes the problem, and then find a near-optimal policy in that discretized MDPU.

We gave a complete characterization of when it is possible to learn to play near-optimally in an MDPU, extending earlier work [4; 31] showing that it is always possible to learn to play near-optimally in an MDP. We extend and generalize these results so as to apply them to the continuous problems of interest to us. We characterize when brute-force exploration can be used to find a near-optimal policy in our setting, and show that a variant of the URMAX algorithm presented in Chapter 2 can find a near-optimal policy. We also characterize the complexity of learning to play near-optimally in continuous problems, when more "guided" exploration is used. Finally, we discuss how MDPUs can be used to solve a real robotic problem: to enable a humanoid robot to learn walking on its own. In our experiment, the robot learned various gaits at multiple discretization levels, including both forward and backward gaits; both efficient and inefficient gaits; and both gaits that resemble human walking, and those that do not.

3.2 Analyzing robotic problems as MDPUs

As we said in the introduction, we apply the MDPU framework to robotic problems such as having a humanoid robot learn to walk. For such problems, we typically have a continuous space of states and actions, where actions take place in continuous time, and actions have a nontrivial duration.

Suppose that the original continuous problem can be characterized by a continuous MDP M_∞ (defined formally below). We would like to find a "good" discretization M of M_∞. "Good" in this setting means that an optimal policy for M is ϵ-optimal for M_∞, for some appropriate ϵ.[1] Clearly the level of discretization matters. Too coarse a discretization results in an MDP whose optimal policy is not ϵ-optimal for M_∞; on the other hand, too fine a discretization results in the problem size becoming unmanageably large. For example, in order to turn a car on a smooth curve (without drifting), the optimal policy is to slowly turn the steering wheel to the left and back, in which the action varies smoothly over time. This can be simulated using a relatively coarse discretization of time. However, in order to make a sharp turn using advanced driving techniques like drifting, the steering wheel needs to be turned at precise points in time, or else the car will go into an uncontrollable spin. In this case, a fine discretization in time is needed.

Unfortunately, it is often not clear what discretization level to use in a specific problem. Part of the DM's problem is to find the "right" level of discretization. Thus, we describe the problem in terms of a continuous MDP M_∞ and a sequence $((M_1, M'_1), (M_2, M'_2), \ldots)$, where M_i is an MDPU with underlying MDP M'_i, for $i = 1, 2, \ldots$. Intuitively, $(M'_1, M'_2, M'_3, \ldots)$ represents a sequence of finer and finer approximations to M_∞.

[1] A policy π is ϵ-optimal for an MDP M if the expected average reward for a policy for M is no more than ϵ greater than the expected average reward of π.

Continuous Time MDP with Continuous Actions over Time: To make this precise, we start by defining our model of continuous MDPs. Let $M_\infty = (S_\infty, A_\infty, g_\infty, P_\infty, R_\infty)$. S_∞ is a continuous state space, which we identify with a compact subset of $\boldsymbol{R}^n$ for some integer $n > 0$; that is, each state can be represented by a vector $(s_1, \cdots, s_n)$ of real numbers. For example, for a humanoid robot, the state space can be described by a vector which includes the robot's (x, y, z) position, and the current positions of its movable joints.

Actions: Describing A_∞ requires a little care. We assume that there is an underlying set of *basic actions* A_B, which can be identified with a compact subset of $\boldsymbol{R}^m$ for some $m > 0$; that is, each basic action can be represented by a vector $(a_1, \cdots, a_m)$ of real numbers. For example, for a humanoid robot, the basic actions can be characterized by a tuple that contains the targeted positions for its movable joints. However, we do not take A_∞ to consist of basic actions. Rather, an action is a *path* of basic actions over time. Formally, an action in A_∞ is a piecewise continuous function from a domain of the form $(0, t]$ for some $t > 0$ to basic actions. Thus, there exist time points $t_0 < t_1 < \ldots < t_k$ with $t_0 = 0$ and $t_k = t$ such that a is continuous in the interval $(t_j, t_{j+1}]$ for all $j < k$. The number t is the *length* of the action a, denoted $|a|$. We use left-open right-closed intervals here; we think of the action in the interval $(t_j, t_{j+1}]$ as describing what the DM does right after time t_j until time t_{j+1}. By analogy with the finite case, $g_\infty(s)$ is the set of actions in A_∞ available at s.

Reward and Transition Functions: We now define R_∞ and P_∞, the reward and transition functions. In a discrete MDP, the transition function P and reward function R take as arguments a pair of states and an action. Thus, for example, $P(s_1, s_2, a)$ is the probability of transitioning from s_1 to s_2 using action a, and $R(s_1, s_2, a)$ is the reward the agent gets if a transition from s_1 to s_2 is taken using action a. In our setting, what matters is the path taken by a transition according to a. Thus, we take the arguments to P_∞ and R_∞ to be tuples of the form (s_1, s_c, a),

where s_1 is a state, a is an action in A_∞ of length t, and s_c is a piecewise continuous function from $(0, t]$ to S_∞. Intuitively, s_c describes a possible path of states that the DM goes through when performing action a, such that before a starts, the DM was at s_1.[2] Note that we do not require that $\lim_{t\to 0^+} s_c(t) = s_1$. Intuitively, this means that there can be a discrete change in state at the beginning of an interval. This allows us to capture the types of discrete changes considered in *semi-MDPs* [43].

We think of $R_\infty(s_1, s_c, a)$ as the reward for transitioning from s_1 according to state path s_c via action a. We assume that R_∞ is bounded: specifically, there exists a constant c such that $R_\infty(s_1, s_c, a) < c \cdot |a|$. For state $s_1 \in S_\infty$ and action $a \in A_\infty$, we take $P_\infty(s_1, \cdot, a)$ to be a probability density function over state paths of length $|a|$ starting at s_1. P_∞ is not defined for transitions starting at terminal states.

We require R_∞ and P_∞ to be continuous functions, so that if (s_i, s_c^i, a_i) approaches (s, s_c, a) (where all the state sequences and actions have the same length t), then $R_\infty(s_i, s_c^i, a_i)$ approaches $R_\infty(s, s_c, a)$ and $P_\infty(s_i, s_c^i, a_i)$ approaches $P_\infty(s, s_c, a)$. To make the notion of "approaches" precise, we need to consider the distance between state paths and the distance between actions. Since we have identified both states (resp., basic actions) with subsets of $\boldsymbol{R}^n$ (resp., $\boldsymbol{R}^m$), this is straightforward. For definiteness, we define the distance between two vectors in $\boldsymbol{R}^n$ using the L_1 norm, so that $d(\vec{p}, \vec{q}) = \sum_1^n |p_i - q_i|$. For actions a and a' in A_∞ of the same length, define $d(a, a') = \int_{t=0}^{|a|} d(a(t), a'(t))dt$. For state paths s_c and s_c' of the same length, define $d(s_c, s_c') = \int_{t=0}^{|s_c|} d(s_c(t), s_c'(t))dt$. Finally, define $d((s_c, a), (s_c', a')) = d(s_c, s_c') + d(a, a')$. This definition of distance allows us to formalize the notion of continuity for R_∞ and P_∞. The key point of the continuity assumption is that it allows us to work with discretizations, knowing that they really do approximate the continuous MDP.

[2] We are thus implicitly assuming that the result of performing a piecewise continuous action must be a piecewise continuous state path.

Constraints on Actions: We typically do not want to take A_∞ to consist of all possible piecewise continuous functions. For one thing, some hardware and software restrictions will make certain functions infeasible. For example, turning a steering wheel back and forth 10^{20} times in one second can certainly be described by a continuous function, but is obviously infeasible in practice. But we may want to impose further constraints on A_∞ and $g_\infty(s)$.

In the discussion above, we did not place any constraints on the length of actions. When we analyze problems of interest, there is typically an upper bound on the length of actions of interest. For example, when playing table tennis using a robotic arm, the basic actions can be viewed as tuples, describing the direction of movement of the racket, the rotation of the racket, and the force being applied to the racket; actions are intuitively all possible control sequences of racket movements that are feasible according to the robot's hardware and software constraints; this includes slight movements of the racket, strokes, and prefixes of strokes. An example of a piecewise continuous action here would be to move the racket forward with a fixed force for some amount of time, and then to suddenly stop applying the force when the racket is close to the ball. We can bound the length of actions of interest to the time that a ball can be in the air between consecutive turns.

Awareness: Even with the constraints discussed above, A_∞ is typically extremely large. Of course, not all actions in A_∞ are "useful". For instance, in the helicopter example, most actions would crash the helicopter. We thus consider *potentially useful* actions. (We sometimes call them just *useful actions*.) Informally, an action is potentially useful if it is not *useless*. A useless action is one that either destroys the robot, or leaves it in an uncontrollable state, or does not change the state. For example, when flying a helicopter, actions that lead to a crash are useless, as are actions that make the helicopter lose control. More formally, given a state s, the set of useful actions at state s are the actions that transit to a different state in which the

robot is neither destroyed nor uncontrollable. Note that an action that crashes the helicopter in one state may not cause a crash in a different state. For robotics applications, we say that a robot is *aware* of an action if it identifies that action as a potentially useful action, either because it has been preprogrammed with the action (we are implicitly assuming that the robot understands all actions with which it has been programmed) or it has simulated the action. For example, a humanoid robot that has been pre-programmed with only simple walking actions, and has never tried running or simulated running before, would be unaware of running actions. Let $\bar{A}_\infty$ denote the useful actions in A_∞, and let $\bar{A}_{\infty 0}$ denote the useful actions that the robot is initially aware of. (These are usually the actions that the robot has been pre-programmed with.)

Discretization: We now consider the discretization of M_∞. We assume that, for each discretization level i, S_∞ is discretized into a finite state space S_i and A_B is discretized into a finite basic action space A_{Bi}, where $|S_1| \leq |S_2| \leq \ldots$ and $|A_{B1}| \leq |A_{B2}| \leq \ldots$. We further assume that, for all i, there exists $d_i > 0$, with $d_i \to 0$, such that for all states $s \in S_\infty$ and basic actions $a_B \in A_B$, there exists a state $s' \in S_i$ and a basic action $a_{B'} \in A_{Bi}$ such that $d(s, s') \leq d_i$, and $d(a_B, a'_B) \leq d_i$. Thus, we are assuming that the discretizations can give closer and closer approximations to all states and basic actions. At level i, we also discretize time into time slices of length t_i, where $T \geq t_1 > t_2 > \ldots$.Thus, actions at discretization level i are sequences of *constant actions* of length t_i, where a constant action is a constant function from $(0, t_i]$ to a single basic action.[3] In other words, the action lengths at discretization level i are multiples of t_i. Thus, at discretization level i, there are $\sum_{l=1}^{\lfloor T/t_i \rfloor} |A_{Bi}|^l$ possible actions. To see why, there are $|A_{Bi}|$ discrete actions at level i, and action lengths must be multiples of t_i. Thus, action lengths must have the form lt_i for some $l \leq \lfloor T/t_i \rfloor$. There are $|A_{Bi}|^l$ actions of length $l \times t_i$ at level i, and thus $\sum_{l=1}^{\lfloor T/t_i \rfloor} |A_{Bi}|^l$ actions at level i. Let A'_i consist of this set of actions. (Note that some actions

[3]Note that we are not assuming that the action space A_{i+1} is a refinement of A_i (which would further require t_{i+1} to be a multiple of t_i).

in A_i' may not be in A_∞, since certain action sequences might be infeasible due to hardware and software constraints.) Let $A_i \subseteq A_i'$ be the set of useful actions at level i.

Let M_i be the MDPU where S_i and A_i are defined above; $A_{\infty 0}$ is the set of useful actions that the DM is initially aware of; $g(s)$ is the set of useful actions at state s; $g_0(s)$ is the set of useful actions that the DM is aware of at state s; and the reward function R_i is just the restriction of R_∞ to A_i and S_i. For $s_1 \in S_i$ and $a \in A_i$, we take $P_i(s_1, \cdot, a)$ to be a probability distribution over $Q_i^{|a|}$, the set of state paths of length $|a|$ that are piecewise constant and each constant section has a length that is a multiple of t_i. For a state path $s_c \in Q_i^{|a|}$, let $P_i(s_1, s_c, a)$ be the normalized probability of traversing a state sequence that is within distance d_i of state sequence s_c when playing action a starting from state s_1. Formally, $P_i(s_1, s_c, a) = (\int_{\{s_c' : d(s_c, s_c') \le d_i\}} dP_\infty(s_1, \cdot, a))/c$, where $c = \sum_{s_c \in Q_i^{|a|}} \int_{\{s_c' : d(s_c, s_c') \le d_i\}} dP_\infty(s_1, \cdot, a)$ is a normalization constant. Since the robot is not assumed to know all useful actions at any specific discretization level, it needs to explore for the useful actions it wasn't aware of. Finally, given a specific exploration strategy, $D_i(j, t)$ describes the probability of discovering a new useful action at discretization level i, given that there are j undiscovered useful actions at level i, and the robot has explored t times without finding a useful action. We model exploration using a_0; every time the robot explores, it is playing a_0.

It remains to define the discretization of an action in A_0. In order to do this, for $a \in A_\infty$, define $a_i \in A_i$ to be *a best approximation to a in level i* if $|a_i|$ is the largest multiple of t_i that is less than or equal to $|a|$, and $\int_0^{(|a_i|)} d(a(t), a_i(t))dt$ is minimal among actions $a' \in A$ of length $|a_i|$. Intuitively, a_i is an action in A_i whose length is as close as possible to that of a and, among actions of that length, is closest in distance to a. The action a_i is not unique. For $a \in A_0$, define its discretization at level i to be a best approximation to a at that level. When there are several best approximations, we choose any one of them.

Policies: As usual, a policy π in M_i is a function from S_i to A_i. We want to compute $U_{M_i}(s, \pi, t)$, the expected average reward over time t of π started in state $s \in S_i$. Let $a_j \in A_i$ and s_{cj} be a state sequence in $Q_i^{|a_j|}$, for $j = 0, \ldots, l$. Say that a sequence $((a_0, s_{c0}), (a_1, s_{c1}), \cdots, (a_l, s_{cl}))$ is *a path compatible with policy π starting at s* if $\pi(s) = a_0$ and $\pi(s_{cj}(|a_j|)) = a_{j+1}$ for all $0 \leq j \leq l-1$. Let $I_{s,t}^{\pi}$ consist of all paths $((a_0, s_{c0}), (a_1, s_{c1}), \cdots, (a_l, s_{cl}))$ starting at s compatible with π such that $\sum_{j=0}^{l} |a_j| \leq t < \sum_{j=0}^{l+1} |a_j|$, where $a_{l+1} = \pi(s_{cl}(|a_l|))$. Essentially, when computing $U_{M_i}(s, \pi, t)$, we consider the expected reward over all maximally long paths that have total length at most t. Thus, $U_{M_i}(s, \pi, t) = \sum_{p \in I_{s,t}^{\pi}} P_i^*(p) \frac{R_i^*(p)}{t}$, where, given a path $p = ((a_0, s_{c0}), (a_1, s_{c1}), \cdots, (a_l, s_{cl}))$, $P_i^*(p) = \Pi_{j=0}^{l} P_i(s_{cj}(0), s_{cj}, a_j)$, and $R_i^*(p) = \sum_{j=0}^{l} R_i(s_{cj}(0), s_{cj}, a_j)$.

Now that we have defined the average reward of a policy at discretization level i, we can define the average reward of a policy in M_∞. Given a discretization level i, let π_i be a *projection* of π_∞ at level i, defined as follows: for each $s_i \in S_i$, define $\pi_i(s_i)$ to be an action $a_i \in A_i$ such that a_i is a best approximation to $\pi(s_i)$ at level i, as defined above. As mentioned, there might be several best approximations; a_i is not unique. Thus, the projection is not unique. Nevertheless, we define $U_{M_\infty}(s, \pi_\infty, t)$ to be $\lim_{i \to \infty} U_{M_i}(s, \pi_i, t)$, where π_i is a projection of π to discretization level i. The continuity of the transition and reward functions guarantees that the limit exists and is independent of the choice of projections.

We now consider how the URMAX algorithm of Chapter 2 can be applied to learn near-optimal policies. We use URMAX at each discretization level. Note that URMAX never terminates; however, it eventually learns to play near-optimally (although we may not know exactly when). The time it takes to learn to play near-optimally depends on the exploration strategy. The next theorem consider brute-force searching, where, at discretization level i, at each discretization level i, all actions in A_i' are exhaustively examined to find useful actions.

(The proof of this and all other theorems can be found in the supplementary material.)

Theorem 3.2.1: *Using brute-force exploration, given* $\alpha > 0$ *and* $0 < \delta < 1$*, we can find an* α*-optimal policy in* M_∞ *with probability at least* $1 - \delta$ *in time polynomial in* l*,* $|A'_l|$*,* $|S_l|$*,* $1/\alpha$*,* $1/\delta$*,* $R^l_{\max}$*, and* T^l*, where* l *is the least* i *such that the optimal policy for* M'_i *is* $(\alpha/2)$*-optimal for* M_∞*,* $R^l_{\max}$ *is the maximum reward that can be obtained by a transition in* M'_l*, and* T^l *is the* ϵ*-return mixing time for* M'_l*.*

Although brute-force exploration always learns a near-optimal policy, the method can be very inefficient, since it exhaustively checks all possible actions to find the useful ones. Thus, at discretization level i, it needs to check $\sum_{l=1}^{\lfloor T/t_i \rfloor} |A_{Bi}|^l$ actions, and as i grows, the method soon becomes impractical. On the other hand, the result is of some interest, since it shows that even when there are infinitely many possible levels of discretizations, a method as simple as brute-force exploration suffices.

When the number of possible actions is huge, the probability of finding a potentially useful action can be very low. In this case, making use of an expert's knowledge or imitating a teacher's demonstration can often greatly increase the probability of finding a useful action. We abstract the presence of an expert or a teacher by assuming that there is some constant $\beta > 0$ such that $D(1, t) \geq \beta$ for all t. Intuitively, the presence of a teacher or an expert guarantees that there is a minimal probability β such that, if there is a new action to be found at all, then the probability of finding it is at least β, no matter how many earlier failed attempts there have been at finding a useful action. For example, Abbeel and Ng (2005) study the problem of robotic helicopter flying. They assume that they have a teacher that will help demonstrate how to fly. Their assumptions imply that there is a constant $\beta > 0$ such that $D(1, t) \geq \beta$.[4]

[4]Specifically, if we take a flight with reward ϵ-close to the flight demonstrated by the teacher to be a useful

Using apprentice learning lets us improve Theorem 3.2.1 by replacing the $|A'_l|$ component of the running time by $|A_l|$; thus, with apprentice learning, the running time depends only on the number of useful actions, not the total number of potential actions. The savings can be huge.

Theorem 3.2.2: *Using an exploration method where $D_i(1,t) \geq \beta$ for all $i, t > 0$ (where $\beta \in (0,1)$ is a constant), for all $\alpha > 0$ and $0 < \delta < 1$, we can find an α-optimal policy in M_∞ with probability at least $1-\delta$ in time polynomial in l, $|A_l|$, $|S_l|$, $1/\beta$, $1/\alpha$, $1/\delta$, $R_{\max}$, and T^l, where l is the smallest i such that the optimal policy for M'_i is $(\alpha/2)$-optimal to M_∞, $R^l_{\max}$ is the maximum reward that can be obtained by a transition in M'_l, and T^l is the ϵ-return mixing time for M'_l.*

3.3 Humanoid Robot Walking

We consider the problem of a humanoid robot with 20 joint motors (which we sometimes call just "joints") learning to walk on its own. More precisely, we require the robot to move from the center of an arena to its boundary; we take any reasonable motion to be "walking". (Figure 3.1 shows the robot and the arena in which it must walk.)

3.3.1 The continuous MDP

We start by defining a continuous M_∞ for the robot problem. A state $s \in S_\infty$ is of the form $s = (w_1, \cdots, w_{23}) \in \boldsymbol{R}^{23}$, where (w_1, w_2, w_3) give the position of the robot's center of mass and

action, and take a_0 be the process of performing h iterations of the main loop in their algorithm, where $h = \frac{64HR_{\max}}{\epsilon}(2 + c\log\frac{64H^2R_{\max}|S|^3|A|}{\epsilon})$, then the probability of finding a useful action is at least $1 - e^{\frac{\epsilon}{(1+c)32HR_{\max}}}$, where $c = \frac{16^2H^2R^2_{\max}|S|^3|A|}{4\epsilon^2}$; H is the horizon, so that the procedure must terminate after H steps; $R_{\max}$ is the maximum reward; $|S|$ is the number of states; and $|A|$ is the number of actions.

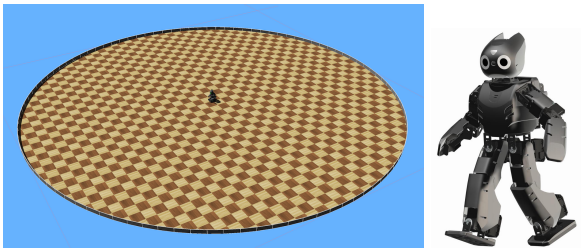

Figure 3.1: The arena with the robot at the center; and the robot.

$(w_4, \cdots, w_{23})$ are the current positions of the robot's 20 joint motors. We define the domain of each dimension as follows: Since the radius of the arena is 5 meters, $w_1, w_2 \in [-5, 5]$; since the robot's height is 0.454 meters, $w_3 \in [0, 0.4]$ (we do not expect the robot's center of mass to be higher than 0.4). Each joint motor has its specific range of mobility, which determines the domain of the corresponding dimension. For example, $w_5 \in [-3.14, 2.85]$ represents the current position of the robot's left shoulder. The mobility range for all joint motors are intervals in $[-\pi, \pi]$.

The basic actions $a \in A_B$ are of the form $a = (v_1, \cdots, v_{20}) \in \boldsymbol{R}^{20}$, where v_i is the target position for the robot's ith joint motor. The domain of each dimension is the mobility range for the corresponding joint motor. For example, v_2, which corresponds to the left shoulder, has mobility range $[-3.14, 2.85]$; $v_2 = 2.85$ means to move the robot's left shoulder forward as far as possible. Since walking is composed of repeated short movements that are typically not much longer than 0.5 seconds, we set $T = 0.512$ seconds. Thus, A_∞, the set of useful actions, consists of piecewise continuous functions that map from time to basic actions and comply with the robot's hardware and software limitations, of length $t \leq 0.512$ seconds.

We now define R_∞ and P_∞. Intuitively, the robot obtains a reward for gaining distance from the center of the arena. If the coordinates of the center of the arena are given by $s_0 = (s_0[1], s_0[2])$, then $R_\infty(s_1, s_c, a) = dis(s_0, s_c(|a|)) - dis(s_0, s_1)$, where $dis(s_0, s_1) =$

$\sqrt{(s_0[1] - s_1[1])^2 + (s_0[2] - s_1[2])^2}$ is the L_2-norm distance between s_0 and s_1 on the (x, y)-plane. The reward could be negative, for example, if the robot moves back towards the center of the arena.

By definition, $P_\infty(s_1, \cdot, a)$ is a probability distribution over state sequences of length $|a|$ starting at s_1. For example, if the robot slowly moves its right leg forward while staying balanced, the state path taken by the robot is a deterministic path. On the other hand, if a is the action of turning around quickly, $P_\infty(s, \cdot, a)$ is distribution over various ways of falling down.

3.3.2 Discretizations

We now define M_i and M_i'. In our experiments we considered only levels 2 and 3 (level 1 is uninteresting since it has just one state and one action), so these are the only levels that we describe in detail here. (These turn out to suffice to get interesting walking behaviors.) At these levels, we discretized more finely the joints corresponding to the left and right upper and lower leg joints and the left and right ankle joints, since these turn out to be more critical for walking. (These are components $(w_{14}, \cdots, w_{19})$ in the state tuples and $(v_{11}, \cdots, v_{16})$ in basic-actions tuples.) We call these the *relevant dimensions*. We assume that the six relevant state and actions components have i possible values at level i, for $i = 2, 3$, as does w_3, since this describes how high off the ground the robot is (and thus, whether or not it has fallen). All other dimensions take just one value. We took $t_2 = t_3$ to be 128ms. Since $T = 0.512$s, an action contains at most $\lfloor T/t_i \rfloor = 4$ basic actions.

$A_{\infty 0}$ is the set of preprogrammed actions. We preprogram the robot with a simple sitting action that lets the robot slowly return to its initial sitting gesture. When we consider apprenticeship learning, we also assume that the robot is preprogrammed with a "stand-up" action,

that enables it to stand up from its initial sitting position. (Intuitively, we are assuming that the expert taught the robot how to stand up, since this is useful after it has fallen.)

A'_i is the set of potential actions at level i. Given our assumptions, for $i = 2, 3$, at level i, there are $(i^6)^4$ potential actions (there are i possible values for each of the six relevant dimensions, and each action is a sequence of four basic actions). Thus, at level 3, there are $(3^6)^4$ =282,429,536,481 potential actions. As we mentioned, a useful action is an action that moves the robot without making it lose control. Here, an action is useful if it moves the robot without resulting in the robot falling down. At both levels 2 and 3, more than 80 useful actions were found in our experiments. The most efficient action found at level 3 was one where the right leg moves backwards, immediately followed by the left leg, in such a way that the robot maintains its balance at all times. By way of contrast, turning the body quickly makes the robot lose control and fall down, so is useless.

For $s_1 \in S_i$, $a \in A_i$, and $s_c \in Q_i^{|a|}$, $P_i(s_1, s_c, a)$ is the normalized probability of traversing a state sequence that is d_i close to s_c, a sequence of states in S_i, where we define $d_i = \frac{12\pi}{i} + 28\pi + 20.4$. So d_i decreases as i increases, and discretizations at a higher level better approximate the continuous problem. All basic actions in A_B are within distance d_i of a basic action in A_{Bi} and all states in S are within d_i of a state in S_i. Let $s \in S$, and let s_i be the closest state to s in S_i. It is easy to check that $d(s, s_i) \leq d_i$ for $i = 2, 3$.

The D_i function depends on the exploration method used to discover new actions. In our experiment, we used two exploration methods: brute-force exploration and apprenticeship-learning exploration.

At discretization level i, using brute-force exploration, we have $D_i(|A_i|, t) = \frac{|A_i|}{|A'_i|}$, since there are $|A_i|$ useful actions and $|A'_i|$ potential actions, and we test an action at random. With ap-

	Brute-force (level 2)	Brute-force (level 3)	Apprenticeship learning (level 2)
$\|S_i\|$	130	1460	3200
$\|A_{Bi}\|$	64	729	1600
$\|A_i\|$	16777216	282429536481	6553600000000
t_i (ms)	124	124	124
Length of action (ms)	496	496	496
Execution time (hours)	24	24	24
Best avg rwd (m/action)	0.043486	0.067599	0.083711
Num of useful actions found	131	89	180

Table 3.1: Performance comparisons.

prenticeship learning, we used following hints from a human expert to increase the probability of discovering new actions: (a) a sequence of moving directions that, according to the human expert, resembles human walking;[5] (b) a preprogrammed stand-up action; (c) the information that an action that is symmetric to a useful action is also likely to be useful (two actions are symmetric if they are exactly the same except that the target values for the left joints and those for the right joints are switched). We also use a different discretization: the ankle joint was discretized into 10 values. The human expert suggests more values in the ankle joints because whether or not the robot falls depends critically on the exact ankle joint position. These hints were provided before the policy starts running; the discretization levels are set then too. There were no further human-robot interactions.

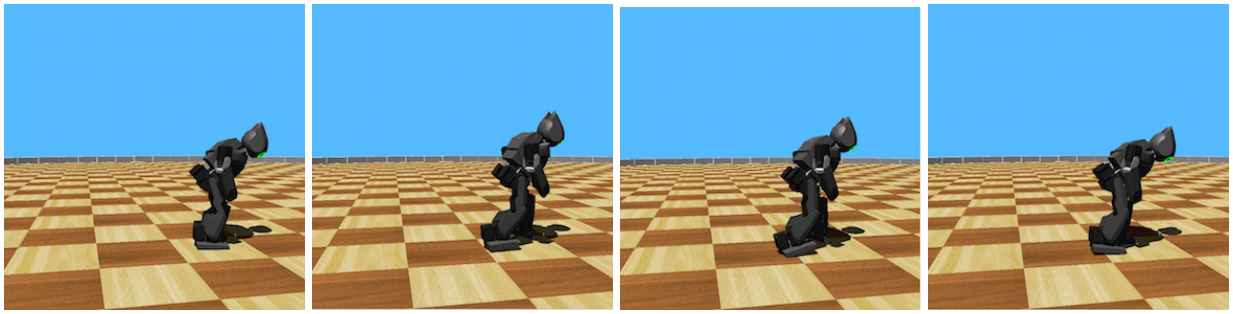

Figure 3.2: A backward gait (from left to right).

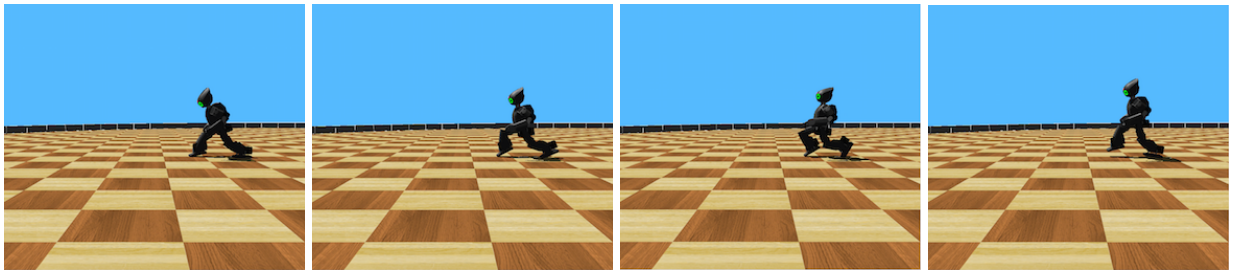

Figure 3.3: A forward gait (from left to right).

3.3.3 Experiments

For our experiments, we simulated DARwIn OP, a commercially available humanoid robot. The simulations were conducted on Webots PRO platform 8.2.1 using a MacBook Pro with 2.8GHz Intel Core i7 Processor, 16GB 1600 MHz DDR3 memory, 0.5TB Flash Storage Mac HD, on OS X Yosemite 10.10.5. We modeled the robot walking problem as an MDPU, and implemented the URMAX algorithm to solve the problem using programming language Python 2.7.7.

As we said, given the number of actions involved, we conducted experiments only for dis-

[5]The sequence gives directions only for certain joints, without specific target values, leaving the movement remaining joints open for experimentation.

cretization level 2 and 3. Both sufficed to enable the robot to learn to walk, using a generous notion of "walk"—more precisely, they sufficed to enable the robot to learn to locomote to the boundary of the arena. As mentioned, two exploration methods were used: brute-force exploration and apprenticeship-learning exploration. One trial was run for brute-force exploration at each of levels 2 and 3, and one trial was run for apprenticeship learning at level 2. Each trial took 24 hours. More than 15 stable gaits were found in total, where a gait is *stable* if it enables the robot to move from the center of the arena to the boundary without falling. In addition, more than 400 useful actions were found. The best gait among all stable gaits achieved a velocity of 0.084m/s, which seems reasonable, given that the best known walking speed of DARwIn-OP is 0.341m/s [6]. Given more time to experiment, we would expect the performance to improve further.

The robot successfully learned gaits of various styles, including both forward and backward gaits (see Figures 3.2 and 3.3), both efficient and inefficient gaits, gaits that resemble human walking and the ones that do not. Somewhat surprisingly, the best gait actually walks backwards. (Videos of some of the gaits and a demo of the learning process can be found at https://youtu.be/qW51iInpdV0.) As shown in Table 3.3.3, as the discretization level increases, both the velocity of the best gait and the number of useful actions found increase. This agrees with the expectation that finer discretization better approximates the continuous problem, and thus gets an expected reward closer to the optimal reward of the continuous problem. Apprenticeship learning resulted in more useful actions than the brute-force exploration and in gaits with a higher average reward. Again, this is hardly surprising; the hints provided by the human expert increases the probability of finding useful actions. On the other hand, when the expert gives "bad" hints, the robot performs worse than with brute-force exploration.

Our approach, using MDPUs, requires no knowledge on the kinematics of the robot other

than the number of joints and the moving range of each joint. Moreover, it makes no assumptions about the moving pattern of the resulting gait; for example, we do not assume that a gait must be cyclic, or symmetric between left and right joints, nor do we specify the length of a gait. Although we do specify the length of a useful action, a gait could be composed of a single or multiple useful actions. Given the few assumptions and little prior knowledge assumed, the performance of the robot seems quite reasonable. More importantly, the experiment proves that the use of MDPUs enables the robot to learn useful new maneuvers (walking, in this case) by itself, with minimum human input.

3.4 Related Work

There has been work on optimal policy learning in MDPs using computational resources. Kearns and Singh's [31] E^3 algorithm guarantees polynomial bounds on the resources required to achieve near-optimal return in general MDPs; variants and extensions of this work can be found in [4; 29; 30]. However, algorithms such as E^3 usually require the exploration of the entire MDP state/action space. This becomes impractical in our setting, where the number of actions is extremely large. In such cases, several exploration methods have been employed to help find useful actions. For example, Abbeel and Ng [1] utilize a teacher demonstration of the desired task to guide the exploration; Dearden et al. [10] utilize the value of information to determine the sequence of exploration. Guestrin et al. [17] make use of approximate linear programming, and focus on exploring states that directly affect the results of the planner; Kakade et al. [29] proved that in certain situations, the amount of time required to compute a near-optimal policy depends on the *covering number* of the state space, where, informally, the covering number is the number of neighborhoods required for accurate local modeling; Other papers (e.g., [9; 23]) consider MDPs with large action spaces.

We are far from the first to consider MDPs with continuous time. For example, *semi-MDPs* (SMPDs) and *continuous-time MDPs* have continuous time [43]. However, these models have discrete actions that can be taken instantaneously, and do not consider continuous actions taken over some duration of time. In *Markov decision drift processes* [25], the state does not have to stay constant between successive actions (unlike an SMDP), and can evolve in a deterministic way according to what is called a *drift function*. But Markov decision drift processes do not have actions with probabilistic outcomes that take place over an interval of time. Hordijk and van der Duyn Schouten [25] make significant use of discrete approximations to compute optimal policies, just as we do. There has also been work on MDPs with continuous state space and action space (e.g., [2; 14]), but with discrete time. For our applications, we need time, space, and actions to be continuous; this adds new complications. In control theory, there are methods for controlling continuous time systems where system transitions are linear function of state and time [55]. These can be extended to non-linear systems [32]. However, the transitions in these systems are usually deterministic, and they do not deal with rewards or policies. Sutton, Precup, and Singh [52] consider high-level actions (which they call *options*) that are taken over a duration of time (such as "opening a door"), but they view time as discrete, which significantly simplifies the model. Rachelson, Garcia, and Fabiani [44] consider continuous actions over some time interval, however, they assume there are decision epochs, which are the only time points where rewards are considered. In our model, the rewards depend on the entire state sequence that the system traverses through while an action is taken. While this makes the model more complicated, it seems more appropriate for the problems of interest to us.

There has also been a great deal of work on bipedal robot walking, since it is a fundamental motor task for which biological systems significantly outperform current robotic systems [53]. There have been three main approaches for solving the task:

- The first approach describes the kinematics of the robot in detail using non-linear and linear equation systems, then solves these systems to obtain desirable trajectories. See, for example, [27; 16; 28; 33; 51; 54].
- The second approach uses genetic algorithms [7; 22; 42]. The traits describing a gait are taken to be the genes in a genetic algorithm. Different gaits (i.e., settings of the parameters) are evaluated in terms of features such as stability and velocity; The most successful gaits are retained, and used to produce the next generation of gaits through selection, mutation, inversion, and crossover of their genes. This approach can also be used for to learn quadrupedal and nine-legged walking. See, for example, [8; 40; 41; 56].
- The third approach uses gradient learning, which starts with either a working gait or a randomly initialized gait. It then improves the gait's performance by changing its parameters, using machine-learning methods (such as neural networks) to find the most profitable set of changes in the parameters. See, for example, [6; 34; 49; 53]. this approach is also used in quadrupedal walking [35].

Since the first approach requires a full description of the robot's kinematics, as well as composing and solving a non-linear system, it requires a great deal of human input. Moreover, its application is limited to walking problems. The approach is unable to produce gaits other than those specified by human (e.g., to walk forward by stepping forward the left and the right legs in turn under a specific speed). Both the second and the third approach require little human input (when starting from random gaits), and may produce a variety of gaits. Both also have the potential to be generalized to problems other than bipedal walking. However, both are heuristic search algorithms, and have no theoretical guarantee on their performance. In contrast, our method produces a variety of gaits, provides a general framework for solving robotic problems, and produces a near-optimal policy in the limit. Moreover, our method

requires minimum human input, although, as the experiments show, it does better with more human input.

A comparison of our method and the genetic algorithm may provide insights into both methods. Although the two approaches seem different, the searching process made possible by selection, mutation, inversion, and crossover in a genetic algorithm can be viewed as a special case of the *explore* action in an MDPU. Conversely, the *explore* action in an MDPU for the robot can be roughly viewed as searching for a set of genes of unknown length (since a gait can be understood as a continuous action over an uncertain amount of time, composed of one or more shorter actions, where each shorter action is described by a set of parameters). Our approach can be viewed as being more flexible than a genetic algorithm; in a genetic algorithm, the length of the chromosome (i.e., the number of parameters that describe the gait) is fixed; only their values that give the best performance are unknown.

The recent work of Mordatch et al [39] also provides a general approach for reinforcement learning in robot tasks. Like us, they require prior knowledge only of the mobility range of each of the robot's joints, and not their kinematics; they also model the problem as an MDP. Their goal is to find an optimal trajectory (i.e., a sequence of states), such that when followed, performs a desired task (such as reaching out the robot's hand to a desired position) with minimal cost. They use neural networks to solve the cost-minimization problem. Thus, their approach does not have any guarantees of (approximate) optimality. Moreover, the complexity of their approach grows quickly as the length of the trajectory grows (while ours is polynomial in the number of useful actions, states visited, and the difficulty of discovering new actions, and thus is not significantly affected by the length of the trajectory). That said, Mordatch et al.'s method has successfully learned a few relatively simple tasks on a physical DARwIn OP2 robot, including hand reaching and leaning the robot's torso to a desired position [39], although

it has not yet been applied to walking.

3.5 Conclusion

We have provided a general approach that allows robots to learn new tasks on their own. We make no assumptions on the structure of the tasks to be learned. We proved that in the limit, the method gives a near-optimal policy. The approach can be easily applied to various robotic tasks. We illustrated this by applying it to the problem of bipedal walking. Using the approach, a humanoid robot, DARwIn OP, was able to learn various walking gaits via simulations (see https://youtu.be/qW51iInpdV0 for a video). We plan to apply our approach to more robotic tasks, such as learning to run and to walk up and down stairs. We believe the process will be quite instructive in terms of adding useful learning heuristics to our approach, both specific to these tasks and to more general robotic tasks. We are also interested in having the robot simulate learning to walk in the same way a baby does, for example, by limiting the robot's abilities initially, so that it must crawl before it walks. Part of our interest lies in seeing if such initial limitations actually make learning more efficient.

CHAPTER 4

BUDGETED LEARNING WITH UNAWARENESS

Lose no time; be always employed in something useful; cut off all unnecessary actions. –Benjamin Franklin

4.1 Introduction

In the last Chapter, we considered how to apply MDPUs to robotic problems; in this Chapter, we consider cases in which the DM has a limited time to learn a near-optimal policy. Recall that in Chapter 2, we completely characterized when a near-optimal policy can be learned for an MDPU in polynomial time; when this can be done in exponential time; and when it is impossible to learn a near-optimal policy. We also provided an algorithm that whenever possible, learns a near-optimal policy for an MDPU in polynomial time.

However, even if the DM can learn to play optimally in polynomial time, this is not helpful if the DM can make only 10 moves. Notice that all the examples given in the previous chapters remain of interest if we are given a budget (i.e., a bound on the number of moves that can be made). Indeed, they are arguably of even more practical interest: the mathematician has a limited working time to solve the math problem; the insurance buyer has a limited time to decide; and the project that develops the robot might have a tight timeline.

There has been work on budgeted learning without unawareness (i.e., where all actions are known in advance) [15; 18; 37; 38]. Madani et al. [38] first defined the *budgeted learning problem*: the problem of learning to play nearly optimally, given a budget. Much of the work on this problem has been done in the context of multi-armed bandits. But now the meaning of

"near optimal" is somewhat different than in the context of MDPUs. Rather than there being an underlying "true" multi-armed bandit problem (with a probability of success for each arm), in which case the goal would be to learn an arm with the highest expected reward (to the extent possible, given the budget), it is assumed that the DM has a prior probability on the expected reward of each arm. Moreover, it is assumed that each arm pays off either 1 ("success") or 0 ("failure"), so the expected reward is just the expected success probability. If the budget is limited, it is clearly unreasonable to expect optimal performance. Thus, given a budget h, the goal is to find a policy whose expected reward is the best among all policies that use only h steps, where the expectation is taken with respect to the DM's beliefs. Madani et al. [38] proved that the problem of finding an optimal policy is NP-hard. Guha and Munagala [18] and Goel et al. [15] each gave a polynomial-time algorithm for the budgeted learning problem that, given a budgeted learning problem B, returns an *approximately optimal policy* for B, that is, a policy whose expected reward is within a constant factor of that of the optimal policy for B.

As we observed above, in many cases of interest, not all the relevant actions are known in advance. In this paper, we consider the budgeted learning problem in the presence of unawareness. We define this formally by considering a variant of MDPUs. There are two key differences: we add a budget and, rather than assuming that there is a "true" underlying MDP, we assume a prior distribution over possible MDPs to get what we call *MDPs with unawareness, a prior, and a budget (MDPUBs)*. We now take an optimal policy in an MDPUB with budget h to be a policy that gives the highest expected reward among all policies that run in h steps, where the expected reward of a policy π is taken over the probability on the possible outcomes of running π for h steps (see Section 4.3 for a discussion).

There are a number of subtleties involved with making this precise. For example, what exactly does it mean for a DM to put a positive probability on an MDP that involves actions

that the DM is not aware of? The DM can consider an MDP that includes such actions possible, but cannot play such actions. Because we allow such actions, the notion of optimal policy used in [20] is different from that used here, even in the special case of an MDPUB M' that places probability 1 on a single MDP M (which we can think of as the "true" MDP) and has an infinite budget.

In this paper, like the earlier literature on budgeted learning, we focus on multi-armed bandits. Moreover, like the earlier literature, we assume that the DM's beliefs about the success probability of each arm is given by an (α, β) prior (also known as a *beta density*) [11] and that the success probability of the arms are independent. Without unawareness, given a policy π, this is enough to determine the probability on the outcomes of π, and hence the expected reward of π. In our setting, we need more information. Specifically, we need to know (a) the probability of there being a new arm; (b) the success probability of new arms, if a new arm is discovered; and (c) the terms $D(1, t)$ described above, which give the probability of discovering a new arm if there is a new arm to be discovered after having played explore t times since an arm was last discovered. We call this restricted class of MDPUBs *budgeted learning problems with unawareness* (BLPUs).

The main contribution of our paper is to define MDPUBs and to give an algorithm that, given a BLPU B returns a policy π that is approximately optimal for B.

4.2 Preliminaries

In this section, we briefly review (α, β)-distribution and the budgeted learning problem.

4.2.1 The (α, β) distribution

Following [15; 18; 37; 38], we use the (α, β) distribution [11] to represent the probability of the probability of success of a bandit-arm. We now describe the (α, β) distribution in more detail.

Let random variable p define the probability of success of a bandit arm. That is, $p = x$ means the bandit arm succeeds with probability x, and fails with probability $1 - x$. Consider the probability $\Pr^{\alpha,\beta}$ (where $\alpha, \beta \in \mathcal{R}^+$) on the probability of p being x whose density function is defined as follows:

$$Pr^{\alpha,\beta}(p = x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{K},$$

where $K = \int_{x=0}^{1} x^{\alpha-1}(1-x)^{\beta-1}dx$ is a normalization factor.[1] As is well known [11], (1) if a success (resp., failure) is observed, then after updating using Bayes' rule, the posterior probability of a success is given by $\Pr^{\alpha+1,\beta}$ (resp., $\Pr^{\alpha,\beta+1}$). and (2) the expected probability of success with respect to $\Pr^{\alpha,\beta}$ is $\alpha/(\alpha+\beta)$; that is, $E_{\Pr^{\alpha,\beta}}[p] = \alpha/(\alpha+\beta)$.

4.2.2 The budgeted learning problem

Previous work on budgeted learning [15; 18; 37; 38] focused on budgeted learning in multi-armed bandits. In this setting, a budgeted learning problem (BLP) can be described by a tuple $(\langle v_1, \ldots, v_n \rangle, h)$. Here there are n arms, v_i is a second-order probability on the success probability of arm i, and $h \in \mathcal{N}^+$ is the budget. Like the rest of the literature, we assume for ease of exposition that the payoff associated with success is 1 and the payoff associated with failure is 0. The BLPs considered in [15; 18; 37; 38] are a special case of the BLPs defined above, where the second-order probabilities are given by (α, β) distributions.

[1] We follow the convention of taking $\mathcal{N}$ to be the set of natural numbers, $\mathcal{R}$ to be the set of real numbers, $\mathcal{N}^+$ to be the set of positive natural numbers, and $\mathcal{R}^+$ to be the set of positive real numbers.

Goel et al. observe that, ignoring the initial state, the play of a BLP B can be defined by an MDP $M_B = (S, A, g_A, P, R)$. We describe the MDP in the special case considered by Goel et al., where v_i is an (α,β) distribution, since that makes the state space and transition probability function particularly easy to describe, but the approach works for an arbitrary second-order prior. We explain the elements in order: The state space $S = \{(\mathcal{R}^+ \times \mathcal{R}^+)^n \times \mathcal{N}, s_{fin}\}$; a state $s \in S$ is a tuple $(\langle u_1^s, u_2^s, \ldots, u_n^s\rangle, h^s)$, where $u_i^s \in \mathcal{R}^+ \times \mathcal{R}^+$ represents an (α,β) distribution (intuitively, the DM's current second-order probability on the probability of arm i returning 1), $h^s \in \mathcal{N}^+$ is the budget remaining in state s, and s_{fin} is the final state; when the MDP reaches this state, it makes no further moves. We call u_i^s the *arm-state of arm i* in s, denoted $s(i)$. The action space $A = \{test_1, \ldots, test_n, exploit_1, \ldots, exploit_n\}$, where $test_i$ is the action of testing arm i (i.e., playing arm i once); $exploit_i$ is the action of returning arm i as the winner. To see which actions can be played at each state, define $g_A(s) = A$ if $h^s \geq 1$, $g_A(s) = \{exploit_1, \ldots, exploit_n\}$ if $h^s = 0$, and $g_A(s_{fin}) = \emptyset$. That is, we assume that in a state where the budget is 0, only $exploit_i$ can be played for some $i \in \{1, \ldots, n\}$ and that no actions can be played in the final state s_{fin}. The transition probability function P reflects how an (α,β) distribution changes when a success/failure is observed. Given state $s = (\langle u_1^s, \ldots, u_n^s\rangle, h^s)$ where $u_j^s = (\alpha_j, \beta_j)$, let $u_j^+ = (\alpha_j + 1, \beta_j)$, $u_j^- = (\alpha_j, \beta_j + 1)$, $s_j^+ = (\langle u_1^s, \ldots, u_{j-1}^s, u_j^+, u_{j+1}^s, \ldots, u_n^s\rangle, h^s - 1)$ and $s_j^- = (\langle u_1^s, \ldots, u_{j-1}^s, u_j^-, u_{j+1}^s, \ldots, u_n^s\rangle, h^s - 1)$. Then, when $h^s > 0$,

$$P(s, s_j^+, test_j) = \frac{\alpha_j}{\alpha_j + \beta_j}, \quad P(s, s_j^-, test_j) = \frac{\beta_j}{\alpha_j + \beta_j}.$$

$P(s, s_{fin}, exploit_i) = 1$; exploiting an arm terminates the MDP.

To define the reward function R, we first define $Rw(u)$, the *expected reward* of arm-state $u = (\alpha,\beta)$, to be $\frac{\alpha}{\alpha+\beta}$ (i.e., the expected success probability of that arm). Define $R(s, exploit_i) = Rw(s(i))$ and $R(s, test_i) = 0$.

A *policy for M_B* is a function that maps each state to an action (in the case of a deterministic

policy) or a distribution over actions (in the case of a probabilistic policy). If π is a probabilistic policy, we write $\pi(s)[a]$ to denote the probability that π plays action a in reduced state s. We are interested in finding a policy whose reward is highest when the budget runs out.

4.3 MDPUBs and BLPUs

As we said in the introduction, our goal in this paper is to extend the work on unawareness to the setting of budgeted learning, with a focus on multi-armed bandit problems where there may be arms that the DM is unaware of. As the examples in the introduction show, this setting arises often in practice. Our first step is to consider a variant of MDPUs that allows for a budget and uncertainty regarding the underlying MDP. For ease of exposition, we associate with every MDP an initial state. This lets us more directly deal with multi-armed bandit problems, where there is an initial state.

Formally, an MDPUB is a tuple $Q = (A_0, a_0, D, h, s_0, \mu)$, where A_0 is a set of actions; a_0 is the special *explore* action; D is a discovery probability function (as in an MDPU); $h \in \mathcal{N}^+$ is a budget, μ is a distribution over MDPs; and s_0 is the initial state in all the MDPs in the support of Q (i.e., all MDPs that have non-zero probability in μ). For each MDP M in the support of μ, we assume that each state s in the state space of M includes actions other than those in A_0 that the DM is aware of, in the order that they were discovered, and the number of times that a_0 has been played since the last time an action was discovered (or since the initial state, if no new action has yet been discovered). We call this the *action component* of state s. Thus, in s_0, the action component is $(\langle\,\rangle, 0)$. In general, the action component has the form $(\langle b_1, \ldots, b_k\rangle, t)$.

The action component serves a number of useful purposes. Suppose that there is some MDP M that the DM considers possible where there are two actions that she is not aware

of, say b and c. If she plays a_0 (the special *explore* action) in a state $(\langle\,\rangle, t)$ and discovers a new action, she will not in general know which of b or c she has discovered. If she in fact discovers b, she will move to a state where the action component is $(\langle b\rangle, 0)$; if she discovers c, the action component will be $(\langle c\rangle, 0)$. Since in an MDP an agent is always assumed to know the current state, rather than having the uncertainty be described by the transition function in M, we assume that rather than M, the DM considers two MDPs, M^b and M^c, possible; in M^b, it is b that is discovered (if an action is discovered at all); in M^c it is c. The DM's uncertainty over which action is discovered is then described by the relative probabilities of M^b and M^c. More generally, we assume that, for each state s in an MDP M, there is a unique action a_s not in the action component of s such that if an action is discovered in state s after playing a_0, it is necessarily a_s.

We can now describe the transition probability function for the action a_0. In a state $s = (s', (\langle b_1, \ldots, b_k\rangle, t))$ in an MDP M where there are no further actions to be discovered, after a_0 is played, the DM transitions to state $(s', (\langle b_1, \ldots, b_k\rangle, t+1))$ with probability 1. If there are m actions to be discovered in the MDP, then the DM transitions to $(s', (\langle b_1, \ldots, b_k\rangle, t+1))$ with probability $1 - D(m, t+1)$ and to $(s', (\langle b_1, \ldots, b_k, a_s\rangle, 0))$ with probability $D(m, t+1)$.[2]

Another way to think about what is going on here is that we really have a POMDP (partially observable MDP). We assume that the DM knows everything about the state except for the action component. To make this precise, let the *reduced state* be the result of replacing the action component $(\langle a_1, \ldots, a_k\rangle, t))$ in a state s with (k, t) (i.e., k is the number of new actions discovered). We assume that all the states that a DM considers possible after playing a_0 at s have the same reduced state; the DM knows how many actions she has discovered and how long it has been since she last discovered an action. Since a DM's policy can depend only on what the DM knows, the policy must depend only on the reduced state. That is, in an

[2]Thus, unlike [20], here, after playing *explore*, the agent always moves to a different state.

MDPUB, a probabilistic policy is a function from reduced states to distributions over actions. In a state where the action component in the reduced state is (k, t), the DM can either play an action $a \in A_0$ (i.e., one of the actions of which she was originally aware) or she can play the jth newly-discovered action, for $1 \leq j \leq k$. That is, the policy can refer to a new action in terms of the order in which new actions were discovered. Playing the first new action in the example above would result in playing either b or c, depending on which was in fact the action discovered.

Just as in the case of a budgeted learning problem without unawareness, playing a policy π induces a distribution over final states (i.e., the states reached after the budget has run out). For each final reduced state, we can (at least in principle) compute the policy with the best expected return, and then take the expected return of π to be the expected return of these optimal policies, where this expectation is taken with respect to the distribution over final reduced states induced by π.

This model greatly simplifies if we consider bandit problems and make some reasonable assumptions. Then we can characterize the probability μ using two features: (1) the probability of there being an undiscovered arm (which we take to be a fixed probability γ, independent of how many arms have been discovered) and (2) a second-order probability v on the probability of success of a new arm, if one is discovered. We take this second-order probability to be an (α, β) probability. Thus, we take a *budgeted learning problem with unawareness* (BLPU) to be a tuple $B = (s_0, h, D, \gamma, v_0)$, where $s_0 = (v_1, \ldots, v_n)$ (i.e., there are initially n known arms) and, just as in the case with complete awareness, v_i is a second-order (α, β) probability on the probability of success of arm i, h is the budget, D is the discovery probability function, $0 \leq \gamma < 1$ is the probability of there being an undiscovered arm, and v is an (α, β) probability on the success probability of each new arm, if there is one. There may be more than one new

arm. The probabilities are taken to be independent, so that the probability of there being at least two additional arms is γ^2, the probability of there being exactly two additional arms is $\gamma^2(1-\gamma)$, and so on. Every time an additional arm is discovered, the second-order probability on the probability of success is initially taken to be v_0.

A BLPU $B = (s_0, h, D, \gamma, v_0)$ can be viewed as describing an MDPUB $M_B = (A_0, a_0, D, h, s_0^+, \mu)$, where $A_0 = \{test_1, \ldots, test_n, exploit_1, \ldots, exploit_n\}$, s_0^+ is the initial state (described in more detail below), and μ puts probability $\gamma^k(1-\gamma)$ on the MDP M^k in which there are exactly $n+k$ arms, where initially arms $a_1, \ldots, a_n$ have a second-order probability of success characterized by s_0^+, and the remaining k arms have a second-order probability of success characterized by v_0. The states s in M^k have the form $(\langle u_1^s, \ldots, u_{n+j}^s\rangle, h^s, \langle b_1^s, \ldots, b_j^s\rangle, t^s)$, where $0 \leq j \leq k$. Intuitively, $\langle b_1^s, \ldots, b_j^s\rangle$ represents the arms not in A_0 that have been discovered, $\langle u_1^s, \ldots, u_{n+j}^s\rangle$ gives the second-order probability of (the probability of) success for the n arms in A_0 and the j new arms, h^s is the remaining budget, and t is the number of times that *explore* has been played since the last time an arm was discovered. All the arms in M^k not explicitly listed in the state are assumed to have second-order probability of success v_0; since they have not been discovered, they have not been played, so this probability has not changed. The initial state s_0^+ is just $\langle s_0, h, \langle\,\rangle, 0)$. The transition probability function in M^k is as discussed above.

By the symmetry of the situation, we can make one further simplifying assumption: we assume without loss of generality that arms are discovered in the same order in all the MDPs. Thus, for example, in all the MDPs that have at least two arms b_1 and b_2 beyond those in A_0, b_1 is always the first new arm discovered and b_2 is always the second new arm discovered. Thus, we can in fact reconstruct the state from the reduced state. Unlike general MDPUBs, there is no uncertainty about which arm was discovered.

While there are quite a few assumptions at play here, we would argue that they are all in the spirit of assumptions that have been made earlier. In particular, the fact that all undiscovered arms have the same second-order probability on success probabilities just says that, a priori, the DM has no way of distinguishing one new arm from another, so she treats them all the same way.

These assumptions allow us to compute the expected profit of a policy π in B in a straightforward way. We simply compute the expected reward of π in each MDP M^k, multiply it by the probability of M^k (i.e., $\gamma^k(1-\gamma)$), and sum over all k.

4.4 An approximately optimal policy for BLPUs

Given a BLPU B, we now provide a way of computing an approximately optimal policy that we denote π^ϵ_B. Since the number of reduced states in B is exponential in the number number of arms initially known in B and the budget h, we cannot explicitly define a policy in B. Rather, what we do is, given B as input, generate an algorithm in polynomial time that takes as input a reduced state s and, in polynomial time, computes a distribution over actions to be played at s. (For reasons that will become clear in the course of the proof, the algorithm also takes a parameter $\epsilon > 0$ as an input.) We then argue that the algorithm represents an approximately optimal policy.

Let R^π_B be the expected reward obtained by policy π starting at the initial state s of B, and let R^*_B be the expected reward obtained by an optimal policy for B.

Theorem 4.4.1: *Given $\epsilon > 0$ and a BLPU $B = (s_0, h, D, \gamma, v_0)$ in which the discovery probability function $D(j,t)$ is a decreasing function of t, there exists a policy π^ϵ_B that can be computed*

in time polynomial in n (the number of initially known arms in B), h, $\log \frac{1}{\epsilon}$, $\log \frac{1}{D(1,1)}$, $\log \frac{1}{Rw(v_0)}$ *and* $\log \frac{1}{\gamma(1-\gamma)}$, *such that* $R_B^{\pi_B^\epsilon} \geq (\frac{\delta_{h-1}}{4\gamma} - \epsilon)R_B^*$ *where* $\delta_{h-1} = \frac{\sum_{i=1}^{\infty} \gamma^i (1-D(i,1))...(1-D(i,h-1))D(i,h)}{\sum_{i=0}^{\infty} \gamma^i (1-D(i,1))...(1-D(i,h-1))}$.

Proof: The proof is somewhat long and complicated. To help the reader navigate the proof, we have structured the proof into sections.

Representing reduced states: Before going into the details of the construction of π_B^ϵ, we need to establish some notation.

Recall that a policy in the BPLU B gets as input a reduced state. Given our assumptions about BLPUs, we can represent a reduced state in a simpler form: as a tuple $(\langle u_1^s, \ldots, u_n^s, \ldots, u_{n+k}^s, (t^s, undisc)\rangle, h^s)$. The interpretation is that k new arms have been discovered; $u_1^s, \ldots, u_n^s, \ldots, u_{n+k}^s$ are the (α,β) distributions describing the success probability of the original n arms and the k new arms; h^s is the remaining budget; and t^s is the number of times that a_0 has been played since the last time a new arm was discovered (or since the beginning, if no new arms have been discovered). We can think of $(t^s, undisc)$ as representing the state of the next undiscovered arm (if there is one). Let s_0 be the initial state of B and let RS be the set of reduced states of B, represented as above.

Constraints on reaching arm states: In order to describe the optimal policy, it is useful to understand the probabilities of reaching various arm-states in a reduced state. Note that if the budget of B is h, there are $O(h^2)$ arm-states for an arm i that can be reached from the initial state of arm i, since an arm-state for i is determined by the number of successes and failures of the arm if i has been discovered, and is determined by the number of time a_0 has been played if i undiscovered. As suggested above, if i is the next arm to be discovered in s, then we define i's

arm-state to be (t^s, undisc); that is, a_0 has been played t^s times without arm i being discovered. Given a policy π, we will be interested in the probability of transitioning from one arm-state u for arm i to another. In the case where i is an arm that has already been discovered and its arm-state is $u = (\alpha, \beta)$, then this is straightforward. To describe these transition probabilities, we need some notation.

Given $s \in RS$, we can compute the probability that s is reached from s_0 in the MDPU M^k (where there are exactly k undiscovered actions) by a policy π, denoted $\Pr^{\pi,k}_{s_0}(s)$, using the transition probabilities of M^k, in a straightforward way. (If more than k arms have been discovered in s, this probability is 0.) The probability that s is reached by π in BLPU B, denoted $\Pr^{\pi}_{s_0}(s)$, is just $\sum_{k=0}^{\infty} \gamma^k (1-\gamma) \Pr^{\pi,k}_{s_0}(s)$.

Let $\mathit{Visit}^{\pi}(u, i)$ be the probability of π reaching arm-state u of arm i (from s_0);

$$\mathit{Visit}^{\pi}(u, i) = \sum_{\{s \in RS : u^s_i = u\}} \Pr^{\pi}_{s_0}(s).$$

Note that if u has the form $(0, \mathit{undisc})$, then we take $\mathit{Visit}^{\pi}(u, i)$ to be the probability of reaching a reduced state where i is the next undiscovered arm, but a_0 has not been played since the last arm was discovered (or a_0 has not been played at all, if i is the first undiscovered arm). Let $\mathit{Test}^{\pi}_i(u, i)$ be the expected probability that π reaches arm-state u of arm i and plays test_i:

$$\mathit{Test}^{\pi}_i(u, i) = \sum_{\{s \in RS : u^s_i = u\}} \pi(s)[\mathit{test}_i] \Pr^{\pi}_{s_0}(s).$$

(Recall the $\pi[s](a)$ denotes the probability that policy π plays action a in state s.) Finally, overloading our earlier notation, if u and v are arm-states of an arm i that has been discovered, if $u = (\alpha, \beta)$ and $v = (\alpha + 1, \beta)$, define $P^{\pi}(u, v, \mathit{test}_i) = \alpha/(\alpha + \beta)$, while if $v = (\alpha, \beta + 1)$, define $P^{\pi}(u, v, \mathit{test}_i) = \beta/(\alpha + \beta)$. If i is initially known and $s_0(i) = (\alpha, \beta)$, then define $U^h_i = \{(\alpha + j, \beta + k) : j + k \leq h\}$; if i is an initially undiscovered arm and v_0 (the arm state after an initially undiscovered arm is discovered) is (α_0, β_0), then $U^h_i = \{(t, \mathit{undisc}) : t \in \{0, \ldots, h -$

$1\}\} \cup \{(\alpha_0 + j, \beta_0 + k) : j + k \leq h - 1\}$. U_i^h can be thought of as the set of arm-states of arm i that can be reached from s_0 in at at most h steps. This is true if i is an initially known arm. If i is an initially undiscovered arm, then U_i^h is actually a superset of the arm-states that can be reached in h steps from s_0. For suppose that i is the second undiscovered arm. Then the first undiscovered arm must be discovered before i can be played, which means that a_0 must have been played at least once before i can be played. This, in turn, means that the budget is at most $h - 2$ once i is discovered, so, for example, $(\alpha_0 + h - 1, \beta_0)$ cannot be reached although it is in U_i^h. It is useful for our proofs to have U_i^h be defined in the same way for all undiscovered arms.

It is easy to check that if $v \in U_i^h$ is a non-initial arm-state of an arm i that has been discovered, then we have

$$Visit^{\pi}(v, i) = \sum_{u \in U_i^h} P(u, v, test_i) Test_i^{\pi}(u, i). \tag{4.1}$$

(Note that the only arm-states u ones that contribute to this sum are the two possible predecessors of v.) The equality (4.1) holds for each MDPU M^k, so it holds for B.

We want to compute the analogous transition probabilities if i is an initially undiscovered arm. We claim that

$$Visit^{\pi}((t + 1, undisc), i) = (1 - \delta_t) Explore^{\pi}((undisc, t), i), \tag{4.2}$$

where

$$Explore^{\pi}(u, i) = \sum_{\{s \in RS : u_i^s = u\}} \pi(s)[a_0] \Pr_{s_0}^{\pi}(s)$$

is the expected probability of π reaching arm-state u for i and playing a_0 (and thus is the obvious analogue of $Test_i^{\pi}(u, i)$) and

$$\delta_t = \frac{\sum_{i=1}^{\infty} \gamma^i (1 - D(i, 1)) \dots (1 - D(i, t)) D(i, t + 1)}{\sum_{i=0}^{\infty} \gamma^i (1 - D(i, 1)) \dots (1 - D(i, t))} \tag{4.3}$$

is the probability of discovering a new arm the $(t + 1)$st time that a_0 is played after the last new arm was discovered, conditional on not discovering a new arm the previous t times that

a_0 was played (we prove this below), assuming that the budget does not run out. (We take $\delta_0 = (1-\gamma)\sum_{i=1}^{\infty}\gamma^i D(i,1)$, since we take the empty product $(1-D(i,1))\ldots,(1-D(i,0))$ to be 1.) Similarly,

$$Visit^{\pi}(v_0, i) = \sum_{t=0}^{h-1} \delta_t Explore^{\pi}((t, undisc), i), \tag{4.4}$$

where v_0 is the arm-state of i when it is first discovered.

We now prove (4.3). Let $E_{dis}(m, t+1)$ be the event of discovering the mth new arm the $(t+1)$st time that a_0 is played after discovering the $(m-1)$st arm (or the $(t+1)$st time that a_0 is played, if $m = 1$). Let $E_{undis}(m, t)$ be the event that the mth arm is not discovered after playing a_0 t times after discovering the $(m-1)$st action. Note that $E_{dis}(m, t+1) \subseteq E_{undis}(m, t)$; in order to discover the mth arm the $(t+1)$st time that a_0 is played, the arm must not have been discovered earlier. The probability of these events depends on the underlying MDP. The probability of $E_{undis}(m, t)$ is 0 in the MDP M^j for $j < m$, and the probability of $E_{dis}(m, t)$ is 0 in M^j for $j \leq m$. By definition, $\delta_t = \Pr_{s_0}^{\pi}(E_{dis}(m, t+1) \mid E_{undis}(m, t))$.

A straightforward computation shows that

$$\begin{aligned}
& \Pr_{s_0}^{\pi}(E_{dis}(m, t+1)) \\
= \; & \textstyle\sum_{j=m}^{\infty} \Pr_{s_0}^{\pi,j}(E_{dis}(m, t+1))\gamma^j(1-\gamma) \quad [\text{since } \Pr_{s_0}^{\pi,j}(E_{dis}(m, t+1)) = 0 \text{ if } j < m] \\
= \; & \textstyle\sum_{j=m}^{\infty}(\prod_{j'=1}^{t}(1 - D(j-(m-1), j')))D(j-(m-1), t+1) \cdot \gamma^j(1-\gamma) \\
= \; & \gamma^{m-1}(1-\gamma)\textstyle\sum_{i=1}^{\infty}\gamma^i(\prod_{j'=1}^{t}(1 - D(i, j')))D(i, t+1).
\end{aligned}$$

The second equality follows from the observation that $\Pr_{s_0}^{\pi,j}(E_{dis}(m, t+1)) = \prod_{j'=1}^{t}(1 - D(j - (m-1), j')))D(j-(m-1), t+1)$, which in turn follows from the fact that in M^j, there are $j-(m-1)$ undiscovered actions after $m-1$ actions have been discovered, so the probability of discovering the mth action the j'th time that a_0 is played is $D(j-(m-1), j')$.

Similarly,

$$\begin{aligned}
& \Pr^{\pi}_{s_0}(E_{undis}(m,t)) \\
= \; & \textstyle\sum_{j=m-1}^{\infty} \Pr^{\pi,j}_{s_0}(E_{undis}(m,t))\gamma^j(1-\gamma) \\
= \; & \textstyle\sum_{j=m-1}^{\infty}(\prod_{j'=1}^{t}(1-D(j-(m-1),j')))\gamma^j(1-\gamma) \\
= \; & \gamma^{m-1}(1-\gamma)\textstyle\sum_{i=0}^{\infty}\gamma^i(\prod_{j'=1}^{t}(1-D(i,j'))).
\end{aligned}$$

Since $E_{dis}(m,t+1) \subseteq E_{undis}(m,t)$, we have

$$\begin{aligned}
\delta_t &= \Pr^{\pi}_{s_0}(E_{dis}(m,t+1) \mid (E_{undis}(m,t)) \\
&= \frac{\Pr^{\pi}_{s_0}(E_{dis}(m,t+1)}{\Pr^{\pi}_{s_0}(E_{undis}(m,t))} \\
&= \frac{\gamma^{m-1}(1-\gamma)\sum_{i=1}^{\infty}\gamma^i(\prod_{j'=1}^{t}(1-D(i,j')))D(i,t+1)}{\gamma^{m-1}(1-\gamma)\sum_{i=0}^{\infty}\gamma^i(\prod_{j'=1}^{t}(1-D(i,j')))} \\
&= \frac{\sum_{i=1}^{\infty}\gamma^i(\prod_{j'=1}^{t}(1-D(i,j')))D(i,t+1)}{\sum_{i=0}^{\infty}\gamma^i(\prod_{j'=1}^{t}(1-D(i,j')))},
\end{aligned}$$

as desired.

Note that the expression for δ_t involves infinite sums. In order to compute an approximation to the optimal policy in B in polynomial time, we will have to approximate δ_t. We show how to approximate δ_t efficiently later; for now, we work with δ_t.

Equation (4.2) follows from the definition of δ_t; for (4.4), note that arm i reaches v_0 exactly if it reaches (t, undisc) for some $0 \le t \le h-1$ and then arm i is discovered when a_0 is played, which happens with probability δ_t.

Lemma 4.4.2: *$\delta_t \ge \delta_{t'}$ for all $0 \le t < t'$.*

Proof: We prove this by showing that $\delta_t \ge \delta_{t+1}$ for all $t \ge 0$. Let $f(i,t) = (1-D(i,1))\ldots(1-D(i,t))$, and let $f'(l,t) = \frac{\sum_{j=1}^{l}\gamma^j f(j,t+1)D(j,t+2)+\frac{f(l+1,t+1)}{f(l+1,t)}\sum_{j=l+1}^{\infty}\gamma^j f(j,t)D(j,t+2)}{\sum_{j=0}^{l}\gamma^j f(j,t+1)+\frac{f(l+1,t+1)}{f(l+1,t)}\sum_{j=l+1}^{\infty}\gamma^j f(j,t)}$, so $f'(\infty,t) = \delta_{t+1}$. We show that $f'(0,t) \le \frac{\sum_{j=1}^{\infty}\gamma^j f(j,t)D(j,t+2)}{\sum_{j=0}^{\infty}\gamma^j f(j,t)}$ and that $f'(l,t)$ decreases in l. Since $D(j,t)$ decreases in t, it then

follows that

$$\delta_{t+1} = f'(\infty, t) \leq f'(0, t) = \frac{\sum_{j=1}^{\infty} \gamma^j f(j,t) D(j,t+2)}{\sum_{j=0}^{\infty} \gamma^j f(j,t)}$$
$$\leq \frac{\sum_{j=1}^{\infty} \gamma^j f(j,t) D(j,t+1)}{\sum_{j=0}^{\infty} \gamma^j f(j,t)} = \delta_t,$$

as desired.

To see that $f'(l,t)$ decreases in l, for $l \geq 0$, let $c_1 = \sum_{j=1}^{l} \gamma^j f(j,t+1) D(j,t+2) + (1 - D(l+1, t+1))\gamma^{l+1} f(l+1,t) D(l+1,t+2)$; $c_2 = \sum_{j=l+2}^{\infty} \gamma^j f(j,t) D(j,t+2)$; $c_3 = \sum_{j=0}^{l} \gamma^j f(j,t+1) + (1 - D(l+1,t+1))\gamma^{l+1} f(l+1,t)$; $c_4 = \sum_{j=l+2}^{\infty} \gamma^j f(j,t)$. So

$$\begin{aligned}
& f'(l,t) - f'(l+1,t) \\
= \; & \frac{c_1 + (1 - D(l+1,t+1))c_2}{c_3 + (1 - D(l+1,t+1))c_4} - \frac{c_1 + (1 - D(l+2,t+1))c_2}{c_3 + (1 - D(l+2,t+1))c_4} \quad [\text{since } \tfrac{f(j,t+1)}{f(j,t)} = 1 - D(j,t+1)] \\
= \; & \frac{(c_2 c_3 - c_1 c_4)(D(l+2,t+1) - D(l+1,t+1))}{(c_3 + (1 - D(l+1,t+1))c_4)(c_3 + (1 - D(l+2,t+1))c_4)} \\
\geq \; & 0, \quad [\text{see below}]
\end{aligned}$$

as desired. For the last inequality, note that $D(l+2, t+1) \geq D(l+1, t+1)$ since $D(l,t)$ increases in l, so we just need to show that $c_2 c_3 - c_1 c_4 \geq 0$. To see this, first observe that

$$\begin{aligned}
c_2 &= \sum_{j=l+2}^{\infty} \gamma^j f(j,t) D(j,t+2) \\
&\geq \sum_{j=l+2}^{\infty} \gamma^j f(j,t) D(l+2,t+2) \\
&= D(l+2,t+2) c_4 \;\; [\text{since } D(j,t) \text{ increases in } j];
\end{aligned}$$

similarly,

$$\begin{aligned}
c_1 &= \sum_{j=1}^{l} \gamma^j f(j,t+1) D(j,t+2) + \gamma^{l+1} f(l+1,t+1) D(l+1,t+2) \\
&\leq \sum_{j=1}^{l} \gamma^j f(j,t+1) D(l+2,t+2) + \gamma^{l+1} f(l+1,t+1) D(l+2,t+2) \\
&= D(l+2,t+2) c_3.
\end{aligned}$$

Thus,

$$c_2 c_3 - c_1 c_4 \geq D(l+2,t+2) c_4 c_3 - D(l+2,t+2) c_3 c_4 = 0.$$

The fact that $f'(0,t) \leq \frac{\sum_{j=1}^{\infty} \gamma^j f(j,t) D(j,t+2)}{\sum_{j=0}^{\infty} \gamma^j f(j,t)}$ is almost immediate from the definition; we leave details to the reader. ∎

A system of linear inequalities: Following Guha and Munagala [18], we construct a system of linear inequalities that describes some of the constraints on policies. To do so, we need just a little more notation. Suppose that initially there are n known arms in the BLPU B. With a budget of h, at most h more arms can be discovered. Let $I^B = \{1, \ldots, n + h\}$; these are the arms that can potentially be played by π. Let $Exploit_i^\pi(u, i)$ denote the probability of π reaching arm-state u for arm i and playing $exploit_i$. Recall that U_i^h is the set of possible arm states for arm $i \in I^B$ reachable from the initial state in at most h steps. It is clear that the expected reward of π is just $\sum_{i \in I^B} \sum_{u \in U_i^h} Exploit_i^\pi(u) Rw(u)$.

The system involves variables $Exploit_i(u, i)$, $Test_i(u, i)$, $Visit(u, i)$, $Explore(u, i)$ for $i \in I^B$ and $u \in U_i^h$. Roughly speaking, the intention is that a solution of the system will suggest a policy π such that for each variable X, X^π will have the value of X in the solution of the system. (For example, $Test_i^\pi(u, i)$ will have the value of the variable $Test_i(u, i)$ in an optimal solution to (4.5).) Let $U_i^h(r)$ consist of all arm-states for i that have the form (α, β); these are the arm-states for i after i has been discovered.

$$\text{Maximize} \sum_{i \in I^B} \sum_{u \in U_i^h} Exploit_i(u, i) Rw(u) \tag{4.5}$$

subject to:

$$\sum_{i \in I^B} \sum_{u \in U_i^h} (Test_i(u,i) + Explore(u,i)) \leq h$$

$$\sum_{i \in I^B} \sum_{u \in U_i^h} Exploit_i(u,i) \leq 1$$

$$\sum_{u \in U_i^h(r)} Test_i(u,i) P(u,v,test_i) = Visit(v,i)$$

if $v \in U_i^h(r) - U_i^0(r)$ and $v \neq v_0$ if i is initially undiscovered

$$Exploit_i(u,i) + Test_i(u,i) + Explore(u,i) \leq Visit(u,i)$$

$$Visit((t+1, undisc), i) = (1 - \delta_t) Explore((t, undisc), i)$$

$$Visit(v_0, i) = \sum_{t=0}^{h-1} \delta_t Explore((t, undisc), i)$$

$$Exploit_i(u,i) = Test_i(u,i) = 0 \text{ if } u \in U_i^h - U_i^h(r)$$

$$Explore(u,i) = 0 \text{ if } u \in U_i^h(r)$$

$$Test_i(u,i) = Explore(u,i) = 0 \text{ if } u \in U_i^h - U_i^{h-1}$$

$$Visit(u,i) = 1 \text{ if } u \in U_i^0$$

$$0 \leq Explore(u,i), Exploit_i(u,i), Test_i(u,i), Visit(u,i) \leq 1.$$

We explain the constraints in order: (i) the total expected number of tests and a_0 played does not exceed the budget h; (ii) the total probability of exploitation is at most 1; (iii) the probability of reaching arm i in an arm-state of the form (α, β), which is not the initial state of i, and is not v_0 if i is initially undiscovered, is determined by equation (4.1); (iv) the total probability of playing $test_i$, a_0 or $exploit_i$ at an arm-state u does not exceed the probability of visiting u; (v) the probability of visiting a state of the form $((t+1), undisc)$ is determined by (4.2), (vi) the probability of discovering an initially undiscovered arm is given by (4.4); (vii) $exploit_i$ or $test_i$ are not played if i is undiscovered; (viii) a_0 is not played at an arm-state of i where i is already discovered; (ix) $test_i$ and a_0 are not played after the budget is exhausted; (x) the probability of visiting the initial arm-state of an arm is 1; and (xi) all the probabilities are between 0 and 1. Clearly, all the constraints in the linear program are satisfied by a legal policy. Thus, the value $\hat{R}$ given by an optimal solution for (4.5) is at least as large as the maximum expected reward R_B^* among all legal policies. That is, $\hat{R} \geq R_B^*$.

Suppose that we have an optimal solution $\vec{x}$ to the linear program (4.5); let x_X denote the value of the variable X in this solution (so that, for example, $x_{Test_i(u,i)}$ is the value of $Test_i(u,i)$ in the vector $\vec{x}$). Note that there is a solution to the variables where, for all initially undiscovered arms i and j, we have $x_{Test_i(u,i)} = x_{Test_j(u,i)}$, $x_{Exploit_i(u,i)} = x_{Exploit_j(u,i)}$, and $x_{Explore(u,i)} = x_{Explore(u,j)}$.

Lemma 4.4.3: *There is an optimal solution $\vec{x}$ to the system (4.5) such that for all initially undiscovered arms i and j, we have $x_{Test_i(u,i)} = x_{Test_j(u,j)}$, $x_{exploit_i(u,i)} = x_{Exploit_j(u,j)}$, and $x_{Explore(u,i)} = x_{Explore(u,j)}$ for $u \in U_i^h$.*

Proof: Note that all the undiscovered arms satisfy exactly the same equations in the system (4.5). Let $\vec{x}'$ be an optimal solution to (4.5). For all initially undiscovered arms i and all $u \in U_i^h$, let

$$x_{Test_i(u,i)} = \frac{\sum_{n+1 \le j \le n+h} x'_{Test_j(u,j)}}{h},$$

$$x_{Exploit_i(u,i)} = \frac{\sum_{n+1 \le j \le n+h} x'_{Exploit_j(u,j)}}{h},$$

$$x_{Explore(u,i)} = \frac{\sum_{n+1 \le j \le n+h} x'_{Explore(u,j)}}{h},$$

$$x_{Visit(u,i)} = \frac{\sum_{n+1 \le j \le n+h} x'_{Visit(u,j)}}{h}.$$

For all other variables $x_k \in \vec{x}$, let $x_k = x'_k$. It is easy to check that $\vec{x}$ satisfies (4.5), and it is an optimal solution (since it has the same value as $\vec{x}'$). It also satisfies the requirements of the lemma. ∎

Approximating the optimal policy: We now use a solution of the linear system satisfying the conditions of Lemma 4.4.3 of inequalities to construct an approximation to the optimal policy. The first step is to construct a policy π_i for each arm $i \in I^B$ such that the expected reward of π_i is the expected reward that arm i contributes in (4.5) (i.e., $\sum_{u \in U_i^h} Exploit_i(u,i) Rw(u)$).

The policies π_i are actually not policies in the BLPU B, but in a related MDP M_B, where we essentially ignore the budget constraints. That is, now moves can be made even if $h^s = 0$, and the transition probabilities are just those that would be obtained if $h^s > 0$ (even if $h^s = 0$. We omit the formal details here. Moreover, in M_B, an additional action *abandon* can be played in every state; if *abandon* is played, then the policy terminates (i.e., transitions to the state s_{fin} with probability 1) and gets reward 0. (We do not use *abandon* in policies for the BLPU; it is just an auxiliary construct used here.)

The policy π_i always plays one of the actions $test_i$, $exploit_i$, a_0, or *abandon*; its action depends at a state s depends only on $s(i)$, i's arm-state in s.

Definition 4.4.4: Let $\vec{x}$ be an optimal solution to (4.5). Policy π_i proceeds as follows: In state $s \neq s_{fin}$,

- If $s(i)$ is defined (i.e., either arm i has been discovered in s or i is the next undiscovered arm, so $s(i) = (0, undisc)$, $s(i) = u$, and $x_{Visit_i(u)} > 0$, then $\pi_i(s)$
  - plays $test_i$ with probability $t_i(u) = \frac{x_{Test_i(u)}}{x_{Visit_i(u)}}$;
  - plays $exploit_i$ with probability $e_i(u) = \frac{x_{Exploit_i(u)}}{x_{Visit_i(u)}}$;
  - plays a_0 with probability $z_i(u) = \frac{x_{Explore(u,i)}}{x_{Visit_i(u)}}$;
  - and plays *abandon* with probability $b_i(u) = 1 - \frac{x_{Test_i(u)} + x_{Exploit_i(u)} + x_{Explore(u,i)}}{x_{Visit_i(u)}}$.
- If $s(i)$ is undefined or $s(i) = u$ and $x_{Visit_i(u)} = 0$, then $\pi_i(s)$ plays *abandon* with probability $b_i(u) = 1$.

□

Let r_i be the expected reward of π_i starting from s_0^i, where $s_0^i = s_0$ if i is initially known, and s_0^i is a state where i is the next arm to be discovered and $s(i) = (0, undisc)$ if i is initially undiscovered. (There are many states s where arm i is the next arm to be discovered and $s(i) = (0, undisc)$. Since π_i acts the same starting from all of these states, r_i is independent of which one is chosen.) Similarly, let e_i be the expected number of times that $exploit_i$ is played by π_i starting from s_0^i, and let c_i be the expected number of times that $test_i$ or a_0 is played by π_i starting from s_0^i. Note for future reference that

$$\begin{aligned} r_i &= \textstyle\sum_{u \in U_i^h} x_{Exploit_i(u,i)} Rw(u) \\ e_i &= \textstyle\sum_{u \in U_i^h} x_{Exploit_i(u,i)}, \text{ and} \\ c_i &= \textstyle\sum_{u \in U_i^h} (x_{Test_i(u,i)} + x_{Explore(u,i)}). \end{aligned} \tag{4.6}$$

Define the *ratio index* of arm i to be $\frac{r_i}{e_i + \frac{c_i}{h}}$.[3] (If $x_{Exploit_i(u,i)} = x_{Test_i(u,i)} = x_{Explore(u,i)} = 0$ for all arm-states u of arm i, then we take the ratio index of i to be 0.) Let $i_1, \ldots, i_{n+h}$ be an ordering of the arms according to decreasing ratio index. Since we are using a solution to (4.5) that satisfies the conditions of Lemma 4.4.3, we can assume without loss of generality that the ordering starts with a (possibly empty) sequence of initially known arms, followed by the h initially undiscovered arms, followed by the remaining initially known arms.

We now define policies π^{i_j}, where $1 \leq j \leq n + h + 1$, for the BLPU B Although there is no arm i_{n+h+1}, for ease of exposition, we define $\pi^{i_{n+h+1}}$ to be the policy that, on all inputs s, plays $exploit_{i^*}$, where i^* is the index of an arm that has the best expected payoff in s. We define the remaining policies by backward induction, starting with $j' = n + h$ and working backwards. Very roughly speaking, $\pi^{i_j}(s)$ plays $\pi_{i_j}(s(i_j))$, except that instead of playing *abandon*, it plays $\pi^{i_{j+1}}$. We give more intuition for these policies below.

[3]The ratio index was originally defined by Goel et al. [15]. We had originally hoped to use the Goel et al. approach in our proof, but we discovered a serious flaw in their argument, which does not at this point seem fixable [A. Goel, private communication, 2016].

The policy π^{i_j} proceeds as follows for $j \leq n + h$. If $s = s_{fin}$, then $\pi^{i_j}(s)$ does nothing (no actions are possible). If $s \neq s_{fin}$,

1. if either $h^s = 0$, $s[i_j]$ is undefined (i.e., i_j is undiscovered, but is not the current undiscovered arm), or $s[i_j] \notin U^h_{i_j}$, then play $\pi^{i_{n+h+1}}(s)$. (We remark that the last two cases do not arise in our analysis; we include them here only for completeness.)
2. if $h^s > 0$, $s[i_j] \in U^h_{i_j}$, and $s[i_j]$ has the form (α, β), then play $test_{i_j}$ with probability $t_{i_j}(s[i_j])$, play $exploit_{i_j}$ with probability $e_{i_j}(s[i_j])$, and play $\pi^{i_{j+1}}(s)$ with probability $b_{i_j}(s[i_j])$.
3. if $h^s > 0$, $s[i_j] \in U^h_{i_j}$, and $s[i_j]$ has the form $(t, undisc)$, then play a_0.

Finally, we define policy π^0_B as follows. Let $i_{last(s)}$ be the last arm played in $s \neq s_{fin}$; formally, $last(s) = 1$ if $s = s_0$ and $last(s) = \max\{l \leq n + h + 1 : s(i_l) \neq s_0(i_l)\}$ otherwise. If $s = s_{fin}$, then $\pi^0_B(s)$ does nothing; if $s \neq s_{fin}$, and $i = last(s)$, then $\pi^0_B(s)$ plays $\pi^i(s)$.

Roughly speaking, π^0_B starts in state s_0 by playing π_{i_1}, except that if π_{i_1} would have played *abandon*, it switches to π_{i_2}; if π_{i_2} would have played *abandon*, it switches to π_{i_3}; and so on. The previous sentence holds until an undiscovered arm is reached. Recall that in the ordering of $i_1, \ldots, i_{n+h}$, the h undiscovered arms are grouped together. So once an undiscovered arm is reached, the next h arms are all undiscovered. If $last(s)$ is an undiscovered arm i_l, then $\pi(s)$ plays a_0 until i_l is discovered (or the budget runs out). After that, it plays π_{i_l}, except that if π_{i_l} would have played *abandon*, it moves to the next arm i_{l+1}, which is also undiscovered, and keeps playing a_0 until i_{l+1} is discovered, and so on. Note that once an undiscovered arm is reached, π^0_B continues to play initially undiscovered arms until the budget runs out, since there are h undiscovered arms in the sequence, and each one must be played at least once. It is easy to check that in every run (execution) of π^0_B, once we start playing arm i_l, we never play arm

$i_{l'}$ for $l' < l$ again. The policy π_B^0 agrees with Guha and Munagala's [18] GREEDY-ORDER policy if there are no undiscovered arms (specifically, if we remove the third clause in the definition of π^{i_j}, then π_B^0 is equivalent to GREEDY-ORDER, although our presentation of it is quite different from that of Guha and Munagala.) Unfortunately, dealing with undiscovered arms adds considerable complexity to the analysis.

We want to show that $R_B^{\pi_B^0} \geq \frac{D(1,h)}{4} R_B^*$. For the remainder of the argument, we need to do rather careful bookkeeping. It is useful for the bookkeeping to partition runs r of π_B^0 into phases. Suppose that i_{l^*} is the first undiscovered arm; that is, i_l is an initially known arm for $l < l^*$, but i_{l^*} is not initially known. For $l < l^*$, phase l occurs while actions of the form $test_{i_l}$ or $exploit_{i_l}$ are played (i.e., while we are playing arm i_l). Phase l ends after $exploit_{i_l}$ is played, if the budget runs out, or just after the last time $test_{i_l}$ is played. Note that here the *exploit* action played by π^{n+h+1} is not counted as part of any phase. Phase l for $l < l^*$ may have length 0 in run r (if arm i_l is not played at all) or may not be reached (if run r terminates before reaching phase l). Phase l begins in the state s immediately after phase $l - 1$ ends provided that $h^s > 0$ and no *exploit* action has been played. For $l \geq l^*$, suppose that the first state in phase l is s and $i_{l'}$ is the current undiscovered arm in s. It easily follows from the construction that phase l always begins with an action a_0. It ends just before a_0 is played a second time (so phase $l + 1$ will start with a_0), if the budget runs out, if $exploit_{i_{l'}}$ is played, or if $\pi_{i_{l'}}$ plays *abandon* after arm $i_{l'}$ is discovered. Thus, if $l \geq l^*$ and phase l is reached in run r, then phase l lasts for exactly one round if $i_{l'}$ is not discovered after a_0 is played, or it lasts until the last time that arm $i_{l'}$ is played. For example, suppose that r is a run where, from the point where all the initially known arms have been played and $\pi^{i_{l^*}}$ starts to play, a_0 is played 3 times, then arm i_{l^*} is discovered, $test_{i^*}$ is played 5 times, then $\pi^{i_{l^*+1}}$ starts to play (because $\pi_{i_{l^*}}$ would have played *abandon*) and a_0 is played twice more before the budget runs out. In r, phases l^* and $l^* + 1$ last one round each, phase $l^* + 2$ lasts for 6 rounds (the round where the third a_0 is played and the five subsequent

rounds where $test_{i_{l^*}}$ is played), and phases $l^* + 3$ and $l^* + 4$ last for one round each.

To analyze π_B^0, it is helpful to consider a slight variant of the policy π_i, denoted π_i' for $i = i_1, \ldots, i_{n+h}$. The policy $\pi_i(s)$ is identical to $\pi_i(s)$ if i is an arm that is initially known or if i is initially undiscovered and $s[i]$ is undefined or $s[i] \neq (0, undisc)$; $\pi'(s) = a_0$ if $s[i] = (0, undisc)$. That is, π_i' starting from s_0^i is the same as π_i except that when i is initially undiscovered, it plays a_0 with probability 1 at the first step (while π_i plays a_0 with probability $z_i(0, undisc)$ and plays *abandon* with probability $1 - z_i(0, undisc)$). Define r_i', e_i', c_i' for π_i' just like r_i, e_i, c_i for π_i. That is, r_i' is the expected reward of π_i' starting from s_0^i, e_i' is the expected number of times that $exploit_i$ is played by π_i', and c_i' is the expected number times that $test_i$ and a_0 are played by π_i'.

Observe that if i_j is initially known, then $r_{i_j} = r'_{i_j}$, $e_{i_j} = e'_{i_j}$, $c_{i_j} = c'_{i_j}$ (since $\pi_{i_j} = \pi'_{i_j}$); and when i_j is initially undiscovered, then $r_{i_j} = z_{i_j}(0, undisc) r'_{i_j}$, $e_{i_j} = z_{i_j}(0, undisc) e'_{i_j}$, $c_{i_j} = z_{i_j}(0, undisc) c'_{i_j}$. To see this, let $p_{i_j} = z_{i_j}(0, undisc)$. We abuse notation slightly, and for π of the form π_{i_j} or π'_{i_j}, we take $Visit^{\pi}(s)$ (resp., $Exploit_i^{\pi}(s)$, $Test_i^{\pi}(s)$, $Explore^{\pi}(s)$) to be the probability of policy π visiting s (resp., visiting s and playing $exploit_i$, $test_i$ or a_0) starting from $s_0^{i_j}$ (rather than s_0). For all states $s \neq s_0^{i_j}$, it is almost immediate that $Visit^{\pi_{i_j}}(s) = p_{i_j} Visit^{\pi'_{i_j}}(s)$. Moreover, $\pi_{i_j}(s) = \pi'_{i_j}(s)$ for all $s \neq s_0^*$, so $Exploit_i^{\pi_{i_j}}(s) = p_{i_j} Exploit_i^{\pi'_{i_j}}(s)$, $Test_i^{\pi_{i_j}}(s) = p_{i_j} Test_i^{\pi'_{i_j}}(s)$, and $Explore^{\pi_{i_j}}(s) = p_{i_j} Explore^{\pi'_{i_j}}(s)$. Therefore, $r_{i_j} = p_{i_j} r'_{i_j}$, $e_{i_j} = p_{i_j} e'_{i_j}$, and $c_{i_j} = p_{i_j} c'_{i_j}$, as desired. Given this observation, it follows that for all arms i_j, the ratio index of π_{i_j} and that of π'_{i_j} are equal (i.e., $\frac{r'_{i_j}}{e'_{i_j} + c'_{i_j}/h} = \frac{r_{i_j}}{e_{i_j} + c_{i_j}/h}$ for all $i_j \in \{1, \ldots, n+h\}$).

Let E_l be a random variable on runs r of policy π_B^0 started in reduced state s_0 (from here on in, we assume without saying it explicitly that π_B^0 starts in reduced state s_0) that counts the number of times that $exploit_j$ for some j is played during phase l in r. We show below that $\mathrm{E}[E_l] \leq e'_{i_l}$. Let $W_l = \sum_{j=1}^{l} E_j$, so that $W_l(r)$ is the number of times that an *exploit* action is played in run r in some phase $j \leq l$. Since a run terminates if an *exploit* action is played,

an *exploit* action can be played at most once in a run, so $W_l(r)$ must be either 0 or 1. Let $w_l = \mathrm{E}[W_l]$. Since $\mathrm{E}[E_j] \leq e'_{i_j}$, we must have $w_l \leq \sum_{j=1}^{l} e'_{i_j}$.

To show that $\mathrm{E}[E_l] \leq e'_{i_l}$, first suppose that i_l is initially known. In that case, we partition the runs into three sets: R_1, the runs where phase l is not reached; R_2, the runs where phase l is reached and r terminates during phase l due to the budget running out; and R_3, the runs where phase l is reached and r does not terminate until at or after the end of phase l. Clearly, if phase l is not reached in r, then $E_l(r) = 0$, so $\mathrm{E}[E_l \mid R_1] = 0 \leq e'_{i_l}$. If r reaches phase l and i_l is an arm that is initially known, then we run π_{i_l} until r terminates or we start to play $\pi^{i_{l+1}}$ (which happens if π_{i_l} would play *abandon*). So clearly $\mathrm{E}[E_l \mid R_2] \leq e_{i_l}$ and $\mathrm{E}[E_l \mid R_3] \leq e_{i_l}$. Since $\pi_{i_l} = \pi'_{i_l}$ when i_l is initially known, $e_{i_l} = e'_{i_l}$. It follows that $\mathrm{E}[E_l] \leq e'_{i_l}$.

Now suppose that i_l is not initially known. We partition the runs into two sets: R_1, the runs where phase l is not reached; R_2, the runs where phase l is reached. Clearly, $\mathrm{E}[E_l \mid R_1] = 0 \leq e'_{i_l}$. We further partition R_2 into sets $R_{l',t}$, where $i_{l'}$ is the currently undiscovered arm at the beginning of phase l (note that $l' \leq l$) and t is the number of times that a_0 has already been played at the beginning of phase l since the last arm was discovered. Conditional on $R_{l',t}$, the expected value of E_l is the probability of discovering a new arm multiplied by the expected number of times that *exploit* is played given that a new arm is discovered. Since a_0 is only played once in phase l, the probability of discovering a new arm is just the probability of reaching a reduced state of the form $(u_1, \ldots, u_{l'-1}, v_0)$ from a reduced state of the form $(u_1, \ldots, u_{l'-1}, (t, \mathit{undisc}))$. This is just δ_t, independent of l'. Let $e^*_{i_{l'}}$ be the expected number of times that *exploit* is played by $\pi'_{i_{l'}}$ once arm $i_{l'}$ is discovered. Note that our assumptions guarantee that $e^*_{i_{l'}} = e^*_{i_l}$. Since π^0_B may run out of budget during phase l if arm $i_{l'}$ is discovered, we have that $\mathrm{E}[E_l \mid R_{l',t}] \leq \delta_t e^*_{i_l}$.

Let $e^j_{i_l}$ be the expected number of times that $\mathit{exploit}_{i_l}$ is played by π'_{i_l} if the true MDP is M^j, in which case there are $j - (l - l^*)$ undiscovered arms at $s_0^{i_l}$, and i_l is discovered with

probability $D(j-(l-l^*),1)$ at the first step. We have that $e^j_{i_l} \geq D(j-(l-l^*),1)e^*_{i_l}$, and $e'_{i_l} = \sum_{j=0}^{\infty} e^j_{i_l}\gamma^j(1-\gamma) \geq \sum_{j=0}^{\infty} D(j,1)e^*_{i_l}\gamma^j(1-\gamma) = \delta_0 e^*_{i_l}$.[4] Since $\delta_0 \geq \delta_t$ by Lemma 4.4.2, $\mathrm{E}[E_l \mid R_{l',t}] \leq e'_{i_l}$ for all l', t, so $\mathrm{E}[E_l \mid R_2] \leq e'_{i_l}$, as desired.

Let C_l be a random variable on runs r of π^0_B that counts the total number of times $test_j$ is played for some arm j or a_0 is played during phase l. Using arguments similar to those in the previous paragraph, we can show that $\mathrm{E}[C_l] \leq c'_{i_l}$. Let $M_l = \sum_{j=1}^{l} \frac{C_{i_j}}{h}$, and let $m_l = \mathrm{E}[M_l]$. Thus, $m_l = \mathrm{E}[M_l] = \sum_{j=1}^{l} \frac{\mathrm{E}[C_j]}{h} \leq \sum_{j=1}^{l} \frac{c'_{i_j}}{h}$. Define $m_0 = w_0 = 0$. Let k' be the least number such that $\sum_{l=1}^{k'}(e'_{i_l} + c'_{i_l}/h) \geq 1$. By definition, we have $\sum_{l=1}^{k'-1}(e'_{i_l} + c'_{i_l}/h) < 1$. Thus, $m_l + w_l < 1$ for all $l < k'$.

Let Y_l be the random variable on runs r of π^0_B that is 1 if phase l is reached in r. Clearly, $\Pr(Y_1 = 1) = 1$. It is easy to see that $Y_j = 1$ for $j > 1$ iff $W_{j-1} = 0$ and $M_{j-1} < 1$: phase l is reached in a run r exactly if during r does not use up the budget or play *exploit* before reaching phase l. Thus, for $i > 1$, $Y_i = 1$ iff $W_{i-1} + M_{i-1} < 1$. Since $\mathrm{E}[W_{i-1} + M_{i-1}] \leq w_{i-1} + m_{i-1}$, by Markov's inequality, we have $\Pr[Y_i = 1] = \Pr[W_{i-1} + M_{i-1} < 1] \geq 1 - w_{i-1} - m_{i-1}$. Since, as we observed above, $w_{i-1} + m_{i-i} < 1$ if $i \leq k'$, it follows that π^* reaches $j = l$ with nonzero probability if $l \leq k'$.

We next need to compute the expected reward obtained by π^0_B in phase l conditional on reaching phase l, except that if π^0_B runs out of budget during phase l, we include the reward obtained by π_{n+h+1} in this expected reward. If i_l is among the arms in A_0 (so that arm i_l is initially known), then we claim that this expected reward is at least r'_{i_l}. We partition the runs where phase l is reached into two sets: R_1, the runs where phase l terminates due to the budget

[4]It is here that it is critical that we are using π'_i rather than π_i. It is *not* the case that i_l is discovered with probability $D(j-(l-l^*),1)$ at the first step of π_{i_l}; rather, it is discovered with probability $z_{i_l}(0,\mathit{undisc})D(j-(l-l^*),1)$, since π_{i_l} plays a_0 with probability $z_{i_l}(0,\mathit{undisc})$ and plays *abandon* with probability $1 - z_{i_l}(0,\mathit{undisc})$ at the first step (while π'_{i_l} plays a_0 with probability 1 at the first step).

running out; and R_2, the runs where phase l is reached and does not terminate due to the budget running out. In runs in R_1, $\pi^{i_{n+h+1}}$ is called and an arm i^* with the highest currently known reward is exploited. It is well known that if π_{i_l} is run starting in state s except that $exploit_{i_l}$ is played instead of *abandon*, then the expected reward of arm i in state s is just the reward it would get by playing $exploit_{i_l}$ in state s. More generally, if a policy π' that plays only $test_{i_l}$ or $exploit_{i_l}$ is played starting in state s, then its expected reward is just the reward of playing $exploit_{i_l}$ in state s. (This is what Goel et al. [15] call the *martingale property* of the (α, β) prior.) This means that the reward obtained by π^{n+h+1} is at least as high. Thus, conditional on R_1, the expected reward of π^0_B in phase l is at least r_{i_l}. Since $\pi_{i_l} = \pi'_{i_l}$ when i_l is initially known, $r_{i_l} = r'_{i_l}$ and the expected reward is at least r'_{i_l} in R_1. In runs in R_2, $\pi^{i_0}(s_0)$ proceeds in phase l just as π_{i_l} does (except that it starts phase $l+1$ when π_{i_l} plays *abandon*). Since the behavior of π_{i_l} is independent of the budget, the expected reward conditional on R_2 is $r_{i_l} = r'_{i_l}$. The result follows.

We claim that if i_l is an initially undiscovered arm, then the expected reward obtained by π^0_B in phase l conditional on reaching phase l is at least $\delta_{h-1} r'_{i_l}/\gamma$. To see this, let $r^j_{i_l}$ be the expected reward of π'_{i_l} if the true MDP is M^j. Since π'_{i_l} starts in a state where arm i_l is in state $(0, undisc)$ (i.e., i_l is the current undiscovered arm), so that $l - l^*$ arms have been discovered. Thus, $j \geq l - l^*$, and we have $r'_{i_l} = \sum_{j=l-l^*}^{\infty} r^j_{i_l} \gamma^{j-(l-l^*)}(1-\gamma)$. Clearly, $r^{l-l^*}_{i_l} = 0$: if there are no further arms to be discovered, then arm i_l will not be discovered, so it will not be exploited. Moreover, if $r^*_{i_l}$ is the expected reward that π'_{i_l} obtains once arm i_l is discovered (which is the same for all MDPs M^j), we clearly have $r^{j'+(l-l^*)}_{i_l} \leq r^*_{i_l}$ that π'_{i_l} obtains once arm i_l is discovered. for $j' \geq 1$. Thus, $r'_{i_l} \leq \gamma r^*_{i_l}$.

Let r_l be the expected reward of π^0_B in phase l, conditional on reaching phase l. To get a lower bound on r_l, we partition the runs of π^0_B where phase l is reached into sets $R_{l',t}$, where $i_{l'}$ is the currently undiscovered arm at the beginning of phase l and t is the number of times

that a_0 has been played since the last arm was discovered. In runs of $R_{l',t}$, $l' - l^*$ arms have been discovered so far. Let $r_{l,l',t}$ denote the expected reward of π_B^0 in phase l conditional on $R_{l',t}$. We have that $r_{l,l',t} \geq \delta_t r^*_{i_{l'}}$, since with probability δ_t, a new arm will be discovered, and once it is discovered, $\pi^{i_1}(s_0)$ plays just like $\pi_{i_{l'}}$ once arm $i_{l'}$ has been discovered, except that if the budget runs out, it exploits the best arm available. Since δ_t decreases in t (by Lemma 4.4.2) and $t \leq h - 1$; moreover, $r^*_{i_{l'}} = r^*_{i_l}$, we have that $r_{l,l',t} \geq \delta_{h-1} r^*_{i_l}$ for all l' and t, so $r_l \geq \delta_{h-1} r^*_{i_l}$. Thus, $r_l \geq \delta_{h-1} r'_{i_l}/\gamma$, as desired.

Let $a_{i_j} = 1$ if i_j is an arm in A_0 and let $a_{i_j} = \frac{\delta_{h-1}}{\gamma}$ if i_j is an initially undiscovered arm. Recall that k' be the least number such that $\sum_{j=1}^{k'}(e'_{i_j} + c'_{i_j}/h) \geq 1$. The discussion above shows that

$$\begin{aligned} & R_B^{\pi_B^0} \\ \geq\ & \textstyle\sum_{j=1}^{k'} a_{i_j}(1 - w_{j-1} - m_{j-1}) r'_{i_j} \\ \geq\ & \tfrac{\delta_{h-1}}{\gamma} \textstyle\sum_{j=1}^{k'}(1 - w_{j-1} - m_{j-1}) r'_{i_j}. \end{aligned} \tag{4.7}$$

Let $\rho' = \frac{1 - \sum_{j=1}^{k'-1}(e'_{i_j} + c'_{i_j}/h)}{e'_{i_{k'}} + c'_{i_{k'}}/h}$. Thus, $\sum_{j=1}^{k'-1}(e'_{i_j} + c'_{i_j}/h) + \rho'(e'_{i_{k'}} + c'_{i_{k'}}/h) = 1$. We claim that

$$\sum_{j=1}^{k'}(1 - w_{j-1} - m_{j-1}) r'_{i_j} \geq (\sum_{j=1}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}})/2 \tag{4.8}$$

and

$$\sum_{j=1}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}} \geq R_B^*/2. \tag{4.9}$$

It follows from (4.7), (4.8), and (4.9) that $R_B^{\pi_B^0} \geq \frac{\delta_{h-1}}{\gamma} R_B^*/4$, as desired.

We now prove (4.8) and (4.9). We remark that Guha and Munagala [18] prove analogues of (4.8) and (4.9) (without the ρ' term). Our proof is similar in spirit to theirs, but somewhat more complicated because of the need to deal with undiscovered arms.

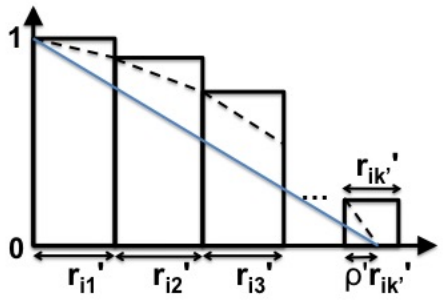

Figure 4.1: $\sum_{j=1}^{k'}(1 - m_{j-1} - w_{j-1})r'_{i_j}$.

For (4.8), note that the sum on the left-hand side is just the sum of the areas of k' rectangles, where the width of the jth rectangle is r'_{i_j} and its height is $1 - m_{j-1} - w_{j-1}$. Consider the dashed lines shown in Figure 4.1, where the jth line goes from the top left of the jth rectangle to the top left of the $(j + 1)$st rectangle, except that the last line goes from the top left of the k'th rectangle to the point $(\sum_{l=1}^{k'-1} r'_l + \rho' r_{k'}, 0)$ (as shown in the figure). The slope of the jth line is

$$\begin{aligned} & \frac{(1-w_j-m_j)-(1-w_{j-1}-m_{j-1})}{r'_{i_j}} \\ = & -\frac{(w_j-w_{j-1})+(m_j-m_{j-1})}{r'_{i_j}} \\ = & -\frac{e'_{i_j}+c'_{i_j}/h}{r'_{i_j}}, \end{aligned}$$

for all $j \in \{1, \ldots, k'\}$. We showed above that $\frac{r'_{i_j}}{e'_{i_j}+c'_{i_j}/h} = \frac{r_{i_j}}{e_{i_j}+c_{i_j}/h}$. Now, by assumption, as j increases, $\frac{r_{i_j}}{e_{i_j}+c_{i_j}/h}$ decreases. Hence, its reciprocal increases, and the negative of its reciprocal decreases; that is, the slopes of these lines becomes more and more negative, and shown in Figure 4.1. It easily follows that the piecewise linear curve defined by these lines lies above the solid line shown in Figure 4.1. The triangle defined by this solid line, the x-axis, and the y-axis has area $(\sum_{j=1}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}})/2$. The sum of the areas of the rectangles is clearly greater than the area of the triangle. This gives us (4.8).

For (4.9), let k be the smallest value such that $\sum_{l=1}^{k}(e_{i_l}+c_{i_l}/h) \geq 1$, and let $\rho = \frac{1-\sum_{j=1}^{k-1}(e_{i_j}+c_{i_j}/h)}{e_{i_k}+c_{i_k}/h}$. Thus, $\sum_{j=1}^{k-1}(e_{i_j} + c_{i_j}/h) + \rho(e_{i_k} + c_{i_k}/h) = 1$. We first show that $\sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k} \geq R^*_B/2$, then show

that $\sum_{j=1}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}} \geq \sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k}$. The desired result follows.

To see why $\sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k} \geq R^*_B/2$, since c_i is the expected number of times that π_i plays $test_i$ or a_0, we have $c_i = \sum_{u \in U_i^h} (x_{Test_i(u)} + x_{Explore(u)})$, so $\sum_{j=1}^{n+h} c_{i_j} \leq h$. Similarly, e_i is the expected number of times that π_i plays $exploit_i$, so we have $e_i = \sum_{u \in U_i^h} x_{Exploit_i(u)}$ and $\sum_{j=1}^{n+h} e_{i_j} \leq 1$. Since $\frac{\sum_{j=1}^{n+h} c_{i_j}}{h} \leq 1$, we have $\sum_{j=1}^{n+h} (e_{i_j} + c_{i_j}/h) \leq 2$. Moreover, since $\sum_{j=1}^{k-1} (e_{i_j} + c_{i_j}/h) + \rho(e_{i_k} + c_{i_k}/h) = 1$, it follows that $(1-\rho)(e_{i_k} + c_{i_k}/h) + \sum_{j=k+1}^{n+h} (c_{i_j} + e_{i_j}/h) \leq 1$, and $\sum_{j=1}^{k-1} (e_{i_j} + c_{i_j}/h) + \rho(e_{i_k} + c_{i_k}/h) \geq (1-\rho)(e_{i_k} + c_{i_k}/h) + \sum_{j=k+1}^{n+h} (c_{i_j} + e_{i_j}/h)$. Thus,

$$
\begin{aligned}
&\sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k} \\
&= \sum_{j=1}^{k-1} \frac{r_{i_j}}{e_{i_j} + c_{i_j}/h} (e_{i_j} + c_{i_j}/h) + \rho r_{i_k} \\
&\geq \sum_{j=1}^{k-1} \frac{r_{i_k}}{e_{i_k} + c_{i_k}/h} (e_{i_j} + c_{i_j}/h) + \rho r_{i_k} \quad [\text{since } \tfrac{r_{i_j}}{e_{i_j} + c_{i_j}/h} \text{ decreases with } j] \\
&= \frac{r_{i_k}}{e_{i_k} + c_{i_k}/h} \left(\sum_{j=1}^{k-1} (e_{i_j} + c_{i_j}/h) + \rho(e_{i_k} + c_{i_k}/h) \right) \\
&\geq \frac{r_{i_k}}{e_{i_k} + c_{i_k}/h} \left((1-\rho)(e_{i_k} + c_{i_k}/h) + \sum_{j=k+1}^{n+h} (e_{i_j} + c_{i_j}/h) \right) \\
&\geq (1-\rho) r_{i_k} + \sum_{j=k+1}^{n+h} \frac{r_{i_j}}{e_{i_j} + c_{i_j}/h} (e_{i_j} + c_{i_j}/h) \quad [\text{since } \tfrac{r_{i_j}}{e_{i_j} + c_{i_j}/h} \text{ decreases with } j] \\
&= (1-\rho) r_{i_k} + \sum_{j=k+1}^{n+h} r_{i_j}.
\end{aligned}
$$

Since $\sum_{j=1}^{n+h} r_{i_j} \geq R^*_B$ and $\sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k} \geq (1-\rho) r_{i_k} + \sum_{j=k+1}^{n+h} r_{i_j}$, we must have $\sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k} \geq R^*_B/2$, as desired.

It now remains to show that $\sum_{j=1}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}} \geq \sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k}$. Recall that i_{l^*} is the first undiscovered arm in the ordering. If $k \leq l^* - 1$, then the first k arms are all initially known, so $e_{i_j} = e'_{i_j}$ and $c_{i_j} = c'_{i_j}$ for all $j \leq k$. Thus, we must have $k' = k$, $\rho' = \rho$, and $\sum_{j=1}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}} = \sum_{j=1}^{k-1} r_{i_j} + \rho r_{i_k}$.

If $k > l^* - 1$, since $\sum_{j=1}^{l^*-1} r'_{i_j} = \sum_{j=1}^{l^*-1} r_{i_j}$, it suffices to show that

$$
\sum_{j=l^*}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}} \geq \sum_{j=l^*}^{k-1} r_{i_j} + \rho r_{i_k}. \tag{4.10}
$$

It is easy to see that $k' \leq l^* - 1 + h$: Since all policies π'_{i_j} where i_j is initially undiscovered play a_0 with probability 1 at the first step, $\sum_{j=l^*}^{l^*+h-1} c'_{i_j} \geq h$. Thus, $\sum_{j=1}^{l^*+h-1} e'_{i_j} + c'_{i_j}/h \geq 1$, so $k' \leq l^* + h - 1$. It follows that the arms $i_{l^*}, \ldots, i_{k'}$ are all initially undiscovered, so the policies $\pi'_{i_{l^*}}, \ldots, \pi'_{i_{k'}}$ all have the same ratio index, say τ^*. Thus, for $j \in \{l^*, \ldots, k'\}$, we have that $\frac{r_{i_j}}{e_{i_j}+c_{i_j}/h} = \tau^*$, so $r_{i_j} = \tau^*(e_{i_j} + c_{i_j}/h)$; similarly, $r'_{i_j} = \tau^*(e'_{i_j} + c'_{i_j}/h)$. And for $j \in \{k'+1, \ldots, k\}$,[5] $\frac{r_{i_j}}{e_{i_j}+c_{i_j}/h} \leq \tau^*$, since arms are ordered in decreasing order of ratio index. (Note that we may have $k > l^* - 1 + h$, so not all these arms necessarily have ratio index τ^*.) So $r_{i_j} \leq \tau^*(e_{i_j} + c_{i_j}/h)$ for all $j \in \{l^*, \ldots, k\}$. Therefore,

$$\begin{aligned}
&\textstyle\sum_{j=l^*}^{k-1} r_{i_j} + \rho r_{i_k} \\
&\leq (\textstyle\sum_{j=l^*}^{k-1}(e_{i_j} + c_{i_j}/h) + \rho(e_{i_k} + c_{i_k}/h))\tau^* \\
&= (\textstyle\sum_{j=l^*}^{k'-1}(e'_{i_j} + c'_{i_j}/h) + \rho'(e'_{i_{k'}} + c'_{i_{k'}}/h))\tau^* \quad \text{[see below]} \\
&= \textstyle\sum_{j=l^*-1}^{k'-1} r'_{i_j} + \rho' r'_{i_{k'}},
\end{aligned}$$

as desired. The third line holds since $\sum_{j=1}^{k'-1}(e'_{i_j} + c'_{i_j}/h) + \rho'(e'_{i_{k'}} + c'_{i_{k'}}/h) = \sum_{j=1}^{k-1}(e_{i_j} + c_{i_j}/h) + \rho(e_{i_k} + c_{i_k}/h) = 1$ and $\sum_{j=1}^{l^*-1}(e'_{i_j} + c'_{i_j}/h) = \sum_{j=1}^{l^*-1}(e_{i_j} + c_{i_j}/h)$. Subtracting the two equations, we get the third line. This completes the argument.

Dealing with the infinite sum: The policy π_B^0 is a $(\frac{\delta_{h-1}}{4\gamma})$-approximation to the optimal policy for B. It is easy to see that π_B^0 can be computed in polynomial time modulo the computation of δ_t, which is an infinite sum. Fortunately, as we show below, ϵ-close approximations to δ_t can be computed in time $O(t \log \frac{1}{\epsilon})$. We make use of this fact and show that an approximately optimal policy for BLPUs can be computed in polynomial time. The idea is to replace δ_t in the linear program by its approximation and construct π_B^0 just as before. We show that the resulting policy is still approximately optimal.

Recall that $\delta_t = \frac{\sum_{i=1}^{\infty} \gamma^i (1-D(i,1))\ldots(1-D(i,t))D(i,t+1)}{1+\sum_{i=1}^{\infty} \gamma^i (1-D(i,1))\ldots(1-D(i,t))}$. Let $\delta_{t,j} = \frac{\sum_{i=1}^{j} \gamma^i (1-D(i,1))\ldots(1-D(i,t))D(i,t+1)}{1+\sum_{i=1}^{j} \gamma^i (1-D(i,1))\ldots(1-D(i,t))}$. We show

[5] We take $\{k'+1, \ldots, k\}$ to be the empty set if $k'+1 > k$.

below that $|\delta_t - \delta_{t,j}| \leq \frac{\gamma^{j+1}}{1-\gamma}$. Thus, as j increases, the error decreases exponentially in γ. It is clear that $\delta_{t,j}$ can be computed in time $O(tj)$.

To see that $|\delta_t - \delta_{t,j}| \leq \frac{\gamma^{j+1}}{1-\gamma}$, let $f(i,t) = \gamma^i(1-D(i,1))\ldots(1-D(i,t))$, and $g(i,t) = f(i,t)D(i,t+1)$. Thus,

$$\begin{aligned}
&\delta_t - \delta_{t,j} \\
&= \frac{\sum_{i=1}^{\infty} g(i,t)}{1+\sum_{i=1}^{\infty} f(i,t)} - \frac{\sum_{i=1}^{j} g(i,t)}{1+\sum_{i=1}^{j} f(i,t)} \\
&\leq \frac{\sum_{i=1}^{\infty} g(i,t)}{1+\sum_{i=1}^{\infty} f(i,t)} - \frac{\sum_{i=1}^{j} g(i,t)}{1+\sum_{i=1}^{\infty} f(i,t)} \\
&= \frac{\sum_{i=j+1}^{\infty} g(i,t)}{1+\sum_{i=1}^{\infty} f(i,t)} \\
&\leq \frac{\sum_{i=j+1}^{\infty} \gamma^i}{1} \quad [\text{since } g(i,t) \leq \gamma^i] \\
&= \frac{\gamma^{j+1}}{1-\gamma}.
\end{aligned}$$

Similarly,

$$\begin{aligned}
&\delta_t - \delta_{t,j} \\
&= \frac{\sum_{i=1}^{\infty} g(i,t)}{1+\sum_{i=1}^{\infty} f(i,t)} - \frac{\sum_{i=1}^{j} g(i,t)}{1+\sum_{i=1}^{j} f(i,t)} \\
&\geq \frac{\sum_{i=1}^{\infty} g(i,t)}{1+\sum_{i=1}^{\infty} f(i,t)} - \frac{\sum_{i=1}^{\infty} g(i,t)}{1+\sum_{i=1}^{j} f(i,t)} \\
&= \frac{-\sum_{i=1}^{\infty} g(i,t) \sum_{i=j+1}^{\infty} f(i,t)}{(1+\sum_{i=1}^{\infty} f(i,t))(1+\sum_{i=1}^{j} f(i,t))} \\
&\geq \frac{-\sum_{i=j+1}^{\infty} f(i,t)}{1+\sum_{i=1}^{j} f(i,t)} \quad [\frac{\sum_{i=1}^{\infty} g(i,t)}{1+\sum_{i=1}^{\infty} f(i,t)} \leq 1 \text{ since } g(i,t) \leq f(i,t)] \\
&\geq \frac{-\sum_{i=j+1}^{\infty} \gamma^i}{1} \quad [\text{since } f(i,t) \leq \gamma^i] \\
&= -\frac{\gamma^{j+1}}{1-\gamma}.
\end{aligned}$$

Thus, $|\delta_t - \delta_{t,j}| \leq \frac{\gamma^{j+1}}{1-\gamma}$.

Given ϵ, let $\theta(\epsilon) = \frac{4\epsilon\gamma^2 D(1,1)Rw(v_0)}{h^2(h+2)^{2(n+h)}}$. Let $j = \lceil \log_\gamma(\theta(\epsilon)(1-\gamma)) \rceil - 1$. By the discussion above, it follows that $|\delta_{t,j} - \delta_t| \leq \theta(\epsilon)$ for all $t \in \{0,\ldots,h-1\}$. Take $\delta_t^\epsilon = \delta_{t,j}$ for this choice of j. As observed above, the values δ_t^ϵ for each $t \in \{0,\ldots,h-1\}$ can be computed in time $O(h \log_\gamma \frac{4\epsilon\gamma^2 D(1,1)Rw(v_0)(1-\gamma)}{h^2(h+2)^{2(n+h)}}) = O(h \log \frac{h^2(h+2)^{2(n+h)}}{4\epsilon\gamma^2 D(1,1)Rw(v_0)(1-\gamma)})$ (since $\gamma < 1$), which is polynomial in $(n+h)$, $\log \frac{1}{\epsilon}$, $\log \frac{1}{D(1,1)}$, $\log \frac{1}{Rw(v_0)}$ and $\log \frac{1}{\gamma(1-\gamma)}$.

Define a linear program just as (4.5), except that we replace δ_t by δ_t^ϵ. Let $\vec{x}^\epsilon$ be an optimal solution of the modified linear system that satisfies Lemma 4.4.3. We can define policies π_i^ϵ and $(\pi_i^\epsilon)'$ using $\vec{x}^\epsilon$, just as π_i and π_i' were defined using the solution $\vec{x}$ to (4.5). We can also define r_i^ϵ, e_i^ϵ, and c_i^ϵ using the analogue of (4.6); that is,

$$\begin{aligned} r_i^\epsilon &= \textstyle\sum_{u \in U_i^h} x^\epsilon_{Exploit_i(u,i)} Rw(u), \\ e_i^\epsilon &= \textstyle\sum_{u \in U_i^h} x^\epsilon_{Exploit_i(u,i)}, \text{ and} \\ c_i^\epsilon &= \textstyle\sum_{u \in U_i^h} (x^\epsilon_{Test_i(u,i)} + x^\epsilon_{Explore(u,i)}). \end{aligned}$$

We order the policies π_i^ϵ in decreasing order of $\frac{r_i^\epsilon}{e_i^\epsilon + c_i^\epsilon/h}$. Using this order, we can then define policies $\pi^{i_j,\epsilon}$ as before, and take π_B^ϵ to be the analogue of π_B^0, defined using $\pi^{i_j,\epsilon}$ instead of π^{i_j}. Since δ_t^ϵ can be computed in time polynomial in $(n+h)$, $\log \frac{1}{\epsilon}$, $\log \frac{1}{D(1,1)}$, $\log \frac{1}{Rw(v_0)}$ and $\log \frac{1}{\gamma(1-\gamma)}$, it is easy to see that π_B^ϵ can be computed in time polynomial in the desired parameters. We now show that $R_B^{\pi_B^\epsilon} \geq (\frac{\delta_{h-1}}{4\gamma} - \epsilon) R_B^*$.

The idea is to show that π_B^ϵ is an almost-optimal policy for an MDP that is "close" to B, and thus is an almost-optimal policy for B. To make this precise, we associate a BLPU B with an MDP M_B whose state space is the set of reduced states of B. In every (reduced) state of M_B, the same actions can be played as in that reduced state of B, and the transition probabilities and rewards are the same. Thus, B and B_m have the same set of policies, and the same maximum expected reward (i.e., $R_B^* = R_{M_B}^*$). Note that if a_0 is played in a reduced state of the form $(u_1, \ldots, u_m, (t, undisc))$, then with probability δ_t, there is a transition to $(u_1, \ldots, u_m, v_0, (0, undisc))$ and with probability $1 - \delta_t$, there is a transition to $(u_1, \ldots, u_m, (t+1, undisc))$.

Let M_B^ϵ be an MDP that is identical to B except that the transition probabilities δ_t and $1 - \delta_t$ when a_0 is played in reduced state $(u_1, \ldots, u_m, (t, undisc))$ are replaced by δ_t^ϵ and $1 - \delta_t^\epsilon$, respectively. Essentially the same argument as that given in the case of B (replacing very term

by the corresponding term with a superscript of ϵ, so that δ_t is replaced by δ_t^ϵ, r_i is replaced by r_i^ϵ, and so on) shows that π_B^ϵ is a $\frac{\delta_{h-1}}{4\gamma}$-optimal policy for M_B^ϵ.

Say that an MDP M_1 is a *c-approximation* of an MDP M_2 if M_1 and M_2 have the same state space, action space, and reward function, and $|P_{M_1}(s, s', a) - P_{M_2}(s, s', a)| \leq c$ for all states s, s' and actions a. Note that M_B^ϵ is a $\theta(\epsilon)$-approximation to M_B. Moreover, if M_1 is a c-approximation, then M_1 and M_2 have the same set of policies.

Lemma 4.4.5: *If MDP M_1 has N states, the maximum reward for a transition in M_1 is at most 1, and M_2 is a c-approximation of M_1, then for all policies π for M_1 (and M_2) that play at most h steps, $|R_{M_2}^\pi - R_{M_1}^\pi| \leq cNh^2$. Moreover, $R_{M_2}^* \geq R_{M_1}^* - cNh^2$.*

Proof: The fact that $|R_{M_2}^\pi - R_{M_1}^\pi| \leq cNh^2$ is a special case of Lemma 4 in [4]. For the second claim, suppose π_{M_1} is an optimal policy for M_1, and π_{M_2} is an optimal policy for M_2. Then

$$R_{M_2}^* = R_{M_2}^{\pi_{M_2}} \geq R_{M_2}^{\pi_{M_1}} \geq R_{M_1}^{\pi_{M_1}} - cNh^2 = R_{M_1}^* - \theta Nh^2.$$

∎

Since M_B^ϵ is a $\theta(\epsilon)$-approximation of M_B, it follows that $R_{M_B^\epsilon}^* \geq R_{M_B}^* - \theta(\epsilon)Nh^2 = R_B^* - \theta(\epsilon)Nh^2$. Since π_B^ϵ is a $\frac{\delta_{h-1}}{4\gamma}$-optimal policy for M_B^ϵ, it follows that

$$R_{M_B^\epsilon}^{\pi_B^\epsilon} \geq \frac{\delta_{h-1}}{4\gamma} R_{M_B^\epsilon}^* \geq \frac{\delta_{h-1}}{4\gamma}(R_B^* - \theta(\epsilon)Nh^2).$$

To complete the argument, we simply observe that

$$\begin{aligned}
& \tfrac{\delta_{h-1}}{4\gamma}(R_B^* - \theta(\epsilon)Nh^2) \\
= \;& \tfrac{\delta_{h-1}}{4\gamma}R_B^* - \tfrac{\theta(\epsilon)}{4\gamma}Nh^2 \quad [\text{since } \delta_{h-1} \leq 1] \\
= \;& \tfrac{\delta_{h-1}}{4\gamma}R_B^* - \epsilon\gamma D(1,1)Rw(v_0) \quad [\text{since } \theta(\epsilon) = \tfrac{4\epsilon\gamma^2 D(1,1)Rw(v_0)}{h^2(h+2)^{2(n+h)}} \text{ and } N \leq (h+2)^{2(n+h)}] \\
\geq \;& \tfrac{\delta_{h-1}}{4\gamma}R_B^* - \epsilon R_B^* \quad [\text{since } \gamma D(1,1)Rw(v_0) \leq R_B^*; \text{ see below}] \\
= \;& (\tfrac{\delta_{h-1}}{4\gamma} - \epsilon)R_B^*,
\end{aligned}$$

as desired. For the second-last line, consider the simple policy that plays a_0 at the first step, exploits the new arm if a new arm is discovered, and otherwise exploits an arm with the highest utility. This policy has an expected reward of at least $\gamma D(1,1)Rw(v_0)$, so $R_B^* \geq \gamma D(1,1)Rw(v_0)$.

CHAPTER 5

CONCLUSION

In this thesis, we defined a model called MDPUs which captures the scenarios in which the DM is unaware of certain states and actions of the problem. We completely characterized the complexity of learning to play near-optimally in MDPUs, and provided an algorithm called URMAX that learns to play near-optimally in polynomial time whenever it is possible to do so.

We consider applying MDPUs to robotic problems by first modeling such problems as continuous MDPs where actions are taken over a continuous period of time, then discretizing them into a sequence of finer and finer MDPUs. We then solve these MDPUs using the URMAX algorithm. We proved that using this approach, a near-optimal policy can be learned for robotic problems in time polynomial in the related parameters.

We applied MDPUs and the URMAX algorithm to a real robotic problem, the bipedal walking problem which aims to enable a humanoid robot to learn walking on its own. Our experiment enabled DARwIn-OP, a humanoid robot, to successfully learn various walking gaits via simulations (see https://youtu.be/qW51iInpdV0 for a video). These results show that our method provides a general approach that allows robots to learn new tasks on their own. We make no assumptions on the structure of the tasks to be learned. The approach can be easily applied to various robotic tasks.

Finally, we considered the problem of learning to play approximately-optimal where the DM is unaware of certain states and actions, and that the DM has a limited budget. We proposed two mathematical models for such problems: MDPUBs and an interesting subclass of MDPUBs called BLPUs. We provided an approximately-optimal policy for BLPUs.

In the future, we plan to apply our approach to more robotic tasks, such as learning to run and to walk up and down stairs. We believe the process will be quite instructive in terms of adding useful learning heuristics to our approach, both specific to these tasks and to more general robotic tasks. We are also interested in having the robot simulate learning to walk in the same way a baby does, for example, by limiting the robot's abilities initially, so that it must crawl before it walks. Part of our interest lies in seeing if such initial limitations actually make learning more efficient.

APPENDIX A

PROOFS FOR THEOREMS IN CHAPTER 2

A.1 Proof of Theorem 2.5.2

THEOREM 2.5.2. *Let* $M' = (S', A', S_0, a_0, g'_A, g_0, D, P', R', G_0)$ *be an MDPU where* $|S'| = N$, $|A'| = k$, *and* $\max(R'(s, s', a) : s, s' \in S', a \in A') = R_{\max}$. *If* $0 < \delta < 1$, $\epsilon > 0$, $K_0 \geq \min\{H : \sum_{t=1}^{H} D(1, t, s) \geq \ln(4Nk/\delta)\}$, *and* $K \geq K_4(T, K_0)$, *then for all MDPs* $M = (S_M, g_M, A_M, P_M, R_M)$ *compatible with* S_0, g_0, G_0, N, k, $R_{\max}$, *and* T *(i.e.,* $S_M \supseteq S_0$, $g_M(s) \supseteq g_0(s)$ *for all* $s \in S_0$, $|S_M| \leq N$, $|A_M| \leq k$, $R_M(s, s', a) \leq R_{\max}$ *for all* $s, s' \in S_M$ *and* $a \in A_M$, *and the* ϵ*-return mixing time* T_M *of* M *is* $\leq T$*), and all states* $s_0 \in S_0$, *with probability at least* $1 - \delta$, $\text{URMAX}_K(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ *running on* M *obtains an expected average reward that is at least* $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$ *in time polynomial in* N, k, T, *and* K.

The basic structure of the proof follows lines similar to the correctness proof of RMAX [4]. (Related results are proved in [1; 29; 30; 31].) We sketch the details here.

Like Brafman and Tennenholtz [4], we first define *α-approximation*. Our definition is exactly the same as theirs, except that we restrict to MDPs (whereas Brafman and Tennenholtz defined it for stochastic games).

Definition A.1.1: Let M and M' be MDPs over the same state and action spaces. M' is an *α-approximation* of M if M and M' are identical except for their transition probability functions, and for all states s, t and every action a we have

$$|P_M(s, t, a) - P_{M'}(s, t, a)| \leq \alpha.$$

□

We now restate the Simulation Lemma in [4], restricted to MDPs.

Lemma A.1.2: *If M and M' are MDPs with N states, and M' is an $\frac{\epsilon}{NTR_{\max}}$-approximation of M, then for every state s and policy π, we have that*

$$|U_{M'}(s, \pi, T) - U_M(s, \pi, T)| \leq \epsilon.$$

Proof: This is a special case of the Simulation Lemma (Lemma 4) in [4]. ∎

For our proofs, it is useful to define a special class of MDPs.

Definition A.1.3: *An a_0-MDP* is an MDP $M = (S, g, A, P, R)$ such that $a_0 \in A$ and for all states $s \in S$, we have that $a_0 \in g(s)$, $P(s, s, a_0) = 1$, and $R(s, s', a_0) = -R_{\max}$ for all $s' \in S$, while $R(s, s', a) \geq 0$ for all states $s, s' \in S$ and $a \in A - \{a_0\}$. □

Let M_1 be an a_0-MDP. An MDP M_2 is *a mirror of M_1 for $B \subseteq S \times A$, $R_{\max}$, and state s_d* if, roughly speaking, M_2 and M_1 are identical for all state-action pairs in B, and, for state-action pairs $(s, a) \notin B$, we have $P_{M_2}(s, s_d, a) = 1$ and $R_{M_2}(s, s', a) = R_{\max}$ for all $s' \in S$. M_2 can be viewed as an optimistic approximation of M_1, because the reward associated with each state-action pair in M_2 is either the same as in M_1 or is the maximum possible reward $R_{\max}$. For example, in URMAX, M^0 is a mirror of the actual MDP for $B = \emptyset$, $R_{\max}$, and the dummy state s_d. In fact, the M' in each iteration of URMAX is a mirror of some $\frac{\epsilon}{4NTR_{\max}}$-approximation of the actual MDP. Our definition of mirror is similar to that of Brafman and Tennenholtz [4], except that we deal only with a_0-MDPs rather than general MDPs, and we allow states in a mirror

that do not exist in the MDP it is mirroring. Since, unlike Brafman and Tennenholtz [4], we allow for uncertainty regarding the number of states, the mirror may have up to the estimated upper bound on the number of states in the underlying MDP. A second difference between our definition and that of Brafman and Tennenholtz is that we allow mirrors to not include some actions in the original MDP, since the DM may be unaware of these actions.

Definition A.1.4: Let $M_1 = (S_1, A_1, g_1, P_1, R_1)$ be an a_0-MDP. MDP $M_2 = (S_2, A_2, g_2, P_2, R_2)$ is a *mirror of* M_1 *for* $B \subseteq S_1 \times A_2$, $R_{\max}$, *and* $s_d \in S_2$ if the following conditions hold:

- $S_2 \supseteq S_1$ and $s_d \in S_2 - S_1$;
- $a_0 \in A_2 \subseteq A_1$;
- $g_2(s) = \{a_0\}$ for all $s \in S_2 - S_1$;
- $g_2(s) = \{a_0\} \cup g'(s)$ for all $s \in S_1$, where $g'(s) \subseteq g_1(s)$;
- if $(s, a_0) \in B$, then $g_2(s) = g_1(s)$;
- for all $(s, a) \in B$, $R_2(s, s', a) = R_1(s, s', a)$ and $P_2(s, s', a) = P_1(s, s', a)$ if $s' \in S_1$, otherwise $P_2(s, s', a) = 0$ and $R_2(s, s', a) = R_{\max}$;
- for all $(s, a) \notin B$, $P_2(s, s_d, a) = 1$ and $R_2(s, s', a) = R_{\max}$ for all $s' \in S$.

M_3 is an ϵ-*mirror of* M_1 *for* $B \in S_1 \times A_3$, $R_{\max}$, *and* $s_d \in S_3$ if there exists an MDP M_2 which is an ϵ-approximation of M_1 such that M_3 is a mirror of M_2 for B, $R_{\max}$, and s_d. □

Using the notation in Definition A.1.4, if M_2 is an ϵ-mirror of M_1 (for B, $R_{\max}$, and s_d), then $S_1 \subseteq S_2$, $A_2 \subseteq A_1$, and M_1 is an a_0-MDP (so that $a_0 \in g(s)$ for all $s \in S$), so a policy π_2 over M_2 can be viewed as a policy over M_1 by simply restricting π_2 to apply only to states in S_1.

Recall that $\Pr_M^{\pi,s}(\rho)$ is the probability that a path ρ is taken when policy π is used in MDP M starting at state s, and $U_M(\rho)$ is the undiscounted average reward of taking path ρ in M. We would like to show that if M_1 is an a_0-MDP with N states, and M_2 is an $(\epsilon/NTR_{\max})$-mirror of M_1 for B, $R_{\max}$, and s_d, then either an optimal T-step policy π_2 for M_2 is near-optimal for M_1, or by executing π_2 in M_1 (recall that we can view any policy for M_2 as a policy for M_1) for T steps, the probability that the path being taken visits a state-action pair not in B is relatively high (i.e., if Φ is the set of all T-paths in M_1 that contain at least one state-action pair not in B, then for all $s \in S_1$, we have that $\Pr_{M_1}^{\pi_2,s}(\Phi)$ is relatively high). This is shown in the following lemma, which is similar to the Explore or Exploit Lemma in [4].

Lemma A.1.5: *If M_1 is an a_0-MDP with N states, M_2 is a $(\beta/NTR_{\max})$-mirror of M_1 for B, $R_{\max}$, and s_d, π_2 is an optimal T-step policy for M_2, π_1 is an optimal T-step policy for M_1, s is a state in M_1, and $0 < \alpha < 1$, then either (a) $U_{M_1}(s, \pi_2, T) \geq U_{M_1}(s, \pi_1, T) - \alpha - 3\beta$; or (b) if Φ is the set of T-paths in M_1 that contain at least one state-action pair not in B, then $\Pr_{M_1}^{\pi_2,s}(\Phi) \geq \frac{\alpha}{2R_{\max}}$.*

Proof: The proof is similar to the corresponding proof in [4].

We first show that for all $s \in S_1$, we have $U_{M_2}(s, \pi_2, T) \geq U_{M_1}(s, \pi_1, T) - \beta$. Since M_2 is a $(\beta/NTR_{\max})$-mirror of M_1 for B, $R_{\max}$, and s_d, there exists an MDP $M_1' = (S_1', A_1', g_1', P_1', R_1')$ such that M_1' is a $(\beta/NTR_{\max})$-approximation of M_1, and M_2 is a mirror of M_1' for B, $R_{\max}$, and s_d. Let π_1' be an optimal T-step policy for M_1'. We claim that $U_{M_2}(s, \pi_2, T) \geq U_{M_1'}(s, \pi_1', T)$. We actually prove a more general statement. Let π_2^t be an optimal t-step policy for M_2, and let π_1^t be an optimal t-step policy for M_1'. We now prove by induction on t that $U_{M_2}(s, \pi_2^t, t) \geq U_{M_1'}(s, \pi_1^t, t)$ for all $t \geq 1$. The claim is the special case where $t = T$.

Base case: $t = 1$. If there exists some action $a \in g_2(s)$ such that $(s,a) \notin B$, then by Definition A.1.4, playing action a at state s in M_2 gives reward $R_{\max}$, and we have $U_{M_2}(s,\pi_2^1,1) = R_{\max} \geq U_{M_1'}(s,\pi_1^1,1)$. Otherwise, for all $a \in g_2(s)$, $(s,a) \in B$. This means $(s,a_0) \in B$, so $g_2(s) = g_1'(s)$ and $R_2(s,s',a) = R_1'(s,s',a)$ for all $s' \in S_1$ and $a \in g_2(s)$ Thus, $U_{M_2}(s,\pi_2^1,1) = U_{M_1'}(s,\pi_1^1,1)$.

Induction step: Suppose that $U_{M_2}(s,\pi_2^t,t) \geq U_{M_1'}(s,\pi_1^t,t)$ for all states $s \in S_1$. We show that $U_{M_2}(s,\pi_2^{t+1},t+1) \geq U_{M_1'}(s,\pi_1^{t+1},t+1)$ for all states $s \in S_1$. So consider $s \in S_1$. Just as in the base case, there are two cases. If there is some action a such that $(s,a) \notin B$, then by Definition A.1.4, playing action a at state s in M_2 gives reward $R_{\max}$ and transits to state s_d with probability 1. Thus, the optimal $(t+1)$-step policy is to play a once, and then to play a_0 t times. This gives an average reward of $R_{\max} \geq U_{M_1'}(s,\pi_1^T,t+1)$. Otherwise, we must have $(s,a) \in B$ for all $a \in g_2(s)$. Thus, $(s,a_0) \in B$. This means that $g_2(s) = g_1'(s)$, thus $(s,a) \in B$ for all $a \in g_1'(s)$. Let a_1 be the first action taken by π_1^{t+1} in state s. Consider the following $(t+1)$-step policy for M_2: first take action a_1, then follow an optimal t-step policy for M_2. It is immediate from the induction hypothesis that the average reward of this policy is at least $U_{M_1'}(s,\pi_1^{t+1},t+1)$. This completes the proof of the claim.

Since M_1' is a $(\beta/NTR_{\max})$-approximation of M_1, by Lemma A.1.2, it follows that $U_{M_1'}(s,\pi_1',T) \geq U_{M_1'}(s,\pi_1,T) \geq U_{M_1}(s,\pi_1,T) - \beta$. Since, as shown above, $U_{M_2}(s,\pi_2,T) \geq U_{M_1'}(s,\pi_1',T)$, we thus have that $U_{M_2}(s,\pi_2,T) \geq U_{M_1}(s,\pi_1,T) - \beta$, as claimed.

We now show that the difference between the reward of π_2 in M_1 and M_2 is smaller than $2R_{\max}\Pr_{M_1}^{\pi_2,s}(\Phi) + 2\beta$. This implies that the probability of visiting a state-action pair not in B is small iff π_2 attains a near-optimal reward in M_1.

We can partition the state-action pairs in M_1 and M_2 into three sets: (a) the ones in B; (b) the ones in $B' = S_2 \times A_2 - B$; (c) the ones in $B'' = S_1 \times A_1 - S_2 \times A_2$. If a path ρ contains a state-action pair that does not exist in M, then we take $\Pr_M^{\pi,s}(\rho) = 0$ and $U_M(\rho) = 0$. Let L consist of all T-paths starting at state s that are either in M_1 or M_2; let L_1 consist of all T-paths ρ starting at s that are in either M_1 or M_2 such that no state-action pair visited in ρ is in B' (i.e., all state-action pairs in ρ must be in B or B''); and let L_2 consist of all T-paths starting from s that are in either M_1 or M_2 that all T-paths starting from s that are in either M_1 or M_2 and that @@@ include at least one state-action pair in B'. Thus, L is the disjoint union of L_1 and L_2, so that given a policy π for $M \in \{M_1, M_2\}$, we have

$$U_M(s, \pi, T) = \sum_{\rho \in L} \Pr_M^{\pi,s}(\rho) U_M(\rho) = \sum_{\rho \in L_1} \Pr_M^{\pi,s}(\rho) U_M(\rho) + \sum_{\rho \in L_2} \Pr_M^{\pi,s}(\rho) U_M(\rho).$$

We now compare the reward of π_2 in M_2 and M_1.

$$\begin{aligned}
& |U_{M_1}(s, \pi_2, T) - U_{M_2}(s, \pi_2, T)| \\
= \; & |\sum_{\rho \in L} \Pr_{M_1}^{\pi_2,s}(\rho) U_{M_1}(\rho) - \sum_{\rho \in L} \Pr_{M_2}^{\pi_2,s}(\rho) U_{M_2}(\rho)| \\
= \; & |(\sum_{\rho \in L_1} \Pr_{M_1}^{\pi_2,s}(\rho) U_{M_1}(\rho) + \sum_{\rho \in L_2} \Pr_{M_1}^{\pi_2,s}(\rho) U_{M_1}(\rho)) - (\sum_{\rho \in L_1} \Pr_{M_2}^{\pi_2,s}(\rho) U_{M_2}(\rho) + \sum_{\rho \in L_2} \Pr_{M_2}^{\pi_2,s}(\rho) U_{M_2}(\rho))| \\
\leq \; & |\sum_{\rho \in L_1} \Pr_{M_1}^{\pi_2,s}(\rho) U_{M_1}(\rho) - \sum_{\rho \in L_1} \Pr_{M_2}^{\pi_2,s}(\rho) U_{M_2}(\rho)| + |\sum_{\rho \in L_2} \Pr_{M_1}^{\pi_2,s}(\rho) U_{M_1}(\rho) - \sum_{\rho \in L_2} \Pr_{M_2}^{\pi_2,s}(\rho) U_{M_2}(\rho)|.
\end{aligned}$$

We claim that

$$|\sum_{\rho \in L_1} \Pr_{M_1}^{\pi_2,s}(\rho) U_{M_1}(\rho) - \sum_{\rho \in L_1} \Pr_{M_2}^{\pi_2,s}(\rho) U_{M_2}(\rho)| \leq \beta. \tag{A.1}$$

Brafman and Tennenholtz [4] have an analogous claim, but we cannot use their argument directly, since they require M_1 and M_2 to have the same action space, while in our case, M_2 is a $\beta/NTR_{\max}$ mirror of M_1, and has a different action space than that of M_1. We thus need to do a little extra work so as to apply their techniques.

We can further partition the paths in L_1 into two sets: L_1', the paths that consist of only state-action pairs in B, and L_1'', the paths that have at least one pair from B'' . Since M_2 is a mirror of M_1' for B, $R_{\max}$, and s_d, by the construction of L_1' and the definition of mirror and T-path, we have that, for all $\rho \in L_1'$, $\Pr_{M_2}^{\pi_2,s}(\rho) = \Pr_{M_{1'}}^{\pi_2,s}(\rho)$. Let L_3 be the set of all T-paths in M_1. Thus, we have

$$\begin{aligned}\sum_{\rho\in L_1'} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_2}^{\pi_2,s}(\rho)| &= \sum_{\rho\in L_1'} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_1'}^{\pi_2,s}(\rho)| \\ &\leq \sum_{\rho\in L_3} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_1'}^{\pi_2,s}(\rho)| \\ &\leq \beta/R_{\max}.\end{aligned}$$

The last inequality follows from Brafman and Tennenholtz's proof of their Simulation Lemma (i.e., Lemma 4 in [4]), which also shows that if M' is a $\beta/NTR_{\max}$-approximation of M, π is a policy for M, and L_M is the set of all T-paths in M, then $\sum_{\rho\in L_M} |\Pr_M^{\pi,s}(\rho) - \Pr_{M'}^{\pi,s}(\rho)| \leq \beta/R_{\max}$. We can apply their lemma here, since M_1' is a $\beta/NTR_{\max}$- approximation of M_1.

Thus,

$$\begin{aligned}|\sum_{\rho\in L_1'} \Pr_{M_1}^{\pi_2,s}(\rho)U_{M_1}(\rho) - \sum_{\rho\in L_1'} \Pr_{M_2}^{\pi_2,s}(\rho)U_{M_2}(\rho)| &\leq \sum_{\rho\in L_1'} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_2}^{\pi_2,s}(\rho)|U_{M_1}(\rho) \\ &\leq (\sum_{\rho\in L_1'} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_2}^{\pi_2,s}(\rho)|)R_{\max} \\ &\leq \beta.\end{aligned}$$

The first inequality holds since the paths that contain only state-action pairs in B have the same reward in M_1 and M_2; the second inequality holds since $R_{\max}$ is the maximum possible reward; finally, the last inequality holds since $\sum_{\rho\in L_1'} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_2}^{\pi_2,s}(\rho)| \leq \beta/R_{\max}$ (as shown above). On the other hand, if ρ has a state-action pair in B'', then this state-action pair does not occur in M_2, so π_2, which is a policy for M_2, cannot generate such a path. Thus, for all $\rho \in L_1''$, we must have $\Pr_{M_1}^{\pi_2,s}(\rho) = \Pr_{M_2}^{\pi_2,s}(\rho) = 0$, so (A.1) holds.

It follows that

$$
\begin{aligned}
& |U_{M_1}(s,\pi_2,T) - U_{M_2}(s,\pi_2,T)| \\
= \;& |\textstyle\sum_{\rho\in L} \Pr_{M_1}^{\pi_2,s}(\rho)U_{M_1}(\rho) - \sum_{\rho\in L} \Pr_{M_2}^{\pi_2,s}(\rho)U_{M_2}(\rho)| \\
\leq \;& |\textstyle\sum_{\rho\in L_1} \Pr_{M_1}^{\pi_2,s}(\rho)U_{M_1}(\rho) - \sum_{\rho\in L_1} \Pr_{M_2}^{\pi_2,s}(\rho)U_{M_2}(\rho)| + \\
& |\textstyle\sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho)U_{M_1}(\rho) - \sum_{\rho\in L_2} \Pr_{M_2}^{\pi_2,s}(\rho)U_{M_2}(\rho)| \\
\leq \;& |\textstyle\sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho)U_{M_1}(\rho) - \sum_{\rho\in L_2} \Pr_{M_2}^{\pi_2,s}(\rho)U_{M_2}(\rho)| + \beta \quad \text{[by (A.1)]} \\
\leq \;& \max(\textstyle\sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho)R_{\max} - \sum_{\rho\in L_2} \Pr_{M_2}^{\pi_2,s}(\rho)(-R_{\max}), \\
& \qquad \textstyle\sum_{\rho\in L_2} \Pr_{M_2}^{\pi_2,s}(\rho)R_{\max} - \sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho)(-R_{\max})) + \beta \\
= \;& \textstyle\sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho)R_{\max} + \sum_{\rho\in L_2} \Pr_{M_2}^{\pi_2,s}(\rho)R_{\max} + \beta \\
\leq \;& \textstyle\sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho)(2R_{\max}) + 2\beta \qquad \text{[see below]}.
\end{aligned}
$$

The third inequality holds since $-R_{\max} \leq U_{M_1}(\rho) \leq R_{\max}$ and $-R_{\max} \leq U_{M_2}(\rho) \leq R_{\max}$; the last inequality follows since, as we now show, $\sum_{\rho\in L_2} \Pr_{M_2}^{\pi_2,s}(\rho) \leq \sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho) + \beta/R_{\max}$. Recall that L is the disjoint union of L_1 and L_2 and, by construction, $\sum_{\rho\in L} \Pr_{M_1}^{\pi_2,s}(\rho) = \sum_{\rho\in L} \Pr_{M_2}^{\pi_2,s}(\rho) = 1$. We observed above that, for all $\rho \in L_1''$, $\Pr_{M_1}^{\pi_2,s}(\rho) = \Pr_{M_2}^{\pi_2,s}(\rho) = 0$; moreover, $\sum_{\rho\in L_1'} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_2}^{\pi_2,s}(\rho)| \leq \beta/R_{\max}$. Hence, $\sum_{\rho\in L_1} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_2}^{\pi_2,s}(\rho)| \leq \beta/R_{\max}$. Thus,

$$
\begin{aligned}
& \textstyle\sum_{\rho\in L_2} \Pr_{M_2}^{\pi_2,s}(\rho) - \sum_{\rho\in L_2} \Pr_{M_1}^{\pi_2,s}(\rho) \\
= \;& \textstyle\sum_{\rho\in L_1} \Pr_{M_1}^{\pi_2,s}(\rho) - \sum_{\rho\in L_1} \Pr_{M_2}^{\pi_2,s}(\rho) \\
\leq \;& \textstyle\sum_{\rho\in L_1} |\Pr_{M_1}^{\pi_2,s}(\rho) - \Pr_{M_2}^{\pi_2,s}(\rho)| \\
\leq \;& \beta/R_{\max},
\end{aligned}
$$

as desired.

Recall that L_2 is the set of T-paths that include at least one state-action pair in B'. Since, by definition, $\Pr_M^{\pi,s}(\rho) = 0$ if ρ contains a state-action pair not in M, $\Pr_{M_1}^{\pi_2,s}(L_2) = \Pr_{M_1}^{\pi_2,s}(\Phi)$ (recall that Φ is the set of T-paths in M_1 that contain at least one state-action pair not in B). Thus, if $\Pr_{M_1}^{\pi_2,s}(\Phi) = \Pr_{M_1}^{\pi_2,s}(L_2) \geq \frac{\alpha}{2R_{\max}}$, then we are done (since condition (b) of Lemma A.1.5 holds).

Suppose that $\sum_{\rho \in L_2} \Pr_{M_1}^{\pi_2, s}(\rho) < \frac{\alpha}{2R_{\max}}$. This implies that

$$|U_{M_1}(s, \pi_2, T) - U_{M_2}(s, \pi_2, T)| \leq \alpha + 2\beta.$$

Since π_1 is an optimal T-step policy for M_1, we have that $U_{M_1}(s, \pi_2, T) \leq U_{M_1}(s, \pi_1, T)$. As shown above, $U_{M_2}(s, \pi_2, T) \geq U_{M_1}(s, \pi_1, T) - \beta$. Thus,

$$\begin{aligned} & |U_{M_1}(s, \pi_1, T) - U_{M_1}(s, \pi_2, T)| \\ = \; & U_{M_1}(s, \pi_1, T) - U_{M_1}(s, \pi_2, T) \\ \leq \; & U_{M_2}(s, \pi_2, T) - U_{M_1}(s, \pi_2, T) + \beta \\ \leq \; & \alpha + 3\beta. \end{aligned}$$

∎

Definition A.1.6: Define a URMAX$_K$ run l to be (s, a)-*good*, where $s \in S_M$ and $a \in A_M$, if the following two conditions hold:

- if $a = a_0$ then, at the first time t that a_0 has been played kK_0 times in state s, we have that $h_t(s) = g_A(s)$ (i.e., the agent knows all the actions at state s);
- if $a \neq a_0$ then, at all times t in run l after a has been played $K_1(T)$ times in state s, the estimated transition probabilities of (s, a) at time t are all within $\epsilon/4NTR_{\max}$ of the transition probabilities in the MDP underlying the MDPU on which the URMAX algorithm is being run (this is the MDPU we work with throughout this section).

□

Note that the first condition above holds vacuously if (s, a_0) is not played kK_0 times in run l, while the second condition holds vacuously if (s, a) is not played $K_1(T)$ times.

We are now ready to prove Theorem 2.5.2. Again, the argument follows lines similar to those of the corresponding result in [4], although we must take unawareness into account.

Proof of Theorem 2.5.2: The assumptions of Theorem 2.5.2 guarantee that the parameters of URMAX$_K$ $(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ are sufficiently large that no inconsistencies will be found, so URMAX$_K$ will run for K iterations. For a URMAX$_K$ run l, let $\pi'_i(l)$ be the policy played in the ith iteration i of l, and let $M'_i(l) = (S_i, A_i, g_i, P_i, R_i)$ be the approximation of M at the beginning of the ith iteration of l. By construction, $\pi'_i(l)$ is an optimal T-step policy for $M'_i(l)$. We now show that if $K \geq K_4(T, K_0)$, then with probability at least $1 - \delta$, the expected reward of URMAX$_K$ is at least $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$. Define L_{s_0} to be the set of URMAX$_K$ runs starting at s_0. Let Pr be the probability distribution on L_{s_0} determined by M' and URMAX$_K$. The argument proceeds in four steps.

(a) Let O_1 consist of all runs l that are (s, a_0)-good for all states s. We show that $\Pr(O_1 \mid L_{s_0}) \geq 1 - \delta/4$.

(b) Let O_2 consist of all runs l that are (s, a)-good for all pairs (s, a) with $a \neq a_0$. We show that $\Pr(O_2 \mid L_{s_0}) \geq 1 - \delta/4$.

(c) Let O_3 consist of all runs l that contain at most $K_2(T, K_0)$ exploration iterations. (Recall that an exploration iteration is one in which the policy being used is not an (ϵ, T)-optimal policy, and an exploitation iteration is one in which the policy is (ϵ, T)-optimal.) We show that $\Pr(O_3 \mid O_1 \cap O_2 \cap L_{s_0}) \geq 1 - \delta/4$.

(d) Let O_4 consist of all runs l such that, if l contains at least K_3 exploitation iterations, then the average reward over all exploitation iterations in l is at least $\text{Opt}(M, \epsilon, T_M) - 3\epsilon/2$. We show that $\Pr(O_4 \mid L_{s_0}) \geq 1 - \delta/4$.

Part (a): To prove (a), recall that $\Gamma^{\vec{n}}_{s,-1,u}$ where $\vec{n} = (n_1, n_2, \ldots, n_i)$, is the set of MDPU runs where the i'th new action at s is discovered the $n_{i'}$th time that a_0 is played at s, for $i' \in \{1, \ldots, i\}$, and a_0 is played at least $n_i + u$ times at s. Let $\Phi_{s,t}$ be the subset of runs in L_{s_0} such that if a_0 is played at least t times at s, then all actions at s are eventually discovered. (Runs where a_0 is not played t times at s are vacuously in $\Phi_{s,t}$.) We prove by induction on j that, for all strictly increasing sequences of integers $\vec{n} = (n_1, n_2, \ldots, n_i)$ where $i \geq 0$ and $|g_A(s) - g_0(s)| - j \leq i \leq |g_A(s) - g_0(s)|$, and all $u \geq 0$, if we take $t = n_i$ when $i > 0$, and $t = 0$ when $i = 0$, then $\Pr(\Phi_{s,t+jK_0} \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \geq 1 - j\delta/4Nk$. As we show later, (a) follows easily from the special cases of the claim where $i = 0$, $u = 0$, and $j = k$.

Base case: $j = 0$. This case is straightforward: Since $|g_A(s) - g_0(s)| - j \leq i \leq |g_A(s) - g_0(s)|$, we must have $i = |g_A(s) - g_0(s)|$. If $i = 0$, there are initially no actions to discover at s; thus, for all $u \geq 0$, $\Pr(\Phi_{s,0} \mid \Gamma^{()}_{s,-1,u} \cap L_{s_0}) = 1$. In general, since $i = |g_A(s) - g_0(s)|$ is the number of actions that initially can be discovered at state s, after the ith new action is discovered at s, there are no more actions to discover at s. Thus, for all strictly increasing sequences $\vec{n} = (n_1, n_2, \ldots, n_i)$ of integers, and all $u \geq 0$, we must have $\Pr(\Phi_{s,n_i} \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) = 1$.

Induction step: Suppose that the claim holds for j. We now show that the claim holds for $j + 1$. That is, for all strictly increasing sequences $\vec{n} = (n_1, n_2, \ldots, n_i)$ where $i \geq 0$ and $|g_A(s) - g_0(s)| - j - 1 \leq i \leq |g_A(s) - g_0(s)|$, and all $u \geq 0$, if we take $t = n_i$ when $i > 0$, and $t = 0$ when $i = 0$, then

$$\Pr(\Phi_{s,t+(j+1)K_0} \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \geq 1 - (j+1)\delta/4Nk. \tag{A.2}$$

There are two cases: (i) $i = |g_A(s) - g_0(s)|$ and (ii) $i < |g_A(s) - g_0(s)|$. We show that, in both cases, inequality (A.2) holds.

If $i = |g_A(s) - g_0(s)|$, then $\Pr(\Phi_{s,t+(j+1)K_0} \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) = 1$, so inequality (A.2) holds. If $i < |g_A(s) - g_0(s)|$, let $E = \bigcup_{n=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1} \cup (L_{s_0} - \Gamma^{()}_{s,-1,n_i+K_0})$. (Recall that in the special case that

$i = 0$, we have taken $\Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1} = \Gamma^{(n)}_{s,-1}$, so $E = \bigcup_{n=1}^{K_0} \Gamma^{(n)}_{s,-1} \cup (L_{s_0} - \Gamma^{()}_{s,-1,K_0})$.) By Theorem 2.5.1, we have

$$\Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \geq 1 - \frac{\delta}{4Nk}. \tag{A.3}$$

We now show that

$$\Pr(\Phi_{s,n_i+(j+1)K_0} \mid E \cap \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \geq 1 - \frac{j\delta}{4Nk}. \tag{A.4}$$

We partition L_{s_0} into two disjoint subsets: $L_1 = L_{s_0} - \Gamma^{()}_{s,-1,n_i+K_0}$, in which runs a_0 is played fewer than $n_i + K_0$ times at s, and $L_2 = L_{s_0} \cap \Gamma^{()}_{s,-1,n_i+K_0}$, in which runs a_0 is played at least $n_i + K_0$ times at s. Clearly, $\Pr(\Phi_{s,n_i+(j+1)K_0} \mid E \cap \Gamma^{\vec{n}}_{s,-1,u} \cap L_1) = 1 \geq 1 - \frac{j\delta}{4Nk}$ (since runs in which a_0 is played less than $n_i + (j+1)K_0$ times at s are vacuously in $\Phi_{s,n_i+(j+1)K_0}$). Thus, in order to prove (A.4), it suffices to prove that

$$\Pr(\Phi_{s,n_i+(j+1)K_0} \mid E \cap \Gamma^{\vec{n}}_{s,-1,u} \cap L_2) \geq 1 - \frac{j\delta}{4Nk}. \tag{A.5}$$

Let $v = \max(u, K_0)$. Note that

$$\begin{aligned} E \cap \Gamma^{\vec{n}}_{s,-1,u} \cap L_2 &= (\textstyle\bigcup_{n=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1}) \cap \Gamma^{\vec{n}}_{s,-1,u} \cap (L_{s_0} \cap \Gamma^{()}_{s,-1,n_i+K_0}) \\ &= \textstyle\bigcup_{n=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1,\max(K_0,u)-n} \cap L_{s_0} \\ &= \textstyle\bigcup_{n=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1,v-n} \cap L_{s_0}. \end{aligned}$$

Thus, in order to prove (A.5), it is sufficient to prove that

$$\Pr(\Phi_{s,n_i+(j+1)K_0} \mid \bigcup_{n=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1,v-n} \cap L_{s_0}) \geq 1 - \frac{j\delta}{4Nk}.$$

Since, by assumption, $i < |g_A(s) - g_0(s)|$ and $|g_A(s) - g_0(s)| - j - 1 \leq i$, it follows that $|g_A(s) - g_0(s)| - j \leq i + 1 \leq |g_A(s) - g_0(s)|$. Thus, by the induction hypothesis,

$$\Pr(\Phi_{s,n_i+n+jK_0} \mid \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1,v-n} \cap L_{s_0}) \geq 1 - j\delta/4Nk$$

for all $n \in [1, K_0]$. Moreover, for all $n \in [1, K_0]$, we have $\Phi_{s,n_i+n+jK_0} \subseteq \Phi_{s,n_i+(j+1)K_0}$, so $\Pr(\Phi_{s,n_i+(j+1)K_0} \mid \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1,v-n} \cap L_{s_0}) \geq 1 - j\delta/4Nk$. Therefore, $\Pr(\Phi_{s,n_i+(j+1)K_0} \mid \bigcup_{n=1}^{K_0} \Gamma^{\vec{n}\cdot(n_i+n)}_{s,-1,v-n} \cap L_{s_0}) \geq 1 - \frac{j\delta}{4Nk}$, and (A.5) holds. It then follows that (A.4) holds.

Therefore,

$$\begin{aligned}
& \Pr(\Phi_{s,n_i+(j+1)K_0} \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \\
\geq\ & \Pr(\Phi_{s,n_i+(j+1)K_0} \mid E \cap \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \cdot \Pr(E \mid \Gamma^{\vec{n}}_{s,-1,u} \cap L_{s_0}) \\
\geq\ & (1 - \tfrac{j\delta}{4Nk})(1 - \tfrac{\delta}{4Nk}) && \text{[by (A.3) and (A.4)]} \\
\geq\ & 1 - \tfrac{(j+1)\delta}{4Nk},
\end{aligned}$$

as desired. This completes the induction.

To prove (a), let $i = 0$, $u = 0$, and $j = k$. Note that $\Gamma^{()}_{s,-1,0} \cap L_{s_0} = L_{s_0}$. Thus, $\Pr(\Phi_{s,kK_0} \mid L_{s_0}) = \Pr(\Phi_{s,kK_0} \mid \Gamma^{()}_{s,-1,0} \cap L_{s_0}) \geq 1 - \frac{k\delta}{4Nk} = 1 - \frac{\delta}{4N}$. In other words, we just proved that for a state s, conditional on L_{s_0}, with probability at least $1 - \frac{\delta}{4N}$, if a_0 is played at least kK_0 times at s, then all actions at s are discovered. Thus, the probability that not all actions are discovered at state s is at most $\frac{\delta}{4N}$. Since there are at most N states, if a is played at least kK_0 time at each state, the probability that not all actions are discovered at some state is at most $\frac{\delta}{4}$. Thus, if a is played at least kK_0 times at all states, then with probability at least $1 - N(\frac{\delta}{4N}) = 1 - \frac{\delta}{4}$, all actions will be discovered at all states. By the definition of (s, a_0)-good, $\Pr(O_1 \mid L_{s_0}) \geq 1 - \frac{\delta}{4}$, as desired.

Part (b): For part (b), we first show that for all state-action pairs (s, a) where $a \neq a_0$, if (s, a) is played $K_1(T)$ times, then its estimated transition probabilities are within $\epsilon/4NTR_{\max}$ of the real transition probabilities with high probability. More precisely, fix a state-action pair (s, a). Let $C_{s,a}$ consist of runs in L_{s_0} in which action a is performed in s at least $K_1(T)$ times. In a URMAX$_K$ run l, let $P_l(s, s', a)$ denote the estimated probability of transitioning from state s to s' when playing action a. We show that, conditional on $C_{s,a}$, for each state s', with probability at least $1 - \frac{\delta}{4N^2k}$,

$$|P_l(s, s', a) - P_M(s, s', a)| \leq \frac{\epsilon}{4NTR_{\max}}.$$

For each state s', let $X_i^{s,a,s'}$ be an indicator random variable such that, for a run $l \in L_{s_0}$, $X_i^{s,a,s'}(l) = 1$ if, on the ith time that a was performed in state s in run l, a transition was taken to s', and is 0 otherwise. If a was performed less than i times in state s in l, define $X_i^{s,a,s'}(l) = 1$. Let $Z^{s,a,s'}(l) = (\sum_{i=1}^{K_1(T)} X_i^{s,a,s'}(l))/K_1(T)$. Note that for $l \in C_{s,a}$, we have $Z^{s,a,s'}(l) = P_l(s, s', a)$. By assumption, $E(Z^{s,a,s'} \mid C_{s,a}) = P_M(s, s', a)$. By the Hoeffding-Chernoff bound, $\Pr(|Z^{s,a,s'} - P_M(s, s', a)| > t \mid C_{s,a}) \leq 2e^{-2t^2 K_1(T)}$. Let $t = K_1(T)^{-1/3}$. Since $K_1(T) \geq (4NTR_{\max}/\epsilon)^3$, it follows that $t \leq \epsilon/4NTR_{\max}$. Thus,

$$\Pr(|Z^{s,a,s'} - P_M(s, s', a)| > \epsilon/4NTR_{\max} \mid C_{s,a}) \leq 2e^{-2K_1(T)^{1/3}}.$$

Since $K_1(T) \geq \frac{1}{8}(\ln \frac{8N^2k}{\delta})^3$, it follows that $K_1(T)^{1/3} \geq \frac{1}{2}(\ln \frac{8N^2k}{\delta})$. Thus, $2e^{-2K_1(T)^{1/3}} \leq \delta/4N^2k$. To conclude, we have that

$$\Pr(|Z^{s,a,s'} - P_M(s, s', a)| > \epsilon/4NTR_{\max} \mid C_{s,a}) \leq \frac{\delta}{4N^2k}.$$

Since there are at most N^2k triples of the form (s, a, s'), by the union bound, with probability at least $1 - \delta/4$, in runs l in $C_{s,a}$, $|Z^{s,a,s'} - P_M(s, s', a)| \leq \epsilon/4NTR_{\max}$ holds simultaneously for all s', s and a. By the definition of (s, a)-good, we have that $\Pr(O_2 \mid L_{s_0}) \geq 1 - \delta/4$, and (b) holds.

Part (c): We now prove part (c). Recall that $M_i'(l) = (S_i', A_i', g_i', P_i', R_i')$ is the approximation of M in iteration i of URMAX$_K$ in run l. Let l be a URMAX$_K$ run in L_{s_0} that is (s, a)-good for all state-action pairs in M (including pairs with $a = a_0$). By the definition of (s, a)-good, this means that if, at the end of iteration i, the state-action pair (s, a) with $a \neq a_0$ is marked as known (i.e., has been visited $K_1(T)$ times), then $|P_i'(s, s', a) - P_M(s, s', a)| \leq \epsilon/4NTR_{\max}$, and if (s, a_0) is marked as known (i.e., has been visited kK_0 times), then $g_i'(s) = g(s)$. The definition of URMAX$_K$ guarantees that if a state-action pair (s, a) is marked as unknown, then $P_i'(s, s_d, a) = 1$ and $R_i'(s, s', a) = R_{\max}$ for all s'; and if a state-action pair (s, a_0) is marked as known, then $P_i'(s, s, a_0) = 1$ and $R_i'(s, s', a_0) = -R_{\max}$ for all s'. Thus, in each iteration of the

execution, $M'_i(l)$ is an $\epsilon/4NTR_{\max}$-mirror of M for B, the set of state-action pairs marked as known in $M'_i(l)$, $R_{\max}$, and the dummy state s_d in the algorithm. Let π^* be the optimal T-step policy for M, and $\pi'_i(l)$ be the optimal T-step policy for $M'_i(l)$. Applying Lemma A.1.5 to $M'_i(l)$ and M with $\alpha = \epsilon/4$ and $\beta = \epsilon/4$ (since $M'_i(l)$ is a $\epsilon/4NTR_{\max}$-mirror of M), we either have

$$U_M(\pi'_i(l), T) \geq U_M(\pi^*, T) - \alpha - 3\beta = U_M(\pi^*, T) - \epsilon,$$

or, taking Φ to be the set of T-paths in M that contain at least one state-action pair not in B, we have

$$\Pr_M^{\pi'_i(l),s}(\Phi) \geq \frac{\epsilon}{8R_{\max}}.$$

Let $F = O_1 \cap O_2 \cap L_{s_0}$; that is, F is the subset of L_{s_0} consisting of the runs that are (s,a)-good for all pairs (s,a) (including ones where $a = a_0$). Let $F_{\leq K_2}$ be the subset of F consisting of the runs with at most $K_2(T, K_0)$ exploration iterations; let $F_{<K_2}$ be the subset of F consisting of the runs with less than $K_2(T, K_0)$ exploration iterations; let $F_{\geq K_2}$ be the subset of F consisting of the runs with at least $K_2(T, K_0)$ exploration iterations; lastly, let F_{K_2} be the subset of F consisting of runs with exactly $K_2(T, K_0)$ exploration iterations. We want to show that $\Pr(F_{\leq K_2} \mid F) \geq 1-\delta/4$. We actually show that $\Pr(F_{K_2} \mid F_{\geq K_2}) \geq 1 - \delta/4$. It then follows that $\Pr(F_{\leq K_2} \mid F_{\geq K_2}) \geq 1 - \delta/4$. Since $\{F_{\geq K_2}, F_{<K_2}\}$ is a disjoint partition of F, and $\Pr(F_{\leq K_2} \mid F_{<K_2}) = 1$, it immediately follows that $\Pr(F_{\leq K_2} \mid F) \geq 1 - \delta/4$.

We start by proving that, conditional on $F_{\geq K_2}$, at the end of the $K_2(T, K_0)$th exploration iteration of a run l, with probability at least $1 - \delta/4$, all state-action pairs in M are known. Let X_i be an indicator random variable such that, for $l \in L_{s_0}$, $X_i(l) = 1$ if, during the ith iteration, an unknown state-action pair is visited, and $X_i(l) = 0$ otherwise.

Let $i > 0$, let Q^i be the set of URMAX_K runs l such that the ith iteration of l is an exploration iteration (which means that $\pi'_i(l)$ is *not* (ϵ, T)-optimal), and let O^i be the set of URMAX_K runs

that visit an unknown state-action pair in their ith iterations. We show that $E(X_i(l) \mid Q^i) \geq \frac{\epsilon}{8R_{\max}}$. In order to show this, we show that for all histories x of length $(i-1)T$ (i.e., the time required for $i-1$ iterations), if s_x is the state that the DM is in at time $(i-1)T$ in history x, and Q^i_x is the subset of runs in Q^i with prefix x, then $E(X_i(l) \mid Q^i_x) \geq \frac{\epsilon}{8R_{\max}}$; it then follows immediately that $E(X_i(l) \mid Q^i) \geq \frac{\epsilon}{8R_{\max}}$. By the definition of URMAX$_K$, $\pi'_i(l)$ only depends on the prefix x. Fix i and x, so that $\pi'_i(l)$ is fixed. The actions taken during the ith iteration of URMAX$_K$ in run l are determined by $\pi'_i(l)$. Thus, the set of runs in O^i with prefix x corresponds to the set of paths in Φ starting at s_x that are generated by $\pi'_i(l)$, and $\Pr_M^{\pi'_i(l),s_x}(\Phi) = \Pr(O^i \mid Q^i_x)$. As observed above, if $\pi'_i(l)$ is not (ϵ, T)-optimal, we must have $\Pr_M^{\pi'_i(l),s}(\Phi) \geq \frac{\epsilon}{8R_{\max}}$ for all $s \in S_M$. By the definition of Q^i_x, $\pi'_i(l)$ is not (ϵ, T)-optimal. Thus, $\Pr(O^i \mid Q^i_x) \geq \frac{\epsilon}{8R_{\max}}$, and $E(X_i(l) \mid Q^i_x) = \Pr(X_i(l) = 1 \mid Q^i_x) = \Pr(O^i \mid Q^i_x) \geq \frac{\epsilon}{8R_{\max}}$, as desired.

If run l has at least $K_2(T, K_0)$ exploration iterations, and the first $K_2(T, K_0)$ exploration iterations in l occur in iterations $j_1, j_2, \ldots, j_{K_2(T,K_0)}$, then define $Z(l) = (\sum_{i=1}^{K_2(T,K_0)} X_{j_i}(l))/K_2(T, K_0)$; If l has fewer than $K_2(T, K_0)$ exploration iterations, then define $Z(l) = 1$. Note that $Z(l)$ is an estimate of the probability of visiting an unknown state-action pair during the first $K_2(T, K_0)$ exploration iterations in a URMAX$_K$ run l. Since we proved above that $E(X_i(l) \mid Q^i) \geq \frac{\epsilon}{8R_{\max}}$ for all i, we must have $E(Z(l) \mid F_{\geq K_2}) \geq \frac{\epsilon}{8R_{\max}}$. By the Hoeffding-Chernoff bound, we have that

$$\Pr(E(Z(l)) - Z(l) > t \mid F_{\geq K_2}) \leq e^{-2t^2 K_2(T,K_0)}.$$

Let $t = \frac{\epsilon}{16R_{\max}}$. Since $E(Z(l) \mid F_{\geq K_2}) \geq \epsilon/8R_{\max}$, it follows that $\Pr(Z(l) < \frac{\epsilon}{16R_{\max}} \mid F_{\geq K_2}) \leq e^{-2(\frac{\epsilon}{16R_{\max}})^2 K_2(T,K_0)}$. Equivalently, $\Pr(Z(l)K_2(T, K_0) < \frac{\epsilon}{16R_{\max}} K_2(T, K_0) \mid F_{\geq K_2}) \leq e^{-2(\frac{\epsilon}{16R_{\max}})^2 K_2(T,K_0)}$. Note that $Z(l)K_2(T, K_0) = \sum_{i=1}^{K_2(T,K_0)} X_{i_j}$ is a lower bound on the number of times that an unknown state-action pair is visited during the first $K_2(T, K_0)$ exploration iterations in a run. Since $K_2(T, K_0) \geq \frac{16R_{\max}}{\epsilon} Nk(K_1(T) + K_0)$, we have that

$$\Pr(Z(l)K_2(T, K_0) < Nk(K_1(T) + K_0) \mid F_{\geq K_2}) \leq e^{-2(\frac{\epsilon}{16R_{\max}})^2 K_2(T,K_0)}.$$

Moreover, since $K_2(T, K_0) \geq \frac{1}{2}(\frac{16R_{\max}}{\epsilon})^2 \ln \frac{4}{\delta}$, we have that $e^{-2(\frac{\epsilon}{16R_{\max}})^2 K_2(T,K_0)} \leq \delta/4$. Thus, we have that

$$\Pr(Z(l)K_2(T, K_0) < Nk(K_1(T) + K_0) \mid F_{\geq K_2}) \leq \delta/4.$$

This shows that, conditional on $F_{\geq K_2}$, at the end of the $K_2(T, K_0)$th exploration iteration in a run l, with probability at least $1 - \delta/4$, the algorithm has visited unknown state-action pairs at least $Nk(K_1(T) + K_0)$ times. It immediately follows that all state-action pairs become known at that point. This is because we can visit each unknown state-action pair (s, a) where $a \neq a_0$ at most $K_1(T)$ times (after $K_1(T)$ visits, it becomes known), and visit each unknown state-action pair for a_0 at most $kK_0(T)$ times (after kK_0 visits, it becomes known). Moreover, there are at most Nk state-action pairs (s, a) where $a \neq a_0$, and there are at most N state-action pairs of the form (s, a_0).

We now show that for a run $l \in F$, if all state-action pairs become known in iteration i, then j must be an exploitation iteration for all $j > i$. Recall that all runs in F are (s, a)-good for all (s, a) pairs. By the definition of (s, a)-good, $M'_j(l)$ must be an $\frac{\epsilon}{4NTR_{\max}}$-approximation to M. Thus, by Theorem A.1.2, we have that

$$U_{M'_j(l)}(s, \pi'_j(l), T) - U_M(s, \pi'_j(l), T) \leq \frac{\epsilon}{4} \tag{A.6}$$

and

$$U_M(s, \pi^*, T) - U_{M'_j(l)}(s, \pi^*, T) \leq \frac{\epsilon}{4}.$$

Furthermore, $U_{M'_j(l)}(s, \pi'_j(l), T) \geq U_{M'_j(l)}(s, \pi^*, T)$, since π'_j is the optimal policy for M'_j. So we must have that

$$U_M(s, \pi^*, T) - U_{M'_j(l)}(s, \pi'_j(l), T) \leq \frac{\epsilon}{4}. \tag{A.7}$$

Adding (A.6) and (A.7), we get

$$U_M(s, \pi^*, T) - U_M(s, \pi'_j(l), T) \leq \frac{\epsilon}{2}.$$

Therefore, for all runs $l \in F$, after all state-action pairs become known, all policies used in later iterations must be (ϵ, T)-optimal for M, and thus all later iterations must be exploitation iterations. Since we proved above that, conditional on $F_{\geq K_2}$, with probability at least $1 - \delta/4$, at the end of the $K_2(T, K_0)$th exploration iteration, all state-action pairs in M are known, we must have that $\Pr(F_{K_2} \mid F_{\geq K_2}) \geq 1 - \delta/4$, and it immediately follows that $\Pr(F_{\leq K_2} \mid F) \geq 1 - \delta/4$. That is, $\Pr(O_3 \mid O_1 \cap O_2 \cap L_{s_0}) \geq 1 - \delta/4$, as desired. Thus, (c) holds.

Part (d): We now prove (d). We want to show that conditional on L_{s_0}, with probability at least $1 - \delta/4$, if a URMAX$_K$ run l contains at least K_3 exploitation iterations, then the average reward over these exploitation iterations is at least $\text{Opt}(M, \epsilon, T) - 3\epsilon/2$. Although the expected reward of an exploitation iteration is (T, ϵ)-optimal, the actual average reward could be lower. We again make use of the Hoeffding-Chernoff bound, and the fact that the standard deviation of the expected reward in an exploitation iteration is bounded, because the maximum reward is bounded by $R_{\max}$. Let j be the number of exploitation iterations in run l. Let the expected reward of the ith exploitation iteration in l be $\mu_i(l)$, and let $X_i(l)$ be a random variable defined on L_{s_0} that gives the average reward in iteration i of run l (if l contains less than i exploitation iterations, then define $X_i(l) = 0$). Let $Y_i(l) = \frac{X_i(l)}{R_{\max}}$. Note that the range of Y_i is $[-1, 1]$. Let $Z(l) = (\sum_{i=1}^{j} Y_j(l))/j$, that is, the average reward of all exploitation iterations in l, divided by $R_{\max}$. Let $\mu(l) = \frac{\sum_{i=1}^{j} \mu_i(l)}{j}$.

We have that $E(Z(l)) = \frac{\sum_{i=1}^{j} \mu_i(l)}{jR_{\max}} = \frac{\mu(l)}{R_{\max}}$. The Hoeffding-Chernoff bound implies that $\Pr(\frac{\mu(l)}{R_{\max}} - Z(l) \geq t) < e^{\frac{-t^2 j}{2}}$. Let $t = j^{-1/3}$. If $j \geq K_3$, then $j \geq K_3 \geq (\frac{2R_{\max}}{\epsilon})^3$. It follows that $t \leq \frac{\epsilon}{2R_{\max}}$. Thus, $\Pr(\frac{\mu(l)}{R_{\max}} - Z(l) \geq \frac{\epsilon}{2R_{\max}}) < e^{\frac{-j^{\frac{1}{3}}}{2}}$. Equivalently,

$$\Pr(\mu(l) - Z(l)R_{\max} \geq \frac{\epsilon}{2}) < e^{\frac{-j^{\frac{1}{3}}}{2}}.$$

Since $j \geq K_3 \geq 8(\ln \frac{4}{\delta})^3$, it follows that $e^{\frac{-j^{\frac{1}{3}}}{2}} \leq \delta/4$. This implies that with probability at

least $1 - \delta/4$, if l contains at least K_3 exploitation iterations, then the average reward of these exploitation iterations is at most $\epsilon/2$ lower than $\mu(l)$, where $\mu(l)$ is at least $\text{Opt}(M, \epsilon, T) - \epsilon$ (since $\mu_i \geq \text{Opt}(M, \epsilon, T) - \epsilon$ for all $i \in [1, K_3]$ by the definition of exploitation iteration). That is, $\Pr(O_4 \mid L_{s_0}) \geq 1 - \delta/4$, and (d) holds.

Putting everything together: We now put everything together. By parts (a) and (b) above and the union bound, we have that

$$\Pr(O_1 \cap O_2 \mid L_{s_0}) \geq 1 - \delta/4 - \delta/4 = 1 - \delta/2.$$

Moreover, by part (c) above, we have that

$$\Pr(O_3 \mid O_1 \cap O_2 \cap L_{s_0}) \geq 1 - \delta/4.$$

Therefore,

$$\begin{aligned} \Pr(O_3 \mid L_{s_0}) &= \Pr(O_3 \mid O_1 \cap O_2 \cap L_{s_0}) \Pr(O_1 \cap O_2 \mid L_{s_0}) \\ &\geq (1 - \delta/2)(1 - \delta/4) \\ &> 1 - 3\delta/4. \end{aligned}$$

Now by part (d) and the union bound, we must have that

$$\Pr(O_3 \cap O_4 \mid L_{s_0}) \geq 1 - 3\delta/4 - \delta/4 = 1 - \delta.$$

We now show that, for all $l \in O_3 \cap O_4 \cap L_{s_0}$, run l has an expected reward of at least $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$. The assumptions of Theorem 2.5.2 guarantee that the parameters of $\text{URMAX}_K(K_0, N, k, R_{\max}, T, \epsilon, \delta, s_0)$ are sufficiently large that no inconsistencies will be found, so a URMAX_K run contains exactly K iterations. Since $K \geq K_4(T, K_0) \geq K_2(T, K_0) + K_3$, and $l \in O_3$ (which means that l contains at most $K_2(T, K_0)$ exploration iterations), l must contain

at least K_3 exploitation iterations. Moreover, $l \in O_4$ (which means that if l contains at least K_3 exploitation iterations, then the expected reward of the exploitation iterations in l is at least $\text{Opt}(M, \epsilon, T_M) - 3\epsilon/2$), thus the average reward of the exploitation iterations in l is at least $\text{Opt}(M, \epsilon, T_M) - 3\epsilon/2$. Finally, the minimum reward is 0, and all iterations in l have T steps, so the average reward of all iterations in l is at least

$$\frac{(\text{Opt}(M, \epsilon, T_M) - 3\epsilon/2)(K - K_2(T, K_0))}{K}. \tag{A.8}$$

Since simple calculus shows that (A.8) is increasing in K, the average reward is at least as large as the result of replacing K in (A.8) by $K_4(T, K_0) = \frac{2R_{\max}}{\epsilon} K_2(T, K_0)$, namely

$$\begin{aligned}
&\frac{(\text{Opt}(M,\epsilon,T_M)-3\epsilon/2)(\frac{2R_{\max}}{\epsilon}K_2(T,K_0)-K_2(T,K_0))}{\frac{2R_{\max}}{\epsilon}K_2(T,K_0)} \\
=\ &\frac{(\text{Opt}(M,\epsilon,T_M)-3\epsilon/2)(\frac{2R_{\max}}{\epsilon}-1)}{\frac{2R_{\max}}{\epsilon}} \\
=\ &\frac{(\text{Opt}(M,\epsilon,T_M)-3\epsilon/2)(2R_{\max}-\epsilon)}{2R_{\max}} \\
=\ &\frac{2R_{\max}\text{Opt}(M,\epsilon,T_M)-3R_{\max}\epsilon-\text{Opt}(M,\epsilon,T_M)\epsilon+\frac{3}{2}\epsilon^2}{2R_{\max}} \\
=\ &\frac{2R_{\max}(\text{Opt}(M,\epsilon,T_M)-2\epsilon)+(R_{\max}\epsilon-\text{Opt}(M,\epsilon,T_M)\epsilon+\frac{3}{2}\epsilon^2)}{2R_{\max}} \\
\geq\ &\frac{2R_{\max}(\text{Opt}(M,\epsilon,T_M)-2\epsilon)}{2R_{\max}} \qquad [\text{since } R_{\max} \geq \text{Opt}(M, \epsilon, T_M] \\
=\ &\text{Opt}(M, \epsilon, T_M) - 2\epsilon.
\end{aligned}$$

Thus, all runs $l \in O_3 \cap O_4 \cap L_{s_0}$ have an expected reward of at least $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$. Since we proved above that $\Pr(O_3 \cap O_4 \mid L_{s_0}) \geq 1 - \delta$, with probability at least $1 - \delta$, the average reward of URMAX_K running on M is at least $\text{Opt}(M, \epsilon, T_M) - 2\epsilon$. Again, since the parameters are sufficiently large, the URMAX_K algorithm runs exactly K iterations. Moreover, it is obvious that the complexity of each iteration is polynomial in N, k, and T. Thus, the running time of the algorithm is polynomial in K, N, k, and T. ∎

A.2 Proof for Theorem 2.5.3

THEOREM 2.5.3. *For all MDPs $M = (S, A, g, P, R)$ compatible with S_0, g_0, and G_0, if $0 < \delta < 1$, $\epsilon > 0$, the maximum possible reward in M is $R^*_{\max}$, and the ϵ-return mixing time of M is T_M, then for all states $s_0 \in S_0$, there exists a time t^* polynomial in $|S|$, $|A|$, T_M, $R^*_{\max}$, $1/\epsilon$, $1/\delta$, and $K^*_0 = \max\{\min\{n_1 : \sum_{t=1}^{n_1} D(1, t, s) \geq \ln(4|S||A|/\delta)\} : s \in S\}$, such that, for all $t \geq t^*$, the expected average reward of running* URMAX(ϵ, δ, s_0) *on M for t steps is at least $(1 - \delta)\text{Opt}(M, \epsilon, T_M) - 3\epsilon$.*

Proof: For convenience, we call an iteration of the loop in Algorithm 2, where URMAX$(T, |S_0| + T - 1, |A_0| + T - 1, T, T, \epsilon, \delta, s_0)$ is executed for some values of T, a *long iteration*, and call an iteration of the loop in Algorithm 3, where the current best guess for the optimal policy is executed for T steps, a *short iteration*.

Since $K_5(T)$ is polynomial in T, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, by the definition of URMAX$_K$, it is immediate that URMAX(ϵ, δ, s_0) takes time polynomial in $|S|$, $|A|$, T_M, $R^*_{\max}$, $1/\epsilon$, $1/\delta$, and K^*_0 to reach the earliest long iteration in which all the relevant parameters of URMAX$(T, |S_0| + T - 1, |A_0| + T - 1, T, T, \epsilon, \delta, s_0)$ are all at least as large as their actual values (i.e., $|S_0| + T - 1 \geq |S|$, $|A_0| + T - 1 \geq |A|$, $T \geq T_M$, $T \geq R^*_{\max}$, and $T \geq K^*_0$). Call this long iteration the *icebreaker* iteration.

Like $H_4(T)$ in RMAX, which includes extra iterations that compensate for the low-reward period, $K_5(T)$ also includes short iterations that are intended to compensate for low-reward periods. However, in RMAX, those extra iterations compensate for the low-reward period in the past, while here, the extra short iterations compensate for the low-reward period in the next long iteration. That is, each long iteration that occurs after the icebreaker iteration compensates for the low reward period in the next long iteration. Using this compensation mechanism, we

show below that the expected average reward of URMAX over any time interval that starts at the beginning of the icebreaker iteration and ends after the end of the icebreaker iteration is above $\text{Opt}(M, \epsilon, T_M) - \frac{5}{2}\epsilon - \text{Opt}(M, \epsilon, T_M)\delta$.

Let the icebreaker iteration be the (T^*)th long iteration. Let $B(i)$ be the time that the (T^*+i)th long iteration begins (and the $(T^* + i - 1)$st long iteration ends). $B(i)$ may depend on the run; it is a random variable. However, $B(i + 1) - B(i)$ is independent of the run; since in long iteration $T^* + i$, all the parameters have values at least as large as their actual values, it follows that $B(i + 1) - B(i) = (T^* + i)K_5(T^* + i)$. Let $A[B(i), B(j)]$ for $j > i$ (resp. $A[B(i), B(i) + t]$) denote the average reward in the interval $[B(i), B(j)]$ (resp. $[B(i), B(i) + t]$). Again, the average reward is a random variable; it depends on the run. We want to show that $E(A[B(0), B(0) + t]) \geq \text{Opt}(M, \epsilon, T_M) - \frac{5}{2}\epsilon - \text{Opt}(M, \epsilon, T_M)\delta$ as long as $t \geq B(1) - B(0)$.

We first prove that $E(A[B(i), B(j)]) \geq \text{Opt}(M, \epsilon, T_M) - 2\epsilon - \text{Opt}(M, \epsilon, T_M)\delta$ if $0 \leq i < j$. By the definition of the icebreaker iteration, the parameters used in the icebreaker iteration T^* and any later long iteration are sufficiently large. Thus, by Theorem 2.5.2, with probability at least $1 - \delta$, $A[B(i), B(i + 1)] \geq \text{Opt}(M, \epsilon, T_M) - 2\epsilon$. Since the minimum reward in M is 0,

$$E(A[B(i), B(i + 1)]) \geq (\text{Opt}(M, \epsilon, T_M) - 2\epsilon)(1 - \delta) \geq \text{Opt}(M, \epsilon, T_M) - 2\epsilon - \text{Opt}(M, \epsilon, T_M)\delta.$$

The desired result easily follows.

Now consider an arbitrary $t \geq B(1) - B(0)$. Suppose that t occurs during the $(T^* + i)$th long interval; that is, $B(i) - B(0) \leq t < B(i + 1) - B(0)$, where $i \geq 1$. Let $t' = B(0) + t - B(i)$, so that $B(0) + t = B(i) + t'$. We consider two cases. First suppose that $t \geq B(i) - B(0) + K_4(T^* + i, T^* + i)(T^* + i)$. Consider the interval $[B(i), B(i) + t']$. For this interval the same arguments as above, using Theorem 2.5.2, show that $E(A[B(i), B(i) + t']) \geq \text{Opt}(M, \epsilon, T_M) - 2\epsilon - \text{Opt}(M, \epsilon, T_M)\delta$. Since $E(A[B(0), B(i)]) \geq \text{Opt}(M, \epsilon, T_M) - 2\epsilon - \text{Opt}(M, \epsilon, T_M)\delta$, it easily

follows that $E(A[B(0), B(0)+t]) \geq \text{Opt}(M, \epsilon, T_M) - 2\epsilon - \text{Opt}(M, \epsilon, T_M)\delta$.

If $t < B(i) - B(0) + K_4(T^*+i, T^*+i)(T^*+i)$, note that $B(0)+t = B(i)+t'$, where $t' < K_4(T^*+i, T^*+i)(T^*+i)$. Consider the interval $[B(i-1), B(i)+t']$. Let $TOT[I]$ represent the total reward in the interval I. Clearly $TOT[B(i-1), B(i)+t'] \geq TOT[B(i-1), B(i)]$, and with probability at least $1-\delta$, $TOT[B(i-1), B(i)] \geq (\text{Opt}(M, \epsilon, T_M) - 2\epsilon)K_5(T^*+i-1)(T^*+i-1)$. Thus, with probability at least $1-\delta$,

$$A[B(i-1), B(i)+t'] \geq \frac{(\text{Opt}(M, \epsilon, T_M) - 2\epsilon)K_5(T^*+i-1)(T^*+i-1)}{B(i) - B(i-1) + t'}.$$

Note that

$$\begin{aligned} B(i) - B(i-1) + t' &= K_5(T^*+i-1)(T^*+i-1) + t' \\ &\leq K_4(T^*+i, T^*+i)\tfrac{2(T^*+i)}{\epsilon}(T^*+i-1) + K_4(T^*+i, T^*+i)(T^*+i) \\ &\leq K_4(T^*+i, T^*+i)(T^*+i)(\tfrac{2(T^*+i-1)}{\epsilon} + 1). \end{aligned}$$

It follows that, with probability at least $1-\delta$,

$$\begin{aligned} A[B(i-1), B(i)+t'] &\geq \frac{(\text{Opt}(M,\epsilon,T_M)-2\epsilon)K_4(T^*+i,T^*+i)\frac{2(T^*+i)}{\epsilon}(T^*+i-1)}{K_4(T^*+i,T^*+i)(T^*+i)(\frac{2(T^*+i-1)}{\epsilon}+1)} \\ &= \frac{2(\text{Opt}(M,\epsilon,T_M)-2\epsilon)(T^*+i-1)}{2(T^*+i-1)+\epsilon}. \end{aligned} \tag{A.9}$$

Easy algebraic manipulations show that

$$\begin{aligned} & 2(\text{Opt}(M, \epsilon, T_M) - 2\epsilon)(T^*+i-1) \\ = \; & (\text{Opt}(M, \epsilon, T_M) - \tfrac{5}{2}\epsilon)(2(T^*+i-1)+\epsilon) + \epsilon(T^*+i-1) - \epsilon\text{Opt}(M, \epsilon, T_M) + \tfrac{5}{2}\epsilon^2. \end{aligned}$$

Since the (T^*)th long iteration is the icebreaker iteration, we must have $T^* \geq R^*_{\max} \geq \text{Opt}(M, \epsilon, T_M)$. Moreover, by assumption, $i \geq 1$. Thus, $\epsilon(T^*+i-1) \geq \epsilon\text{Opt}(M, \epsilon, T_M)$, so

$$2(\text{Opt}(M, \epsilon, T_M) - 2\epsilon)(T^*+i-1) \geq (\text{Opt}(M, \epsilon, T_M) - \frac{5}{2}\epsilon)(2(T^*+i-1)+\epsilon).$$

It now follows from (A.9) that, with probability at least $1 - \delta$,

$$A[B(i-1), B(i) + t'] \geq \mathrm{Opt}(M, \epsilon, T_M) - \frac{5}{2}\epsilon.$$

Thus, $E(A[B(i-1), B(i)+t']) \geq \mathrm{Opt}(M, \epsilon, T_M) - \frac{5}{2}\epsilon - \mathrm{Opt}(M, \epsilon, T_M)\delta$. Since, as we have shown, if $i > 1$, $E(A[B(0), B(i-1)]) > \mathrm{Opt}(M, \epsilon, T_M) - 2\epsilon - \mathrm{Opt}(M, \epsilon, T_M)\delta$, it easily follows that

$$E(A[B(0), B(i) + t']) \geq \mathrm{Opt}(M, \epsilon, T_M) - \frac{5}{2}\epsilon - \mathrm{Opt}(M, \epsilon, T_M)\delta.$$

Thus, we have shown that the expected average reward of URMAX(ϵ, δ, s_0) over any time interval that starts at the beginning of the icebreaker iteration and ends after the end of the icebreaker iteration is at least $\mathrm{Opt}(M, \epsilon, T_M) - \frac{5}{2}\epsilon - \mathrm{Opt}(M, \epsilon, T_M)\delta$. We want to find a time t^* such that, for all $t \geq t^*$, the expected average reward of URMAX(ϵ, δ, s_0) in the interval $[0, t]$ is at least $\mathrm{Opt}(M, \epsilon, T_M) - 3\epsilon - \mathrm{Opt}(M, \epsilon, T_M)\delta$. Let $t^* = B(0)(\frac{2R^*_{\max}}{\epsilon} + 1)$. Since $B(0)$ is polynomial in $|S|$, $|A|$, T_M, $1/\epsilon$, $1/\delta$, $R^*_{\max}$, and K^*_0, so is t^*. Suppose that $t \geq t^*$. Then

$$\begin{aligned} E(A[0,t]) &\geq \frac{(t-B(0))E(A[B(0),t])}{t} \\ &\geq \frac{(t-B(0))(\mathrm{Opt}(M,\epsilon,T_M)-\frac{5}{2}\epsilon-\mathrm{Opt}(M,\epsilon,T_M)\delta)}{t}. \end{aligned}$$

Easy calculus shows that a function $f(t) = (t-a)b/t$ is increasing as a function of t if $ab > 0$, so it follows that

$$\begin{aligned} E(A[0,t]) &\geq \frac{(t^*-B(0))(\mathrm{Opt}(M,\epsilon,T_M)-\frac{5}{2}\epsilon-\mathrm{Opt}(M,\epsilon,T_M)\delta)}{t^*} \\ &= \frac{(t^*-B(0))(\mathrm{Opt}(M,\epsilon,T_M)-3\epsilon-\mathrm{Opt}(M,\epsilon,T_M)\delta)+\frac{\epsilon}{2}(t^*-B(0))}{t^*} \\ &= \frac{(t^*-B(0))(\mathrm{Opt}(M,\epsilon,T_M)-3\epsilon-\mathrm{Opt}(M,\epsilon,T_M)\delta)+\frac{\epsilon}{2}(B(0)(\frac{2R^*_{\max}}{\epsilon}+1)-B(0))}{t^*} \\ &= \frac{(t^*-B(0))(\mathrm{Opt}(M,\epsilon,T_M)-3\epsilon-\mathrm{Opt}(M,\epsilon,T_M)\delta)+B(0)R^*_{\max}}{t^*} \\ &\geq \frac{(t^*-B(0))(\mathrm{Opt}(M,\epsilon,T_M)-3\epsilon-\mathrm{Opt}(M,\epsilon,T_M)\delta)+B(0)\mathrm{Opt}(M,\epsilon,T_M)}{t^*} \\ &\geq \frac{t^*(\mathrm{Opt}(M,\epsilon,T_M)-3\epsilon-\mathrm{Opt}(M,\epsilon,T_M)\delta)}{t^*} \\ &= \mathrm{Opt}(M,\epsilon,T_M) - 3\epsilon - \mathrm{Opt}(M,\epsilon,T_M)\delta. \end{aligned}$$

∎

APPENDIX B

PROOFS FOR THEOREMS IN CHAPTER 3

In this chapter, we provide proofs of Theorems 3.2.2 and 3.2.1. We start with Theorem 3.2.2. We repeat the statement of the theorem for the reader's convenience.

B.1 Proof for Theorem 3.2.2

Theorem 3.2.2: *Using an exploration method where $D_i(1,t) \geq \beta$ for all $i, t > 0$ where $\beta \in (0,1)$ is a constant, for any $\alpha > 0$ and $0 < \delta < 1$, the robot can obtain an α-optimal policy to M_∞ with probability at least $1-\delta$ in time polynomial in l, $|A_l|$, $|S_l|$, $1/\beta$, $1/\alpha$, $1/\delta$, $R_{\max}$ and T^l, where l is the smallest i such that the optimal policy for M'_i is $(\alpha/2)$-optimal to M_∞, $R^l_{\max}$ is the maximum reward (that a transition can obtain) in M'_l, and T^l is the ϵ-return mixing time for M'_l.*

Proof: Since $D_i(1,t) \geq \beta$ for $i \geq 1$, for all levels $i \geq 1$, we have $\Psi(t) \geq \beta \cdot t \geq \beta \ln(t)$ for all $t \geq 1$. By Theorem 2.5.5, for each level i, with probability at least $1-\delta$, we can obtain a policy π_i that is $(\alpha/2)$-optimal for M'_i in time polynomial in $|A_i|$, $|S_i|$, $1/\beta$, $2/\alpha$, $1/\delta$, $R^i_{\max}$, and T^i using the URMAX algorithm, where $R^i_{\max}$ is the maximum reward in M'_i and T^i is the $(\alpha/2)$-return mixing time for M'_i.

However, we are not interested in obtaining a near-optimal policy for M'_i; we want a near-optimal policy for M_∞. Let l be the smallest i such that the optimal policy for M'_i is $(\alpha/2)$-optimal for M_∞. Such a level l must exist, due to the continuity of the reward and transition functions of M_∞. For suppose that π is the optimal policy in M_∞. By continuity, there exists a

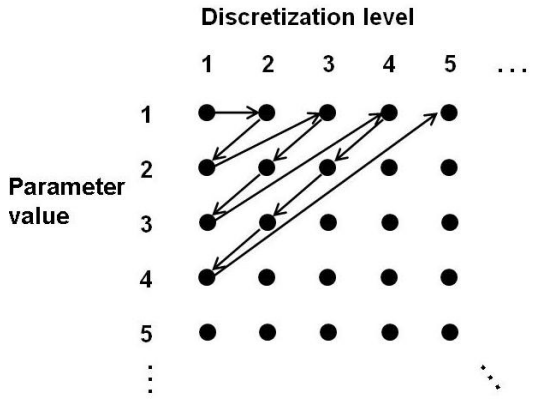

Figure B.1: The diagonal execution of URMAX.

level l and a policy π' at level l such that the expected reward of π' is within $\alpha/2$ of that of π. But then the optimal policy at level l must have expected reward within $\alpha/2$ of that of π.[1] Since π_l is $(\alpha/2)$-optimal for M'_l, and the optimal policy for M'_l is $(\alpha/2)$-optimal for M_∞, π_l must be α-optimal for M_∞.

Thus, if we knew l and the values of all the relevant parameters, then by running URMAX at each level from 1 to l, we could obtain an α-optimal policy for M_∞ in time polynomial in l, $|S_l|$, $1/k$, $2/\epsilon$, $1/\delta$, $R^j_{\max}$, $|A_j|$ and T^j for all $j \in [1, l]$. Note that we include $|A_j|$ and T^j for all $j \in [1, l]$ here. This is because we did not assume each discretization level $j+1$ is a refinement of the discretization level j; thus, it could happen that $R^l_{\max} < R^j_{\max}$ for some $j < l$, or $|A_l| < |A_j|$ for some $j < l$, or that $T^l < T^j$ for some $j < l$. However, we show below that we actually need only $R^l_{\max}$, $|A_l|$, and T^l (instead of all $R^j_{\max}$, $|A_j|$, and T^j).

The problem with the approach described in the previous paragraph is that we do not know l nor the values $|S_l|$, $|A_j|$, T^j and $R^j_{\max}$ for $j \in [1, l]$. We solve this problem just as how we solved

[1] Actually, there may not be an optimal policy in M_∞. That is, there may be a reward γ such that no policy in M_∞ has a reward of γ or greater, and a sequence of policies $\pi^1, \pi^2, \ldots$ such that the expected reward of the π^j approaches γ, although no policy in the sequence actually has expected reward γ. But essentially the same argument still applies: We take a policy π in M_∞ that has a reward greater than $\gamma - \alpha/4$ and and choose a level l that has a policy approximating π within $\alpha/4$.

the analogous problem when running the URMAX algorithm (see Chapter 2): we diagonalize. Specifically, we start running URMAX at discretization level 1 under the assumption that the parameters ($|A_1|$, $|S_1|$, $R^1_{\max}$, $|S_1|$) all have value 1. level 2 with the parameters set to 1, and run URMAX at level 1 with parameters set to 2; and so on. The process is described in Figure B.1. A similar approach is used in Chapter 2 to increase the value of unknown parameters. We call running URMAX at a specific particular discretization level using a particular setting of the parameters an *iteration* of URMAX. For example, running URMAX at discretization level 3 with the parameters set to 4 is one iteration of URMAX.

To deal with the fact that we do not know l, we always keep track of the current candidate for best policy. (That is, the policy that is optimal given the current set of assumptions about values of the parameters, given the actions that have been discovered and our current estimate of the transition probabilities.) At the end of each iteration of URMAX we run the current candidate optimal policy a sufficient number of times so as to guarantee that the average payoff of URMAX is close to optimal, if the current candidate optimal policy is indeed α-optimal.[2] Eventually, URMAX will reach a stage where it is exploring discretization level l using values for the parameters $|A_l|$, $|S_l|$, $R^l_{\max}$ and T^l that are at least as high as the actual values. At that point, the candidate for optimal policy is almost certain to be α-optimal for M_∞. (Recall that l is defined to be the discretization level at which the optimal policy is $\alpha/2$-optimal for M_∞.) After this point, we always run a policy that is at least α-optimal to M_∞. Note that this happens even if we do not know the value of l or any of the relevant parameters.

Thus, although URMAX runs forever, from some point in time it has discovered an α-optimal algorithm. Moreover, due to the "diagonal" manner in which URMAX is run, the candidate optimal policy is (with probability $1 - \delta$) α-optimal after time polynomial in l, $|A_l|$, $|S_l|$, T^l, $R^l_{\max}$, k, $1/\delta$, and $2/\alpha$. From that point on, we are guaranteed to run policies that are

[2]The number of times that we need to run the policy is computed in Chapter 2.

α-optimal for M_∞. ∎

B.2 Proof for Theorem 3.2.1

Theorem 3.2.1: *Using brute-force exploration, for any* $\alpha > 0$ *and* $0 < \delta < 1$*, a DM can find an* α*-optimal policy in* M_∞ *with probability at least* $1 - \delta$ *in time polynomial in* l, $|A'_l|$, $|S_l|$, $1/\alpha$, $1/\delta$, $R^l_{\max}$ *and* T^l*, where* l *is the least* i *such that the optimal policy for* M'_i *is* $(\alpha/2)$*-optimal for* M_∞, $R^l_{\max}$ *is the maximum reward (that a transition can obtain) in* M'_l*, and* T^l *is the* ϵ*-return mixing time for* M'_l.

Proof: At any discretization level, there are only finitely many possible actions. Since the brute force exploration examines all possible actions at each level, it is guaranteed to find all useful actions, and thus the near-optimal policy for that level.

We apply the URMAX algorithm diagonally as described in the Proof for Theorem 3.2.2, so that sooner or later, we will reach the discretization level i such that the optimal policy for M'_i is ϵ-close to P^*, and run URMAX in that level with parameters values no less than the real values of $|A_i|$, $|S_i|$, T_i and $R^i_{\max}$. Thus, we are guaranteed to obtain an ϵ-near-optimal policy. ∎

BIBLIOGRAPHY

[1] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proc. 22nd Int. Conf. on Machine Learning (ICML '05)*, pages 1–8, 2005.

[2] A. Antos and R. Munos. Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems 20 (Proc. of NIPS 2007)*, pages 9–16, 2007.

[3] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957.

[4] R. I. Brafman and M. Tennenholtz. R-MAX: A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[5] D. Brown, L. Cantillo, and G. Webster. Nasa confirms evidence that liquid water flows on today?s mars, 2015.

[6] D. Budden, J. Walker, M. Flannery, and A. Mendes. Probabilistic gradient ascent with applications to bipedal robot locomotion. In *Australasian Conference on Robotics and Automation (ACRA 2013)*, pages 37–45, 2013.

[7] M.-Y. Cheng and C.-S. Lin. Genetic algorithm for control design of biped locomotion. *Journal of Robotic Systems*, 14(5):365–373, 1997.

[8] S. Chernova and M. Veloso. An evolutionary approach to gait learning for four-legged robots. In *Proc. International Conference on Intelligent Robots and Systems (IROS 2004) Vol. 3*, pages 2562 – 2567, 2004.

[9] T. Dean, K. Kim, and R. Givan. Solving stochastic planning problems with large state and action spaces. In *Proc. 4th Int. Conf. on Artificial Intelligence Planning Systems*, pages 102–110, 1998.

[10] R. Dearden, N. Friedman, and D. Andre. Model based bayesian exploration. In *Proc. 15th Conf. on Uncertainty in AI (UAI '99)*, pages 150–159, 1999.

[11] J. DeVore. *Probability and Statistics for Engineering and The Sciences*. Duxbury Press, 1995.

[12] Y. Feinberg. Subjective reasoning—games with unawareness. Technical Report Research Paper Series #1875, Stanford Graduate School of Business, 2004.

[13] Y. Feinberg. Games with unawareness. http://www.stanford.edu/ yossi/Files/Games With Unawareness.pdf, 2009.

[14] Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous Markov decision problems. In *In Proc. 20th Conf. on Uncertainty in Artificial Intelligence*, pages 154–161, 2004.

[15] A. Goel, S. Khanna, and B. Null. The ratio index for budgeted learning, with applications. In *Proc. 9th Symp. on Discrete Algorithms (SODA '09)*, pages 18–27, 2009.

[16] J. B. Gonalves and D. E. Zampieri. An integrated control for a biped walking robot. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 28:453 – 460, 12 2006.

[17] C. Guestrin, R. Patrascu, and D. Schuurmans. Algorithm-directed exploration for model-based reinforcement learning in factored mdps. In *Proc. International Conference on Machine Learning (ICML 2002)*, pages 235–242, 2002.

[18] S. Guha and K. Munagala. Approximation algorithms for budgeted learning problems. In *Proc. 39th Symp. on Theory of Computing (STOC '07)*, pages 104–113, 2007.

[19] J. Y. Halpern and L. C. Rêgo. Extensive games with possibly unaware players. In *Proc. Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 744–751, 2006. Full version available at arxiv.org/abs/0704.2014.

[20] J. Y. Halpern, N. Rong, and A. Saxena. Mdps with unawareness. In *Proc. 26th Conf. on Uncertainty in AI (UAI'10)*, 2010.

[21] J. Y. Halpern, N. Rong, and A. Saxena. MDPs with unawareness. *CoRR*, abs/1006.2204, 2010.

[22] Y. Hasegawa, T. Arakawa, and T. Fukuda. Trajectory generation for biped locomotion robot. *Mechatronics*, 10:67–89, 2000.

[23] M. Hauskrecht, N. Meuleau, L.P. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Proc. 14th Conf. on Uncertainty in AI (UAI '98)*, pages 220–229, 1998.

[24] A. Heifetz, M. Meier, and B. Schipper. Unawareness, beliefs and games. In *Theoretical Aspects of Rationality and Knowledge: Proc. Eleventh Conference (TARK 2007)*, pages 183–192, 2007. Full paper available at www.econ.ucdavis.edu/faculty/schipper/unawprob.pdf.

[25] A. Hordijk and F. A. Van der Duyn Schouten. Discretization and weak convergence in Markov decision drift processes. *Mathematics of Operations Research*, 9:112–141, 1984.

[26] R.A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[27] Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie. Planning walking patterns for a biped robot. *IEEE Transactions on Robotics and Automation*, 17:280–289, 2001.

[28] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Proc. International Conference on Robotics and Automation (ICRA 2003), Vol. 2*, pages 1620 – 1626, 2003.

[29] S. Kakade, M. Kearns, and J. Langford. Exploration in metric state spaces. In *Proc. 20th Int. Conf. on Machine Learning (ICML '03)*, pages 306–312, 2003.

[30] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *Proc. Sixteenth Int. Joint Conf. on Artificial Intelligence (IJCAI '99)*, pages 740–747, 1999.

[31] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[32] H. K. Khalil. *Nonlinear Systems*. Prentice-Hall, 2002.

[33] J. Kim, I. Park, and J. Oh. Walking control algorithm of biped humanoid robot on uneven and inclined floor. *Journal of Intelligent and Robotic Systems*, 48(4):457–484, 2007.

[34] M. S. Kim and W. Uther. Automatic gait optimisation for quadruped robots. In *Proc. Australasian Conference on Robotics and Automation (ACRA)*, 2003.

[35] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 2004)*, pages 2619–2624, 2004.

[36] M.L. Littman, T.L. Dean, and L.P. Kaelbling. On the complexity of solving Markov decision problems. In *Proc. 11th Conf. on Uncertainty in AI (UAI '95)*, pages 394–402, 1995.

[37] D. J. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naive-Bayes classifiers. In *Proc. 19th Conf. on Uncertainty in AI (UAI'03)*, pages 378–385, 2003.

[38] O. Madani, D.J. Lizotte, and R. Greiner. Active model selection. In *Proc. 20th Conf. on Uncertainty in AI (UAI '04)*, pages 357–365, 2004.

[39] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. Preprint, http://www.eecs.berkeley.edu/ igor.mordatch/darwin/paper.pdf, 2016.

[40] G. Parker and W. Tarimo. Using cyclic genetic algorithms to learn gaits for an actual quadruped robot. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011)*, pages 1938 – 1943, 2011.

[41] G. B. Parker, W. T. Tarimo, and M. Cantor. Quadruped gait learning using cyclic genetic algorithms. In *Proc. IEEE Congress on Evolutionary Computation (CEC 2011)*, pages 1529 – 1534, 2011.

[42] H. Picado, M. Gestal, N. Lau, L. P. Reis, and A. M. Tome. Automatic generation of biped walk behavior using genetic algorithms. In *Proc. 10th International Work-Conference*

on Artificial Neural Networks, pages 805–812, 2009.

[43] M. L. Puterman. *Markov Decision Processes*. John Wiley and Sons, Inc., 1994.

[44] E. Rachelson, F. Garcia, and P. Fabiani. Extending the bellman equation for MDPs to continuous actions and continuous time in the discounted case. In *10th Int. Symp. on Artificial Intelligence and Mathematics*, 2008.

[45] N. Rong and J. Y. Halpern. Budgeted learning with unawareness. Unpublished manuscript, 2016.

[46] N. Rong, J. Y. Halpern, and A. Saxena. Mdps with unawareness in robotics. Unpublished manuscript, 2016.

[47] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[48] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent ASIMO: system overview and integration. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, volume 3, pages 2478–2483, 2002.

[49] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. In *Proc. 31st International Conference on Machine Learning (ICML '15)*, pages 1889–1897, 2015.

[50] S. P. Soundararaj, A. Sujeeth, and A. Saxena. Autonomous indoor helicopter flight using a single onboard camera. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 5307–5314, 2009.

[51] J. Strom, G. Slavov, and E. Chown. Omnidirectional walking using ZMP and preview control for the nao humanoid robot. In *RoboCup 2009: Robot Soccer World Cup XIII*, pages 378–389, 2010.

[52] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[53] R. Tedrake, T. W. Zhang, and H. S. Seung. Learning to walk in 20 minutes. In *Proc. Fourteenth Yale Workshop on Adaptive and Learning Systems*, 2005.

[54] F. Xue, X. Chen, J. Liu, and D. Nardi. Real time biped walking gait pattern generator for a real robot. In T. Röfer, N. M. Mayer, J. Savage, and U. Saranlı, editors, *RoboCup 2011: Robot Soccer World Cup XV*, pages 210–221. Springer, 2012.

[55] A. S. Zinober. Deterministic control of uncertain systems. In *Proc. Int. Conf. on Control and Applications (ICCON '89)*, pages 645–650, 1989.

[56] V. Zykov, J. Bongard, and H. Lipson. Evolving dynamic gaits on a physical robot. In *Proc. Genetic and Evolutionary Computation Conference, Late Breaking Paper (GECCO'04)*, 2004.