The Dissertation Committee for Shivaram Kalyanakrishnan
certifies that this is the approved version of the following dissertation:

Learning Methods for Sequential Decision Making with Imperfect Representations

Learning Methods for Sequential Decision Making with Imperfect Representations

by

Shivaram Kalyanakrishnan, B.Tech.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2011

Acknowledgments

This dissertation marks the culmination of a defining period of my life. I arrived in America in August 2004; the several years I have spent thereafter as a graduate student at the University of Texas have seen me mature as a person and find my feet as a researcher. The most prized possession I have acquired during this period is a deep and enduring respect for the academic process: for the purity it represents in the pursuit of knowledge. I did not have to scour the far corners of the earth to find such profound enrichment, I only had to be Peter Stone's student. Words will scarcely convey the regard and gratitude I reserve Peter Stone for the countless ways in which he has influenced me and nurtured my growth. As my advisor, he has provided me invaluable guidance and unwavering support. Without the freedom and confidence he gave me, without the honesty, trust, and respect that bound us, surely I could not have undertaken this journey with the zeal I have, nor derived fulfillment from its every moment. The completion of the journey reminds me how blessed I am.

I thank the other members my thesis committee—Raymond J. Mooney, Risto Miikkulainen, Dana H. Ballard, and Ronald Parr—whose varied recommendations have significantly benefitted this dissertation. In particular, Chapter 4 has materialized primarily in response to suggestions the committee made during my Ph.D. proposal.

I was fortunate to spend my early days as a researcher under the patient and encouraging mentorship of Yaxin Liu. It was from him that I grasped the meaning of a Ph.D., and learned the various skills it demanded. For the worthy role model that he has been, I will always be in his debt.

My research has been shaped by a number of telling forces. The person whose technical ideas have most influenced mine, and have indeed sown the seeds for this thesis, is Shimon Whiteson, with whom I am fortunate to have shared an office and several illuminating conversations. My perspectives on research were significantly widened when I visited the Honda Research Institute for an internship in 2008. My mentor there, Ambarish Goswami, has been an inspiring researcher and a patient collaborator. Only in the later stages of my Ph.D. did I venture into theoretical research topics. Any progress I have subsequently made therein owes to meticulous tutorship I have received from Ambuj Tewari. A substantial portion of Chapter 5 is drawn from our collaboration. Curiously, the greatest joys and sorrows I have experienced as a graduate student have been on the soccer pitch (a simulated one, albeit). I am grateful to Peter Stone for introducing me to RoboCup, and especially for shepherding me through sustained periods of failure at the competitions. For their hard work and the change in fortunes they have ushered in, colleagues in the UT Austin Villa team have my wholehearted appreciation. For several years, I have organized and attended the UTCS Reinforcement Learning Reading Group. Not only has this forum advanced my technical knowledge, it has taught me the important lesson that talking and listening are vital for research to thrive.

Without all these fertilizing influences over the years, my work would have remained much the poorer. For all their contributions on my behalf, my colleagues have my most sincere gratitude. Specifically, thanks are due to my many collaborators: Peter Auer, Yinon Bentor, Ian Fasel, Ambarish Goswami, Todd Hester, Yaxin Liu, Patrick MacAlpine, Michael Quinlan, Peter

Stone, Ambuj Tewari, and Daniel Urieli. Exchanges with Doran Chakraborty, Matthew Hausknecht, W. Bradley Knox, and Matthew E. Taylor also resonate through the pages of this dissertation. I am most grateful to Stacy Miller for managing my administrative affairs.

Could I even have dreamt of pursuing a Ph.D. if not for the nurture and inspiration I received from my family and teachers right from an early age? This thesis is my offering to my parents, Meenakshi and R. Kalyanakrishnan. It is also my tribute to my dear sister, Charukesi Saishankar. At Vidya Mandir Adyar, my English teacher, Nirmala Jayaraj, and my mathematics teacher, R. Rajendran, command my utmost respect and gratitude. At IIT Madras, I am indebted to Deepak Khemani, who introduced me to machine learning; and to the late Dilip Veeraraghavan, who taught me Indian history and much more.

Spending seven years in a foreign land (and coping with Bellman's equations on top of it) is a rather unnatural exercise, and it is inconceivable that I could have endured its rigors without the uplifting presence of friends. I specially cherish the friendships I formed while in Austin with Shweta Agrawal, Meghana Deodhar, and Juhyun Lee. I have also benefitted from the company of long-time roommate Aditya Gopalan. Without doubt, the contents of the forthcoming pages of this dissertation are far outweighed by the extent America has educated me—through her national parks and her coffee shops! Austin has been a welcoming home, and this dissertation may well be construed as my attempt to Keep Austin Weird.

Shivaram Kalyanakrishnan
Austin, November 2011

Learning Methods for Sequential Decision Making with Imperfect Representations

Shivaram Kalyanakrishnan, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Peter Stone

Sequential decision making from experience, or reinforcement learning (RL), is a paradigm that is well-suited for agents seeking to optimize long-term gain as they carry out sensing, decision, and action in an unknown environment. RL tasks are commonly formulated as Markov Decision Problems (MDPs). Learning in finite MDPs enjoys several desirable properties, such as convergence, sample-efficiency, and the ability to realize optimal behavior. Key to achieving these properties is access to a perfect representation, under which the state and action sets of the MDP can be enumerated. Unfortunately, RL tasks encountered in the real world commonly suffer from state aliasing, and nearly always they demand generalization. As a consequence, learning in practice invariably amounts to learning with imperfect representations. In this dissertation, we examine the effect of imperfect representations on different classes of learning methods, and introduce techniques to improve their practical performance. We make four main contributions.

First we introduce “parameterized learning problems”, a novel experimental methodology facilitating the systematic control of representational aspects such as state aliasing and generalization. Applying this methodology, we compare the class of on-line value function-based (VF) methods with the class of policy search (PS) methods. Results indicate clear patterns in the effects of representation on these classes of methods. Our second contribution is a deeper analysis of the limits imposed by representations on VF methods; specifically we provide a plausible explanation for the relatively poor performance of these methods on Tetris, the popular video game.

The third major contribution of this dissertation is a formal study of the “subset selection” problem in multi-armed bandits. This problem, which directly affects the sample-efficiency of several commonly-used PS methods, also finds application in areas as diverse as industrial engineering and on-line advertising. We present new algorithms for subset selection and bound their performance under different evaluation criteria. Under a PAC setting, our sample complexity bounds indeed improve upon existing ones. As its fourth contribution, this dissertation introduces two hybrid learning architectures for combining the strengths of VF and PS methods. Under one architecture, these methods are applied in sequence; under the other, they are applied to separate components of a compound task. We demonstrate the effectiveness of these methods on a complex simulation of robot soccer.

In sum, this dissertation makes philosophical, analytical, and methodological contributions towards the development of robust and automated learning methods for sequential decision making with imperfect representations.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In this introductory chapter, we describe the motivations and foundational assumptions underlying this thesis. We argue that representations used for sequential decision making are invariably imperfect in practice, and therefore, learning methods must aim to maximize rewards in spite of them. We proceed to summarize the contributions of the thesis and to provide a guide to the subsequent chapters.

Sequential decision making from experience, or reinforcement learning (RL) (Sutton and Barto, 1998), is an ideal problem formulation for agents seeking to optimize long-term gains as they carry out sensing, decision, and action in an unknown environment.[1] RL tasks are commonly formulated as Markov Decision Problems (MDPs). The analysis of learning and planning algorithms in MDPs has benefited immensely from a strong theoretical framework that has been developed over the years. The cornerstone of this framework is the *value function* of the MDP (Bellman, 1957), which encapsulates the long-term utilities of decisions.

[1]Throughout this thesis, we use the term "reinforcement learning" to refer to the *problem* of sequential decision making from experience, and not to any specific class of learning methods addressing that problem. Such an interpretation is consistent with Sutton and Barto (1998, see chapter 3).

Value functions naturally give rise to control policies. In the context of *finite* MDPs, several learning algorithms provably converge to *optimal* policies (Watkins and Dayan, 1992; Singh et al., 2000), and indeed while being efficient in terms of samples, computation, and memory (Strehl et al., 2006; Szita and Szepesvári, 2010a). Yet, in the RL tasks we face in the real world, learning seldom proceeds with the same exactness and efficiency that the aforementioned results guarantee. Not only do the traditional objectives of convergence and optimality not apply to a vast majority of these tasks, in many of them, we cannot even ascertain the best performance that can be achieved, or how much training is necessary to achieve satisfactory behavior. What, then, is the chief difference between the mainstream theory of RL, as typified within methods seeking to learn value functions, and its practice? In this dissertation, we argue that the difference is *representation* (as of the policy, value function, or model), which is invariably *imperfect* in practice.

1.1 Imperfect Representations

The MDP formulation elegantly captures the very crux of sequential decision making: the quest for taking actions that will bring long-term benefit. However, the assumption that a learner can enumerate the states or state-action pairs in an MDP, such as for associating utilities or distributions with them, is rarely true in practice. In other words, the classical "tabular" representation that is used for most theoretical analysis—which is "perfect" in its capacity to enumerate states or state-action pairs—is most often inapplicable in practice. For illustration consider Table 1.1, which lists a variety of

RL applications from the last two decades.[2] The table draws our attention to two major factors that introduce "imperfection" in the representations agents use in practice: state aliasing and generalization.

A majority of the examples listed in Table 1.1 are affected by state aliasing (or *partial observability*). In complex systems such as stock markets (Nevmyvaka et al., 2006), physical environments (Gomez and Miikkulainen, 2003), and cellular tissue (Guez et al., 2008), available measurements seldom suffice to capture all the information that can affect decision making. Nearly every agent embedded in the real world (Kwok and Fox, 2004; Ng et al., 2004; Lee et al., 2006) receives noisy sensory information. The inadequacy of the sensory signal in identifying the underlying system state hinders the assumption of a Markovian interaction between the agent and the environment, on which the theoretical guarantees associated with most learning methods rely. Whereas coping with state aliasing in a systematic manner is a well-studied problem, it is yet to scale to complex tasks with high-dimensional, continuous state spaces (Chrisman, 1992; Cassandra et al., 1994; Bakker et al., 2003; Pineau et al., 2006).

Of the 25 applications listed in Table 1.1, 15 involve continuous state spaces, which force the use of generalization in the representation. General-

[2]This author has populated Table 1.1 by browsing several relevant journals, conference proceedings and citations contained therein. For every entry in the table, this author has judged that the application itself, rather than the RL algorithm employed, is the primary focus of the relevant publication. By no means comprehensive, Table 1.1 at least represents applications from a wide spectrum of domains. Other independently-compiled surveys of sequential decision making applications corroborate the observations we proceed to draw based on Table 1.1. Langley and Pendrith (1998) describe several RL applications presented at a symposium organized around the topic; Szepesvári lists numerous applications from the control and approximate dynamic programming literature at this URL: `http://www.ualberta.ca/~szepesva/RESEARCH/RLApplications.html`.

Table 1.1: Characterization of representative real-world applications of RL. "Policy structure" describes the underlying representation from which the policy is derived. A "neural network" representation is non-linear, incorporating at least one hidden layer of units. Under tile coding and Gaussian Process, the number of "features" indicates the number of state variables, rather than the number of individual tiles or basis functions.

Task	**State aliasing**	**State space**	**Policy structure (Number of features)**
Backgammon (Tesauro, 1992)	Absent	Discrete	Neural network (198)
Job-shop scheduling (Zhang and Dietterich, 1995)	Absent	Discrete	Neural network (20)
Tetris (Bertsekas and Tsitsiklis, 1996)	Absent	Discrete	Linear (21)
Elevator dispatching (Crites and Barto, 1996)	Present	Continuous	Neural network (46)
Acrobot control (Sutton, 1996)	Absent	Continuous	Tile coding (4)
Dynamic channel allocation (Singh and Bertsekas, 1997)	Absent	Discrete	Linear (100's)
Active guidance of finless rocket (Gomez and Miikkulainen, 2003)	Present	Continuous	Neural network (14)
Fast quadrupedal locomotion (Kohl and Stone, 2004)	Present	Continuous	Parameterized policy (12)
Robot sensing strategy (Kwok and Fox, 2004)	Present	Continuous	Linear (36)
Helicopter control (Ng et al., 2004)	Present	Continuous	Neural network (10)
Dynamic bipedal locomotion (Tedrake et al., 2004)	Present	Continuous	Feedback control policy (2)
Adaptive job routing/scheduling (Whiteson and Stone, 2004)	Present	Discrete	Tabular (4)

(Table continued on next page.)

Table 1.1 – continued

Task	**State aliasing**	**State space**	**Policy structure (Number of features)**
Robot soccer Keepaway (Stone et al., 2005)	Present	Continuous	Tile coding (13)
Robot obstacle negotiation (Lee et al., 2006)	Present	Continuous	Linear (10)
Optimized trade execution (Nevmyvaka et al., 2006)	Present	Discrete	Tabular (2-5)
Blimp control (Rottmann et al., 2007)	Present	Continuous	Gaussian Process (2)
9×9 Go (Silver et al., 2007)	Absent	Discrete	Linear ($\approx$1.5 million)
Ms. Pac-Man (Szita and Lőrincz, 2007)	Absent	Discrete	Rule list (10)
Autonomic resource allocation (Tesauro et al., 2007)	Present	Continuous	Neural network (2)
General game playing (Finnsson and Björnsson, 2008)	Absent	Discrete	Tabular (over part of state space)
Soccer opponent "hassling" (Gabel et al., 2009)	Present	Continuous	Neural network (9)
Adaptive epilepsy treatment (Guez et al., 2008)	Present	Continuous	Extremely randomized trees (114)
Computer memory scheduling (İpek et al., 2008)	Absent	Discrete	Tile coding (6)
Motor skills (Peters and Schaal, 2008b)	Present	Continuous	Motor primitive coefficients (100's)
Combustion Control (Hansen et al., 2009)	Present	Continuous	Parameterized policy (2-3)

ization refers to the phenomenon wherein a parameter (or a "degree of freedom") in the representation influences the choice of action from more than one state. In the context of value function-based learning methods, generalization is typically given attention as the problem of "function approximation". Even among the ten applications in Table 1.1 that have discrete state spaces, seven use some form of generalization, as their state spaces are too large to be enumerated. The use of generalization negates the theoretical guarantees of achieving optimal behavior. Often the generalization scheme used is not capable of representing an optimal policy for a task; even when it is, typically it cannot be guaranteed that a learning algorithm will discover such a policy. Although there do exist convergence guarantees for certain algorithms that use linear function approximation schemes (Konda and Tsitsiklis, 2003; Perkins and Precup, 2003; Maei et al., 2010), such guarantees seldom come with effective lower bounds on the values of the learned policies. Further, convergence results rarely extend to situations in which non-linear representations such as neural networks are used to approximate the value function; yet non-linear representations are used commonly in practice, as apparent from Table 1.1.

In summary, we may conclude from Table 1.1 that hardly is it accurate or even feasible to view practical sequential decision making tasks as finite MDPs with enumerable states. In practice, state aliasing and generalization introduce inevitable imperfections in the representation with which an agent is constrained to learn.

Before continuing, we might reflect that imperfect representations are not peculiar to sequential decision making, but in fact they ail all walks of artificial intelligence and machine learning. As a *positive* example—demonstrating that the capability of representations often determines the overall effectiveness of a methodology—consider how the Scale Invariant Feature Transform (Lowe, 2004) (commonly identified through its practical utility: “SIFT features”) has effected a seismic shift in the capacity to solve problems in computer vision. Such successes are the exception: predominantly it is far from clear, in problems of any substantial complexity, how best to constrain and bias the relationship between input and output variables.

1.2 Learning with Imperfect Representations

An agent that comes endowed with an imperfect representation could still actively improve its representation as it interacts with the environment. Conceptually we can imagine the agent’s memory as divided into two portions: one for dealing with representational aspects (such as for disambiguating aliased states, identifying features, building control hierarchies, and so on), and the other devoted to learning (typically by updating a real-valued vector of weights). Such a view is not universally accurate: under certain architectures, the elements of representation discovery and learning are hard to separate. Nevertheless, in a majority of cases, a meaningful distinction can be drawn between the two aspects, and indeed this distinction is made quite commonly in the literature (Van Roy, 1998, see page 13). Likewise for the purposes of this thesis, we assume that an agent’s internal representation is a template for mapping its raw sensory input to its actions; the learning procedure updates real-valued parameters in this template to determine the precise mapping.

This author’s overarching view is that representation discovery and learning must be interactive processes that complement each other towards an agent’s ultimate benefit, as illustrated schematically in Figure 1.1. From a practical standpoint, that benefit is quite well-defined: to accrue high rewards in a given amount of time on a specified task. Unfortunately the process of representation discovery typically operates on a much slower time scale than learning. In practical applications such as those listed in Table 1.1, there is little hope that an agent can substantially improve its representational architecture in the time it interacts with the environment—typically a few hours or days on a robot or computer. In such cases, the apt assumption to make is that the agent must make do with the imperfect representation it has been provided.

The main philosophical choice this thesis advocates is first to acknowledge that over short time scales, representation discovery is often an intractable problem, and having done so, to explicitly consider devising learning methods

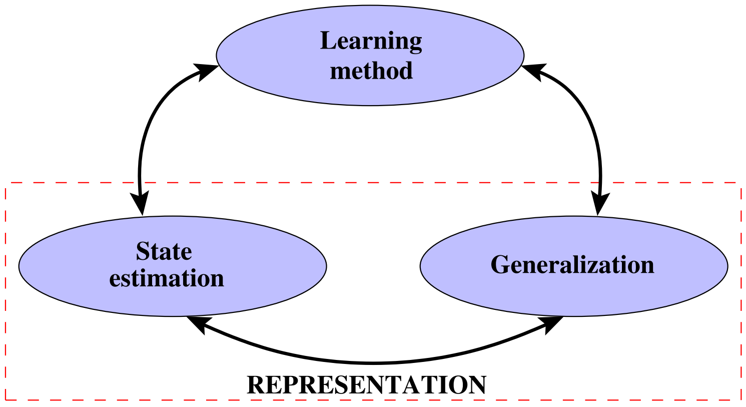


Figure 1.1: A schematic depiction of the conjunctive relationship between learning and representation discovery (itself comprising state estimation and generalization). The modules need to complement each other in order to achieve high long-term reward on a specified task. Representation discovery typically operates at a slower time scale than learning, and so is rendered infeasible in many practical tasks.

that would be effective when applied with imperfect representations. Specifically this dissertation examines the following question:

Thesis Question: How well do different learning methods for sequential decision making perform in the *presence* of state aliasing and generalization; can we develop methods that are both sample-efficient and capable of achieving high asymptotic performance in their presence?

The problem of sequential decision with imperfect representations is a general problem underlying a large section of the RL tasks we encounter in practice. In such tasks, our formulation calls for the learning module in Figure 1.1 to bear extra burden in working together with a representation module that is in general imperfect. This approach is in contrast with the common practice (such as evidenced in many of the examples listed in Table 1.1) whereby learning methods originally developed to work with perfect representations are applied unchanged in imperfect-representation settings.

In seeking answers to its motivating question, the main thrust of this dissertation is in understanding the nature of the dependence of various learning methods on observed state transitions. Solutions to sequential decision making problems are policies, which are mappings from states to actions. In order to learn policies, methods generalize based on a set of experiences that register the effects of actions an agent has taken from the states it has visited. Different RL methods generalize based on observed state transitions in different ways. For example, fully bootstrapping temporal difference (TD) learning methods such as Sarsa(0) (Rummery and Niranjan, 1994) use the estimated values of next states in order to estimate values of states, while Monte Carlo methods such as Sarsa(1) estimate state values based on samples of long-term reward. Policy search approaches such as NEAT (Stanley, 2004) and CMA-ES (Hansen, 2009) do not even estimate state values; rather, they directly evaluate policies.

It is natural to expect that the fewer the assumptions a method makes about associations between states (and between states and values), the less it will suffer due to state aliasing and generalization, both of which constrain the scope of functions that can be learned over the state space. On the other hand, exploiting the relationships between states occurring in a sequence is precisely the reason for the sample-efficiency of bootstrapping TD methods, which Monte Carlo and policy search methods are not likely to match. Thus, learning needs to cope with a tension between sample-efficiency on the one hand, and robustness towards imperfect representations on the other. Existing methods are not specifically designed to achieve satisfactory tradeoffs between these conflicting objectives: rather than hoping that they will in an incidental manner on a given problem, this thesis examines when and why different methods succeed, and how their strengths can be synthesized.

1.3 Contributions of Thesis

The contributions of this dissertation fall into three categories: (1) the conceptual formulation of learning with imperfect representations, (2) experimental evaluation and analysis of the relationship between different learning methods and representations, and (3) additions to learning methodology directly motivated by practical concerns of sample-efficiency and performance. Below is a summary of the contributions in these categories.

1.3.1 Conceptual Framework

As explained in Section 1.2, recall that the central principle behind this thesis is that learning methods must not rely on having perfect representations; rather, they must strive to maximize performance even when representations are imperfect. This novel formulation naturally engenders a unified view of various existing classes of learning methods against the backdrop of representation quality. The pragmatic sense of the formulation, coupled with the research it encourages towards unifying the strengths of contrasting learning approaches, signifies a new conceptual framework and a positive step towards scaling RL to increasingly complex applications.

1.3.2 Evaluation and Analysis

This thesis undertakes two studies to evaluate the relationships between representations and learning methods.

Experimental comparison of learning methods. We devise *parameterized learning problems* as a methodology to systematically control representational aspects such as state aliasing and generalization, and measure their effects on learning methods through targeted studies. Apart from providing precise control of the parameters that affect learning, parameterized learning problems enable benchmarking against optimal behavior; their relatively small sizes facilitate extensive experimentation.

Using parameterized learning problems, we compare two qualitatively distinct classes of algorithms for RL: on-line value function-based methods and policy search methods. Empirical comparisons among various methods within each of these classes project Sarsa(λ) (Rummery and Niranjan, 1994;

Sutton and Barto, 1998) and Q-learning(λ) (Watkins, 1989; Rummery, 1995; Peng and Williams, 1996), as the best performers among the former class, and CMA-ES (Hansen, 2009) as the winning candidate in the latter. Comparing Sarsa(λ) and CMA-ES further on relevant problem instances, our study highlights regions of the problem space favoring their contrasting approaches. In particular we find that Sarsa(λ) is able to learn quickly, reaching near-optimal performance when provided near-perfect representations; however, CMA-ES is able to outperform Sarsa(λ) when the quality of the generalization is downgraded. Short run-times for the experiments allow for an extensive search procedure that provides additional insights on relationships between method-specific parameters—such as eligibility traces, initial weights, and population sizes—and problem instances.

Future work might extend the experimental comparisons described above to other classes of learning methods: in particular, model-based and batch RL methods, actor-critic methods, and policy gradient techniques. A brief survey of these classes of learning methods is included in the thesis, but it exceeds the scope of the thesis to subject them to systematic empirical evaluation.

Effect of representations in Tetris. We follow our extensive comparative study with a more targeted analysis of the role of representations in determining the success of learning methods. Specifically we consider Tetris, the popular video game and RL benchmark. The literature shows that value function-based methods perform significantly worse on this task than policy search methods; yet the reasons for this shortcoming of value function-based methods have not been fully understood. We design experiments to test the hypothesis that the representation used for learning imposes fundamental limits on the performance of value function-based methods on the Tetris task. We find evidence in support of this hypothesis. In particular we demonstrate that policy iteration with an approximate architecture could actually lead to a degradation in performance on this task.

1.3.3 Practice-driven Learning Methodology

The experimental and analytical studies described in Section 1.3.2 provide a number of insights about the characteristics of different learning methods in the presence of imperfect representations, highlighting possibilities for tailoring learning methods to perform better in practice. We pursue two such possibilities, which have resulted in concrete algorithmic and methodological contributions. Below we briefly describe these elements of the "constructive" contribution of this dissertation.

Subset selection and efficient policy search. One of the observations resulting from our experimental study is that policy search methods, while they can achieve relatively high rewards even in the presence of poor representations, typically require a large number of samples to achieve their asymptotic performance. Can they be made more sample-efficient? A key step that affects the sample-efficiency of policy search methods such as CMA-ES and the cross-entropy method (de Boer et al., 2005) is "subset selection": from a population of candidate policies, selecting a subset of those with the highest fitness values, based on *sampling* the fitness of the policies (running Monte Carlo rollouts of episodes). Since fitness evaluations are noisy, policies must be evaluated multiple times in order to reliably select the best subset. How must we allocate fitness evaluations among the policies in order to obtain a good selection, while keeping the total number of fitness evaluations minimal?

We formalize the above subset selection problem using stochastic multi-armed bandits, where each arm corresponds to a real-valued random variable. We consider three relevant operational settings: probably approximately correct (PAC) selection, simple regret (SR) minimization, and cumulative regret (CR) minimization. Under all three settings, we generalize existing problem formulations that focus on the selection of a single arm (that is, $m = 1$) to instead deal with subsets of arms. Under the PAC setting, we provide algorithms for subset selection that improve existing sample complexity bounds when instantiated with $m = 1$. Under the SR and CR settings, our algorithms and regret bounds essentially generalize existing algorithms and bounds.

Subset selection is a fundamental problem in exploration; the subset-selection algorithms developed in this thesis apply to problems in a variety of areas, such as simulation, industrial engineering, and on-line advertising. Experimental results demonstrate that indeed they improve the sample-efficiency of policy search methods such as CMA-ES and the cross-entropy method.

Hybrid learning methods. In addition to our theoretical work on the subset selection problem, we present two case studies involving learning architectures that integrate the strengths of qualitatively different learning algorithms.

First, as a means to combine the sample-efficiency of Sarsa(λ) with the robustness of CMA-ES to imperfect representations, we devise a natural "sequencing" algorithm wherein Sarsa(λ) is run for a certain number of episodes, following which the learned weights are used to initialize a run of CMA-ES. Tests over a range of parameterized learning problems establish that the resulting algorithm performs at least as well as CMA-ES, and often performs better. A similar algorithm sequencing Sarsa(0) and the cross-entropy method performs better than either of those methods on the Keepaway soccer benchmark (Stone et al., 2005), where the policy is represented using a neural network.

As we encounter increasingly complex sequential decision making tasks, it is likely that the diversity of the challenges they pose can only be addressed by combining the strengths of different learning algorithms. We examine this aspect of learning in our second case study, which involves the Keepaway task. Whereas previous successful results in Keepaway have limited learning to an isolated, infrequent decision that amounts to a turn-taking behavior (passing), we expand the agents' learning capability to include a much more ubiquitous action (moving without the ball, or getting open), such that at any given time, multiple agents are executing learned behaviors simultaneously. We introduce a policy search method for learning "GetOpen", to complement the TD learning approach employed for learning "Pass". Empirical results indicate that the learned GetOpen policy matches the best hand-coded policy for this task, and outperforms the best policy found when Pass is learned. We demonstrate that Pass and GetOpen can be learned simultaneously to realize tightly-coupled soccer team behavior. The interleaved schedule employed for learning Pass and GetOpen can be viewed as an application of the "layered learning" paradigm (Stone, 1998) to high-level sequential decision making in a multiagent environment.

In summary, this thesis (1) defines a scope for learning in practical tasks, (2) applies novel experimental methodology towards characterizing the influence of representation quality on the performance of different learning methods, and (3) contributes new methods designed to improve the performance of existing ones. Application domains covered in the dissertation include Tetris and simulation robot soccer.

1.4 Summary of Chapters

Below is a summary of the remaining chapters in this dissertation. The dissertation can be divided into four logical parts, as indicated in Table 1.2.

Chapter 2 provides a background on sequential decision making, and specifically considers the effect of state aliasing and generalization on RL methods. The chapter builds on an argument that leads to a formal statement of the problem of learning in the presence of imperfect representations.

Chapter 3 introduces parameterized learning problems as a methodology for evaluating learning methods while systematically varying representational aspects. We employ this methodology to compare on-line value function-based methods with policy search methods across an extensive suite of problem instances.

Chapter 4 presents a more focused investigation on the role of representations in limiting the performance of value function-based methods on the Tetris task.

Chapter 5 is devoted to a formal study of the subset selection problem under the PAC, simple regret, and cumulative regret settings. Apart from novel theoretical results, we present evidence that efficient subset selection can indeed improve the performance of certain policy search methods.

Chapter 6 presents two case studies in which on-line value function-

Table 1.2: Organization of chapters.

Part	**Chapter**
Conceptual framework	2. Background and problem definition
Evaluation and analysis	3. Experimental comparison of learning methods 4. Limits of representation: an illustration
Practice-driven learning methodology	5. Subset selection and efficient policy search 6. Hybrid learning methods: two case studies
Discussion	7. Conclusion and future work

based methods and policy search methods are combined in innovative ways successfully tackle complex sequential decision making tasks. In particular both case studies are evaluated on the Keepaway robot soccer task.

Chapter 7 summarizes and concludes the thesis. The chapter also highlights several important areas for future research to extend the work presented in this thesis.

The dissertation does not have a separate chapter dedicated to related work. Instead, we discuss related work in detail in each of the individual chapters.

Chapter 2

Background and Problem Definition

This chapter presents a detailed description of the problem of sequential decision making with imperfect representations, which the remainder of the dissertation addresses through analysis and the design of learning methods. We begin with an account of learning in finite MDPs (Section 2.1), which is an idealized abstraction of sequential decision making. We then consider how the phenomena of state aliasing (Section 2.2) and generalization (Section 2.3) affect learning. Learning algorithms operating with the imperfect representations these factors induce are less amenable to theoretical analysis. We argue that the performance of learning algorithms can yet be evaluated precisely, and that the evaluation framework should inspire their design, too. This observation leads to a formal problem definition for sequential decision making with imperfect representations (Section 2.4).

Sequential decision making is traditionally formalized using the framework of Markov Decision Problems (MDPs). An MDP $M = (S, A, T, R, \gamma, D)$ comprises a set of states S and a set of actions A available from each state. A transition function T generates next states stochastically: the probability of reaching state s' by taking action a from state s is given by $T(s, a, s')$. A reward function R assigns numerical rewards to state transitions; $R(s, a, s')$ is the reward for reaching state s' by taking action a from state s. A (deterministic) policy $\pi : S \to A$ specifies the action a to take from a given state s; associated with π is its *action value function* $Q^\pi : S \times A \to \mathbb{R}$, which gives the expected long-term discounted reward for taking action a from state s. Q^π can be defined recursively as:

$$Q^\pi(s,a) = \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma Q^\pi(s', \pi(s')) \right], \forall s \in S, \forall a \in A,$$

where $\gamma \in [0,1)$ is the discount factor (γ can be set to 1 in episodic MDPs). The *value function* of the policy π, $V^\pi : S \to \mathbb{R}$, associates with every state s, the expected long-term discounted reward that will be accrued by starting at s and following π: it is given by $V^\pi(s) = Q^\pi(s, \pi(s))$. Provided a distribution D over S to probabilistically pick start states, the *value* of a policy π is given by $V(\pi) = \sum_{s \in S} D(s) V^\pi(s)$. Among the set of all policies Π applicable to the MDP M, an *optimal policy* π^* is one whose value is not exceeded by that of any other: that is, $V(\pi^*) = max_{\pi \in \Pi} V(\pi)$. The action value function corresponding to π^* is denoted Q^*, and the value function of π^* is denoted V^*. Note that there could exist multiple optimal policies π^*; however, the optimal action value function Q^* and the optimal value function V^* are unique in every MDP.

Figure 2.1 schematically depicts the interaction between an agent and its environment, formulated as an MDP. The agent, aware of its current state s_t, takes the action a_t prescribed by its policy π_t. The transition function T and the reward function R are embedded in the environment, which stochastically picks a next state s_{t+1} and generates the corresponding reward r_{t+1}. On receiving these signals, the agent resumes its interaction, and over time gathers a sequence of "experiences", which are transitions (or *samples*) of the form $(s_t, a_t, r_{t+1}, s_{t+1})$. The agent's objective is to update π_t such that over time, the value of π_t increases towards its maximum.

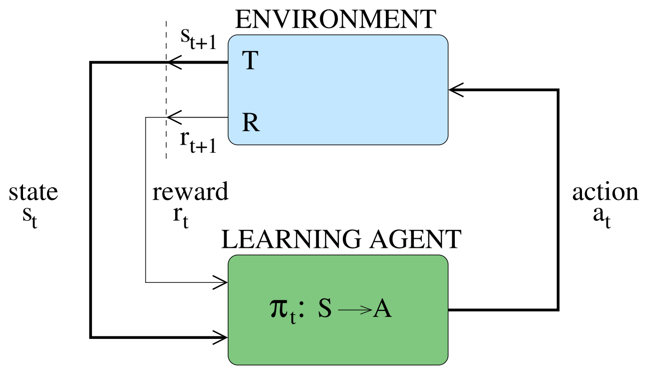


Figure 2.1: Schematic view of agent learning to perform sequential decision making from experience (figure adapted from the textbook by Sutton and Barto (1998, see chapter 3)).

2.1 Finite MDPs

A seminal result due to Watkins and Dayan (1992) establishes that if S and A are finite, an agent can learn the optimal action value function Q^*, provided it takes every action from every state infinitely often in the limit. In order to do so, the agent maintains a table with an entry for each state-action pair, to which it performs a "Q-learning" update every time the state is visited and the action taken. If the learning rate in the updates is annealed according to a certain schedule, the entries in the table converge to Q^* values. Acting greedily with respect to these converged action values induces optimal behavior; that is, $\pi^*(s) = argmax_{a \in A} Q^*(s,a), \forall s \in S$. Singh et al. (2000) derive a similar proof of convergence to the optimal action value function and optimal policy for the Sarsa(0) learning algorithm.

Subsequent work has provided learning algorithms that achieve near-optimal behavior after collecting a number of samples that is polynomial in the size of the state space ($|S|$) and the number of actions ($|A|$) (Kearns and

Singh, 2002; Brafman and Tennenholtz, 2003; Strehl and Littman, 2005; Szita and Szepesvári, 2010a); indeed such behavior can be achieved using a memory bounded in size by $O(|S||A|)$ (Strehl et al., 2006). Learning algorithms have also been devised for achieving near-optimal "regret" (Jaksch et al., 2010); in other words, the losses they accrue *while learning* are essentially minimal. In short, an agent interacting with a discrete, finite MDP with fully observable states can achieve optimal behavior in the limit, while being efficient in terms of samples, memory, and regret.

2.2 State Aliasing

In an MDP, the state signal summarizes all the information necessary to predict the future. However, in a predominant number of sequential decision making tasks encountered in practice, the information relayed to the agent through its sensors at every decision cycle only provides an occluded view of the underlying state. Thus, it becomes necessary to distinguish the agent's *observation* at any instant of time from the environment's *state* at the same instant. For example, in a soccer game, the state of the system might be fixed by the positions and velocities of the players and the ball. However, a player with a restricted field of view might not always perceive all the objects on the playing field. Even if the player does, a single visual snapshot would not identify the velocities of the other players and the ball. Further, a camera with finite resolution and noisy pixels would lead to incorrect estimates of the objects' positions, too. In all these cases, and in many other practical applications, the agent's instantaneous observations do not always uniquely identify the underlying system state.

The term "state aliasing" traces back to Whitehead and Ballard (1991), who introduce the term *perceptual aliasing* to describe the phenomenon of multiple states manifesting as the same observation to an agent. They argue that perceptual aliasing could indeed be *beneficial* if there is no need to distinguish aliased states; that is, if all the aliased states have the same optimal action. The advantage of such a situation is that the agent can use a simpler representation for decision making than one that can discriminate between any set of states in the task. In several practical tasks, aliased states do need to be distinguished, and so state aliasing becomes more of a worry than a blessing.

In general an agent augmented with a short-term memory can remember sequences of observations, and use this history as a means to differentiate aliased states. Lin and Mitchell (1993) provide empirical evidence suggesting that several models of memory—such as finite windows of past experiences, and recurrent neural networks computing the model or the action value function—can be used in conjunction with learning to boost the performance of an agent acting in a partially observable world. Several researchers have indeed applied memory-based approaches to overcome state aliasing (Glickman and Sycara, 2001; Todd et al., 2009). Chrisman (1992) shows similar results obtained by explicitly modeling the underlying MDP and observation function, which together constitute a Partially Observable Markov Decision Problem (POMDP) (Monahan, 1982). McCallum (1993) designs a Utile Distinction Memory (UDM) in which states are distinguished only if doing so will increase the utility of decision making. The appeal of this approach is that the agent has a dynamic memory size that is adapted based on the needs of decision making, rather than on the underlying complexity of the state space. A shortcoming of the UDM approach is that it learns very slowly, because it needs large amounts of experience to provide the statistical confidence for deciding which states to distinguish. However, McCallum (1996) shows that the adoption of a prediction suffix tree (PST) as the data structure indexing memory can significantly speed up learning. Recent work (Holmes and Isbell, Jr, 2006) provides a proof that looping PSTs can exactly represent finite POMDPs with deterministic transition and observation functions, and indeed that these PSTs can be learned from experience.

The mathematical principles for solving POMDPs stem from the seminal work of Åström (1965). The key idea contributed by Åström is the use of belief states, which are probability distributions over the state space of the underlying MDP. To solve a POMDP P, Åström constructs an MDP M' whose states are belief states over the states in P. Transitions in M' are defined such that an optimal policy for M' induces an optimal policy for P. Crucially, the next state in M' (the next belief state in P) depends only on the current state in M' (the current belief state in P) and the most recent action and observation in P. In other words, belief states summarize all the information necessary for optimal decision making in P.

Since a belief state in P is a probability distribution over its underlying state space, the set of belief states is continuous. Hence, M' is an MDP with a continuous state space. There has been active research over the last few decades focusing on solving M' efficiently. It has been shown that the value functions of a certain class of policies for M' are convex and piecewise linear (Sondik, 1978), which allows more efficient solution than arbitrary continuous MDPs. Much of the work on POMDPs is in the context of planning: assuming that P is known, and also that it is finite. An early, landmark result from this body of research relates to the Witness algorithm provided by Cassandra et al. (1994), which can compute policies with values arbitrarily close to the value of an optimal policy for P. It remains that even in the context of planning, finding the optimal policy for a POMDP is very computationally expensive. For instance, the effectiveness of the point-based value iteration method, a sophisticated technique recently proposed by Pineau et al. (2006), has only been demonstrated on finite POMDPs with a few hundreds of states.

In a learning setting, samples are typically expensive to obtain, and even for problems with small, finite, state spaces, it becomes impractical to implement POMDP solution techniques that rely on reasoning exactly about belief states. Further, several problems in practice have continuous state spaces, which further erode the applicability of classical POMDP algorithms. Indeed there have been some attempts to tackle state aliasing using "approximate belief states" (Rodríguez et al., 1999), which can achieve a more favorable tradeoff between computational time and solution quality, but nor has this approach been demonstrated effective on tasks with more than a few hundreds of states.

The failure of theoretically-grounded POMDP learning and planning techniques to scale to practice has not derailed research seeking to apply RL methods to tasks with state aliasing. Indeed a significant fraction of sequential decision making tasks encountered in practice do not have fully observable states, and yet several successes have been noted on such problems (as Table 1.1 attests). In practice significant manual effort is expended in designing an "observed state" signal that encapsulates as much information as possible about the underlying state. Past observations and actions, and other useful sources of information, are combined to construct the observed state, making extensive use of domain knowledge. For example, it is common for robot soccer agents to use particle filtering to estimate positions and velocities of objects based on their past configurations, and applying the laws of physics. Little is known about the factors influencing the dynamics of a rat's nervous system when it is subjected to electrical stimuli; in designing a stimulation schedule for reducing the incidence of epileptic seizures, the only measurements available to Guez et al. (2008) are a sequence of electrical activations in the tissue, of which they treat a finite window as the observed state. It is so common in the practice of RL to use a well-designed observed state as a surrogate for the system state that the important distinction between them—the preservation of the Markov property!—is often forgotten. The (typically implicit) assumption underlying this practice is declared explicitly as follows by Nevmyvaka et al. (2006, see Section 3), who design a learning algorithm for the task of optimized trade execution:

> "*We note that where we say state we more precisely mean observed state; we in no way suggest that our state representations are sufficient to render a system as complex as modern financial markets truly Markovian. In what follows we will essentially be treating a partially observable environment as if it were fully observable to us; the test of this assumption will lie in our empirical evaluation.*"

In this thesis, we adopt the perspective that in general, a learning agent in a practical task can only access an observed state signal. Figure 2.2 shows a schematic view of a state-estimator module that derives this signal based on past observations and actions, as well as the current observation recorded by the agent's sensors. The key point to note is that the observed state s' is not necessarily a sufficient statistic for decision making, unlike a belief state. While observed states often yield "reasonable" policies when they are treated as states, an explicit acknowledgement of the modeling mismatch leads to

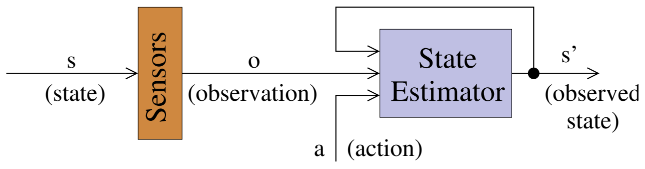


Figure 2.2: In practical tasks, we assume that the agents' sensors provide an observation signal o, which does not always identify the underlying system state s. A state estimator computes an observed state s' based on o and the sequence of past observations and actions in memory. This figure is based on one by Cassandra et al. (1994), in which s' is a belief state, rather than an observed state.

the following question: can better policies be learned by using optimization methods that do not treat observed states as states in an MDP? It does not seem ideal to completely ignore the "somewhat Markovian" nature of observed states either, as the sequence in which they occur encapsulates some amount of information for learning. Thus, a more pointed question to consider is whether we can learn by exploiting the properties of MDPs to the extent that observed states exhibit the Markov property, and use general optimization methods to the extent that they do not. Again, it is not obvious how to precisely measure to what extent the observed state space follows the Markov property; yet it appears reasonable to believe that an observed state that mixes X amount of noise to the true state signal will be more Markovian than an observed state that mixes $2X$ amount of noise.

In our experiments, we evaluate learning methods against representations of different quality by systematically varying sensor noise to implement different "levels of Markovianness". Our experiments also control generalization in a similar manner to gauge its effect on learning. In the next section, we proceed to consider the role of generalization in RL.

2.3 Generalization

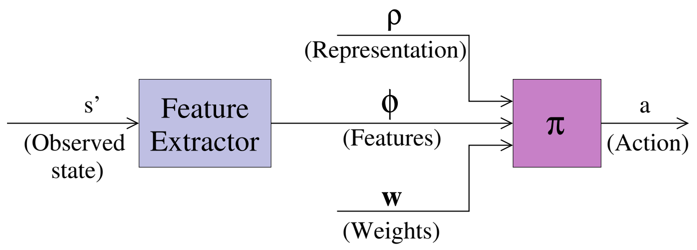
State aliasing handicaps learning by invalidating the Markov property in observed state transitions. However, even in fully observable worlds—where an agent's observation can unambiguously identify the system state—the sheer size of the state space can pose a formidable challenge to learning. Consider: a policy maps observed states to actions, and hence a general representation for learning must maintain a data structure that can assign an *arbitrary* action to *every* observed state. The number of parameters that would have to be stored and updated during learning in order to do so would be too large for most tasks occurring in practice. The impracticability of maintaining an index of individual states necessitates the use of generalization, whereby a smaller set of parameters approximates the learned policy. Even if generalization is strictly not necessary for learning over finite state and action spaces (as it is for infinite state and action spaces), it can still promote quicker learning in finite MDPs.[1]

Figure 2.3(a) depicts a typical architecture for learning a policy that generalizes over states and actions. Perhaps the most influential aspect in the success of a generalization scheme is the list of features made available to the learner. We can assume that a feature extraction scheme generates a vector of features $\boldsymbol{\phi}$ based on the agent's state. For example, in the Keepaway task (Stone et al., 2005), $\boldsymbol{\phi}$ is a 13-dimensional vector describing distances and angles among estimates of the players' and ball's positions, as shown in Figure 2.3(b). It is expected that these 13 features will facilitate better

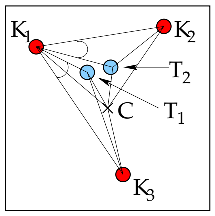
[1]Here, and often in the remainder of this text, we use the term *state* informally, without explicitly differentiating *states* from *observed states*. We do so to ease the reader's burden; we still assume that observed states and states are properly distinguished as in Section 2.2.

generalization than, say, x and y coordinates of the players and the ball, which would, in aggregate, convey the same information.

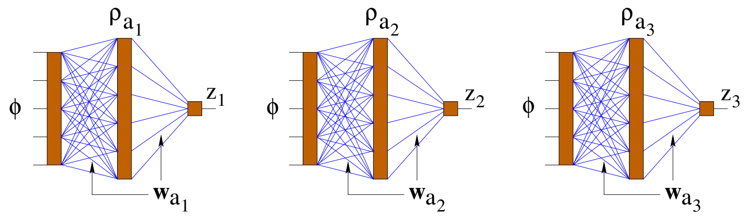
Given a feature vector corresponding to the state—which serves as a "factored" representation of that state—a parameterized functional form ρ accepts this feature vector as an input, and produces outputs that can be interpreted using some simple rule to pick an action for that state, thereby implementing a policy. Figure 2.3(c) shows a typical generalization scheme



(a)



(b)



(c)

Figure 2.3: (a) Typical generalization architecture used in practice, in which a policy is represented using a functional form ρ parameterized by a set of weights $\mathbf{w}$. Given a feature vector $\boldsymbol{\phi}$ as input, the outputs computed by ρ determine the action to select, implementing a policy π. (b) 13 features corresponding to distances and angles among (estimated) positions of the players in the Keepaway task. (c) A possible representation for a policy in Keepaway, which has three actions. For each action a_i, a separate neural network ρ_{a_i} computes an output z_i, given the current feature vector $\boldsymbol{\phi}$ and network weights $\mathbf{w}_{a_i}$. The policy is to choose an action solely based on the values of z_i.

adopted for learning in Keepaway (used, for example, by Stone et al. (2006)). Under this particular scheme, ρ comprises three neural networks, each producing an output corresponding to one of the three actions in Keepaway. The inputs to the neural networks are the 13 features, and the policy is to pick the action corresponding to the neural network with the highest activation. From Table 1.1, we observe that indeed neural networks are a common choice for ρ. Other function approximators used in practice are tile coding, decision trees, and linear combinations. Note that in nearly every task using generalization in Table 1.1, the feature extraction module and the functional form ρ are carefully designed by the programmer *a priori*. In other words, these components of the learning agent stay constant during learning. What is "learned" is the vector of parameters to ρ, which we denote $\mathbf{w}$. In the case of neural networks, $\mathbf{w}$ could serve as the network's weights and biases; if ρ is linear, its output is given by $\mathbf{w}^T\boldsymbol{\phi}$.

In general, $\boldsymbol{\phi}$ and ρ, which are components of the agent's representation, can also be adapted along with $\mathbf{w}$ as transition data become available. Indeed "representation discovery" has received steady attention over the years as a research area. Important topics that have been considered therein include, among others, state abstraction (Dietterich, 2000), temporal abstraction (Sutton et al., 1999), subgoal discovery (Digney, 1998), hierarchical control (Parr, 1998), feature selection and discovery (Ormoneit and Sen, 2002; Girgin and Preux, 2008; Kolter and Ng, 2009; Petrik et al., 2010), neuroevolution (Whiteson, 2007), and variable resolution discretization (Munos and Moore, 2002). It remains that while these studies make very important conceptual contributions, representation discovery is yet to be demonstrated as a general methodology that can be applied successfully across a wide range of sequential decision making problems. As stated in Chapter 1, we contend that representation discovery is likely to be successful in general only if an agent has a lot of data—orders of magnitude more than agents in today's practical applications can gather. We proceed to consider, therefore, how learning methods (for adapting $\mathbf{w}$ alone) must be designed under the assumption that the representation (combining $\boldsymbol{\phi}$ and ρ) is useful, but not necessarily perfect.

2.4 Demands of a Practical Learning Method

Continuing from Section 2.3, let us assume that a learning agent employs an underlying function representation ρ, and at each time step t, receives as input a vector of features $\boldsymbol{\phi}_t$, depending on the state s_t. The applications mentioned in Table 1.1 and the discussion in sections 2.2 and 2.3 affirm that it is realistic, and perhaps most appropriate, to expect that ρ and $\boldsymbol{\phi}_t$ are affected to some extent by state aliasing, facilitate *some* amount of useful generalization, but are not perfect. Given this limitation, what is demanded of the learning method in adapting the representation parameters $\mathbf{w}$?

The output of the learning agent at time t is an action a_t, based on which the environment will generate a next state s_{t+1} (partially visible to the agent through features $\boldsymbol{\phi}_{t+1}$) and a reward r_t. Perhaps the greatest appeal of RL lies in the manner desirable behavior can be specified succinctly through a scalar-valued reward function (Sutton and Barto, 1998). It is natural, therefore, that the objective of learning must be to maximize expected long-term reward. Surely the developers of the 25 applications mentioned in Table 1.1 will agree! Formalizing the motivating question laid down in Section 1.2, this dissertation concretely specifies the objective of learning in practical sequential decision making problems as follows.

Problem Definition: Given $\boldsymbol{\phi}$, ρ, and the number of samples that can be gathered, adapt $\mathbf{w}$ based on experience such that the expected long-term reward of the resulting policy is maximized.

This definition appears quite natural, and at first glance, the reader might wonder what is novel about it. Has the maximization of long-term reward not always been the central tenet of RL? If anything, the definition above underscores that it has, and that it must continue to be even when the representation is imperfect. From the definition, it is clear that if an agent possesses a perfect representation, maximizing long-term reward is exactly achieved by learning the optimal action value function (Bellman, 1957), which it is possible to achieve in a sample-efficient manner (Brafman and Tennenholtz, 2003). Thus, if representations are perfect, the problem definition above essentially becomes equivalent to learning a model or a value function exactly and efficiently. However, the key implication of the problem definition is on learning with *imperfect* representations. If state aliasing and generalization handicap the representation, how can $\mathbf{w}$ be adapted to reap the greatest rewards after a given amount of training time? *Approximating* the model or value function is no longer the obvious, provably correct answer. In some cases, the predictive power of such approaches might be desired as an end in itself, but here we solely demand performance, a pragmatic measure.

Several problems in machine learning have profited through concrete specifications of an objective function to optimize by adapting a vector of parameters. For example, classification is commonly posed as the problem of updating real-valued coefficients to maximize a separating margin; clustering and regression are implemented as the minimization of appropriate loss functions (Bishop, 2006). Our problem definition above for sequential decision making with imperfect representations is in the same spirit. However, it is important to note that whereas for several problems, setting up the right objective function naturally gives rise to solution strategies, it is not obvious how we may devise learning methods for sequential decision making under imperfect representations to maximize expected long-term reward.

We view the problem statement above as a fair reflection of the true needs of a majority of practical learning agents. Even if solution strategies are not readily forthcoming based on this definition, we believe that the quest for answers must be pursued in the spirit exhorted by the noted statistician John Tukey (1962, see pages 13–14):

> *"... Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise."*

This thesis primarily adopts experimental means to look for answers to the "right question" of learning with imperfect representations. Before proceeding to the next chapter, we consider one final aspect concerning the evaluation of learning algorithms, which, on purpose, we have left unspecified in our problem definition. In some tasks involving agent learning, it is necessary to measure the rewards accrued while learning (or to measure the *on-line* performance) (İpek et al., 2008). In other situations, it is more appropriate to assume that the learner outputs an entire policy after learning, which can then be evaluated off-line, with no further learning (Crites and Barto, 1996; Ng et al., 2004; Guez et al., 2008; Gabel et al., 2009). Either of these criteria can be applied in our problem definition to properly define expected long-term reward. As a concrete evaluation criterion for the experiments in this thesis, we adopt the latter strategy; that is, we treat the training and the deployment of a learning agent as distinct phases, measuring performance as the value of the policy that is generated at the end of the learning phase. Having a separate learning phase before the agent is deployed in practice finds appeal when it is not feasible to learn during an agent's long-term operation, for example, because of computational considerations, or because learning simply cannot be trusted in the agent's deployed scenario (such as a helicopter in flight (Ng et al., 2004)). Turning off learning, and testing what has been learned, also facilitates meaningful comparisons with policies that have been obtained through ways other than learning.

This chapter concludes the first part of the dissertation, wherein we have argued the inevitability of having to work with imperfect representations. After presenting some fundamental results related to learning in finite MDPs, we have described the phenomena of state aliasing and generalization, and have thereafter proceeded to define the problem of sequential decision making with imperfect representations. In chapters 3 and 4, we experimentally evaluate the performance of different existing classes of learning methods under this problem definition. In chapters 5 and 6, we present ideas for improving the performance of learning methods that are constrained to work with imperfect representations.

Chapter 3

Experimental Comparison of Learning Methods

This chapter is the first one in a series (which also includes chapters 4, 5, and 6) devoted to describing the individual technical contributions of the thesis. Specifically this chapter presents the first contribution introduced in Section 1.3.2, which constitutes a major and defining portion of the dissertation. As a first step towards engaging with the problem definition in Section 2.4—of how we might devise effective learning methods for imperfect representations—in this chapter, we devise a thorough experimental study to evaluate existing classes of RL methods under imperfect representations.

First we propose "parameterized learning problems" as a novel experimental methodology, which enables us to systematically control representational aspects such as state aliasing and generalization, and to characterize their effects on learning methods (Section 3.1). Employing this methodology, we compare two qualitatively distinct classes of algorithms: on-line value function-based methods and policy search methods (Section 3.2). Empirical comparisons among various methods within each of these classes determine Sarsa(λ) and Q-learning(λ) as winners among the former, and CMA-ES as the winner in the latter. Comparing Sarsa(λ) and CMA-ES further on relevant problem instances, we find conclusive evidence that any one of these methods outperforms the other in some part of the problem space, thereby affirming our founding hy-

pothesis that different representations call for different learning methods (Section 3.3). We discuss several new insights relating problems, learning methods, and method-specific parameters (Section 3.4), and provide references to related work (Section 3.5).

In chapters 1 and 2, we laid down the conceptual framework of this thesis, which wrestles with the question of devising learning methods to be effective in practice (that is, when working with imperfect representations). As with any endeavor in science and engineering, the first step in our quest for answers has to be that of exploring and mapping the landscape that we set forth to penetrate. In this chapter, we undertake such an exercise. Specifically we conduct an extensive experimental study to compare existing classes of RL methods under various settings of "representational quality". In short we seek to ascertain: in what ways, and to what extent, does the representation used for learning determine the dominance of one class of learning methods over another?

In a formal sense, the "No Free Lunch" theorems of Wolpert and Macready (1997) establish that for any optimization algorithm, an elevated performance in one class of problems is offset by worse performance in some other class. Even so, the enterprise of machine learning rests on the assumption that classes of problems of any practical interest tend to possess regularities: we desire learning methods that can perform well on such practically-relevant problems by actively characterizing and exploiting the problems' regularities. Consequently, to the extent that the relationships between *problem* instances and the performance properties of *algorithms* are unclear, it becomes a worthwhile pursuit to uncover them. The need for such research has been advocated ever since the inception of machine learning as a field; in an early editorial in the *Machine Learning* journal, Langley (1988, see page 7) writes:

> *"For instance, one might find that learning method A performs better than method B in one environment, whereas B fares better than A in another. Alternatively, one might find interactions between two components of a learning method or two domain characteristics. We believe the most unexpected and interesting empirical results in machine learning will take this form."*

In line with Langley's vision, the practice of supervised learning has benefitted from a number of empirical studies that seek to identify the strengths and weaknesses of learning methods. For example, Caruana and Niculescu-Mizil (2006) undertake a detailed comparison involving a number of supervised learning methods, test problems, and evaluation metrics. Caruana et al. (2008) present empirical results demonstrating that random forests (Breiman, 2001) are typically more effective than several other classification methods on problems with high dimensionality (greater than 4000). Although the canonical boosting algorithm (Freund and Schapire, 1996) enjoys desirable theoretical properties and is predominantly effective in practice, studies comparing it with other ensemble schemes such as bagging (Quinlan, 1996; Bauer and Kohavi, 1999) hint at its vulnerability in the presence of noisy training data. Banko and Brill (2001) advance the case that for problems with very large data sets (for example, natural language applications on the Internet), simple classifiers such as Winnow (Littlestone, 1987) can be the most effective, and that voting-based ensemble schemes do not retain their attractiveness.

By associating *problem characteristics* with the strengths and weaknesses of supervised learning *methods*, the studies listed above provide useful “rules of thumb” to a practitioner who must choose a method to apply to a problem. Unfortunately the much broader scope of the RL problem leaves the practitioner of RL with few such guidelines. Faced with designing an agent for a sequential decision making problem, not only does a designer need to pick a learning algorithm, he/she has to address the related issues of state estimation, exploration, and function approximation, while possibly satisfying computational and memory constraints.

The experimental study described in this chapter is among early steps towards the the eventual development of a “field guide” for the practice of RL, which would both inform the choices made by designers of RL solutions, and identify promising directions for future research. Ultimately, a field guide would be evaluated based on the extent to which it can expedite the process of designing solutions for full-scale deployed applications. Nevertheless, such applications are themselves too complex and constrained to provide reliable data from which the principles for a field guide can be inferred. Rather, there is a need for simpler, more transparent problems through which we, as designers, can systematically sort through the complex space of interactions between RL problems and solution strategies.

Existing work on the subject of comparing RL algorithms has primarily relied on standard, benchmarking tasks, with possibly a small number of variations (Moriarty et al., 1999; Gomez et al., 2008; Heidrich-Meisner and Igel, 2008a; Whiteson et al., 2010). By contrast, we design a synthetic, *parameterized learning problem* with the explicit purpose of ascertaining the “working regions” of learning algorithms in a space that is carefully engineered to span the dimensions of the task and the learning architecture. Our approach enjoys the following merits:

1. The designed task and learning framework are easy to understand and can be controlled precisely.
2. We may examine the effect of subsets of problem parameters while keeping others fixed.
3. We can benchmark learned policies against optimal behavior.
4. The learning process can be executed in a relatively short duration of time, thereby facilitating extensive experimentation.

Whereas parameterized learning problems can be designed for testing virtually any problem characteristic, in keeping with the pursuit of this thesis, we employ them to study two key representational aspects: state aliasing and generalization. In our study, these factors are systematically controlled in order to gauge their effect on different learning methods. While state aliasing and generalization can be construed as aspects internal to the agent, our study also considers task-specific characteristics such as the size of the state space and the stochasticity of actions. Any fixed setting for the parameters that control these factors determines a *learning problem*, on which different learning methods can be compared.[1]

[1]The term "parameterized learning problem" is quite generic; such problems have been used in the past both in RL and in other fields. For some examples, see our discussion of related work in Section 3.5 We apply the term to our framework here to underscore that problem parameters are its very crux; they are not secondary as in related work.

In our study, we compare learning methods from two contrasting classes of algorithms. The first class corresponds to (model-free) on-line value function-based methods, which learn by associating utilities with action choices from individual states. The second class of algorithms we examine are policy search methods. Rather than learn a value function, policy search methods seek to directly optimize the parameters representing a policy, treating the expected long-term reward accrued as an objective function to maximize. First we evaluate several methods *within* each of the above classes, and based on their empirical performance, pick one method from each class to further compare across a suite of problem instances. The representatives thus chosen are Sarsa(λ) (Rummery and Niranjan, 1994; Sutton and Barto, 1998) from the class of on-line value function-based methods, and CMA-ES (Hansen, 2009) from the class of policy search methods. In evaluating a method on a problem instance, our experimental framework allows us to extensively search for the method-specific parameters (such as learning rates, eligibility traces, and sample sizes for fitness evaluation) that lead to the method's best performance on that instance. Our experiments identify regions of the problem space that are better suited to on-line value function-based and policy search methods, and yield insights about the effect of algorithm-specific parameters.

While the careful design of a synthetic learning problem empowers us with a high degree of control in our experimentation, equally it qualifies the extent to which our conclusions may generalize in practice. Thus, the results from our study are to be taken as starting points for further empirical investigation, rather than treated as well-grounded final products in themselves. In this sense, the methodology we put forth enjoys a *complementary* relationship with the research strategy of evaluating RL methods on more realistic problems. The subsequent chapters of this thesis indeed pay close attention to some complex sequential decision making problems: Tetris (Bertsekas and Tsitsiklis, 1996) (Chapter 4) and robot soccer Keepaway (Stone et al., 2005) (Chapter 6). Lessons drawn from the present chapter do find validation and inspire new algorithmic contributions in these more complex tasks.

Whereas the experiments in this chapter are restricted to two classes of learning methods—on-line value function-based methods and policy search methods—the experimental framework developed here can easily support other classes of learning methods. In particular some relevant classes of methods include model-based and batch RL methods, actor-critic methods, and policy gradient techniques. We provide a brief survey of these classes of learning methods in Appendix C, and discuss relationships between them in Chapter 7. However, it exceeds the scope of the thesis to subject these methods to systematic empirical evaluation.

This chapter is organized as follows. In Section 3.1, we describe the detailed design of our parameterized learning problem. Section 3.2 provides brief descriptions of the methods compared in the study. In Section 3.3, we present detailed results from our experiments, which we follow with a discussion in Section 3.4. Related work is discussed in Section 3.5. We summarize and conclude the chapter in Section 3.6.

3.1 A Parameterized Sequential Decision Making Problem

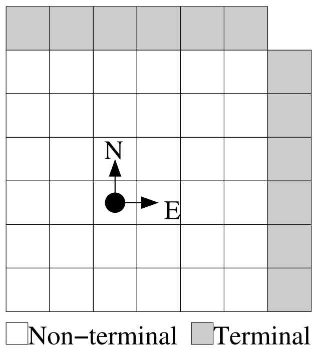
In this section, we describe the construction of our parameterized learning problem, which is composed of a task MDP and an accompanying learning framework that incorporates state aliasing and generalization.

3.1.1 Problem Size and Stochasticity

The class of tasks we design consists of simple, square grids, each having a finite number of states. An example of such a task is illustrated in Figure 3.1. The size of the state space is $s^2 - 1$, where s, the side of the square, serves as a parameter to be varied. Each episode begins with the agent placed in a start state chosen uniformly at random from among the set of non-terminal states, as depicted in Figure 3.1(a). The north and east sides of the grid are lined with terminal states, of which there are $2(s-1)$. From each state, the agent can take either of two actions: **North** (**N**) and **East** (**E**). On taking **N** (**E**), the agent moves north (east) with probability p and it moves east (north) with probability $1-p$. The variable p, which essentially controls the stochasticity in the transitions, is also treated as a parameter of the task MDP. Note that irrespective of the value of p, the agent always moves either north or east on each transition before reaching a terminal state. Consequently episodes never last more than $2s-3$ steps.

Through the course of each episode, the agent accrues rewards at the states it visits. Each MDP in our class is initialized with a fixed allotment of rewards drawn uniformly from $[0, 1]$, as illustrated in Figure 3.1(b). In general the rewards in an MDP can themselves be stochastic, but in our tests, we find that the effect of stochastic rewards on the performance of our learning algorithms is qualitatively similar to the effect of stochastic state transitions, which are controlled by the parameter p. Thus, we keep the rewards deterministic. Figures 3.1(c) and 3.1(d) show the optimal values and the actions to which they correspond under the reward structure shown in Figure 3.1(b) (assuming $p = 0.1$).

We do not discount rewards in the computation of values. Notice that



(a)

.77	.33	.59	.34	.82	.39	
.79	.01	.26	.03	.14	.55	.57
.84	.03	.44	.96	.83	.53	.23
.99	.55	.75	.69	.05	.23	.05
.58	.96	.36	.18	.85	.92	.82
.30	.69	.84	.05	.23	.95	.22
	.35	.28	.32	.90	.45	.64

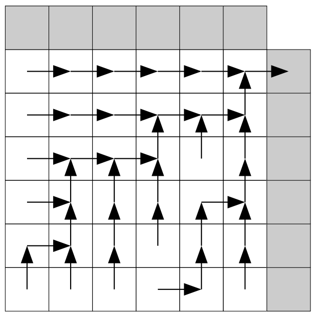
Example Rewards

(b)

0	0	0	0	0	0	
1.2	1.3	1.1	1.1	1.1	0.6	0
3.1	3.2	3.0	2.2	1.5	1.0	0
4.8	4.4	3.7	3.1	2.3	1.4	0
5.9	4.9	4.4	3.7	2.5	1.5	0
6.5	5.8	4.7	3.8	3.3	2.2	0
6.8	6.4	5.5	4.4	3.5	2.9	0

Values of optimal actions

(c)



Optimal Policy (p = 0.1)

(d)

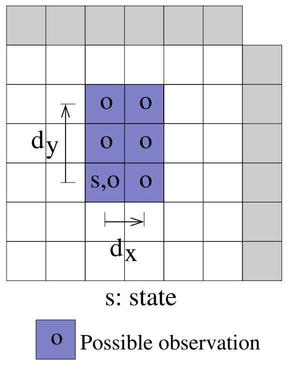
Figure 3.1: (a) Example of parameterized MDP example with $s = 7$; the number of non-terminal states is 36. (b) Rewards obtained at "next states" of transitions. (c) Optimal action values from each state when $p = 0.1$. (d) Corresponding optimal policy.

the variation in values along the north and east directions is gradual, thereby supporting the scope for generalization between neighboring cells. The values marked in Figure 3.1(c) are obtained using dynamic programming. Indeed it is also straightforward under this setup to learn the optimal policy based on experience, for example, by using a table of action values updated through Q-learning. However, the objective of our study is to investigate situations in which table-based approaches are not guaranteed to succeed. In the remainder of this section, we specify the aspects of our learning problem that, in ways similar to real-world problems, render table-based approaches infeasible.

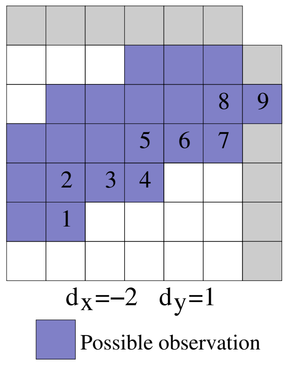
3.1.2 State Aliasing

As argued in Section 2.2, state aliasing is widespread in practice, forcing agents to use observed states as surrogates for the true system state. We adopt an appropriate encoding of state aliasing in our parameterized learning problem. Each cell in our task MDP corresponds to a state. In order to model state aliasing, we constrain the learner to use an observed state o, which, in general, can be different from the true state s. Our scheme to pick o based on s is depicted in Figure 3.2. Given s (say, with coordinates (x, y)), we consider all the cells with x coordinates between x and $x + d_x$ (both inclusive) and with y coordinates between y and $y + d_y$ (both inclusive). From among these cells, we pick one uniformly at random to serve as the corresponding observed state o. By controlling d_x and d_y, we vary the extent of state aliasing.

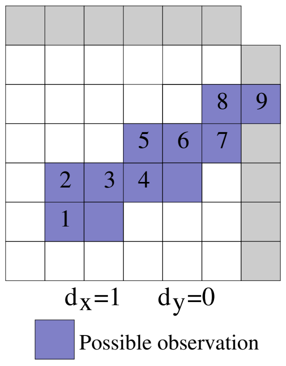
Before starting a learning run, we fix d_x and d_y: each is sampled from a Gaussian distribution with zero mean and a standard deviation equal to σ, and then rounded to the nearest integer. Note that d_x and d_y can be positive, negative, or zero. Figures 3.2(b) and 3.2(c) show an illustrative trajectory of states numbered 1 through 9. Under different settings of d_x and d_y, the figures show the set of all possible observed states that could result while the agent traces its trajectory. As is apparent from the figures, by keeping d_x or d_y fixed for the entire course of a learning run (that is, by not changing them from episode to episode), the state aliasing (or *state noise*) encountered by the agent during its lifetime is *systematic* in nature. Informal experimentation with a



(a)



(b)



(c)

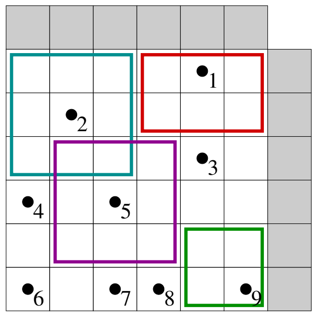
Figure 3.2: An implementation of state aliasing in the example MDP from Figure 3.1. (a) Variables d_x and d_y (themselves generated randomly based on parameter σ) define a rectangle with the true state at a corner; cells within this rectangle are picked uniformly at random to constitute observed states. (b) A trajectory of true states 1 through 9, and the set of all possible observed states that could be encountered during this trajectory when $d_x = -2$ and $d_y = 1$. (c) For the same trajectory, the set of possible observed states when $d_x = 1$ and $d_y = 0$.

number of schemes for implementing state noise suggests that biased noise tends to affect learning more severely than zero-mean noise. The magnitude of the noise, implemented through d_x and d_y, is controlled by the single free parameter σ, which we vary in our experiments. Setting σ to 0 removes state aliasing. Progressively larger values of σ lead to observed states that are farther apart from the agent's true state, and render the agent's interaction with the environment non-Markovian.

3.1.3 Generalization

Recall from Section 2.3 that generalization is the phenomenon wherein a free parameter in the policy representation might influence the choice of action from more than one state. The generalization scheme in our learning problem is motivated by "CMAC" (Albus, 1981), a popular method that is used in a number of RL applications (Singh and Sutton, 1996; Stone et al., 2005; İpek et al., 2008). At each decision making step, we provide the learning agent a vector of n_f features to describe its observed state. Each feature is a square "tile", with a binary activation: 1 within the boundary of the tile and 0 outside. Tiles have a fixed width w, which serves as a parameter in our experiments that determines the extent of generalization between states while learning. The centers of the tiles are chosen uniformly at random among non-terminal cells in the MDP. Figure 3.3 continues the example from Figure 3.1, describing our generalization architecture. In Figure 3.3(a), nine tiles (numbered 1 through 9) are used by the generalization scheme. The tile width w is set to 3; for illustration, four among the nine tiles are shown outlined.

Notice that every non-terminal cell in Figure 3.3(a) is covered by at least one tile: every such cell has at least one feature that is active. Indeed we ensure that complete coverage is always achieved, in order that non-trivial decisions can be made at every cell. Clearly, not all the cells will be covered if the number of tiles (n_f) and the width of each tile (w) are both small compared to the number of non-terminal states ($(s-1)^2$). Therefore, in all our experiments, we set these parameters such that in conjunction they can facilitate complete coverage of all non-terminal cells. The placement of the n_f tiles is performed randomly, but preserving the constraint that all non-terminal cells are covered. In order to implement this constraint, we first place the tiles in regular positions that guarantee complete coverage, and then repeatedly shift tiles, one at a time, to random positions, while still preserving complete coverage. Rather than treat n_f directly as a parameter in our experiments, we



Centers of tiles ($n_f = 9$)

(a)

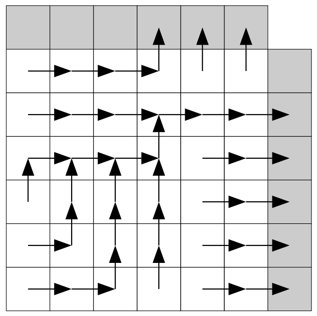
	N	E
f_1	2	−1
f_2	−8	3
f_3	0	6
f_4	5	5
f_5	4	−6
f_6	−3	2
f_7	2	3
f_8	4	1
f_9	−2	7

(b)

3	3	3	2	2	2
3	3	3	5	5	5
8	2	−3	4	6	6
5	9	4	4	6	6
7	8	10	10	8	7
2	5	6	6	8	7

Larger of activation values

(c)



Greedy policy

(d)

Figure 3.3: Generalization scheme in example MDP from Figure 3.1. (a) A randomly chosen subset of cells (numbered 1 through 9) are the centers of overlapping tiles (giving $\chi = \frac{9}{36} = 0.25$). The tile width w is set to 3; tiles 1, 2, 5, and 9 are shown outlined (and clipped at the boundaries of the non-terminal region). (b) Table showing coefficients associated with each tile for actions **N** and **E**. (c) The activation value of each cell for an action is the sum of the weights of the tiles to which it belongs. The figure shows the higher activation value (among **N** and **E**) for each cell. (d) Arrows mark a policy that is greedy with respect to the activations: that is, in each cell, the action with a higher activation value is chosen. In general the agent will take the greedy action from its *observed* state, which is determined as described in Section 3.1.2.

normalize it by dividing by the number of non-terminal cells: $(s-1)^2$. The resulting quantity, $\chi = \frac{n_f}{(s-1)^2}$, lies in the interval $(0, 1]$, and is more appropriate for comparisons across different problem sizes. In Figure 3.3(a), $n_f = 9$ and $s = 7$, yielding $\chi = 0.25$. We treat χ as a parameter in our experiments. As we shortly describe, χ determines the resolution with which independent actions can be taken from neighboring cells. In this sense, χ measures the "expressiveness" of the generalization scheme.

Given the vector of features for its observed state, the agent computes a separate linear combination for each action, yielding a scalar "activation" for that action. For illustration consider Figure 3.3(b), which shows a possible assignment of coefficients for each feature and action. It is these coefficients (or "weights") that the agent updates when it is learning. Figure 3.3(c) shows the *higher* of the activations for the two possible actions at each cell in our illustrative example; Figure 3.3(d) shows the action with the higher activation. While learning, the agent may take any action from the states it visits. However, while evaluating learned behavior, we constrain the agent to take the action with the higher activation, breaking ties evenly. This strategy effectively implements the choice we made in Section 2.4 of evaluating behavior off-line.

In effect, the only free parameters for the learning agent to update are the coefficients corresponding to each action. By keeping other aspects of the representation—such as the features and policy structure—fixed, we facilitate a fair comparison between different learning methods. In general, value function-based methods such as Sarsa(λ) seek to learn weights that approximate the action value function. We expect that setting $\sigma = 0$ and $\chi = 1$ would favor them, as the optimal action value function can then be represented. While this is so under any value of w, setting $w = 1$ replicates the case of table-based learning with no generalization. Higher settings of w enforce generalization. Increasing σ or reducing χ would likely shift the balance in favor of policy search methods, under which activations of actions are merely treated as action preferences. As Baxter and Bartlett (2001) illustrate, even in simple 2-state MDPs, with generalization it is possible that the optimal action value function cannot be represented, even if an optimal policy can be represented.

In summary the design choices listed in this section are the end products of a process of trial and error directed towards constructing a suite of instances that allow us to study trends in learning algorithms, rather than constructing instances that are challenging in themselves. Table 3.1 summarizes the parameters used in our framework. Parameters s, p, σ, χ, and w, along with a random seed, fix a learning problem for our experiments. By averaging over multiple runs with different random seeds, we estimate the mean performance achieved by learning methods as a function of s, p, σ, χ, and w. Note that even if these parameters do not perfectly replicate an instance of any *specific* sequential decision making in practice, they are capable of being *varied* in a controlled manner to measure their effect on learning methods.

It must be noted that the parameterized learning problem described above is limited in several respects. While it enables the study of the most central problem parameters—problem size, stochasticity, state aliasing, and generalization—it does not likewise isolate several other aspects influencing practical implementations of RL. Foremost is the question of exploration, which is not very crucial in our setup due to the occurrence of start states uniformly at random. The learning agent only has two actions; in practice large or continuous action spaces are quite common. Understanding the ef-

Table 3.1: Summary of learning problem parameters. The last column shows the ranges over which each parameter is valid and meaningful to test.

Parameter	**Property of:**	**Controls:**	**Range**
s	Task	Size of state space	$\{2, 3, \ldots, \infty\}$
p	Task	Stochasticity of transitions	$[0, 0.5)$
σ	Agent/task interface	State aliasing	$[0, \infty)$
χ	Agent	Expressiveness of generalization scheme	$(0, 1]$
w	Agent	Width of generalization	$\{1, 3, \ldots, 2s - 3\}$

fects of other aspects such as computational and memory constraints, the variation among action values from a state, different types of state noise, the sparsity and spread of the rewards, and the average episode length, would also be important for designing better algorithms in practice. We hope that the experimental methodology introduced in this chapter will facilitate the investigation of such questions in the future.

In the next section, we provide brief descriptions of the learning algorithms used in our experiments; in Section 3.3, the algorithms are compared at a number of different parameter settings drawn from the ranges provided in Table 3.1. Along with the parameterized learning problem itself, the results of these experiments constitute an important contribution of this thesis.

3.2 Methods in Study

As noted earlier, we compare two contrasting classes of learning methods in our study: on-line value function-based (VF) methods, and policy search (PS) methods. Methods from both classes find wide use in practice, for example, in several applications from Table 1.1. With the aim of comparing these classes themselves, we first evaluate various methods *within each class* to pick a representative. In this section, we describe the learning methods thus considered, and include relevant implementation-specific details. Experimental comparisons follow in Section 3.3.

3.2.1 On-line Value Function-based (VF) Methods

We compare three learning methods from the VF class: Sarsa(λ) (Rummery and Niranjan, 1994; Rummery, 1995), Q-learning(λ) (Watkins, 1989; Watkins and Dayan, 1992; Rummery, 1995; Peng and Williams, 1996; Sutton and Barto, 1998), and Expected Sarsa(λ) (abbreviated "ExpSarsa(λ)") (Rummery, 1995; van Seijen et al., 2009). These methods are closely related: they all continually refine an approximation of the action value function, making a constant-time update every time a new state is encountered. Yet the methods are distinguished by subtle differences in their update rules. We include these methods in our study to examine how their differences affect learning under state aliasing and generalization: settings under which theoretical analysis is limited. We proceed to describe the methods themselves.

Sarsa(λ) is a model-free value function-based method, which makes on-line, on-policy, temporal difference (TD) learning updates. The learning agent maintains an estimate of an action value function, Q, which is updated as it encounters sequences of states (s), actions (a) and rewards (r). In particular assume that the agent encounters the following trajectory, in which suffixes index decision steps:

$$s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, a_{t+2}, r_{t+3}, \ldots.$$

The agent updates $Q(s_t, a_t)$ by computing a target, $Q_{Target}(s_t, a_t)$, and taking an incremental step towards it as follows:

$$Q(s_t, a_t) \leftarrow (1 - \alpha_t)Q(s_t, a_t) + \alpha_t Q_{Target}(s_t, a_t),$$

where $\alpha_t \in (0, 1]$ is the learning rate for the update. Recall that in our architecture, Q is represented as a linear function approximator; hence, the learning update is implemented through gradient descent. Under Sarsa(0), the "fully bootstrapping" version of Sarsa, the target is computed as follows:

$$Q_{Target}^{Sarsa(0)}(s_t, a_t) = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}),$$

where $\gamma \in [0, 1)$ is a discount factor.[2] Note that the target does not count the actual rewards accrued beyond time step $t + 1$; rather, the discounted sum of these "future" rewards is substituted with its current estimate: $Q(s_{t+1}, a_{t+1})$. By contrast, a Monte Carlo method, Sarsa(1) computes its estimates wholly from sample returns, as:

$$Q_{Target}^{Sarsa(1)}(s_t, a_t) = r_{t+1} + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+1+k}.$$

[2]It is legitimate to use $\gamma = 1$ in episodic tasks. We do so in our experiments (see Section 3.3).

This target of Sarsa(1) would not change depending on the actual states that the trajectory visited, but only based on the sequence of rewards obtained, making Monte Carlo methods less dependent on the state signal than fully bootstrapping methods. Both methods still try to estimate state-action values, and therefore rely on being able to precisely detect s_t and represent $Q^{Sarsa}_{Target}(s_t, a_t)$. In general, intermediate methods that implement varying extents of bootstrapping can be conceived by varying the "eligibility trace" parameter $\lambda \in [0, 1]$. The estimated target for $Q(s_t, a_t)$ used by Sarsa(λ) is:

$$Q^{Sarsa(\lambda)}_{Target}(s_t, a_t) = r_t + \gamma\{(1 - \lambda)Q(s_{t+1}, a_{t+1}) + \lambda Q^{Sarsa(\lambda)}_{Target}(s_{t+1}, a_{t+1})\}.$$

For the case of discrete MDPs, in which Q can be maintained as a table, Singh et al. (2000) show that by following a policy that is "greedy in the limit" with respect to Q, and which performs an infinite amount of exploration, Sarsa(0) will ultimately converge to the optimal action value function Q^*, from which the optimal policy π^* can be derived by acting greedily. For linear function approximation schemes such as in our parameterized learning problem, Perkins and Precup (2003) show that convergence to a fixed point can be achieved by following a method similar to Sarsa(0).

We use a standard on-line implementation of Sarsa(λ) with binary features, a linear representation, and *replacing* eligibility traces (Sutton and Barto, 1998, see page 212). While learning, the agent follows an ϵ-greedy policy. We treat both the exploration strategy and the schedule for annealing the learning rate as *parameterizable* processes. We follow an ϵ_u-greedy exploration policy during episode u, keeping ϵ_0 as a free parameter, and $\epsilon_U = 0.01$, where U is the total number of training episodes. Intermediate values of ϵ_u are set based on a harmonic sequence going from ϵ_0 to 0.01. We use such a schedule based on empirical evidence of its effectiveness. Interestingly, informal experimentation shows that a similar annealing schedule is also the most effective for the learning rate α; that is, we keep α_0 as a free parameter and anneal it harmonically to 0.01 at the end of training. Since features are binary, we divide the mass of each update equally among the features that are active under the state-action being updated. We note that theoretically-motivated update rules do exist for annealing the learning rate. For example, Hutter and Legg (2008) derive a rule based on minimizing the squared loss between estimated and true values. However, their approach only applies with tabular representations of Q, and only in continuing (rather than episodic) tasks.

Apart from λ, ϵ_0, and α_0, yet another parameter influencing Sarsa(λ) is the setting of the initial weights (coefficients in the linear representation). In our experiments, we set all the weights initially to θ_0, which is our final method-specific parameter. Table 3.2 summarizes the parameters defining Sarsa(λ). These parameters also apply to other methods in the VF class, which we now describe.

Whereas Sarsa(λ) computes its target for time t based on the action to be taken at time $t+1$—a_{t+1}—ExpSarsa(λ) and Q-learning(λ) compute their targets (and make learning updates) *before* a_{t+1} is chosen. Once s_{t+1} is reached, ExpSarsa(λ) computes its target based on an expectation over the possible choices of a_{t+1} while following the current ϵ-greedy policy π_{t+1}:

$$Q^{ExpSarsa(\lambda)}_{Target}(s_t, a_t) = r_t +$$
$$\gamma \left\{ (1-\lambda) \sum_{a \in A} \mathbb{P}\{a | s_{t+1}, \pi_{t+1}\} Q(s_{t+1}, a) + \lambda Q^{ExpSarsa(\lambda)}_{Target}(s_{t+1}, a_{t+1}) \right\}.$$

Table 3.2: Summary of parameters used by methods within VF. The last column shows the ranges over which we tune each parameter.

Parameter	**Controls:**	**Range**
λ	Eligibility traces	$[0, 1]$
α_0	Initial learning rate	$[0.1, 1]$
ϵ_0	Initial exploration rate	$[0.1, 1]$
θ_0	Initial weights	$[-10.0, 10.0]$

This alteration leads to a reduced variance in the update, as a sampled action value is now replaced with a smoothed-out estimate. It is shown by van Seijen et al. (2009) that like Sarsa(0), ExpSarsa(0) can also be made to converge to the optimal policy in discrete, finite MDPs. Q-learning(λ) differs from Sarsa and ExpSarsa in that it is an *off-policy* method: rather than learning the action value function of the policy being followed, π_t, Q-learning(λ) seeks to directly learn the action values of the optimal policy π^*. This objective is achieved by computing the target as follows:

$$Q^{\textit{Q-learning}(\lambda)}_{Target}(s_t, a_t) = r_t + \gamma\{(1-\lambda)\max_{a \in A} Q(s_{t+1}, a) + \lambda Q^{\textit{Q-learning}(\lambda)}_{Target}(s_{t+1}, a_{t+1})\}.$$

Sutton and Barto (1998, see page 184) refer to the update rule resulting from the target above as a "naïve" implementation of Q-learning with eligibility traces, because the rule lacks technical justification as a proper TD learning update. By contrast, there do exist some sound variations of Q-learning with eligibility traces (Watkins, 1989; Peng and Williams, 1996), under which updates additionally have to account for whether chosen actions are greedy or non-greedy. In fact, a wide array of possibilities exists for devising TD learning algorithms: several variants of basic algorithms such as Q-learning and Sarsa can be realized by making subtle alterations to their update rules. For example, Nissen (2007) introduces a method called Q-Sarsa(λ), which takes an additional parameter to determine a balance between the on-policy and off-policy nature of updates. In his Ph.D. thesis, van Hasselt (2011) introduces several novel TD learning rules—such as Double Q-learning and QV-learning—and analyzes the estimation biases in their update rules.

It exceeds the scope of our dissertation to undertake an extensive study comparing all possible variants of TD update rules. Rather, a novel contribution of our experiments is to consider three among them—Sarsa(λ), ExpSarsa(λ), and (naïve) Q-learning(λ)—in the presence of state aliasing and generalization. Indeed our results show that under the influence of these factors, uncharacterized patterns in performance emerge. We refer the reader to the Ph.D. thesis of Rummery (1995, see Chapter 2) for an excellent presentation of various TD learning rules. Note that Rummery refers to Sarsa as "modified Q-learning", and to Expected Sarsa as "summation Q-learning".

As with Sarsa(λ), we parameterize ExpSarsa(λ) and Q-learning(λ) to control their learning and exploration rates, as well as their initial weights. The corresponding parameters, α_0, ϵ_0, and θ_0, are summarized in Table 3.2. Henceforward, we drop the "λ" from Sarsa(λ), ExpSarsa(λ), and Q-learning(λ), and refer to these methods simply as Sarsa, ExpSarsa, and Q-learning, respectively. We do so to highlight that these methods are no longer only parameterized by λ in our experiments—so are they by α_0, ϵ_0, and θ_0.

Note that setting $w > 1$ in our parameterized learning problem introduces generalization, and further, setting $\chi < 1$ reduces the expressiveness of the generalization scheme. Thus, in general, the approximate architectures used are incapable of representing the optimal action value function Q^*. Even with full expressiveness ($\chi = 1$), if using generalization ($w > 1$), methods from VF are not guaranteed to converge to the optimal action value function. Further, even if these methods approximate the action value function well, as defined through the Bellman error, greedy action selection might yet pick suboptimal actions in regions of inaccurate approximation, resulting in low long-term returns. The next chapter focuses on the manifestation of this phenomenon in sequential decision making algorithms that have been applied to Tetris (Bertsekas and Tsitsiklis, 1996), the popular computer video game.

A bulk of the research in RL with linear function approximation has been in the context of prediction: estimating the value function of a fixed policy (without policy improvement). An early result due to Sutton (1988) establishes that TD(0) with linear function approximation converges when the features used are linearly independent. Dayan (1992), and Dayan and Sejnowski (1994), extend this result to TD(λ), $\forall \lambda \in [0, 1]$, while Tsitsiklis and Van Roy (1997) show convergence for the more realistic case of infinite state spaces and linearly dependent features. Although most results for the convergence of linear TD learning are for estimating values of the policy that is used to gather experiences, the more general (and useful) case of off-policy learning has also been addressed (Precup et al., 2001; Sutton et al., 2009).

The problems in learning approximate value functions on-line primarily arise due to the nonstationarity and bias *in the targets* provided to the function approximator (Thrun and Schwartz, 1993). The best theoretical guarantees for learning control policies with approximate schemes come with several restrictions. Most results are limited to linear function approximation schemes; in addition some methods make demands such as Lipschitz continuity of the policy being learned (Perkins and Precup, 2003) and favorable initial conditions (Melo et al., 2008). Results tend to guarantee convergence of certain updating schemes, but invariably lack desirable guarantees about the long-term reward that will be accrued at convergence (Sabes, 1993; Perkins and Pendrith, 2002; Perkins and Precup, 2003).

In recent work, Maei et al. (2010) introduce the Greedy-GQ algorithm, which provably converges while making off-policy learning updates to a linear function approximator. Unfortunately Greedy-GQ requires that the policy followed *while learning* stay fixed, preventing the agent from actively exploring based on the experiences it gathers. Thus, for example, ϵ-greedy exploration with $\epsilon < 1$ violates the conditions needed for Greedy-GQ to converge; our informal experiments confirm that such a version of Greedy-GQ does not perform on par with the other methods we consider within the VF class. Thus, we do not include Greedy-GQ in our extensive comparisons.

3.2.2 Policy Search (PS) Methods

We include three methods from the PS class in our study: the cross-entropy method (CEM) (de Boer et al., 2005), the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen, 2009), and a genetic algorithm (GA). In addition we implement random weight guessing (RWG) to compare as a baseline.

CEM is a general optimization algorithm that has been used effectively as a policy search method on RL problems (Szita and Lőrincz, 2006). In our linear representation, the vector of weights constitute the policy parameters to be adapted. The objective function, or "fitness" function, to be maximized is the expected long-term reward accrued by following the greedy policy that is derived from the weights. An iterative algorithm, CEM maintains and updates a parameterized distribution over the multi-dimensional search space. During each generation, a population of *#pop* points is sampled from the current distribution. Each point is evaluated, and the μ points with the highest fitness values are used to determine the distribution parameters for the next generation. The update rule is such that with time, the variance of the distribution shrinks and its mean gravitates towards regions of the parameter space with high fitness values. As is a common choice, in our experiments, we use a Gaussian distribution to generate sample points. We initialize the mean of this distribution to be the zero vector; along each dimension, the variance is set to 1 (with no covariance terms). The update rule for Gaussian distributions is such that at every generation, the updated distribution has as its mean and variance, the sample mean and variance of the μ selected points (independently for each parameter). In general the update can also depend on the current distribution's mean and variance; further, noise can be added to the variance at each generation to prevent premature convergence (Szita and Lőrincz, 2006). We do not implement these variations in our experiments as they do not have an appreciable effect in our domain.

Like CEM, the CMA-ES method also employs the principle of updating a distribution at each generation to maximize the likelihood of the μ points with the highest fitness values being generated. However, unlike CEM, CMA-ES tracks covariances across dimensions and actively monitors the search path in the parameter space leading up to the current generation. Handling several aspects in the search procedure, CMA-ES is a fairly sophisticated optimization technique (rather than furnish a detailed explanation here, we refer the reader to descriptions from Hansen (2009) and Suttorp et al. (2009)). Yet, we find it surprisingly straightforward to implement the algorithm based on existing code, which automatically sets most of the method-specific parameters.[3] We set the initial distribution identically to the one set under CEM.

We implement GA in a manner akin to CEM and CMA-ES. On each generation, we spawn and evaluate *#pop* policies; of these, the μ with the highest fitness values are selected to generate the next population. Specifically pairs are chosen uniformly at random from the selected μ and crossed over to produce two offspring each. Policies are real-valued vectors over the space of parameters searched. Each parameter, restricted to the interval $[-1, 1]$, is represented using a 32-bit Gray-coded string. To implement crossover between two individuals, the bit strings corresponding to each parameter are cut at a random location and matched across individuals, thereby yielding two offspring. To implement mutation, individuals are picked from the population with a small probability (0.05), and once picked, have each bit flipped with a small probability (0.1). Both under CEM and GA, we set μ, the number of policies selected every generation to seed the next, to 15% of the population size *#pop*. Experiments suggest that these methods are not very sensitive to μ values in this vicinity. CMA-ES uses a default value for μ depending on *#pop*. Under all three methods, we return the policy that registers the highest empirical performance (across all generations) as the output of learning.

In general, PS methods can work with a variety of representations. An illustrative example is the PS framework implemented by Kohl and Stone

[3]See: `http://www.lri.fr/~hansen/cmaes_inmatlab.html`.

(2004) to optimize the forward walking speed of an Aibo robot. The gait they design has parameters describing trajectory positions and timings, which are combined using manually designed sets of rules. In order to maintain a fair comparison with the VF methods in this study, we enforce that the methods chosen from PS employ the same representation, under which real-valued parameters are to be optimized (Section 3.1.3). In principle numerous evolutionary and optimization techniques apply to this problem: among others, amoeba, particle swarm optimization, hill climbing, simulated annealing, and several variants of genetic and "estimation of distribution" algorithms. The reason we choose CEM and CMA-ES in our comparison is due to the several successes these methods have achieved in recent times (Szita and Lőrincz, 2006, 2007; Hansen et al., 2009), which partly owes to their mathematically-principled derivation. We implement GA on the grounds that although it optimizes exactly the same parameters, it employs a bit string-based *internal* representation during its search, and thus is qualitatively different. Note that all the methods described above only use the ranks among fitness values in a generation to determine the population in the next generation. In this manner, these methods differ from canonical policy gradient methods for RL (Sutton et al., 2000; Baxter and Bartlett, 2001; Kakade, 2001), which rely on the gradient of the value of a policy with respect to the policy parameters to identify a direction for policy improvement. Since it is deterministic, our policy is not analytically differentiable.

The three PS methods described above each take two parameters, listed in Table 3.3. Since fitness is defined as the expected long-term reward accrued by a policy, we estimate it by averaging the returns from $\#trials$ episodes. The other method-specific parameter, $\#gens$, is the number of generations

Table 3.3: Summary of parameters used by methods from PS. The last column shows the ranges over which we tune each parameter. The range shown for *#trials* is used when the total number of episodes is 50,000, as in a majority of our experiments (see Section 3.3). The range is scaled proportionately with the total number of training episodes. Under RWG, *#trials* is the only method-specific parameter.

Parameter	**Controls:**	**Range**
#trials	Samples per fitness evaluation	$[25, 250]$
#gens	Generations	$[5, 50]$

undertaken during the learning period. As a consequence, note that if the total number of training episodes is U, the population size in each generation is given by $\frac{U}{\#trials \times \#gens}$. Under RWG, we repeatedly generate policies, evaluate each for *#trials* episodes, and retain the policy with the highest fitness. Thus, *#trials* is its only method-specific parameter. Informal experimentation shows that for RWG, it is more effective to sample policies based on a Gaussian distribution for each parameter, rather than a uniform distribution.

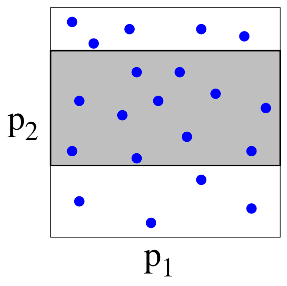
3.3 Experiments and Results

In this section, we present experimental results. First, in Section 3.3.1, we describe our experimental methodology. In Section 3.3.2, we perform comparisons *within* the VF and PS classes to pick representative methods from each. These representative methods are further compared across a series of experiments in Sections 3.3.3 through Sections 3.3.6 to ascertain their interactions with parameters of the learning problem.

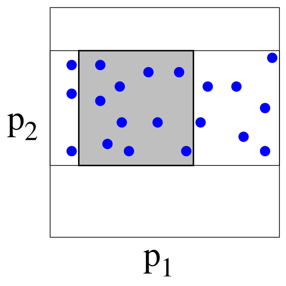
3.3.1 Experimental Methodology

As defined in Section 3.1, a learning problem is fixed by setting s, p, σ, χ, w, and a random seed. Additionally before conducting an experiment, we fix U, the total number of learning episodes conducted. Recall from tables 3.2 and 3.3 that learning methods themselves have parameters: λ, α_0, ϵ_0, θ_0, $\#trials$, and $\#gens$. In some experiments, we study the learning performance at fixed values of these method-specific parameters. However, note that for a given method (say Sarsa), its best performance at different problem settings will invariably be achieved under different settings of its method-specific parameters (λ, α_0, ϵ_0, θ_0). In response we conduct a search over the method-specific parameter space (4-dimensional for Sarsa) to find a configuration that yields the highest learned performance for a given problem instance and number of training episodes. The search procedure, illustrated schematically in Figure 3.4 for a 2-dimensional parameter space, involves evaluating a number of randomly generated points in the space and iteratively halving the search volume, always retaining the region with the highest performance density. The procedure is necessarily inexact due to stochasticity in evaluations, and since performance might not be smoothly varying over the region searched. Yet in practice we find that with sufficient averaging (2000 points per generation) and enough splits (5 times the number of dimensions searched), the procedure yields fairly consistent results.

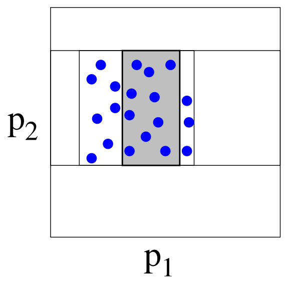
We suffix the method-specific parameter configurations returned by the search "$*$" to indicate that they have been optimized for some task setting and number of training episodes. Thus, Sarsa* refers to an instance of Sarsa identified through the search procedure, its parameters being λ^*, α_0^*, ϵ_0^*, and θ_0^*. Under Sarsa$(\lambda)^*$, λ is fixed, and only α_0, ϵ_0, and θ_0 are optimized.



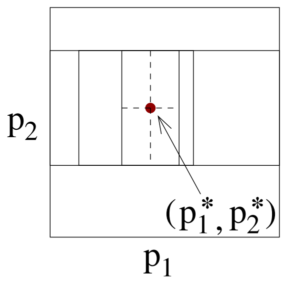
(a) Stage 1



(b) Stage 2



(c) Stage 3



(d) Final solution

Figure 3.4: Illustration of a search over two method-specific parameters, p_1 and p_2, to optimize the performance achieved after learning for a given number of episodes on a given task. Initial ranges for each parameter are specified as inputs to the search. To begin, points are sampled uniformly from within the specified ranges. At each sampled point, a *single* learning run is conducted and the performance of the final policy it returns evaluated. Subsequently a split is performed to halve the search volume, retaining an axis-aligned region with the highest density of performance among all such regions. The process is repeated several times: with each split, attention is focused on a smaller part of the search space empirically found to contain the most successful learning runs. Note that at each stage, any parameter could lead to the best split (p_2, p_1, and p_1 at stages 1, 2, 3, respectively, in the illustration). At termination, the midpoint of the surviving volume is returned.

For clarity, below we enumerate the sequence of steps undertaken in each of our experiments.

1. We fix learning problem parameters s, p, σ, χ, and w.
2. We fix the total number of training episodes U.
3. Either we manually specify an instance of a learning method, or search for one, as described above, to maximize performance for the problem parameters and number of training episodes set in steps 1 and 2.
4. With the chosen method instance, we conduct at least 1,000 independent trials of learning runs. Each trial is fixed by setting a different random seed, which can generate additional seeds for the learning problem (to determine features and rewards) and the learning method (to explore, sample, etc.).
5. Each learning trial above results in a fixed policy. We estimate the performance of this policy through 1,000 Monte Carlo samples. (Although sometimes a policy can be evaluated exactly through dynamic programming, the presence of state aliasing and generalization make it necessary to estimate performance through sampling.) Note that methods from VF and PS are both evaluated based on a greedy policy with respect to the learned weights.
6. Since all the rewards in our parameterized learning problem are non-negative, we find that problems with larger state spaces invariably lead to policies with higher absolute rewards. To facilitate meaningful comparison across problems with different parameter settings, we scale the

performance of a policy such that 0 corresponds to the value, under the same settings, of a random policy, and 1 to that of an optimal policy. In our graphs, we plot this normalized performance measure. Note that our careful design of the task MDP allows us to compute the performance values of random and optimal policies at each setting, even if the settings themselves might preclude the *learning* of optimal behavior by an agent. Policies that are "worse than random" have normalized performance values less than zero.

7. We report the normalized performance achieved (over all trials), along with one standard error (typically these are small and sometimes difficult to distinguish visually in our graphs). Note that standard errors do not apply to the results of our parameter search, such as to find λ^* under some problem instance. For any task instance, the method-specific parameter search is conducted exactly once.

In summary: the steps outlined above aim to provide each method the best chance of success for a given problem instance and training time, and then to fairly evaluate and compare competing methods. Having specified our methodology, we proceed to describe the results from our experiments.

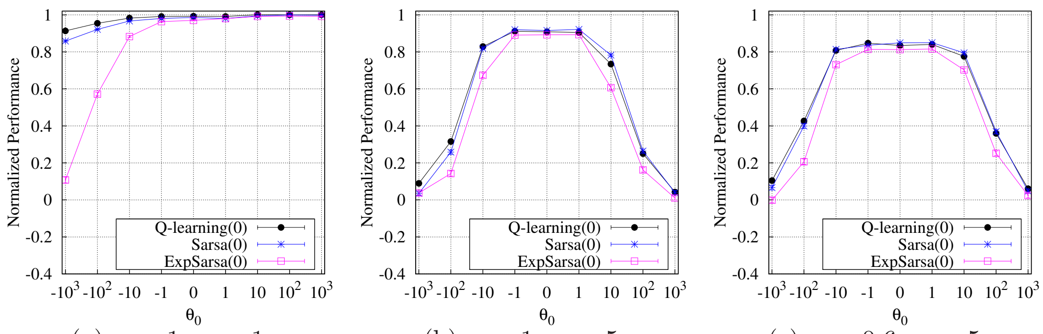
3.3.2 Picking Representative Methods, Setting Training Period

The first phase in our experiments is to pick representative learning methods from the VF and PS classes. We now present comparisons among methods from these classes (sections 3.3.2.1 and 3.3.2.2). We also describe how the number of training episodes for learning runs is set in our experiments (Section 3.3.2.3).

3.3.2.1 Picking a Representative Method from VF

In comparing methods from the VF class, we observe that the method-specific parameter with the most dominant effect on performance is the setting of initial weights, θ_0. For illustration consider Figure 3.5. In the experiments reported therein, we compare Sarsa(0), ExpSarsa(0), and Q-learning(0). For all these methods, we find that a broad range of the parameters α_0 and ϵ_0 yield policies with high performance; we manually pick favorable settings from among these ranges. Q-learning(0) and Sarsa(0) use $\alpha_0 = 0.8$, $\epsilon_0 = 0.8$, while ExpSarsa(0) uses $\alpha_0 = 0.8$, $\epsilon_0 = 0.2$. The total number of training episodes U is set to 50,000.

The three methods show qualitatively similar patterns in performance as θ_0 is varied. In Figure 3.5(a), we find that all of them achieve near-optimal behavior at large settings of θ_0, directly reflecting the merits of optimistic



(a) $\chi = 1$, $w = 1$. (b) $\chi = 1$, $w = 5$. (c) $\chi = 0.6$, $w = 5$.

Figure 3.5: [$s = 10$, $p = 0.2$, $\sigma = 0$.] Plots showing the effect of the initial weights θ_0 on the performance of on-line value function-based methods. Note the irregular spacing of points on the x axis. Plot (a) corresponds to an exact tabular representation with no generalization. Generalization is introduced in (b) by increasing w; additionally the expressiveness χ is reduced in (c).

initialization (Even-Dar and Mansour, 2001). Action values tend to lie in the range $[0, 20]$; correspondingly we notice that "pessimistic" initialization of weights to lower values leads to noticeable degradation in performance. Note that the settings in Figure 3.5(a) correspond to a fully expressive tabular representation with no generalization. As we introduce generalization by increasing w to 5 (Figure3.5(b)), we observe a significant change in trend: both very high and very low initial weights lead to a marked decrease in the final performance. This trend persists as the expressiveness χ is reduced (Figure 3.5(c)).

In figures 3.5(b) and 3.5(c), it is apparent that ExpSarsa(0) falls below Sarsa(0) and Q-learning(0) at most settings of θ_0. We posit that since it performs a weighted average over all next state-action values, updates under ExpSarsa(0) are likely to propagate error from state-actions that are encountered less frequently. For a perfect tabular representation, such as in Figure 3.5(a), van Seijen et al. (2009) prove that ExpSarsa(0) updates have the same bias, but a lower variance, compared to updates under Sarsa(0). However, our results appear to suggest that when generalization is present (as w is increased), and learning starts with a stronger initial bias (by setting θ_0 farther away from the true action values), ExpSarsa(0) suffers more from the error in its updates. Extending this argument, we could expect ExpSarsa, based on its learning update, to perform worse at high values of α_0 and ϵ_0 when generalization is used. Shortly we present the evidence for such a phenomenon.

We design three *problem* instances to further investigate differences between Sarsa, ExpSarsa, and Q-learning. Table 3.4 summarizes these problem instances. Instance I_1 corresponds to a fully-expressive tabular representation with no generalization, under which all three methods enjoy provable convergence guarantees. Expressiveness is reduced and generalization introduced in

Table 3.4: Parameter settings for illustrative problem instances I_1, I_2, and I_3.

Problem instance	**s**	**p**	$\boldsymbol{\chi}$	**w**	$\boldsymbol{\sigma}$
I_1	10	0.2	1	1	0
I_2	10	0.2	0.5	7	0
I_3	10	0.2	1	1	4

I_2. While I_1 and I_2 are both devoid of state noise, I_3 is identical to I_1 except for its higher setting of σ.

Figure 3.6 plots the performance of Sarsa*, ExpSarsa* and Q-learning* on I_1, I_2, and I_3. Notice that under I_1, all the methods achieve near-optimal behavior at the end of 50,000 episodes of training. While optimal behavior is not to be expected under I_2, it becomes apparent that ExpSarsa* trails the other methods in this problem (p-value $< 10^{-4}$).[4] This finding parallels the inference we draw from Figure 3.5: generalization adversely affects ExpSarsa, as its learning updates propagate more bias than either Sarsa or Q-learning.

Recall that I_3 is identical to I_1 except that it features state noise. Thus, when compared with I_1, we observe that all three methods suffer a significant drop in performance under I_3. Yet, the introduction of state noise does not appear to disadvantage any of the methods more than the others. Table 3.5 reports the optimized method-specific parameters found by our search strategy under the three problem instances. From the table, we see that for all three methods, the values of λ^* found under I_3 are significantly higher than the

[4]Throughout this thesis, p-values are reported based on an unpaired two-tailed t-test.

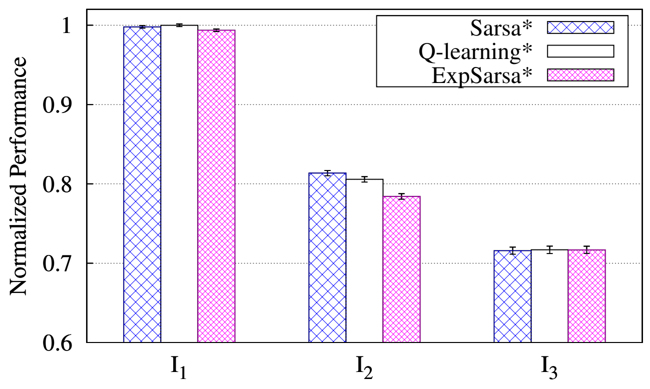


Figure 3.6: Comparison of the performance of different VF methods on the three problem instances from Table 3.4. Under each instance, and for each of the methods—Sarsa, Q-learning, and ExpSarsa—a systematic search (see Section 3.3.1) identifies the method-specific parameter settings (α_0, ϵ_0, θ_0, and λ) yielding the highest performance after 50,000 episodes of training. The methods are marked "$*$" as they are run under these optimized parameter settings.

values found under I_1 and I_2. We may infer that reducing the reliance on bootstrapped estimates (by setting high values of λ) counteracts the error introduced in TD updates due to state noise. We also observe from Table 3.5 that the θ_0^* values found by our search strategy for each method and problem are as one may expect based on Figure 3.5. These results affirm the reliability of our search strategy.

Predominantly we find that the VF methods compared above are not very sensitive to the learning rate parameter α_0 and the exploration parameter ϵ_0 within the ranges in which we optimize them: $[0.1, 1]$ for both parameters. The only significant exception, to which we alluded earlier, is the case of ExpSarsa under I_2, which strongly favors lower α_0 and ϵ_0 settings. For reference we provide graphs plotting the performance of VF methods as a function of α_0 and ϵ_0 in Appendix A.1.

Table 3.5: For each of three methods—Sarsa, Q-learning, and ExpSarsa—the method-specific parameters yielding the highest performance (at 50,000 episodes of training), under problem instances I_1, I_2 and I_3. Figures are rounded to one place of decimal.

Problem	**Sarsa***				**Q-learning***				**ExpSarsa***			
instance	λ^*	α_0^*	ϵ_0^*	θ_0^*	λ^*	α_0^*	ϵ_0^*	θ_0^*	λ^*	α_0^*	ϵ_0^*	θ_0^*
I_1	0.4	0.4	0.5	9.0	0.4	0.4	0.7	9.4	0.4	0.4	0.5	3.9
I_2	0.2	0.6	0.8	0.4	0.2	0.7	0.8	0.4	0.2	1.0	0.1	0.3
I_3	0.8	0.5	0.8	5.6	0.8	0.6	0.7	8.6	0.8	0.5	0.5	7.2

In summary: we find that Sarsa and Q-learning (albeit with a "naïve" implementation of eligibility traces) perform equally well on all our experiments; both methods outperform ExpSarsa on problems in which generalization is employed. Arguing that Sarsa and Q-learning would continue to register quite similar performance in our subsequent experiments (see sections 3.3.3 through 3.3.6), we break ties arbitrarily to pick Sarsa as a representative method from the VF class.

3.3.2.2 Picking a Representative Method from PS

We reuse problem instances I_1, I_2, and I_3 to compare methods from the PS class. As noted in Section 3.2.2, two parameters have to be set for methods from this class: *#trials* and *#gens*. Optimizing over these parameters, we plot the performance of CEM*, CMA-ES*, GA* and RWG* in Figure 3.7. Unlike with the VF class, the ordering among the methods from PS stays consistent across the problem instances. In all cases, CEM* and CMA-ES*

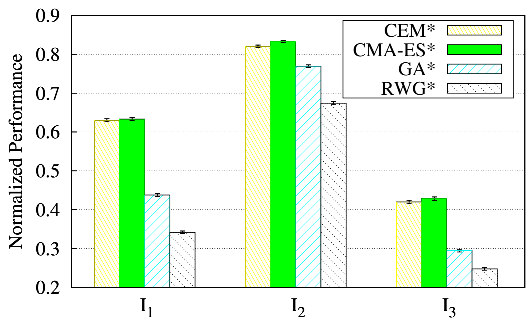


Figure 3.7: Comparison of the performance of different PS methods on the three problem instances from Table 3.4. Methods are marked "$*$" to denote that method-specific parameters—$\#trials$ and $\#gens$ (except $\#gens$ for RWG)—have been optimized for each task instance.

outperform GA* and RWG* (p-value $< 10^{-4}$). However, CEM* and CMA-ES* themselves register virtually identical performance: they cannot be separated with statistical significance (p-value < 0.05) on instances I_1 and I_3, although on I_2, CMA-ES* emerges the winner (p-value < 0.005).

It is worth noting that whereas all the VF methods in our study achieve their highest performance on instance I_1, all the methods from PS achieve theirs on I_2. 50,000 episodes is a relatively short duration of training for PS methods, which do not make effective use of individual transition samples, but rather, aggregate them in evaluating fitness. Greater generalization across the state space (as in I_2, where $w = 5$) enables them to learn more quickly. In Section 3.3.6, we observe that if optimized for 500,000 episodes instead, PS methods do perform better at $w = 1$.

The best parameter settings found for each PS method, under the three chosen problem instances, are listed in Table 3.6. Although we search over $\#trials$ and $\#gens$, note that thereby we implicitly set up a search over the population size $\#pop$ used in every generation. This is a consequence of the

Table 3.6: For policy search methods, the method-specific parameters yielding the highest performance (at 50,000 episodes of training) for problem instances I_1, I_2, and I_3. Figures are rounded to the nearest integer. Under CEM, CMA-ES, and GA, the parameters searched are $\#gens$ ("$\#g$") and $\#trials$ ("$\#t$"). These parameters automatically fix the population size ("$\#p$"), which is suffixed with "D" to denote that it is implicitly *derived*. Under RWG, only $\#t$ is optimized; $\#g$ is implicitly derived. Derived parameters values are shown for reference.

Problem	**CEM****			**CMA-ES***			**GA***			**RWG***	
instance	$\#g^*$	$\#t^*$	$\#p^*_D$	$\#g^*$	$\#t^*$	$\#p^*_D$	$\#g^*$	$\#t^*$	$\#p^*_D$	$\#g^*_D$	$\#t^*$
I_1	17	53	55	23	51	42	22	14	162	120	416
I_2	12	117	35	30	84	20	22	19	120	241	207
I_3	16	52	60	16	131	24	14	46	78	288	174

relation that $\#trials \times \#gens \times \#pop = U$, the total number of training episodes. In Table 3.6, we see that CMA-ES* typically has a smaller population size than CEM*, which itself has a smaller population size than GA*.

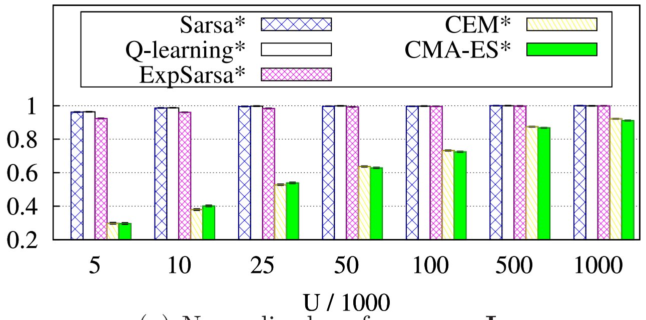
Appendix A.2 displays the performance of the various PS methods as a function of their input parameters. We observe a noticeable variance in the performance of all the methods over the parameter ranges considered. While CMA-ES* and CEM* have comparable performance on all three problem instances (I_1, I_2, and I_3), it is apparent that CMA-ES is more robust to parameter settings; that is, it registers a higher performance over a wider range of settings. This makes CMA-ES overall a slightly more favorable candidate than CEM to represent the class of PS methods. Therefore, we select CMA-ES for our further experiments.

3.3.2.3 Setting the Training Period

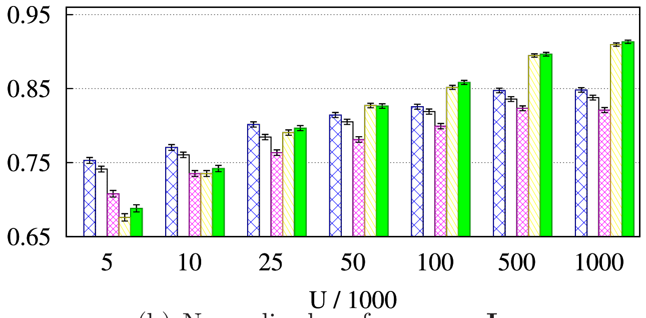
Even if the problems used in our experiments are themselves reasonably small, the extensive search and evaluation processes incur a significant amount of time during each experiment. One factor that plays a major role in determining the experimental running time is U, the total number of training episodes in each run. Setting $U = 50,000$, as we have in the experiments reported thus far, it takes us roughly 1-2 hours to complete a single search and evaluation procedure, such as, for example, identifying Sarsa* and evaluating it under I_2. In this duration, we have roughly 200 processes running in parallel on a computing cluster with 2GHz CPUs. In general we do not find it feasible to conduct extensive experimentation under higher values of U (although we do undertake such investigation under *some* interesting cases, such as in Section 3.3.6).

To gauge the implications of consistently setting $U = 50,000$ in our subsequent comparisons, we run a single suite of experiments at multiple settings of U. Figure 3.8 shows the performance of Sarsa*, Q-learning*, ExpSarsa*, CEM*, and CMA-ES*; under problem instances I_1, I_2, and I_3; optimized for various settings of U. As expected we find that all the methods improve their performance with longer training periods. The gains from a longer training period are more marked among the methods from PS, as in general, methods from VF appear to plateau within a few thousands of episodes.

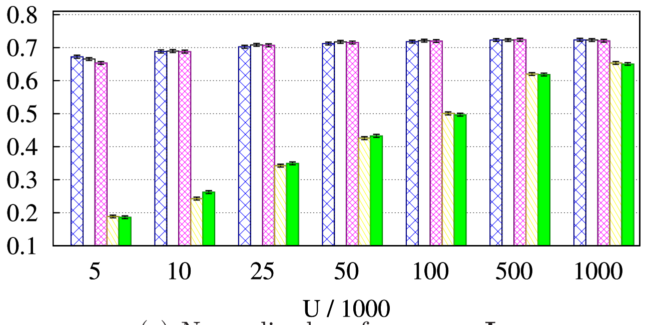
We observe from Figure 3.8 that under all problem instances, the trend *within* methods in VF remains roughly the same at all values of U: under I_1, the methods all achieve comparable performance; under I_2, ExpSarsa* performs poorest; and under I_3, Q-learning*. Likewise no outright winner among CEM* and CMA-ES* emerges in any of the instances, for any setting



(a) Normalized performance: $\mathbf{I_1}$



(b) Normalized performance: $\mathbf{I_2}$



(c) Normalized performance: $\mathbf{I_3}$

Figure 3.8: Performance of different learning methods as the number of training episodes U is varied. Each plot corresponds to a problem instance from Table 3.4. Note the irregular spacing of points on the x axis. At each point, the best performance achieved by three learning methods from VF (Sarsa*, Q-learning*, and ExpSarsa*) and two from PS (CEM*, CMA-ES*) is shown (key specified in plot (a)).

of U. Therefore, we may conclude that our choice of picking Sarsa* and CMA-ES* for further comparisons is justified. However, the choice of U does affect comparisons between Sarsa* and CMA-ES* themselves. Notice that up to 25,000 episodes, Sarsa* consistently outperforms CMA-ES*. Yet, from 50,000 episodes onward, CMA-ES* overtakes Sarsa* on I_2 (p-value < 0.008). Under I_1 and I_3, CMA-ES* narrows the margin with Sarsa* at $U = 1,000,000$, although it does not reach comparable performance.

The trends in Figure 3.8 inform our interpretation of the results to follow in the remainder of this section. In general we expect that Sarsa will not significantly improve its performance beyond 50,000 episodes of training, whereas CMA-ES consistently improves at least up to 1,000,000 episodes. Even so, in several problem instances, we find that CMA-ES outperforms Sarsa even at 50,000 episodes, validating this choice of U as a meaningful comparison point between the methods.

In summary: our "within class" comparisons in VF and PS provide convincing evidence that Sarsa and CMA-ES are respectively the best methods to represent these classes when being evaluated within our parameterized learning problem (except that Q-learning performs as well as Sarsa in VF). We now proceed to compare these methods as relevant problem parameters are varied. In each comparison (excepting cases in Section 3.3.6), the normalized performance of these methods after 50,000 episodes of training is considered while evaluating them.

3.3.3 Effect of Problem Size and Stochasticity

In our first set of "VF versus PS" experiments, we evaluate our learning methods as the size of the state space and the stochasticity of transitions in the task MDP are varied. Conjunctions of three settings of s (6, 10, 14) and three settings of p (0, 0.2, 0.4) are compared; results are plotted in Figure 3.9. With complete expressiveness ($\chi = 1$), no generalization ($w = 1$), and full observability ($\sigma = 0$), all nine cases are akin to learning with a classical "tabular" representation.

The most striking observation from the plots in Figure 3.9 is the disparity in the learning rates of Sarsa* and CMA-ES*. In all nine cases, Sarsa* reaches near-optimal behavior, and typically so within a few thousands of episodes. At 50,000 episodes of training, in none of the problems does CMA-ES* match the performance of Sarsa* (p-value $< 10^{-4}$). The gap between the methods is to be expected, as by making learning updates based on every transition, VF methods make more efficient use of experience for learning than PS methods do. Note that Sarsa is still on-line and model-free; we could expect model-based methods (Sutton, 1990) and batch methods (Lin, 1992; Lagoudakis and Parr, 2003) to further improve sample-efficiency.

Under both Sarsa* and CMA-ES*, we notice a decrease in performance as s is increased. This decrease is more marked for CMA-ES*, as the dimensionality of the parameter space it searches increases quadratically with s. The effect of the stochasticity parameter p in widening the gap between Sarsa* and CMA-ES* is also significant. The error bars plotted in the graphs show one standard deviation in performance (in all other graphs in the chapter, one standard error is shown). We observe that for both methods, the variance in performance increases as p is increased, and further, that for any given problem, CMA-ES* displays a slightly higher variance than Sarsa*. As described earlier in Section 3.1, note that even at $p = 0$, there is stochasticity in the task, as the start state for each episode is picked uniformly at random.

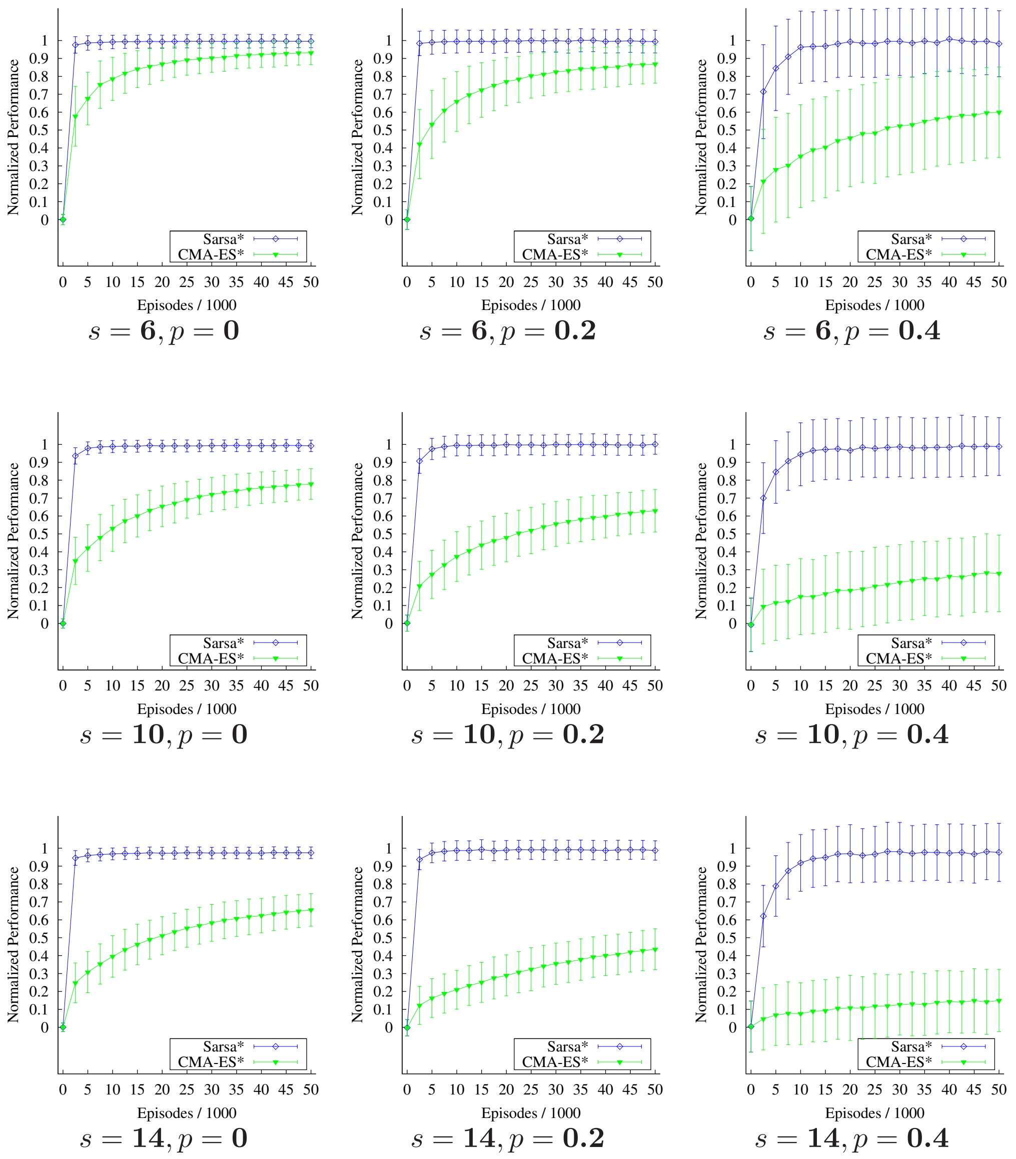


Figure 3.9: [$\chi = 1$, $w = 1$, $\sigma = 0$.] Sarsa* and CMA-ES* (optimized for 50,000 episodes of training) compared at different settings of s and p. Unlike other plots in this section, in these learning curves, we plot one *standard deviation* in the performance (instead of one standard error).

Recall that the method-specific parameters of Sarsa* and CMA-ES* have been optimized for each problem and training period. While we do not notice any significant patterns among the method-specific parameters thereby found under Sarsa*, we note that under CMA-ES*, $\#trials^*$ gets consistently higher as p is increased. For example, at $s = 10$, the settings of $\#trials^*$ found by our search procedure are 44, 70, and 166 for $p = 0$, $p = 0.2$, and $p = 0.4$, respectively. In other words, CMA-ES* benefits from more evaluation trials in evaluating fitness values as the task stochasticity increases. In Chapter 5, we consider how the total number of trials in a generation of CMA-ES (or other similar PS methods) can be minimized by allocating trials intelligently to different population members, rather than using the uniform allocation strategy employed here.

The primary inference from the set of experiments above is that Sarsa has significant advantages both in terms of the performance achieved and the variance in performance as problem size and stochasticity are increased. Not only is CMA-ES slower to learn, it demands better tuning of $\#trials$ across different problem instances. To characterize the reasons underlying these observations, we turn to Cobb (1992), who separates the inductive biases in a reinforcement learner into "language" and "procedural" biases. The former corresponds to the representation used by the learner, which in this study, we have fixed to be the same for the methods compared. VF and PS methods are essentially separated by their procedural bias: how they updates weights in the representation. The language bias in the problem instances above—$\chi = 1$, $w = 1$, $\sigma = 0$—strongly favors the procedural bias of Sarsa. How would the methods fare if the language bias is changed? The experiments to follow examine the effects of state noise, generalization, and expressiveness.

3.3.4 Effect of State Aliasing

In our second set of experiments, we study the effect of state aliasing by increasing σ. In so doing, we notice a conjunctive relationship between σ and w, the generalization width. In response we conduct experiments with three settings of σ (0, 2, 4) and three settings of w (1, 5, 9). Results are plotted in Figure 3.10: in each graph therein, the performance of Sarsa(λ)* is plotted at six values of λ (0, 0.2, 0.4, 0.6, 0.8, 1). The performance of CMA-ES* (which does not depend on λ) is also shown.

In general the best memoryless policies for Partially Observable MDPs (POMDPs) can be stochastic (Singh et al., 1994). Perkins and Pendrith (2002) show that in order to converge in POMDPs, it is necessary for methods such as Sarsa and Q-learning to follow policies that are *continuous in the action values*, unlike the ϵ-greedy policies used by the VF methods in our experiments. However, we do not observe any divergent behavior for Sarsa(λ) in the experiments reported here.

We notice that when $\sigma = 0$ and $w = 1$, the effect of λ on the performance of Sarsa(λ)* is not very pronounced. As soon as either σ or w is increased, intermediate values of λ predominantly yield the highest performance for Sarsa(λ)*. These results echo the findings of Loch and Singh (1998), who demonstrate that deterministic policies learned using Sarsa(λ) with ample exploration perform quite well on a suite of benchmark POMDPs. Key to their success is the high values of λ used (between 0.8 and 0.975), which weight true returns from actions much higher than estimated values.

As the generalization width w is increased, notice that there is no longer a single winner between Sarsa* and CMA-ES*: VF methods no longer dominate PS methods completely when state aliasing occurs. An intriguing trend

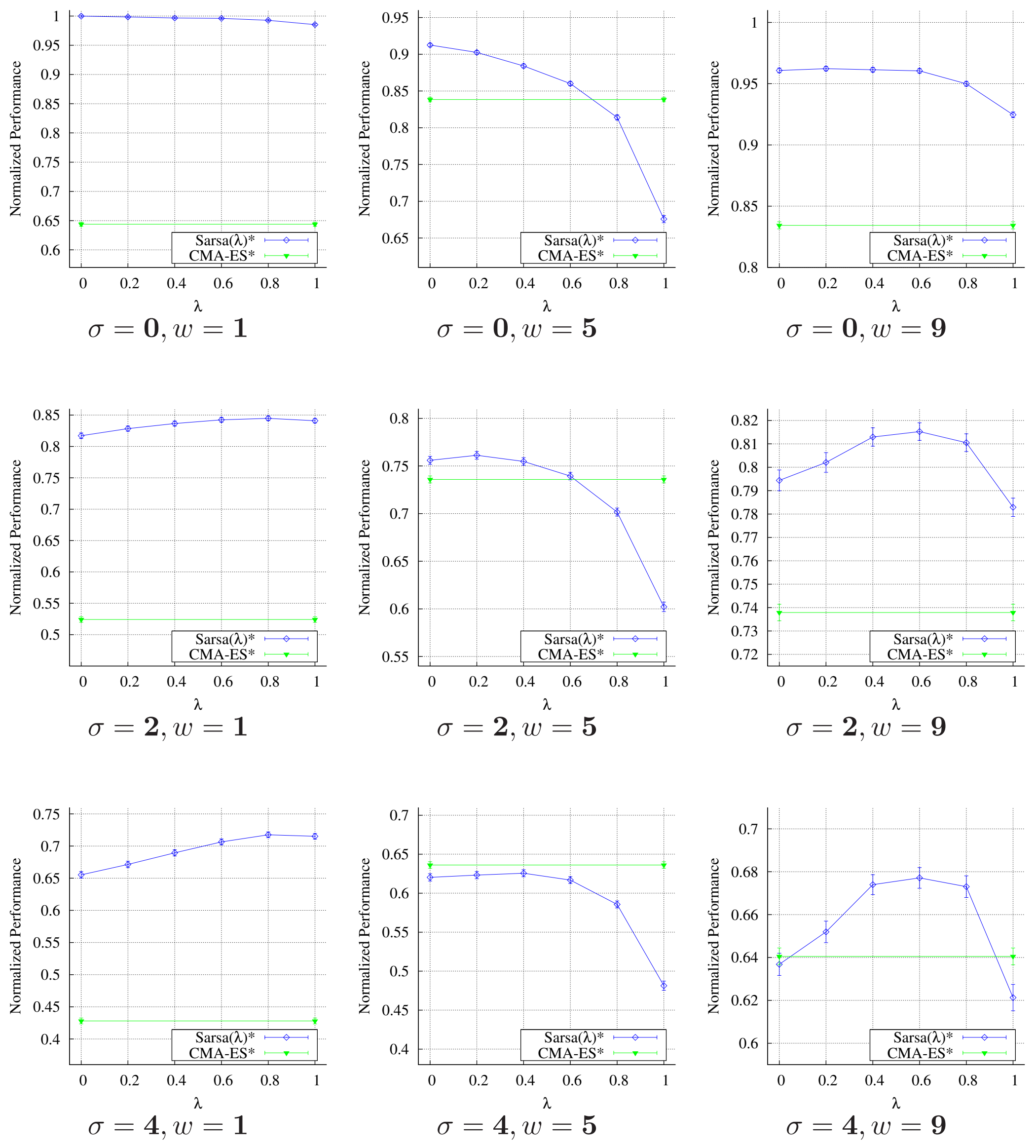


Figure 3.10: $[s = 10,\ p = 0.2,\ \chi = 1.]$ Sarsa$(\lambda)^*$ and CMA-ES* compared at different settings of σ and w. Under each plot, six regularly spaced values of λ are chosen and the corresponding Sarsa$(\lambda)^*$ evaluated. CMA-ES* appears as a line, as it does not depend on λ.

that becomes apparent from Figure 3.10 is that the performance of Sarsa(λ)* is not monotonic with respect to w: for most settings of σ and λ, the highest performance is achieved at $w = 1$, followed by $w = 9$, with the lowest performance at $w = 5$. In Section 3.3.6, we find further evidence of such anomalous patterns in the performance of Sarsa as w is varied. Interestingly CMA-ES* registers its highest performance, for any fixed σ, at $w = 5$ or $w = 9$. This trend arises as 50,000 episodes is a relatively short training duration for PS methods in this domain; generalization promotes quick initial learning. Experiments in Section 3.3.6 show that with more episodes of training, the performance of CMA-ES under $w = 1$ begins to catch up with its performance at higher settings of w.

A recent variant of Sarsa(λ) applied to POMDPs is SarsaLandmark (James and Singh, 2009), in which λ is set to 0 (full bootstrapping) when special "landmark" states (which are perfectly observable) are visited, but λ remains 1 at all other times (Monte Carlo). SarsaLandmark is not directly applicable in our domain as the agent receives no special information about landmark states. In recent work, Downey and Sanner (2010) propose a method to adaptively tune λ while learning. Formally derived under a Bayesian framework, their algorithm—Temporal Difference Bayesian Model Averaging (TD-BMA)—is shown to outperform Sarsa(λ) for *any* fixed value of λ on illustrative grid-world tasks. Our results highlight that tuning λ is of particular relevance in problems with state noise and generalization; our parameterized learning problem therefore becomes an ideal testbed for evaluating adaptive approaches.

In Table 3.7, we report the best initial weights, θ_0^*, found for Sarsa(λ)*, under various settings of λ, σ, and w. The most noticeable pattern from the

Table 3.7: [$s = 10$, $p = 0.2$, $\chi = 1$.] θ_0^* (initial weights under Sarsa(λ)*) for different problem instances. Each cell in the table corresponds to a setting of σ, w (problem parameters), and λ (Sarsa parameter); entries correspond to the value of θ_0 found by searching for Sarsa(λ)*. Note that each search is only performed once.

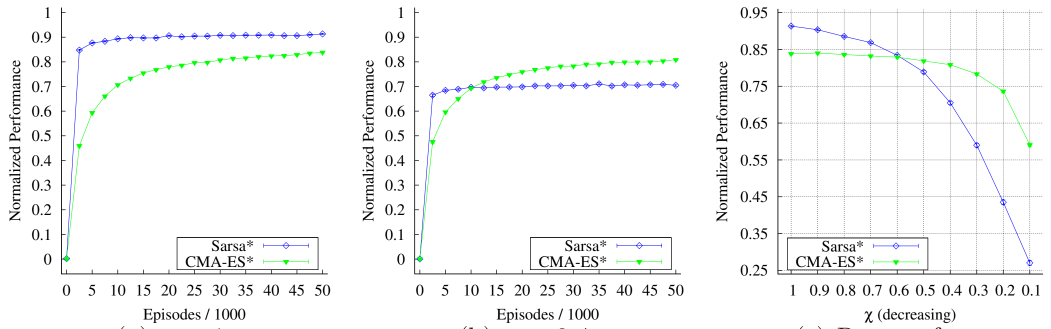
w	$\lambda = 0$			$\lambda = 0.4$			$\lambda = 1.0$		
	$\sigma = 0$	$\sigma = 2$	$\sigma = 4$	$\sigma = 0$	$\sigma = 2$	$\sigma = 4$	$\sigma = 0$	$\sigma = 2$	$\sigma = 4$
1	9.5	7.3	5.9	8.3	6.9	7.5	8.1	8.3	6.0
5	-0.5	-0.1	-0.4	0	-0.5	-1.2	2.0	1.8	2.2
9	0.1	-0.4	-0.1	0.1	0.3	-1.3	-0.1	-0.1	0

table is the favor for lower settings of θ_0 as w is increased. The best initial weights do not appear to change much as state noise and eligibility traces are varied.

3.3.5 Effect of Expressiveness of Generalization Scheme

Continuing our study, we conduct experiments to gauge the role of the expressiveness parameter χ in determining the performance of learning methods. Again, we find no single winner among Sarsa* and CMA-ES* as χ is varied. Results in Figure 3.11 are obtained under $\sigma = 0$ and $w = 5$; the qualitative nature of the results does not change as σ and w are varied.

In the learning curve in Figure 3.11(a), under $\chi = 1$ (which allows the optimal action value function to be represented), Sarsa* displays quick learning to reach a normalized performance close to 0.9, whereas CMA-ES* fails to achieve comparable performance after 50,000 episodes. By contrast,



(a) $\chi = 1$ (b) $\chi = 0.4$ (c) Range of χ

Figure 3.11: $[s = 10,\ p = 0.2,\ w = 5,\ \sigma = 0.]$ Plots (a) and (b) show learning curves of Sarsa(λ)* and CMA-ES* at different values of χ. Plot (c) shows the performance achieved after 50,000 episodes of training at different values of χ.

at $\chi = 0.4$ (Figure 3.11(b)), we notice that Sarsa* suffers a dramatic drop in performance, plateauing at a normalized performance value close to 0.7. At the same setting of χ, CMA-ES* overtakes the learning curve of Sarsa* and reaches a significantly higher performance at 50,000 episodes (p-value $< 10^{-4}$).

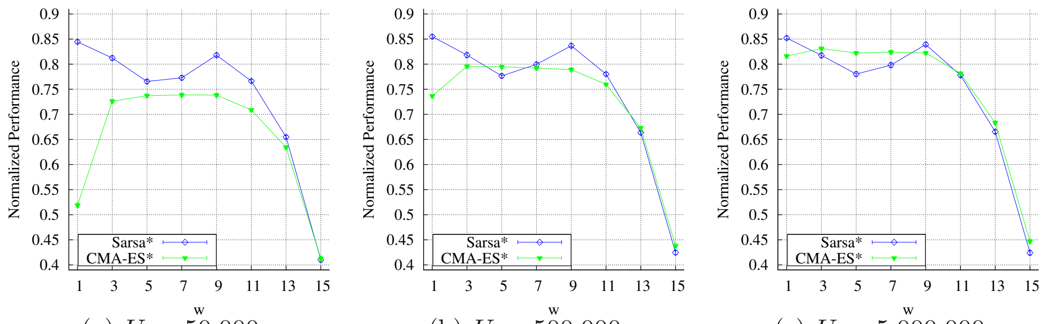
As χ is decreased, the representation for the value function and policy becomes increasingly handicapped. In Figure 3.11(c), we observe that both under Sarsa and CMA-ES, the performance achieved after 50,000 episodes of training decreases monotonically as χ is reduced. However, of the two methods, Sarsa suffers the more significant drop in performance as χ is reduced. Whereas Sarsa outperforms CMA-ES for $\chi \geq 0.7$ (p-value $< 10^{-4}$), the opposite is true when $\chi \leq 0.5$ (p-value $< 10^{-4}$). We do not observe any striking trends in the method-specific parameters of Sarsa* and CMA-ES* as χ is varied.

To the best of our knowledge, prior literature does not compare methods from VF and PS while constraining them to use the same representation. Our finding that CMA-ES is able to achieve good performance even under a representation that is extremely impoverished for approximating the value function suggests that it is a promising candidate in a large number of real-world domains in which feature engineering and representations are deficient. We posit that like the example constructed by Baxter and Bartlett (2001), many of the cases with $\chi < 1$ allow for the representation of high-reward policies, but only admit poor approximations of the action value function. Notice that we do not have any *irrelevant* features in our learning problem: in the future it would be useful to incorporate such a setting, which is often encountered in practice. Non-linear function approximation would be an equally important avenue to explore.

3.3.6 Effect of Generalization Width

In Section 3.3.4, we noted that the "width of generalization" parameter w plays a role in determining the relative order between Sarsa and CMA-ES at different values of σ. In examining the effect of w more closely, we notice that although its effect on CMA-ES is fairly regular, its interaction with Sarsa is less predictable. Figure 3.12 shows the normalized performance of these methods at settings of w varying from 1 (no generalization) to 15 (very broad generalization). The three plots in the figure correspond to progressively increasing settings for U, the total number of training episodes.

When trained for 50,000 training episodes, CMA-ES* performs its best at $w = 7$ (*not* statistically significant in dominating $w = 5$ and 9 at p-value < 0.05, but statistically significant in dominating $w = 1, 3, 11, 13,$ and 15 at p-value < 0.02). However, after 500,000 episodes of training, CMA-ES* registers its best performance at $w = 3$ (indistinguishable from $w = 5, 7,$ and 9 at p-value < 0.05, but dominating $w = 1, 11, 13,$ and 15 at p-value $< 10^{-4}$).



(a) U = 50,000 (b) U = 500,000 (c) U = 5,000,000

Figure 3.12: [$s = 10$, $p = 0.2$, $\chi = 1$, $\sigma = 2$.] Performance of Sarsa* and CMA-ES* at different values of w, optimized for (a) 50,000, (b) 500,000, and (c) 5,000,000 training episodes.

It is apparent from Figure 3.12(c) that CMA-ES continues to improve its performance even at 5,000,000 episodes of training, with its performance at $w = 1$ catching up with the plateau visible across $w = 3, 5, 7,$ and 9. We ascribe the pattern in the performance of CMA-ES with respect to w and U to the benefits of generalization early during the search—in quickly identifying the most promising actions in localized regions of the state space. It is not surprising that both for Sarsa and CMA-ES, the performance begins to drop sharply for $w > 9$. Since in this problem, $s = 10$, some non-terminal cells in the task MDP necessarily get activated by *all* the tiles present if the tile width exceeds 9. Indeed beyond $w = 19$ (not shown in figure), no two cells in the MDP remain distinguishable.

Interestingly Sarsa* presents a less regular pattern in performance as w is varied, as evinced by Figure 3.12(a). We find that Sarsa* is most effective at $w = 1$, but its performance suffers a dip until $w = 5$; again a rise until $w = 9$; before monotonically decreasing again. 50,000 episodes is a fairly long duration by VF standards, as apparent from several learning curves shown in the chapter (for instance, see Figure 3.11(a)). It is clear that the irregular performance pattern of Sarsa* is not an artefact of training time, as the pattern essentially persists up to 5,000,000 episodes of training (figures 3.12(b) and 3.12(c)). Also, notice the small error bars in the plots: the pattern is systematic.

We investigate whether the irregular pattern in the performance of Sarsa persists as the problem size is increased beyond $s = 10$. Figure 3.13 (see top row) indeed affirms that at $s = 14$ and $s = 18$, too, multiple local minima emerge in the performance as w is varied. Curiously, under all three settings of s, we observe that as w is varied, a correlation exists—up to $w < s$—between the performance of Sarsa* and λ^*, the eligibility trace parameter optimized for each problem setting. The bottom row in Figure 3.13 shows the values of λ^* under each setting. Observe that for $w < s$ the local maxima and minima in λ^* predominantly coincide with those of the normalized performance (recall that for $w \geq s$, states necessarily become aliased, and so it is not surprising to find no apparent correlation between the performance and λ^*).

At present we do not have a conclusive explanation for the phenomenon described above. Since CMA-ES shows predictable variation with w, we surmise that the variation shown by Sarsa ultimately arises from its on-line updates to the value function. We speculate that "edge effects" in our tiling scheme, whereby states on the periphery of the grid have fewer neighbors, might induce irregular patterns in the trajectory taken by the value function. Nevertheless, closer inspection would be necessary to fully explain such behavior, which at present remains rather intriguing.

In the context of kernel-based methods, Ormoneit and Sen (2002) discuss the "bias-variance tradeoff" induced by generalization widths. Munos

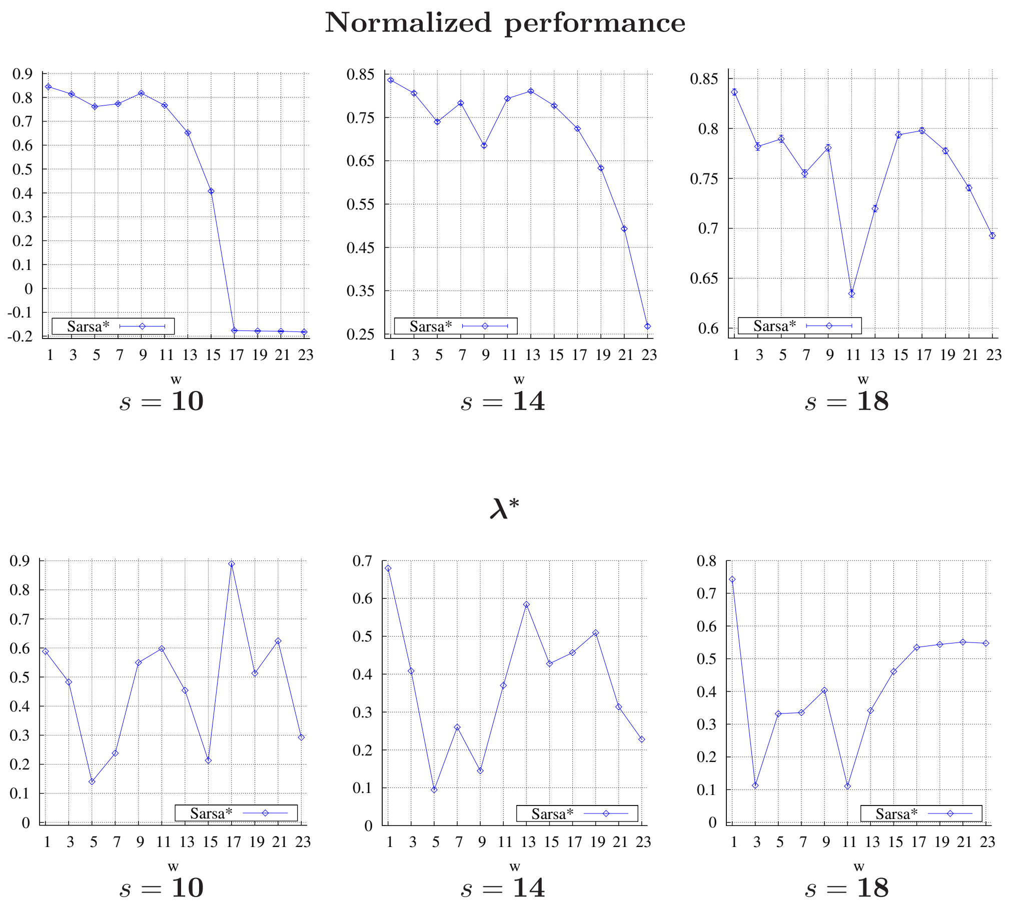


Figure 3.13: [$p = 0.2$, $\chi = 1$, $\sigma = 2$.] Analysis of Sarsa* as s (columns) and w (x axis in each plot) are varied. The top row shows the normalized performance achieved at each setting; correspondingly the bottom row shows λ^*—the value of λ found by searching for Sarsa*—for the same settings.

and Moore (2002), and Sherstov and Stone (2005), devise schemes for setting different generalization widths in different parts of the state space. Our parameterized learning problem becomes a valuable testbed to evaluate this line of work, which our results hint needs attention.

3.4 Discussion

The extensive suite of experiments reported in Section 3.3 uncovers several interesting patterns characterizing the interaction between problem parameters and method-specific parameters in the context of sequential decision making from experience. Overall, it is clear that there is no easy "one-size-fits-all" solution to the problem of picking or devising learning methods to work in tandem with different representational settings. In this section, we highlight some of the main questions brought to relevance by our study.

Generalization. Our results consistently indicate that generalization significantly alters the landscape while evaluating learning algorithms, in particular those from VF. For instance, Section 3.3.2 presents conclusive evidence that ExpSarsa suffers more severely due to the bias introduced by generalization than either Sarsa or Q-learning. Section 3.3.6 brings into focus an irregular—yet systematic—pattern in the performance of Sarsa as the tile width w is increased. Interestingly this pattern is correlated with the best eligibility trace settings. To the best of our knowledge, generalization has not been given explicit attention in the context of PS methods. Our results show that generalization can benefit PS methods, too, when the training duration is short.

As motivated in Chapter 1, generalization is necessary in nearly every practical application of RL. Thus, the importance of understanding its effects on algorithms cannot be understated. In future work, the experimental framework introduced here can be used to probe more deeply into this subject.

Optimistic Initialization. Of special significance among the ramifications of learning with generalization is its effect on the common practice of optimistic initialization. Optimistic initialization (of action values) has long been employed as a mechanism to promote exploration. In the context of finite MDPs, elegant proofs of convergence of VF methods have been derived on the basis of optimistic initial values (Even-Dar and Mansour, 2001; Szita and Lőrincz, 2008). Grześ and Kudenko (2009) provide experimental justification, again on finite (or suitably discretized) MDPs, for schemes that refine the basic optimistic initialization framework.

Our experiments in Section 3.3.2 convey that optimistic initialization is only effective in the fully tabular case: for $w > 1$, the error introduced into TD updates by high action values invariably degrades the performance of VF methods. A question that arises in response is how we may initialize action values when learning *with* generalization. Research in this direction appears particularly relevant to algorithms that are guaranteed to reach fixed points under linear function approximation (Perkins and Precup, 2003; Sutton et al., 2009; Maei et al., 2010). Which reasonable strategies for setting initial weights would profit such methods the most?

Meta-learning and Algorithm Portfolio Design. "Meta-learning" (Vilalta and Drissi, 2002) is the enterprise of (1) characterizing the strengths and weaknesses of learning methods vis-à-vis problem characteristics, with the intent of (2) designing adaptive schemes that, given a problem, apply the method best suited for it (Brodley, 1995; Pfahringer et al., 2000). Similarly "algorithm portfolio" methods (Gomes and Selman, 2001) are those that rely on applying several candidate algorithms to a problem (either in series or in parallel) before identifying the most effective choice or combination. While existing work on the topics of meta-learning and algorithm portfolio design has largely been in the context of supervised learning and search (Leyton-Brown et al., 2003; Xu et al., 2008), the work described in this chapter is motivated by the meta-learning problem within sequential decision making.

Our experiments clearly show that there is a need for meta-learning within RL. For example, Figure 3.11(c) succinctly conveys that Sarsa outperforms CMA-ES when expressiveness in the generalization scheme is above a certain threshold, but that the opposite is true below the threshold. Indeed our experiments unearth several other strengths and weaknesses of methods within the VF and PS classes. Our results are validated on a parameterized learning problem that is specifically designed to implement a methodology for meta-learning within RL. We believe that this methodology can support the eventual development of effective algorithm portfolio designs for sequential decision making, which currently appears a rather formidable undertaking.

Automatic Parameter Tuning. While meta-learning operates at the *macro* scale of choosing between methods, our experiments also underscore the gains obtained at a *micro* scale by tuning method-specific parameters. In this work, we have employed a search technique to optimize method-specific parameters such as learning rates and population sizes. In practice an agent would need to automatically tune these parameters *while learning.* In the context of PS methods, it is worth repeating that we find existing code for CMA-ES quite adept in automatically setting and tuning several internal parameters in the algorithm. For VF methods, techniques for tuning learning rates (Sutton and Singh, 1994; George and Powell, 2006; Hutter and Legg, 2008) and eligibility traces (Downey and Sanner, 2010) have predominantly been derived and validated for the case of finite MDPs. Our parameterized learning problem serves as an excellent mechanism for prototyping adaptive schemes for more realistic settings involving state aliasing and generalization.

Learning and Representation. With the purpose of solely comparing the "learning" behavior of VF and PS methods—how they adapt a vector of weights—in this study, we have forced them to share a fixed, common representation. In particular we adopt a linear function approximation scheme, whose expressiveness and generalization width can be carefully controlled. Our results show that VF and PS methods dominate at different settings of these problem parameters, thereby affirming our research hypothesis.

While our approach makes for sound experimental methodology, it must be noted that in general the greatest success can be achieved by adapting the representation itself while learning (Whiteson and Stone, 2006a; Nissen, 2007). Indeed Cobb and Bock (1994) argue that representations favoring an expert agent might be unfavorable for an agent beginning to learn. Integrating learning with adaptive representation is yet another area of future work that our parameterized learning problem enables. In pursuing such work, we would treat χ and w as *internal* to the learning agent, rather than extraneous. The agent could potentially adapt these representational aspects by applying methods from feature selection (Kolter and Ng, 2009; Petrik et al., 2010), structure learning (Degris et al., 2006; Diuk et al., 2009) and manifold learning (Mahadevan, 2009).

In taking steps towards the automated application of RL methods to problems, the issues discussed above are all relevant to consider. We hope that future work will make progress along all these directions by extending the ideas presented in this chapter. For a start, the next three chapters to follow in this dissertation pursue some important questions arising from the experimental study presented here; below is a summary.

- Chapter 4 undertakes an investigation into *how* representations impose limitations on the performance of learning methods. In particular the chapter analyzes the game of Tetris as a case study, providing an explanation for the stark failure of VF methods to perform well in this task, given that PS methods have been rather successful.

- It is natural to expect that because they disregard potentially useful information contained in individual state transitions, PS methods will be slower to learn than VF methods, which indeed make learning updates based on atomic state transitions. This difference in performance is clearly visible in Figure 3.9. Chapter 5 considers one way to speed up PS methods such as CMA-ES by being judicious in allocating fitness evaluations among the population being evaluated (rather than allocating each member the same number of samples, as done in this chapter). Adopting a general, abstract view of the "subset selection" problem, Chapter 5 makes novel contributions to the literature of multi-armed bandits, applying the results to improve policy search methods.

- From Section 3.3.5, we clearly see that the inferior sample-efficiency of PS methods is compensated by their relative robustness in the face of poor representations. Can we develop a class of methods that is both sample-efficient and robust to inadequate representations? Chapter 6 presents two case studies in which the strengths of VF and PS methods are integrated.

We now proceed to the next section in this chapter, where we discuss related work in the context of parameterized learning problems and empirical evaluations in RL.

3.5 Related Work

In an early article, Cohen and Howe (1988) consider the strong coupling that exists in many disciplines of artificial intelligence between problem types and method instances. While formulating guidelines for the evaluation of methodological contributions to the field, they argue the need to precisely characterize the set of problems on which a method is expected to be successful, and symmetrically, the approaches that are likely to succeed on a given class of problems. As noted in the introduction to this chapter, Langley (1988) makes a similar observation in the specific context of machine learning.

Parameterized learning problems have been used in the literature to study the effects of factors such as dimensionality and noise. For example, Spall (2003) extensively uses the "Rosenbrock" function while comparing the performance of optimization algorithms in his textbook. The "Sphere" function discussed by Beyer (2000) has long served as a standard benchmark for evolutionary algorithms.

The work within the RL literature that is philosophically the closest to the contribution of this chapter is the notion of "generalized environments" proposed by Whiteson et al. (2011). Here, too, the authors argue against "environment overfitting", whereby methods tend to get evaluated on problems that favor them, but the broader scope of their applicability, especially that of their weaknesses, is not easy to gauge. A generalized environment represents a formally defined distribution of environments: the objective is to develop methods that perform well over the entire distribution. Whereas the motivation for generalized environments comes from realistic tasks such as helicopter control and Tetris, the apparatus developed in this chapter examines the performance of learning methods in a carefully-designed, controllable, abstract setting. Our results underscore that qualitatively different learning methods excel in different regions of the problem space, while teasing apart effects that method-specific parameters introduce.

The empirical approach we adopt here to characterize the interactions between learning problems and methods is complemented by theoretical formulations designed with a similar objective. Littman (1993) characterizes agents and environments based on the amount of memory they can use and the length of the horizon for which they seek to optimize rewards. He then considers several interesting classes of problems—for example, those with nonstationary environments—that fit within this formalization. Ratitch and Precup (2003) define environmental properties such as *state transition entropy* and *forward controllability*, and investigate how these properties bear on the exploration strategy of a learning agent. To the best of our knowledge, the issue of generalization has not been addressed through similar theoretical formulations.

The main difference between our comparative study and others in the RL literature is that our parameterized learning problem enables us to evaluate the effects of individual parameters while keeping others fixed. For example, in most related studies, methods typically use different generalization schemes, thereby introducing an additional qualification in any comparisons. Also, our formulation allows us to control problem parameters continuously along a scale from “high” to “low”; in the studies we shortly list, comparisons are typically between two or three distinct task settings. In this sense, this chapter answers the call put forth by Togelius et al. (2009) for parameterizable benchmarks, and affirms their basic conjectures on the strengths and weaknesses of "ontogenetic" (similar to VF) and "phylogenetic" (similar to PS) methods. In addition our results shed light on hitherto unexplored questions such as the effects of optimistic initialization when used in conjunction with generalization. We now list a number of studies comparing RL algorithms.

Moriarty et al. (1999) apply a suite of "Evolutionary Algorithms for Reinforcement Learning" (EARL) to a simple grid-world MDP, and compare results with Q-learning. Policies are represented using lists of rules or neural networks, which are evolved using standard genetic operators. The main conclusion of their study is that EARL is more suited to tasks with large state spaces (but represented compactly), tasks with incomplete state information, and tasks with nonstationary returns. Whiteson et al. (2010) undertake a comparative study between Sarsa(0) and NEAT (Stanley, 2004), a policy search method. These methods are compared on the benchmark tasks of Keepaway soccer (Stone et al., 2005) and Mountain Car (Sutton and Barto, 1998). Their findings are that sensor noise affects the final performance of Sarsa(0) more than NEAT, and indeed that stochasticity has the opposite effect, as policy evaluations under NEAT become more noisy.

Heidrich-Meisner and Igel (2008a) compare the "natural actor-critic" method with CMA-ES on a pole-balancing task. Both methods are "variable metric"; that is, they are insensitive to linear transformations of the parameter space. The methods achieve comparable results, but CMA-ES is found to be less sensitive to initial settings for the policy, which has a small number of parameters. Similar results are registered in the noisy Mountain Car task (Heidrich-Meisner and Igel, 2008b). Gomez et al. (2008) carry out an extensive suite of comparisons on single and double pole-balancing tasks (with hidden state). The methods compared are evolutionary algorithms such as CoSyNE, NEAT, ESP, and SANE, along with Q-learning, Sarsa, recurrent policy gradient, random weight guessing, and "Value and Policy Search" (VAPS) (Baird and Moore, 1999). The authors' findings reinforce the expectation that under the presence of state aliasing, evolutionary algorithms dominate model-free value function-based methods such as Q-learning and Sarsa. Heidrich-Meisner and Igel (2009b) conduct a further series of experiments on a similar suite of single and double pole-balancing tasks, on which they find CMA-ES significantly more successful than several competing policy search methods. The underlying representation used is a neural network; the authors find that networks without "bias" connections perform better than those with biases, likely because bias breaks the inherent symmetry in pole-balancing.

Lucas (2010) applies novel information-theoretic techniques for comparing evolutionary algorithms with TD learning methods. He shows that the theoretical *upper bound* on the information that can be learned by a TD learner after a certain number of environmental interactions far exceeds that of an evolutionary algorithm on the same problem (in his case a simple grid world task). Further, experiments demonstrate that indeed the "amount of learning" a TD method achieves in practice is of the same order as its theoretical upper bound; an evolutionary algorithm (CMA-ES), though, falls well short of its upper bound. These experimental results are obtained under a perfect tabular representation. Interestingly when function approximation is introduced (by way of a neural network or an interpolated table), the order between the performance of TD learning and CMA-ES no longer remains consistent. It does appear very promising to integrate the experimental approach adopted in this chapter with the theoretical ideas conceived by Lucas in further investigating the role of representations in sequential decision making. We comment further on this possibility in Chapter 7 (Section 7.2.1, page 257).

The recent thesis of van Hasselt (2011) shares many of the motivations of our current dissertation. After evaluating several learning algorithms (a majority of them TD learning methods) both analytically and experimentally, van Hasselt (2011, see Section 8.3) derives rules of thumb for choosing among these methods based on problem characteristics such as the size and continuity of the state and action spaces. The work in this chapter differs from van Hasselt's contribution mainly in that (1) we carefully vary representation quality (along the lines of state noise, expressiveness, and width of generalization) while evaluating methods; the dramatic effect of these representational aspects on learning methods makes (2) comparisons among qualitatively distinct classes of methods (VF and PS) our primary focus (even if we do perform intra-class comparisons in Section 3.3.2 to pick representatives). By contrast, van Hasselt's thesis takes a detailed look mostly at methods that learn (action) value functions, and does not dissect representational aspects as minutely as we do.

3.6 Summary

In this section, we summarize the contributions of this chapter, which serves as the groundwork for our dissertation. In particular the chapter undertakes an extensive experimental study to characterize the interactions between learning methods and the representations used for learning.

As a general framework to conduct empirical studies in RL, we introduce *parameterized learning problems*, in which the factors influencing the performance of a learning algorithm can be controlled systematically through targeted studies. The main merits of our experimental methodology are that (1) the designed task and learning framework are easy to understand; (2) we may examine the effect of subsets of problem parameters while keeping others fixed; (3) we can benchmark learned policies against optimal behavior; and (4) the learning process can be executed in a relatively short duration of time, thereby facilitating extensive experimentation. The purposefulness of parameterized learning problems—in getting to the heart of the phenomena being investigated—distinguishes them from the empirical studies on more realistic (and less controllable) problems.

Consistent with the theme of the thesis, we design a parameterized learning problem to evaluate the effects of state aliasing and generalization: representational aspects that are imperfect in a sizeable fraction of realistic RL tasks. On this problem, we evaluate various methods from the classes of on-line value function-based (VF) methods and policy search (PS) methods. Through a series of carefully-designed experiments, we obtain clear patterns separating the learning methods considered. A novel aspect of our study is a search procedure that enables us to find the best method-specific parameters (such as learning rates and population sizes) for a given problem instance. Largely made possible by the relative simplicity of our simulation, the search procedure uncovers interesting patterns relating problem instances and method-specific parameters.

Within the VF class, we find that Sarsa and Q-learning perform better than Expected Sarsa (ExpSarsa) when learning with generalization and function approximation. Within the PS class, we find that CMA-ES and the cross-entropy method (CEM) achieve significantly better performance than a genetic algorithm (GA); CMA-ES is more robust to its parameter settings than CEM. Comparing Sarsa (from VF) and CMA-ES (from PS), we find that the former enjoys a higher speed of learning, and also better asymptotic performance, when the learner is provided an expressive representation. On the other hand, CMA-ES is significantly more robust to severely deficient representations. Both methods suffer noticeably when state noise is added; their relative performance is additionally determined by the width of generalization in the representation. Our experiments highlight several promising lines of inquiry involving generalization, representation, meta-learning, initial weight settings, and parameter tuning.

It would be ideal for future work to extend the experiments reported in this chapter to other classes of learning methods, specifically model-based and batch methods, actor-critic architectures, and policy gradient techniques. A lack of time prevents us from incorporating these methods into the already extensive body of work contained in this chapter. In Chapter 7, we present a unifying view of several classes of learning methods against the backdrop of representation.

The next chapter continues our examination of the relationship between learning and representation. Adopting a more analytical approach, the chapter presents results seeking to *explain* how representations might impose limits on the performance of learning methods. Specifically this analysis is performed in the context of Tetris, the popular computer video game.

Chapter 4

Limits of Representation: An Illustration

This chapter continues the line of inquiry initiated in Chapter 3: how does representation affect the performance of different learning methods? In the previous chapter, recall that we carefully designed a parameterized abstraction of learning problems, on which we compared the performance of various VF and PS methods, while varying representation quality. By contrast, in this chapter, we focus on a concrete, well-known sequential decision making task: the popular computer video game Tetris.

Over the last two decades, numerous learning algorithms have been tested on Tetris. The most striking pattern among the results reported in previous studies is the dramatic gap that exists between the performance of VF and PS methods on this task. Whereas VF methods have, at best, learned policies that can clear a few thousands of rows in the Tetris game, PS methods, using similar representations, have been able to clear millions of rows. What explains this glaring disparity in the performance of these methods?

Several factors confound learning in Tetris: for example, high stochasticity and long episodes. In this chapter, we present experimental analysis showing that even if such factors were somehow to be circumvented, value function learning in Tetris is still likely to fail due to the fundamental limits imposed by the representation that has typically been used with this task (a linear architecture with a few tens of features). In this sense, the Tetris task and its accompany-

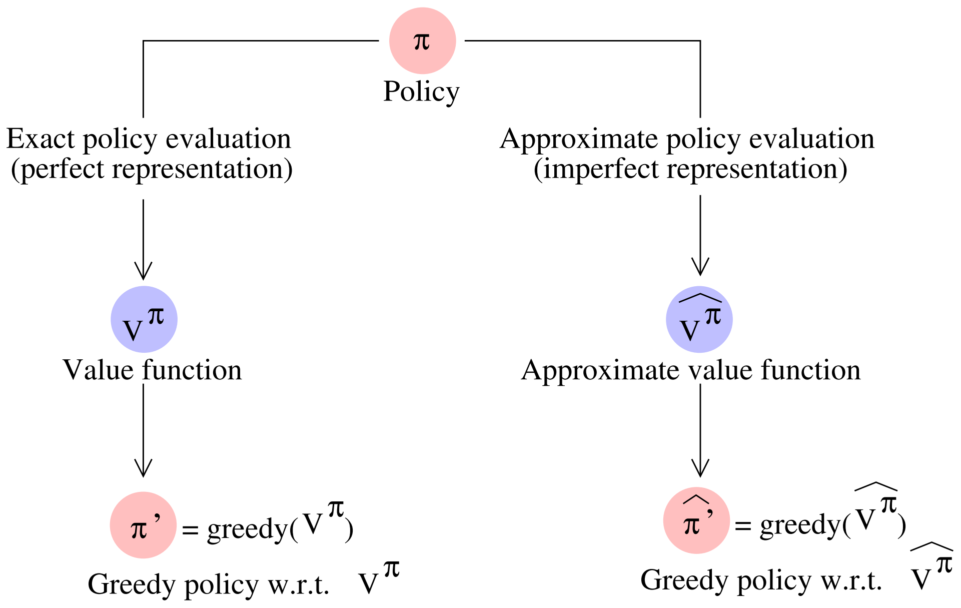
ing linear representation provide a stark example of representations strongly favoring one class of learning methods over another.

After describing the Tetris task in Section 4.1, we present a perspective view of previous work, highlighting the choices in picking learning methods and representations (Section 4.2). For one common choice of representation, we design experiments to ascertain the "best" approximation a VF method could possibly achieve in a policy evaluation setting (Section 4.3). Results indicate that even this best approximation would lead to a worsening of performance if the policy improvement operation were then applied, as under standard policy iteration. The results also provide ideas towards effectively combining VF and PS methods; we investigate these ideas further in Chapter 6.

In Chapter 3, we saw extensive evidence of representational aspects—expressiveness (χ), generalization width (w), and state noise (σ)—determining the dominance of one class of learning methods over another on a given MDP. In particular consider Figure 3.11 (see page 83), which clearly demonstrates that (1) Sarsa, a VF method, outperforms CMA-ES, a PS method, at high settings of χ, but (2) the opposite is true at low settings of χ. In the commentary following Figure 3.11, we speculated on reasons for the latter result. Specifically we hypothesized that low settings of χ were probably capable of representing reasonably good policies, but they could not support "good enough" approximations of value functions.

Now, it is true that acting greedily with respect to the value function V^{π} of a policy π will yield higher expected long-term reward than π itself accrues (or, if π is optimal, yield the same expected long-term reward) (Bellman, 1957). However, acting greedily with respect to $\widehat{V^{\pi}}$, an *inexact approximation* of V^π, could indeed result in *lower* expected long-term reward than π achieves. For clarity we depict this contrast between exact and approximate policy evaluation through a sketch (see Figure 4.1).

For a more concrete example of the phenomenon illustrated in Figure 4.1, we refer the reader to Baxter and Bartlett (2001, see Appendix A). In the example they construct, a linear function approximator (with one feature) is employed to approximate value functions in an MDP with two states



π
Policy
Exact policy evaluation (perfect representation)
Approximate policy evaluation (imperfect representation)
V π
Value function
V π
Approximate value function
π' = greedy(V π)
Greedy policy w.r.t. V π
π' = greedy(V π)
Greedy policy w.r.t. V π

Figure 4.1: Contrast between exact and approximate value functions. For a policy π, consider (left branch) its value function V^π and a policy π' that is greedy with respect to V^π; and (right branch) $\widehat{V^\pi}$ an approximation of V^π, and the corresponding greedy policy $\widehat{\pi'}$. Provably, we have $V(\pi') \geq V(\pi)$: that is, π' leads to at least as high expected long-term reward as π. However, due to the approximation error in $\widehat{V^\pi}$, indeed it could happen that $V(\widehat{\pi'}) < V(\pi)$. This chapter investigates the occurrence of this phenomenon when learning approximate value functions for Tetris.

and two actions. In particular the authors focus on approximating the optimal value function V^*, which is the value function of any optimal policy π^*. Among the functions that the linear architecture can represent, the authors show that $\widehat{V^*}$, the one providing the best approximation of the optimal value function, itself induces a sub-optimal policy.[1] Ironically, the same linear architecture still gives rise to an infinite number of other functions that do not approximate V^* nearly as well, but that still induce the optimal policy!

Any attempt to learn value functions in practical tasks is likely to suffer, to varying extents, the ailment of Baxter and Bartlett's example. How bleak are the consequences? On the one hand, enough practical successes of VF methods have been reported in the scientific literature (several among the applications listed in Table 1.1) to assure us that learning through value function approximation is a viable enterprise. On the other hand, one must take into account that examples of VF methods failing are less likely to be published in the literature on equal terms with cases where they succeed. Over the years, this author has heard anecdotally of numerous such negative examples from colleagues, and has personally borne witness to many. Can we scientifically characterize the reasons VF methods often fail to perform well? In this chapter, we consider a long-established failure case of VF methods, and examine if the failure indeed owes to the phenomenon highlighted in Baxter and Bartlett's example.

The example we study is the popular computer video game Tetris,

[1]Here approximation error is defined as the squared distance between the optimal and approximate value functions, weighted by the stationary distribution of states under the optimal policy. By this definition, $\widehat{V^*}$ would be the function to which, say, TD(1) would converge if evaluating the optimal policy with a linear representation. A policy "induced" by a function f over the state space takes actions greedily with respect to f.

which has served as a benchmark for evaluating RL methods for over a decade (Bertsekas and Tsitsiklis, 1996; Böhm et al., 2005; Szita and Lőrincz, 2006; Thierry and Scherrer, 2010a). Nearly every instance of learning applied to Tetris in the literature has employed a linear representation with a few tens of features. In the case of VF methods, this representation is used to approximate the value function. Under PS methods, the representation gives rise to an evaluation function (of afterstates), which, coupled with a simulation model, induces a corresponding policy. For the experiments in this chapter, we adopt the set of 22 features originally introduced by Bertsekas and Tsitsiklis (1996). With this fixed representation, we compare the following policies:

$\underline{\pi^{HC}}$, a hand-coded policy that clears about $1,000$ lines on average in a Tetris game, and

$\underline{\pi^{HC-vf-greedy}}$, a policy that is greedy with respect to the best approximation of the $V^{\pi^{HC}}$ among the functions our linear architecture can represent.

In defining "best" approximation in the latter case, it is typical to apply the squared error norm weighted by the stationary distribution of π^{HC} (Tsitsiklis and Van Roy, 1997). Nevertheless, other choices of distribution and norm apply, and could potentially induce better policies (Koller and Parr, 2000). In our experiments, we consider three natural distributions for weighting the squared error norm, and show that in all three cases, the resulting instance of $\pi^{HC-vf-greedy}$ performs significantly worse than π^{HC} (the best among the three clears 70 lines on average in a Tetris game). In short, these results indicate that the linear representation of Bertsekas and Tsitsiklis places a significant handicap on value function-learning in this task.

The remainder of this chapter is organized as follows. We begin with a concise description of the Tetris task in Section 4.1. For learning this task, several choices present themselves in terms of learning methods and representational aspects. In Section 4.2, we describe these possibilities through a survey of previous work on Tetris. For our experiments, we pick the linear representation of introduced by Bertsekas and Tsitsiklis; we evaluate the capacity of this representation to approximate value functions in Tetris. Experiments and results are presented in Section 4.3. A summary and discussion in Section 4.4 concludes the chapter.

4.1 Tetris: Task Description

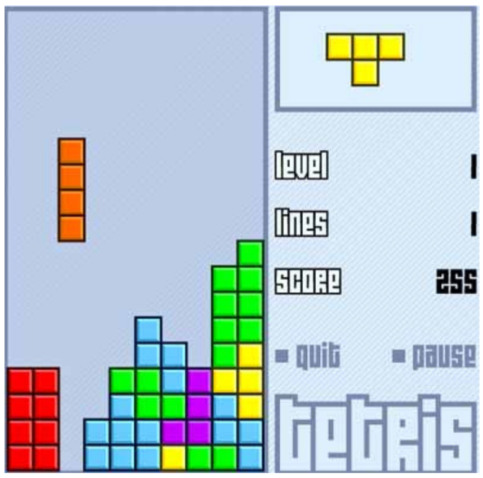
Tetris was a introduced in the mid-1980's as a video game, and it has been an extremely popular one ever since. Alexey Pajitnov and Dmitry Pavlovsky are credited with the creation of this game while working as engineers in the Russian Academy of Sciences.[2]

In this section, we describe our implementation of Tetris. First we relate the basic idea behind the game in Section 4.1.1. Several choices arise towards fully specifying the rules of the game. In Section 4.1.2, we describe our implementation, which matches one that has widely been used by the RL community (Thierry and Scherrer, 2010a). In Section 4.1.3, we briefly sketch how our implementation maps to a formulation of Tetris as an MDP. The reader familiar with the Tetris may skip Section 4.1.1, and directly proceed to sections 4.1.2 and 4.1.3.

[2]Fahey's web page on Tetris (see `http://www.colinfahey.com/tetris/tetris_en.html`) is an excellent resource documenting the history of the game, its several variations, standards, scientific aspects, and so on.

4.1.1 Basics of Game

Figure 4.2 shows a screenshot from a typical Tetris game. The game comprises a rectangular board, whose standard dimensions are a width of 10 and height of 20. Each cell in the board can be filled or empty. Each game (or *episode*) starts with an empty board. Subsequently, one of seven fixed "tetromino" pieces is generated (uniformly at random) at the top of the board, in line with the central columns. Tetrominoes are contiguous patterns filling up four cells; the seven standard Tetris tetrominoes are listed in Figure 4.2. Once generated, a tetromino starts falling from the top of the board towards the bottom ("vertically" downward, as if under the influence of gravity), until it gets lodged either because some cell underneath it is already filled, or the



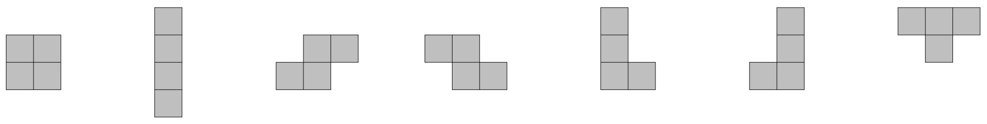


Figure 4.2: (Top) Screenshot from a Tetris game (source: `http://vuigame.net/`). (Bottom row) List of tetrominoes, left to right: `O`, `I`, `S`, `Z`, `L`, `J`, and `T`.

tetromino has reached the bottom of the board. Once a tetromino gets lodged, a new one is generated, and the above sequence repeats. Thus, whereas the first tetromino descends into an empty board, in general, a "wall" of previously lodged tetrominoes is already present when a new one gets generated.

A player (in our case a computer program) can alter the trajectory of a falling tetromino by shifting it laterally or rotating it as it descends. These actions, which have deterministic consequences, enable the player to control where the piece gets lodged, thereby determining the shape of the resulting wall. The defining aspect of the Tetris game is that if in getting lodged, a piece results in one or more rows in the wall getting *completely* filled, then those rows are cleared, and the portion of the wall above them descends by as many rows. In effect, the wall grows in size as pieces get lodged, and shrinks as rows get cleared. The player gets rewarded for each row cleared during a game: the player's objective is to maximize the number of rows cleared until the episode ends (as described in formally in section 4.1.2 and 4.1.3).

4.1.2 Some Choices in Implementation

So far we have described the overall scheme within a Tetris game. Below we complete our description by specifying particular choices made in our implementation.

Actions. In most Tetris interfaces intended for humans to play the game, the player is allowed to execute an action on the falling piece every time it descends by a cell. In particular the player may shift the piece left or right; or rotate it clockwise or counter-clockwise; or do nothing. Thus, the player has to pick one of five actions. The main drawback of the above scheme is the time consumed in computing the next configuration after each action is taken. Not only does the simulator need to consistently check for collisions between the falling piece and the wall, the sheer number of actions between the generation of a piece and its eventual getting lodged can be as high as 20.

Like other researchers before us, we opt for a simpler set of actions that significantly speeds up simulations. In particular we only admit actions to be taken when a tetromino is first generated. The action merely specifies a column and an orientation for the tetromino. Such an action has the deterministic effect of dropping the tetromino from the top of the specified column, in the specified orientation, until it gets lodged. Figure 4.3 illustrates this process through an example. Note that the number of distinct actions in a situation will depend on the piece that is falling, The number of column settings for a piece can at most be 10 (for an `I` piece), and the number of orientations at most 4 (for `L`, `J`, and `T` pieces).

Our simplified action scheme indeed allows for faster simulations than the alternative of applying shift or rotate actions to a piece as it falls. It is worth bearing in mind that nevertheless, the latter scheme allows the player more options in the placement of the tetromino on the wall. For example, shifting a piece even as it falls makes it possible to fill up holes, which cannot be achieved by merely dropping a piece from the top.

Episode termination. In most human-friendly versions of Tetris, newly generated pieces are actually generated *inside* the 10×20 board, within the top few rows. The game is then deemed to have ended when a newly generated piece collides with the existing wall. In our implementation, though, we deem an episode to have terminated only when there is no action that will result in

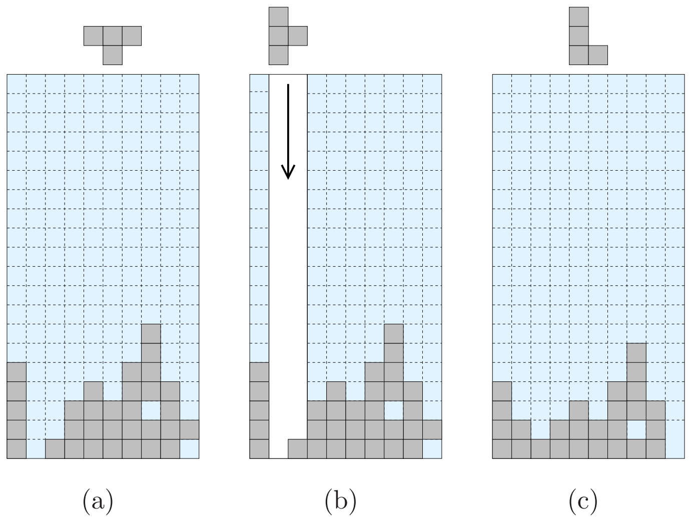


Figure 4.3: Example sequence from the middle of a Tetris episode. (a) New piece (`T`) generated. (b) Action (column, orientation) specified. (c) Transition effected, including the clearing of a row, and next piece (`L`) generated.

the newly-generated piece getting properly lodged within the 10×20 board; in other words, the piece will necessarily overshoot into an imaginary 21^{st} row.

Whereas both the schemes described above have the same qualitative intent of penalizing walls that exceed a certain size, experiments from Thierry and Scherrer (2010a) show that such subtle differences in implementation can have a significant effect on the performance of players. Even so, we do not believe that the *relative order* between the performance of different learning algorithms changes with the implementation choice; this chapter is primarily concerned only with the relative order.

Visibility of next piece. The essential decision for an agent to make while playing Tetris is where to place the newly-generated piece. In some versions of Tetris, the player's decision can benefit from the additional knowledge of what piece the simulator will generate next, following the current one. The reader might have noticed the presence of this feature in the screenshot from Figure 4.2, wherein it is visible that a `T` piece is due to fall next, even as an `I` piece is currently descending. In our implementation, consistent with other RL researchers', the player only gets to see one piece at a time: the piece that currently needs to be controlled.

Rewards. How is a Tetris player evaluated? In our implementation, the player (or a policy) gets a reward of 1 for every row cleared. Thus, the value of a policy becomes the expected number of rows it will clear before the episode terminates. In some variants of the game, players have additional incentives for clearing multiple rows at a time (the maximum possible is four rows, which can sometimes be cleared when lodging an `I` piece). We do not provide additional rewards if multiple rows are cleared at the same time.

Before proceeding further, we summarize how our Tetris implementation maps to an MDP formulation.

4.1.3 Formulation as MDP

We can treat Tetris as an MDP, in which each state corresponds to a configuration of the wall and the index of the piece that is due to fall. Assuming all possible wall configurations are reachable, we find that the size of the state space is $7 \times 2^{10 \times 20}$, which is more than 10^{61}. Up to 34 actions can apply from a state. Note that on applying an action from a state, we may *deterministically* compute the resulting configuration of the wall, which it is convenient to think of as an *afterstate*. The transition from a state to an afterstate also generates a deterministic reward based on the number of rows cleared. Figure 4.4 presents a schematic depiction of trajectories in Tetris.

It has formally been shown that regardless of the actions taken, almost all Tetris episodes must eventually terminate (Burgiel, 1997). The proof hinges on the occurrence of a long sequence of alternating `S` and `Z` pieces, which it can be shown cannot be contained within a finite board height. Importantly, this result implies that the value functions of policies are well-defined even when no discounting is used (indeed we use $\gamma = 1$ in our evaluation), and therefore, that there is an optimal value function for Tetris. However, very little is known about the structure of optimal policies. Breukelaar et al. (2004) show that even if the entire sequence of tetrominoes in a game is known in advance, computationally verifying whether the tetrominoes can be placed to

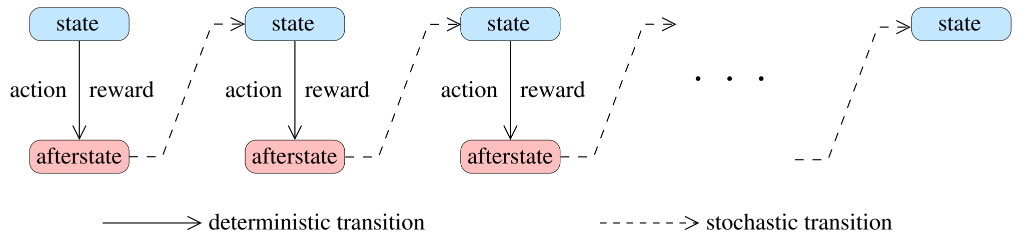


Figure 4.4: Structure of a trajectory in Tetris. The first state is always one of a fixed seven: an empty wall with one of the standard seven tetrominoes due to fall. (Conceptually, one might imagine every trajectory as actually starting with an afterstate corresponding to the empty wall.) An action taken from a state leads to a deterministic reward and afterstate. From an afterstate $\bar{s}$, the next state is determined by picking one of the seven standard tetrominoes uniformly at random, and retaining the wall corresponding to $\bar{s}$. A state is terminal if no action taken from it will lead to the lodgment of the associated tetromino within the 10×20 board.

clear a certain number of rows is an NP-complete problem.

The main advantages of Tetris as a benchmark problem—for evaluating sequential decision making algorithms—are its discrete state and action sets, and its concise forward model. Its main drawbacks include the large sizes of its state and action spaces, and the stochasticity in the transitions, which additionally gets exacerbated by the very long trajectories that result from good policies. In our experiments in Section 4.3, we mitigate this high stochasticity by running a large number of Monte Carlo simulations. Our experiments analyze the interaction between representations and learning methods in Tetris; the next section describes these aspects in some detail.

4.2 Choices in Representation and Learning

The large size of the state space in Tetris rules out the use of a tabular representation for storing value functions or policies. Rather, a compact representation is needed. Recall from Figure 4.4 that trajectories in Tetris have a special structure that is commonly found in board games. Specifically, at every decision step, the agent has (1) a full description of the state, and (2) a deterministic forward model to compute the afterstate that will result from taking some action. As a result, it becomes natural to represent policies through an evaluation function over afterstates. As far as we are aware, every instance of learning applied to Tetris in the literature has adopted such a representation, which we illustrate through a picture in Figure 4.5.

In the case of VF methods, the evaluation function over afterstates is learned to approximate the value function under some policy (the expected long-term reward to follow from each afterstate). On the other hand, PS methods seek to tune the parameters of the evaluation function such that they

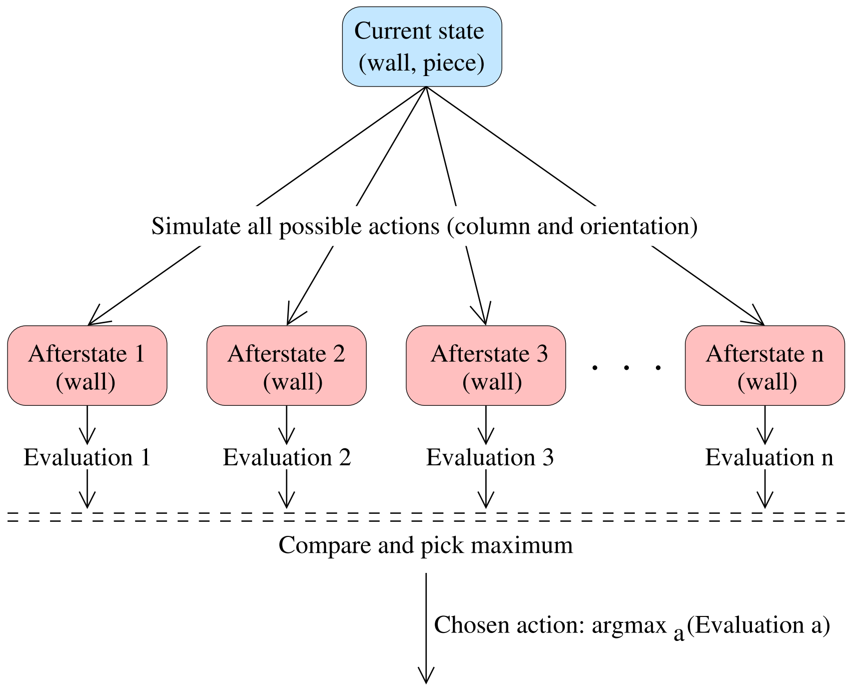


Figure 4.5: Typical structure of a Tetris policy based on an evaluation function (figure adapted from one presented by Thierry and Scherrer (2010a, see Figure 2)). From the current state, all possible actions are simulated using a one-step forward model to determine afterstates. An evaluation function associates each afterstate with a real number (the *evaluation*). The policy then is to pick an action leading to an afterstate with the highest evaluation (with ties broken uniformly at random). VF methods attempt to match the evaluation function with the value function being learned; PS methods search for parameters of the evaluation function that maximize the resulting performance. Note that some researchers have considered performing deeper-ply simulations before applying the evaluation function; that is, they consider all the states that could result from an afterstate, the afterstates that would result from those states, and so on. While deeper-ply simulations are more computationally demanding, they can lead to slightly better performance (Böhm et al., 2005; Kistemaker, 2008). Consistent with the vast majority of RL applications to Tetris in the literature, we stick with the single-ply simulations depicted in the figure.

induce the highest long-term reward. Thus, under PS methods, the evaluation of an afterstate bears no special semantics.

Interestingly, nearly every application of learning to Tetris in the literature has used a linear architecture to represent the evaluation function. Thus, in all previous work, the evaluation function learned is a linear combination of features derived from the afterstate: learning amounts to adapting the coefficients (or "weights") in the linear combination. As a concrete example, consider the early efforts of Bertsekas and Tsitsiklis (1996), in which the features used are the column heights of the wall, differences in the heights of consecutive columns, the maximum column height, and the number of "holes" in the wall (empty cells with at least one filled cell above them in the same column). Additionally a "bias" feature (always 1) is used to aid value function approximation. Table 4.1 lists the features introduced by Bertsekas and Tsitsiklis, of which there are 22.

For the experiments in this chapter, indeed we use the representation developed by Bertsekas and Tsitsiklis (1996). However, we must note that apart from the 22 features included therein, researchers in the past have considered several others, and often with demonstrable performance improvements (Thierry and Scherrer, 2010a, see Table 1). It exceeds the scope of this chapter to undertake systematic feature engineering and selection for Tetris. Rather than develop the most effective learning algorithms for the task, our objective here is to evaluate existing algorithms on an existing representation. For this purpose, we consider the architecture introduced by Bertsekas and Tsitsiklis appropriate, as (1) it continues to be used by other researchers (Farias and Van Roy, 2006; Szita and Lőrincz, 2006), and (2) the alternative representations proposed (Lagoudakis et al., 2002; Boumaza, 2009; Thierry and

Table 4.1: Features used by Bertsekas and Tsitsiklis (1996). For illustration we list the values these features assume on afterstates encountered in Figure 4.3.

Feature		**Value: wall in Figure 4.3(a)**	**Value: wall in Figure 4.3(c)**
Column height	h_1	5	4
	h_2	0	2
	h_3	1	1
	h_4	3	2
	h_5	4	3
	h_6	3	2
	h_7	5	4
	h_8	7	6
	h_9	4	3
	h_{10}	2	0
Successive column height difference	$\|h_1 - h_2\|$	5	2
	$\|h_2 - h_3\|$	1	1
	$\|h_3 - h_4\|$	2	1
	$\|h_4 - h_5\|$	3	1
	$\|h_5 - h_6\|$	1	1
	$\|h_6 - h_7\|$	2	2
	$\|h_7 - h_8\|$	2	2
	$\|h_8 - h_9\|$	3	3
	$\|h_9 - h_{10}\|$	2	3
Number of holes ($numHoles$)		2	1
Maximum height ($maxHeight$)		7	6
Bias		1	1

Scherrer, 2010b) remain *qualitatively* similar to it: linear combinations of a few tens of features.

For a comprehensive description of previous learning approaches applied to Tetris, we refer the reader to the excellent survey by Thierry and

Scherrer (2010a). For convenient reference, we list the methods described therein, along with the scores they achieve, in Table 4.2. Broadly, we may divide the methods in the table into two groups: methods that seek to learn or compute the value function (VF methods), and those that directly tune the weights of a controller based on feedback from a suitable fitness function (PS methods).

As Thierry and Scherrer (2010a) correctly observe, subtleties in implementing Tetris (such as in the rule to determine episode termination, or the lack of sufficient averaging to diminish noise) can significantly affect the observed results. Even so, we believe that the difference in the scales of performance achieved by VF and PS methods on Tetris—clearly visible in Table 4.2—cannot be due to noise or subtle differences in the implementation. For example: using the features listed in Table 4.1, the λ-policy iteration method of Bertsekas and Tsitsiklis (1996), a VF method, clears 3,000–5,000 lines. Using exactly the same features, the noisy cross-entropy method of Szita and Lőrincz (2006), a PS method, clears nearly 350,000 lines. It is unlikely that this "orders of magnitude" gap in the performance of the methods can be attributed solely to evaluation noise and variations in implementation.

In the next section, we carry out experiments to examine the limits imposed by the representation scheme on the accuracy with which value functions can be represented. Using the representation scheme of Bertsekas and Tsitsiklis (1996), we find evidence of the phenomenon illustrated by Baxter and Bartlett: that the greedy policy resulting from an approximate value function performs much worse than the policy whose value function is being approximated. As a result, the "policy improvement" step that is crucial to control is likely to become counter-productive. We proceed to our experiments.

Table 4.2: List of learning methods applied to Tetris. All the methods use a linear representation for the evaluation function, but the features they employ vary. The performance reported is rounded to a nearby multiple of a power of 10, with ranges provided when multiple variants of the learning method have been tested. The natural policy gradient algorithm due to Kakade (2001) is an actor-critic approach, wherein a parameterized policy is optimized by following a gradient that is itself obtained from a stored value function (rather than solely from observed rewards).

Method	**Class**	**Mean lines per episode**
λ-policy iteration (Bertsekas and Tsitsiklis, 1996)	VF	3,000–5,000
Least-squares policy iteration (Lagoudakis et al., 2002)	VF	1,000-3,000
Relational RL with Gaussian Process regression (Ramon and Driessens, 2004)	VF	50
Approximate dynamic programming (Farias and Van Roy, 2006)	VF	3,500–5,000
Approximate dynamic programming (Petrik and Scherrer, 2009)	VF	20,000–25,000
Natural policy gradient with compatible function approximation (Kakade, 2001)	VF/PS	5,000–7,000
Dellacherie's controller from 2003 (Thierry and Scherrer, 2010a)	Manual design	660,000
Evolutionary algorithm (Böhm et al., 2005)	PS	1,400,000
Noisy cross-entropy method (Szita and Lőrincz, 2006)	PS	350,000
CMA-ES (Boumaza, 2009)	PS	36,000,000
Noisy cross-entropy method (Thierry and Scherrer, 2010b)	PS	36,000,000

4.3 Experiments and Results

In this section, we present the experimental analysis that constitutes the core of this chapter. Let us take a moment to recall the observations that bring us here, and the agenda we set ourselves.

1. As evidenced by Table 4.2, in general, VF methods perform significantly worse than PS methods on the Tetris task.
2. We aim to investigate to what extent the poor performance of VF methods owes to the phenomenon illustrated by Baxter and Bartlett (2001), of approximate policy iteration leading to a deterioration of the policy (also depicted in Figure 4.1).
3. Specifically for our experiments, we adopt the representation introduced by Bertsekas and Tsitsiklis (1996) (a linear combination of 22 features, described in Table 4.1).

The sequence of experiments to follow in this section essentially parallels the demonstration provided by Baxter and Bartlett (2001, see Appendix A). In both cases, the purpose is to compare a policy π with another policy π', which is a greedy policy with respect to an *approximate* value function of π. However, several aspects of our work distinguish it from Baxter and Bartlett's concise illustration. Below we list four important differences.

Size and nature of task. Baxter and Bartlett carefully design a "toy" MDP with two states and two actions to establish that approximate value functions can induce inferior policies. Assuming a complementary stance, we set out to ascertain to what extent this phenomenon pervades existing large-scale, complex sequential decision decision making tasks. In particular we carry out our tests on Tetris, an MDP with nearly 10^{61} states and 34 actions.

Solution versus estimation. The simplicity of their MDP allows Baxter and Bartlett to perform exact calculations and "solve for" the best approximation of the value function. The complexity of Tetris forces us to *estimate* this best approximation itself using Monte Carlo simulation.

Policy being evaluated. In Baxter and Bartlett's example, π, the policy being evaluated, is an optimal policy. By contrast, we restrict ourselves to picking a "good" policy for Tetris and examining the consequences of approximating its value function. One obvious reason for our choice is that we do not know of any optimal policies for Tetris. However, even if we did, it would be impractical for us to deal with policies that clear more than a few thousands of lines, simply due to the time it would take to get reliable estimates from our Monte Carlo estimation. For our experiments, we hand-code a policy, π^{HC}, that clears about 1,000 lines per episode. Despite running Monte Carlo evaluations on roughly 100 computers for close to a week, even with this policy, a small amount of error remains in our estimation of the "best approximation" value function (see Section 4.3.2).

Definition of approximation error. If we are attempting to approximate V^π, the value function of a policy π, and we are limited to working with functions from a certain parameterized class F (such as the class of linear functions in this chapter), how must we define the "best approximation" $\widehat{V^\pi}$

contained in F? One natural definition would be based on the squared error norm weighted by D_1, the stationary distribution of states under π.[3] In other words, $\widehat{V^\pi} \overset{\text{def}}{=} \min_{f \in F} \sum_{s \in S} D_1(s) \left(V^\pi(s) - f(s)\right)^2$. This definition of "best approximation" is indeed the one that Baxter and Bartlett invoke; it is also the approximation to which methods such as TD(1) and LSTD(1) would converge when using a linear function approximator (Tsitsiklis and Van Roy, 1997; Boyan, 2002).

Koller and Parr (2000) observe that while the definition of $\widehat{V^\pi}$ above is appropriate for policy evaluation purposes, in practice, other choices could result in better success for the subsequent policy improvement step. In particular, minimizing a max-norm ($\mathcal{L}_\infty$ norm) error instead of the weighted $\mathcal{L}_2$ norm error defined above can lead to better bounds on the sub-optimality of policy improvement (Guestrin et al., 2001a). Unfortunately, general methods for efficiently minimizing the max-norm remain unknown.

Yet another crucial element in defining the error term is the weighting distribution: above we use D_1, the stationary distribution induced by π. Koller and Parr (2000) suggest that the use of alternative weighting distributions, even if in tandem with the $\mathcal{L}_2$ norm, could potentially yield better results for policy improvement. Correspondingly they construct a policy iteration framework that can be instantiated with arbitrary weighting distributions. While this framework becomes very useful once we know which weighting distribution to use, most often it is unclear which weighting distributions would

[3]In our general commentary, often we refer to value functions and distributions that are functions over *state* spaces. In a formal sense, recall that we learn these functions over the *afterstate* space in Tetris. The distinction between states and afterstates does not have any significant implications on learning; for the rest of this chapter, the reader may think of states and afterstates as "essentially the same."

be the most successful for policy improvement. Further, commonly used policy evaluation algorithms such as TD(λ) are known to be unstable when sampling is performed using distributions other than D_1 (Tsitsiklis and Van Roy, 1997).

In our experiments, we define approximation error through the weighted $\mathcal{L}_2$ norm, and consider three choices for the weighting distribution: D_1, D_2, and D_3. These distributions over the afterstate space are described in Table 4.3. Our experiments show that under each of the resulting definitions of approximation error, the best approximation of the value function of π^{HC} induces a policy that is substantially inferior to π^{HC}.

Table 4.3: Distributions used for for weighting the $\mathcal{L}_2$ norm.

Distribution	**To obtain sample:**
$\mathbf{D_1}$: Stationary distribution of afterstates under hand-coded policy π^{HC} (described in Section 4.3.1).	Record afterstates along a trajectory, starting with the empty wall, following π^{HC}, and until episode termination. Of the afterstates recorded, pick one uniformly at random.
$\mathbf{D_2}$: Stationary distribution of afterstates under π^{Random}, which selects an afterstate $\bar{s}$ uniformly at random from among the distinct afterstates reachable from the current state, and picks an action that reaches $\bar{s}$.	Record afterstates along a trajectory, starting with the empty wall, following π^{Random}, and until episode termination. Of the afterstates recorded, pick one uniformly at random.
$\mathbf{D_3}$: Uniform distribution over set of afterstates.	Generate a 200-bit string corresponding to a wall, setting each bit to 0 or 1 with equal probability. If no row in the wall is fully filled, pick the wall as the sample afterstate. Otherwise repeat. (This procedure is approximate: at present we are unaware of a concise *necessary and sufficient* condition for the reachability of Tetris afterstates.)

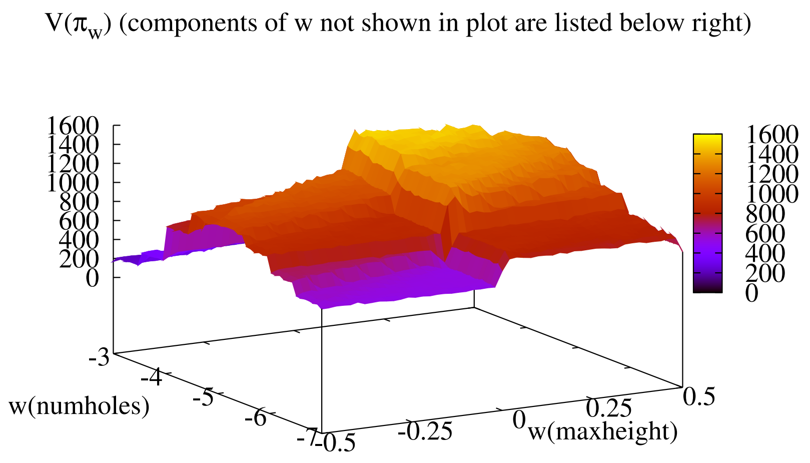
The remainder of this section is in three parts. In Section 4.3.1, we describe our hand-coded policy, π^{HC}. Section 4.3.2 provides the details of our Monte Carlo simulation to estimate $\mathbf{w}_{(D_1)}^{HC-vf}$, $\mathbf{w}_{(D_2)}^{HC-vf}$, and $\mathbf{w}_{(D_3)}^{HC-vf}$, the coefficients best approximating $V^{\pi^{HC}}$ with respect to corresponding weighting distributions. In Section 4.3.3, we examine the greedy policies that these coefficients induce (when combined linearly with the features given by Bertsekas and Tsitsiklis).

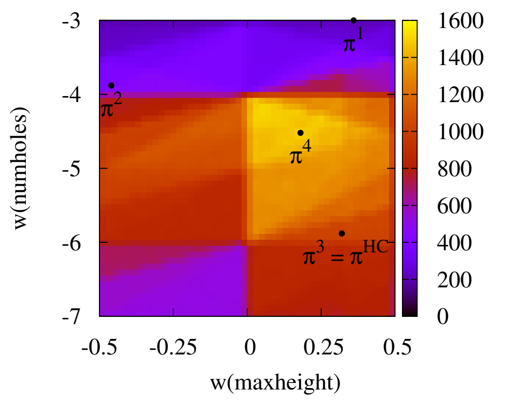
4.3.1 Hand-coding a "Good" Policy

We hand-code a policy for Tetris by directly manipulating the 22 coefficients in the linear combination for the evaluation function. We hypothesize that the better our policy performs, the starker its contrast would be with a greedy policy obtained from approximating its value function. On the other hand, policies that clear more than a few thousand lines per episode are unsuitable for our experiments, owing to the sheer time it would take to obtain a reliable Monte Carlo estimate of their approximate value functions. As an intermediate choice, we seek to hand-tune a policy that can clear about 1,000 lines per episode. We find it relatively straightforward to identify such a policy by tuning a small subset of the 22 coefficients after fixing the others to be either 0 or -1. Our tuning procedure is detailed in Figure 4.6.

In order to gain some familiarity with the nature of Tetris policies, we isolate four policies encountered during our tuning process. These policies, π^1 through π^4, are indicated in the bottom left plot of Figure 4.6. The policies are indexed in the order of their values; it is π^3 that clears close to 1,000 lines per episode, and which we take as our hand-coded policy π^{HC}.

In Figure 4.7, we plot the distributions of the episodic reward (lines





Feature	**Coefficient**
$h_i, \forall i \in \{1, 2, \ldots, 10\}$	0
$\lvert h_i - h_{i+1} \rvert, \forall i \in \{1, 2, \ldots, 9\}$	-1
$maxHeight\ (\pi^{HC})$	0.32
$numHoles\ (\pi^{HC})$	-5.88
bias	0

Figure 4.6: Procedure to determine a suitable hand-coded policy π^{HC}. As shown at bottom right, we set the 10 coefficients corresponding to column heights to 0, and the 9 coefficients corresponding to differences in the heights of successive columns to -1. We then search over a grid for the coefficients corresponding to the $numHoles$ and $maxHeight$ features. The bias coefficient does not affect the policy, which is greedy with respect to the evaluation function over afterstates. At each setting of $\mathbf{w}(numHoles)$ and $\mathbf{w}(maxHeight)$, the graphs show the value of the corresponding policy, estimated from 10,000 independent episodic runs. For illustration, we highlight four policies in the bottom left plot, whose values (with one standard error) are: $V(\pi^1) = 182 \pm 1$, $V(\pi^2) = 600 \pm 5$, $V(\pi^3) = 1009 \pm 10$, and $V(\pi^4) = 1527 \pm 15$. Of these policies, we pick π^3 to be our hand-coded policy π^{HC}.

cleared in an episode) under policies π^1 through π^4. Note that the value of a policy is its mean episodic reward. Thierry and Scherrer (2010a) conjecture that the distribution of episodic rewards for a fixed Tetris policy closely resembles a geometric distribution. However, our plots in Figure 4.7 appear to suggest that a negative binomial distribution would be a more appropriate fit. The negative binomial distribution is a generalization of the geometric distribution.[4] If episodic rewards indeed followed a geometric distribution, the most probable episodic reward would be 0, and the probability distribution function would monotonically decrease. However, under a negative binomial distribution, the mode lies between 0 and the mean, as it does in all our plots. The formal characterization of the distribution of episodic rewards in Tetris is an interesting challenge for future scholarship. However, the more relevant concern with respect to *learning* Tetris is that policies with higher values also have higher variances in the episodic reward. In the plots in Figure 4.7, we find that the standard deviations of the distributions are of the same magnitude as their means.

Before proceeding, we leave the reader with a few illustrations of Tetris afterstates and their values under π^{HC}. Figure 4.8 presents four examples. From the figure, we do observe an inverse correlation between the $maxHeight$ feature and the value of afterstates. However, it does not appear easy to pinpoint how other features influence the afterstate value. Under approximate policy evaluation, the objective is to approximate afterstate values using a linear combination of the 22 features.

[4]The geometric and negative binomial distributions, which are discrete probability distributions, can be construed as analogues of corresponding continuous probability distributions: respectively, the exponential and gamma distributions.

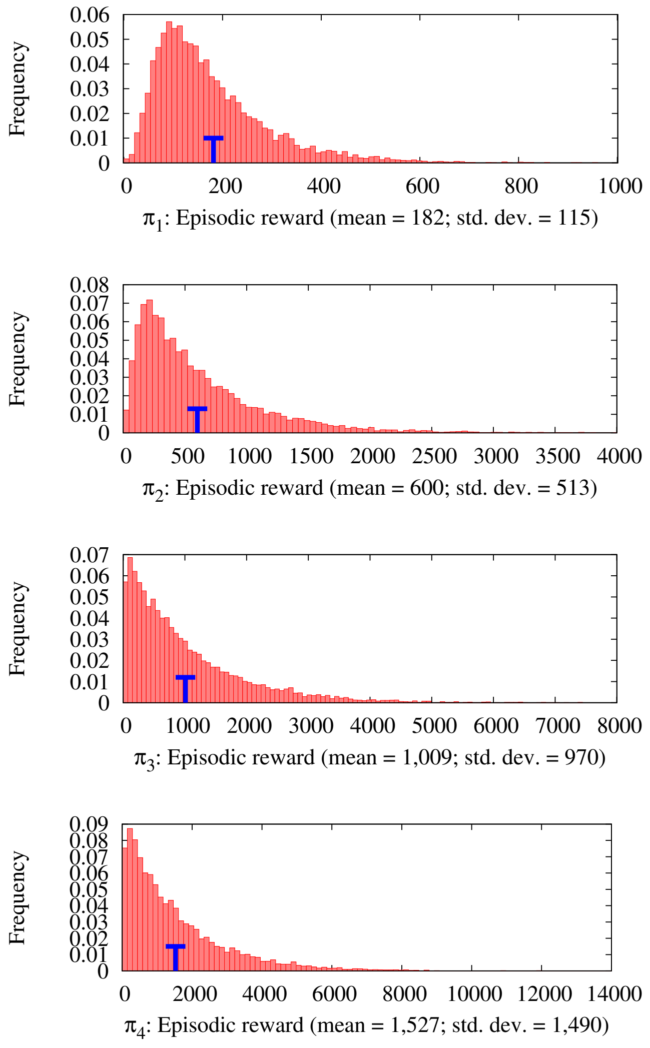


Figure 4.7: Distribution of episodic reward in the four policies highlighted in Figure 4.6. The x axis in each plot shows the episodic reward, and the y axis the frequency of occurrence (obtained by binning episodic rewards into 100 equal-length intervals). Note that the x axis scales on the plots are different, and that the means of the distributions (shown as T's) progressively *increase* from π^1 through π^4. From the plots, it appears that the episodic rewards from a fixed Tetris policy follow a negative binomial distribution.

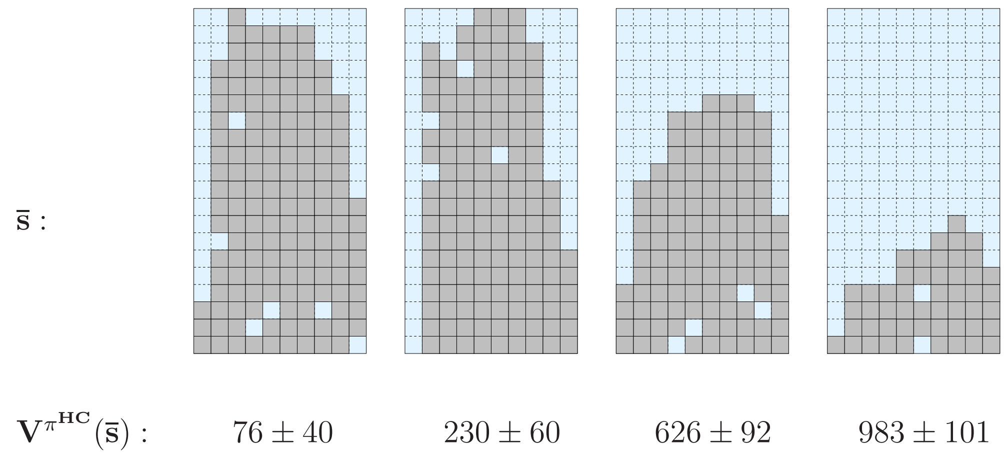


Figure 4.8: Four wall configurations (afterstates) and their values (and one standard error) under π^{HC}. The estimates are based on 100 independent evaluations.

4.3.2 Limit of Approximate Policy Evaluation

Given π^{HC}, the objective of policy evaluation is to determine its value function $V^{\pi^{HC}}$. Since our representation is not fully expressive, value function learning has to make do with approximating $V^{\pi^{HC}}$. The objective of approximate policy evaluation, therefore, is to learn $\widehat{V^{\pi^{HC}}}$. As we have discussed earlier in this section, $\widehat{V^{\pi^{HC}}}$ can be defined based on multiple error norms. In this work, we consider the $\mathcal{L}_2$ norm weighted by the distribution D over afterstates, giving:

$$\widehat{V^{\pi^{HC}}_{(D)}} = \min_V \sum_{\bar{s} \in \bar{S}} D(s)\{V^{\pi^{HC}}(\bar{s}) - V(\bar{s})\}^2,$$

where $\bar{S}$ is the set of all afterstates. In particular the set of functions from which we select $\widehat{V^{\pi^{HC}}_{(D)}}$ here is those that linearly combine the feature vector $\boldsymbol{\phi}$. Thus, equivalently, the objective of approximate policy evaluation is to find a coefficient vector $\mathbf{w}_{(D)}^{HC-vf}$ such that:

$$\mathbf{w}_{(D)}^{HC-vf} = \min_{\mathbf{w}\in\mathbb{R}^{22}} \sum_{\bar{s}\in\bar{S}} D(\bar{s})\{V^{\pi^{HC}}(\bar{s}) - \mathbf{w}^T\boldsymbol{\phi}(\bar{s})\}^2.$$

Figure 4.9 provides a geometric illustration of $\mathbf{w}_{(D)}^{HC-vf}$, which leads to the best approximation of the value function of π^{HC} when weighted by distribution D. If $D = D_1$, policy evaluation methods such as TD(1) and LSTD(1) indeed converge to $\mathbf{w}_{(D)}^{HC-vf}$, while TD(λ) and LSTD(λ), for $\lambda \in [0,1)$, converge to points whose distance from $\mathbf{w}_{(D)}^{HC-vf}$ can be bounded in terms of λ (Tsitsiklis and Van Roy, 1997).

The method we adopt for estimating $\mathbf{w}_{(D)}^{HC-vf}$, for $D \in \{D_1, D_2, D_3\}$, is akin to LSTD(1): we collect a large number of afterstates, estimate their values under π^{HC}, and use linear regression to determine the best coefficients. Algorithm 4.1 describes our estimation method, which takes three parameters.

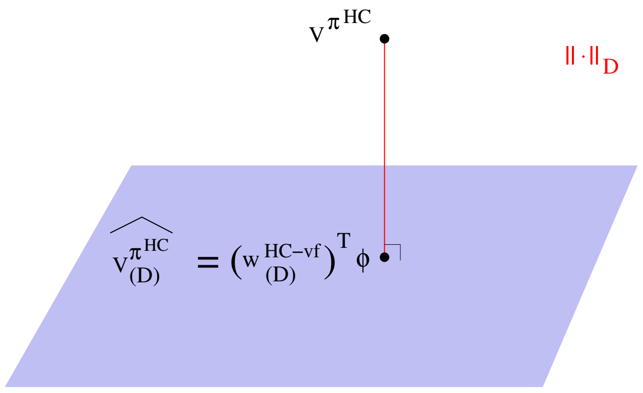


Figure 4.9: Best approximation of $V^{\pi^{HC}}$ under weighted $\mathcal{L}_2$ norm. In particular if the weighting distribution is D, then $\mathbf{w}_{(D)}^{HC-vf}$ corresponds to the weight vector of the best approximation value function $\widehat{V^{\pi^{HC}}_{(D)}} \overset{\text{def}}{=} (\mathbf{w}_{(D)}^{HC-vf})^T\boldsymbol{\phi}$.

The first parameter, D, determines the distribution from which afterstates are sampled. The other parameters are N, the number of afterstates to sample, and M, the number of trials for estimating the long-term return from each afterstate. Access to a forward model allows us to set $M > 1$ and revisit afterstates, thereby reducing the variance in their value estimates. Regardless of the setting of M, our estimate of $\mathbf{w}^{HC-vf}_{(D)}$ remains unbiased; the higher we set N, the smaller the estimation error.

For our experiments, under each of D_1, D_2, and D_3, we collect $N = 1,000,000$ afterstates, each of whose values we average based on $M = 100$ roll-outs. In total, this process takes us roughly 10-12 days on a computing cluster with close to 100 nodes. Are the estimates we make based on the collected data reliable estimates of $\mathbf{w}^{HC-vf}_{(D_1)}$, $\mathbf{w}^{HC-vf}_{(D_2)}$, and $\mathbf{w}^{HC-vf}_{(D_3)}$? We have no independent way to estimate these vectors, and therefore, we adopt a cross-validation approach to ascertain the reliability of our estimates. Below we

Algorithm 4.1 Estimation of $\mathbf{w}^{HC-vf}_{(D)}$

```
Input: $D$ (distribution of afterstates), $N$ (number of states to record), $M$ (number of roll-outs from each state).
Output: Estimate of $\mathbf{w}^{HC-vf}_{(D)}$.
for $i = 1, 2, \ldots, N$ do
    Draw afterstate $\bar{s}_i$ based on the distribution $D$ (independently).
    Simulate $M$ independent trajectories starting from $\bar{s}_i$, and following policy $\pi^{HC}$, until a terminate state is reached. Let the mean long-term reward accrued along these trajectories be $v_i$.
Let $X$ denote the $N \times 22$ matrix $[\boldsymbol{\phi}(\bar{s}_i)]_{i=1}^{N}$.
Let $Y$ denote the $N \times 1$ matrix $[v_i]_{i=1}^{N}$.
Return $\left(X^T X\right)^{-1} X^T Y$ as an estimate of $\mathbf{w}^{HC-vf}_{(D)}$.
```

describe this approach.

Suppose we have a data set Z with N pairs of the form (afterstate features, afterstate value under π^{HC}). Our main operation is to perform linear regression on this data to obtain a 22-dimensional weight vector $\mathbf{w}$. Now consider splitting Z into two halves, Z_1 and Z_2, performing linear regression independently on each, and obtaining corresponding vectors $\mathbf{w}_1$ and $\mathbf{w}_2$. Surely we may conclude that we have not collected a sufficient amount of data—that is, N is not large enough—if we find that $\mathbf{w}_1$ and $\mathbf{w}_2$ are "substantially different". On the other hand, if $\mathbf{w}_1$ and $\mathbf{w}_2$ are "close enough", we can be confident that N is large enough. To improve our confidence that $\mathbf{w}_1$ and $\mathbf{w}_2$ are not close enough merely due to a lucky split of Z, we can perform several independent splits of Z, and average the similarity between the $\mathbf{w}_1$ and $\mathbf{w}_2$ arising from each split. Straightforward indicators of the similarity between $\mathbf{w}_1$ and $\mathbf{w}_2$ could be, for instance, the angle between these vectors, or some norm of $\mathbf{w}_1 - \mathbf{w}_2$. However, the more appropriate similarity index should compare not $\mathbf{w}_1$ and $\mathbf{w}_2$ by themselves, but the predictions that result from them. We define two such similarity measures, which we describe next.

Mean Split Prediction Difference: We define the "prediction difference" between vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ as:

$$\sqrt{\frac{1}{|Z|}\sum_{\bar{s}\in Z}\{(\mathbf{w}_1)^T\boldsymbol{\phi}(\bar{s}) - (\mathbf{w}_2)^T\boldsymbol{\phi}(\bar{s})\}^2}.$$

This formulation leads to a natural scaling of the elements of $\mathbf{w}_1$ and $\mathbf{w}_2$ by the feature vector $\boldsymbol{\phi}$, before the weighted $\mathcal{L}_2$ norm of the resulting difference vector is computed. In our experiments, we average the prediction difference over 100 independent splits of our data, and denote the resulting average the Mean Split Prediction Difference (MSPD). As we collect more data—that is, as the size of Z increases—we expect the MSPD of Z to converge to 0.

Mean Split Policy Similarity: Recall that ultimately we want to evaluate the performance of the greedy policy that $\mathbf{w}_{(D)}^{HC-vf}$ induces. Correspondingly we define a second measure to compare the policies that result from splitting Z and performing linear regression on the splits. Let $\mathbf{w}_1$ and $\mathbf{w}_1$ be the coefficients obtained from the split, and let π_1 and π_2 be greedy policies with respect to $(\mathbf{w}_1)^T\boldsymbol{\phi}$ and $(\mathbf{w}_2)^T\boldsymbol{\phi}$, respectively. Let us say that policies π_1 and π_2 *agree* on some state s iff $\pi_1(s) = \pi_2(s)$. To assess the similarity between π_1 and π_2, we measure the fraction of states on which they agree. Naturally, this measurement would be most meaningful when we weight individual states by the probability π_1 or π_2 would visit them. Formally we define $sim(\pi_1, \pi_2)$ to be the expected fraction of states on which π_1 and π_2 agree, among the states encountered while following π_1 through an entire episode. We define $sim(\pi_2, \pi_1)$ similarly, and take the "policy similarity" of π_1 and π_2 to be $\frac{sim(\pi_1,\pi_2)+sim(\pi_2,\pi_1)}{2}$. We compute the Mean Split Policy Similarity (MSPS) of a data set Z by averaging the policy similarity resulting from 100 random splits. We expect that as more data get collected in Z, its MSPS will tend towards 1.

Despite collecting 1,000,000 data points under each of our distributions, unfortunately we do not observe perfect scores either for the MSPD or MSPS of our data. Figure 4.10 plots these measurements as a function of the amount of data used in our Monte Carlo simulations. In all cases, while we do find steady improvement in the measurements as we train on more data, we also find that there is significant room for further improvement. Unless we exceed machine precision, it is unlikely that we will get an MSPD score of zero. However, we do expect that weight vectors in some small neighborhood of $\mathbf{w}_{(D)}^{HC-vf}$ will all induce the same policy, and so, indeed MSPS can converge to 1 in practice. Based on the trends in the graphs, one might conjecture that about ten times as much data would be needed for such convergence to occur.

In summary our Monte Carlo estimate of $\mathbf{w}_{(D)}^{HC-vf}$, and specifically of $\pi_{(D)}^{HC-vf-greedy}$, the greedy policy thereby induced, remains imperfect as a consequence of the high stochasticity in Tetris. Nevertheless, note that the MSPS under all distributions is still greater than 90%. For the remainder of this chapter, we treat our Monte Carlo estimates of $\mathbf{w}_{(D_1)}^{HC-vf}$, $\mathbf{w}_{(D_2)}^{HC-vf}$, and $\mathbf{w}_{(D_3)}^{HC-vf}$ as though they are perfect. We believe that this concession does not alter the nature of the results to follow.

4.3.3 Greedy Policy Induced by Approximate Value Function

Having estimated $\mathbf{w}_{(D_1)}^{HC-vf}$, $\mathbf{w}_{(D_2)}^{HC-vf}$, and $\mathbf{w}_{(D_3)}^{HC-vf}$ to a reasonable degree of precision, we confront the question that this chapter set out to answer: how well do the greedy policies resulting from these weight vectors perform? Table 4.4 records the mean lines per episode cleared by policies $\pi_{(D_1)}^{HC-vf-greedy}$, $\pi_{(D_2)}^{HC-vf-greedy}$, and $\pi_{(D_3)}^{HC-vf-greedy}$. In all three cases, the performance is significantly lower than that of π^{HC}. Taken together, these results establish that Tetris, when considered with the representation introduced by Bertsekas and Tsitsiklis, suffers the same fate of Baxter and Bartlett's example. That approximate policy improvement fails under three distinct weighting distributions is an indication of the primacy of representation in determining its success.

It must be noted that practical constraints on time prevent us from replicating our experiments on policies other than π^{HC}. Even so, we consider

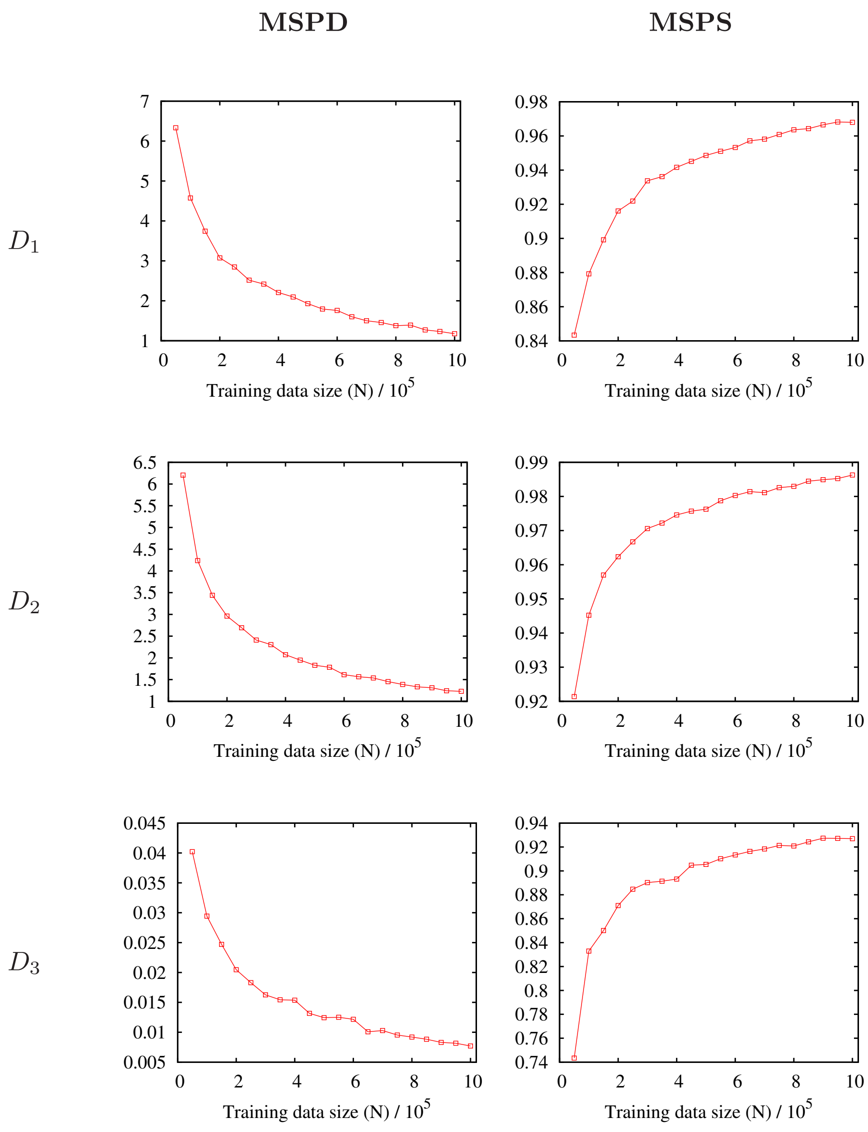


Figure 4.10: Convergence of Monte Carlo simulation. Under each of three weighting distributions, the Mean Split Prediction Difference (MSPD) and the Mean Split Policy Similarity (MSPS) (explained in Section 4.3.2) as a function of training data size.

Table 4.4: Results of approximate policy improvement.

Policy	Mean lines per episode
$\pi_{(D_1)}^{HC-vf-greedy}$	70 ± 1
$\pi_{(D_2)}^{HC-vf-greedy}$	28 ± 1
$\pi_{(D_3)}^{HC-vf-greedy}$	29 ± 1
π^{HC}	1009 ± 10

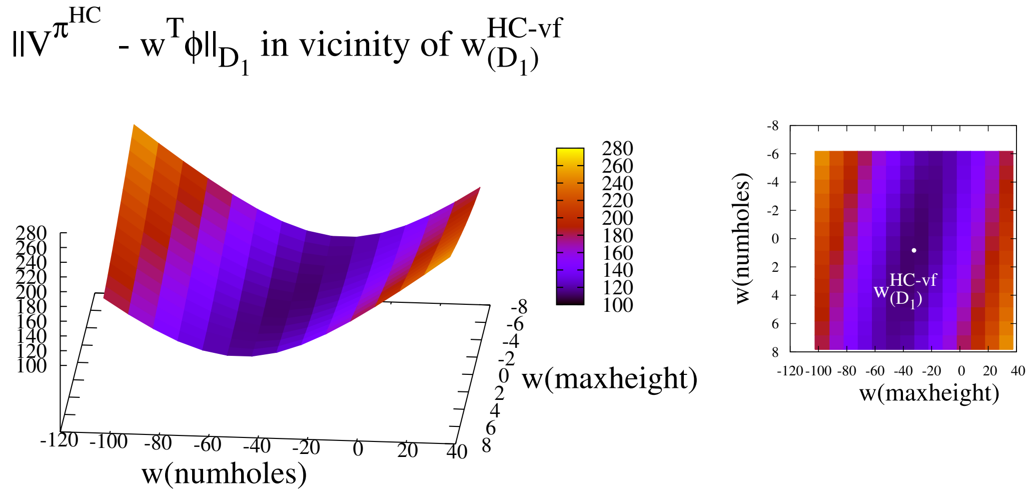
it unlikely that approximate policy improvement would succeed when applied to other well-performing Tetris policies. In fact, in their experiments applying the λ-policy iteration algorithm to this task, Bertsekas and Tsitsiklis (1996, see page 438) report a gradual *degradation* in performance beyond a certain number of iterations. Based on the experiments in this chapter, we hypothesize that the reason for such degradation is the incapacity of the representation to consistently support successful policy improvement.

We conclude this section with a series of graphs that visually depict the mismatch between the objectives of (1) approximating some value function well, as defined through a suitable norm, and (2) realizing policies with high values. These graphs are displayed in figures 4.11, 4.12, and 4.13. Below we describe Figure 4.11, which shows the value function approximation error under the D_1-weighted norm at different weight settings in the vicinity of $\mathbf{w}_{(D_1)}^{HC-vf}$, and also the performance of the policies induced by the same weight settings. Figures 4.12 and 4.13 follow the same scheme as Figure 4.11, but corresponding to weighting distributions D_2 and D_3, respectively.

Figure 4.11 comprises two sets of plots: the top two plots provide alternate views of the same data, and likewise, the bottom two plots display the same data. The top plots show the approximation error (the $\mathcal{L}_2$ norm weighted by D_1) associated with different weight vectors. The center of the plots corresponds to $\mathbf{w}_{(D_1)}^{HC-vf}$; the other weight vectors shown in the graphs vary from $\mathbf{w}_{(D_1)}^{HC-vf}$ only along the dimensions corresponding to *numHoles* and *maxHeight*. As we might expect, the approximation error achieves its lowest value at the setting corresponding to $\mathbf{w}_{(D_1)}^{HC-vf}$, and gradually increases as we move away from the minimum.

The x and y axes in the two bottom plots in Figure 4.11 exactly match those of the top plots. However, while the top plots show the value function approximation error, the bottom plots show the mean lines per episode achieved by policies that are greedy with respect to the corresponding weight vectors. As reported in Table 4.4, we observe that $\pi_{(D_1)}^{HC-vf-greedy}$, the policy induced by $\mathbf{w}_{(D_1)}^{HC-vf}$, clears about 70 lines per episode.

An interesting phenomenon becomes apparent as we view the neighborhood of $\mathbf{w}_{(D_1)}^{HC-vf}$ in the bottom plots. In particular as the *numHoles* and *maxHeight* coefficients are both decreased, we observe a dramatic increase in the value of the resulting policies. It is rather ironic that in such close proximity of $\mathbf{w}_{(D_1)}^{HC-vf}$, which approximates $V^{\pi^{HC}}$ well but clears only 70 lines per episode, there exists a weight configuration that does not approximate $V^{\pi^{HC}}$ nearly as well, but clears more than 7,000 lines per episode! The plots in figures 4.12 and 4.12 also display a similar pattern. In all these cases, it appears that it would be a profitable strategy to search for weight configurations in the proximity of $\mathbf{w}_{(D)}^{HC-vf}$ with the aim of maximizing the performance of



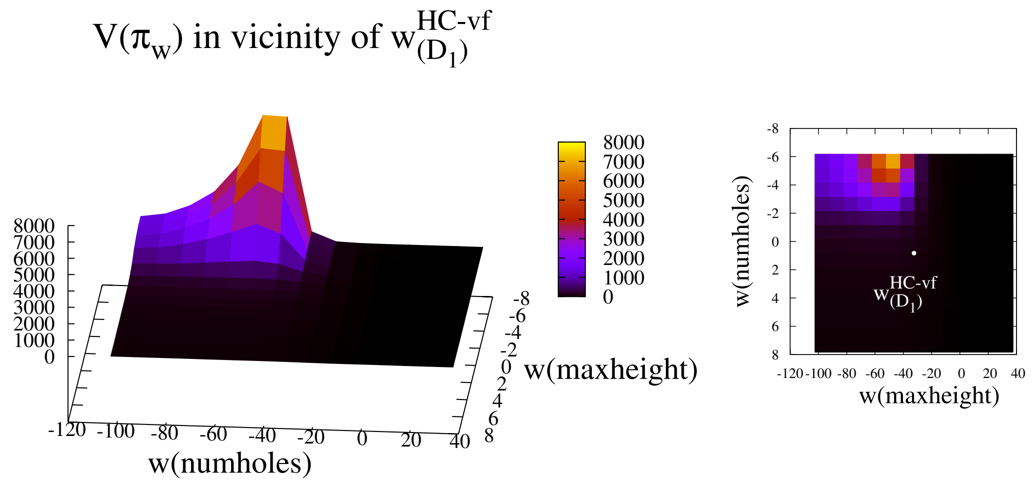
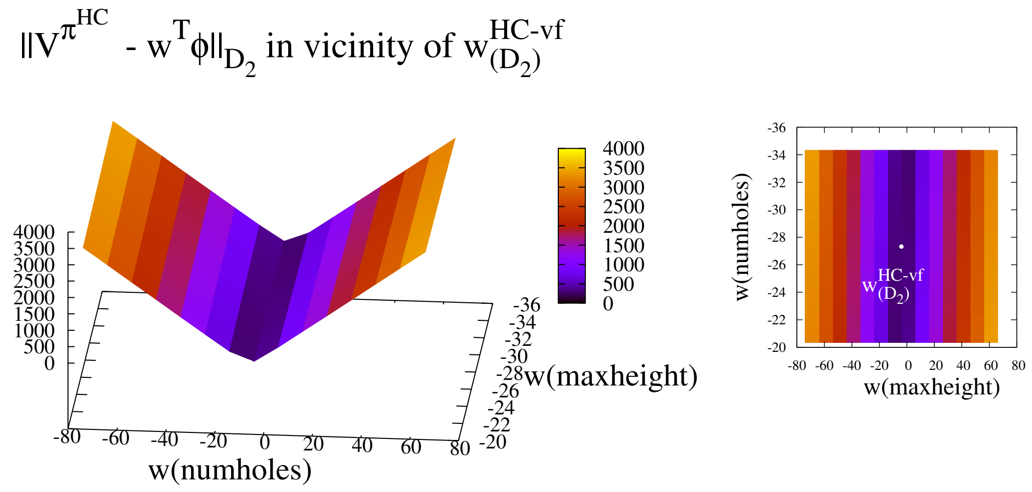


Figure 4.11: Plots showing the value function approximation error (above) and the value of the induced policy (below) for a range of weight vectors in the vicinity of $\mathbf{w}_{(D_1)}^{HC-vf}$. In all the plots, the x and y axis vary the coefficients corresponding to the $numHoles$ and $maxHeight$ features, respectively. The 20 elements of the weight vector not shown in the plots correspond to those under $\mathbf{w}_{(D_1)}^{HC-vf}$. Detailed explanations are provided in the text.



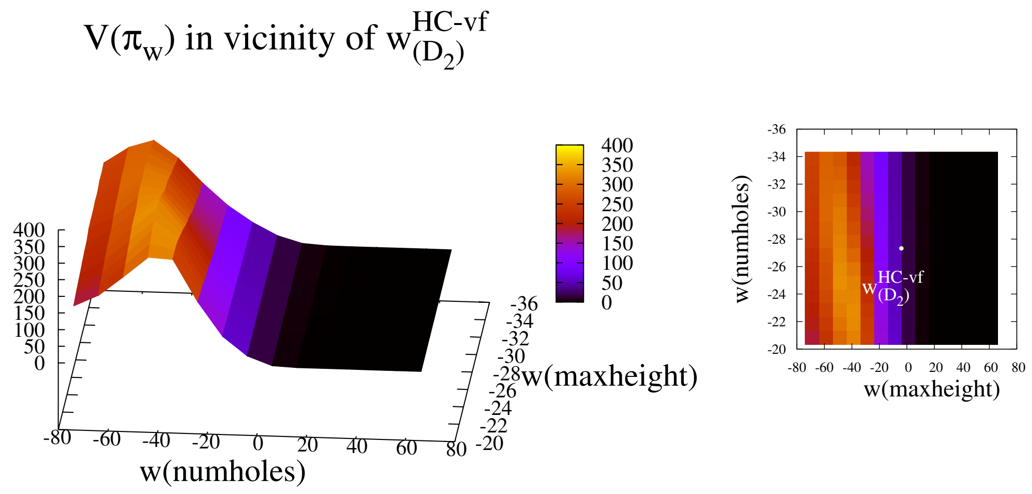
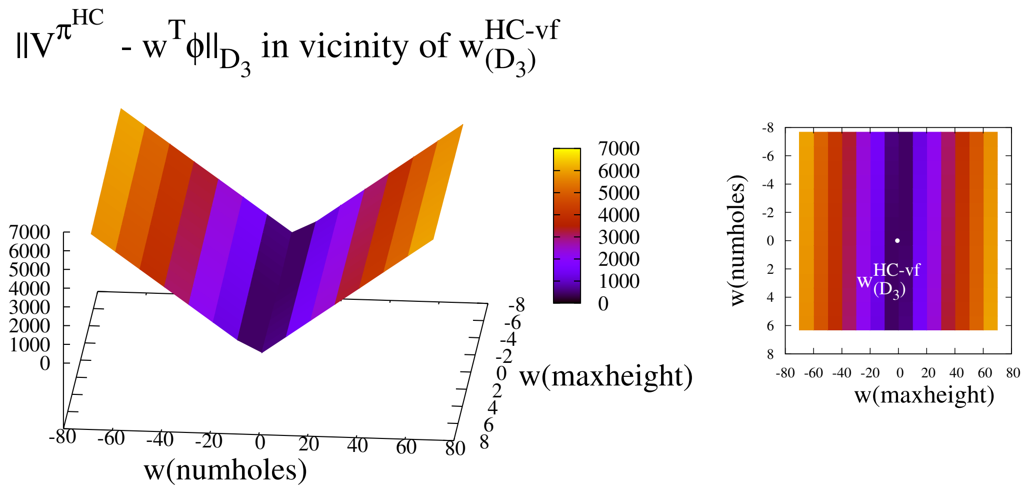


Figure 4.12: Similar plots as those in Figure 4.11, but with D_2 as the weighting distribution.



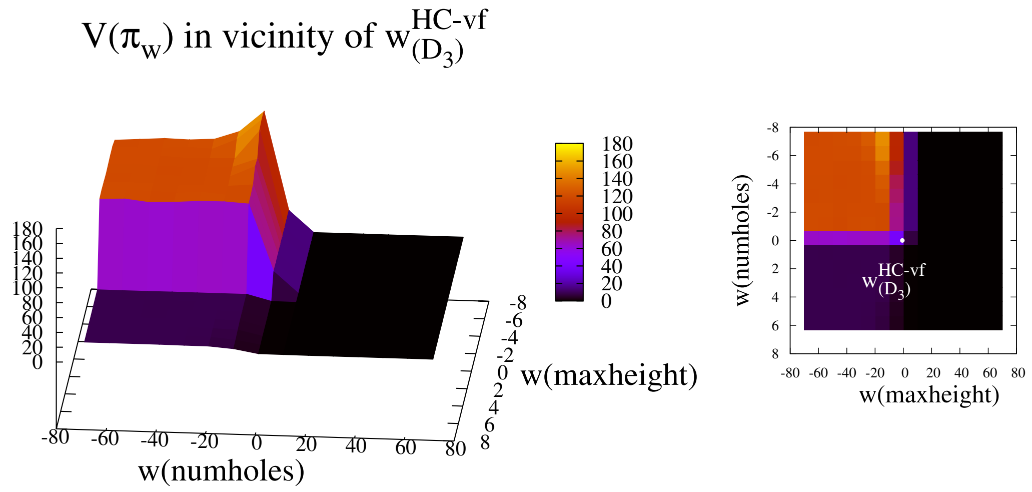


Figure 4.13: Similar plots as those in Figure 4.11, but with D_3 as the weighting distribution.

the resulting policy. In Chapter 6, we develop a method wherein VF and PS methods are applied in sequence, and which indeed proves fairly successful.

4.4 Summary and Discussion

This chapter, like the previous one, investigates the relationship between representations and learning methods. In Chapter 3, the performance of VF and PS methods was compared on an abstract problem class, in which representation could be systematically varied. A clear result that emerged from these experiments was that VF methods do not perform as well as PS methods when the representation (in particular the expressiveness of the generalization scheme) is deficient. In this chapter, we have gone a step further to analyze *why* VF methods are affected more adversely by poor representations than are PS methods. Rather than study this phenomenon in a general and simplified context, we have focused on an existing benchmark task, on which previous work points to a significant gap between the performance of VF and PS methods. Specifically we consider Tetris, the popular video game, which has been used as a benchmarking task for over a decade.

In their early work outlining the limitations of value function-learning with imperfect representations, Baxter and Bartlett (2001) provide a concise illustration of an MDP and a representation scheme under which the best approximation of the optimal value function gives rise to a sub-optimal policy. In this chapter, our aim is to verify whether a similar phenomenon is behind the failure of VF methods in Tetris.

Several attributes of the Tetris MDP make it a hard problem for learning. These attributes include the large number of actions that can be taken from each state, the high stochasticity in transitions, and importantly, that "better performance" implies "longer episodes". Rather than testing the actual performance of a VF learning method that has to deal with these problematic factors, we directly test the limit that such a method can achieve for a given choice of representation. Specifically we use a linear representation introduced by Bertsekas and Tsitsiklis (1996) in our evaluations. Under the resulting architecture, the large number of actions from each state is not a concern (except for a computational overhead) since a forward model is available, and we can learn an evaluation function over afterstates. To cope with the high stochasticity in transitions, we perform extensive Monte Carlo simulations (for several days on a computing cluster). Also, conceding the impracticality of evaluating policies that clear more than a few thousands of lines per episode, we hand-code a policy, π^{HC}, that clears roughly 1,000 lines per episode.

It appears unlikely that VF learning algorithms—without access to restarts and to the sheer scale of Monte Carlo simulations we undertake in our experiments—can converge in a reasonable amount of time to $\widehat{V^{\pi^{HC}}}$, the best approximation of the value function of π^{HC}. However, our philosophy is to grant such methods the improbable concession that indeed they will converge to $\widehat{V^{\pi^{HC}}}$, and then consider the consequences of having done so. We note that $\widehat{V^{\pi^{HC}}}$ could be defined based on multiple weighing distributions, and therefore examine three natural choices: D_1, the stationary distribution of afterstates under π^{HC}; D_2, the stationary distribution of afterstates under a random policy; and D_3, a uniform distribution over the afterstate space. For each of these weighting distributions, we estimate the best approximation value functions: $\widehat{V^{\pi^{HC}}_{(D_1)}}$, $\widehat{V^{\pi^{HC}}_{(D_2)}}$, and $\widehat{V^{\pi^{HC}}_{(D_3)}}$, respectively. Next we consider greedy policies resulting from these approximate value functions, and find that they clear 70, 28, and 29 lines per episode, respectively. In short our experiments demonstrate that approximate policy improvement starting with π^{HC} results in policies that are significantly inferior. The implication is that the linear generalization scheme employed is incapable of delivering policy improvement, which is a key element of value function-based learning of control.[5]

The work reported in this chapter is predated by a similar line of research undertaken by Szita and Szepesvári (2010b). Unfortunately this author remained unaware of their work until after completing the experiments for this chapter. Szita and Szepesvári introduce "SZ-Tetris", a simplified form of the game in which only `S` and `Z` tetrominoes are generated (with equal probability). These tetrominoes are the most "unwieldy" among the standard seven, and consequently, even well-tuned policies for SZ-Tetris do not clear more than a few hundred lines. The authors argue that the short episodes and the reduced stochasticity make SZ-Tetris a useful benchmark for understanding the limitations of value function-based methods for RL. Using the benchmark, they conduct experiments with different features, learning methods, and exploration techniques. The fundamental difference between the experiments reported in this chapter and those of Szita and Szepesvári is that rather than evaluate learning methods, we directly test the limit of the best results a VF method might achieve (in terms of approximating the value function). We demonstrate that subsequent approximate policy improvement could indeed degrade performance. Our conclusion concurs with Szita's succinct summing-up in a presentation on the subject: "with lame features, good approximation $\neq$ good control."[6] Had we conducted the experiments reported in this chap-

[5]Note that we classify LSTD(1) as a batch method (under the category "MB") in Appendix C. With the linear architecture used in our experiments, MB methods could be expected to suffer the same fate as VF methods.

[6]See: `http://barbados2011.rl-community.org/program/SzitaTalk.pdf`.

ter on SZ-Tetris, rather than on (standard) Tetris, it is likely that we would have obtained more reliable Monte Carlo estimates of the best value function approximation (as measured using quantities MSPS and MSPD defined in 4.3.2).

We view Tetris as a fairly extreme example of a task in which imperfect representations thwart VF methods from learning effective policies. Nevertheless, anywhere VF methods are applied in practice, they are likely to be affected, to varying degrees, by the phenomenon isolated here. We hope that our illustration draws the attention of researchers to the important role representations play within the overall architecture of a learning agent.

One important line of research to extend our work is that of developing representations for Tetris that can enable VF methods to learn much better policies than they current are able. We hope that our descriptions in this chapter of the task, learning architectures, and related work, will aid interested researchers in pursuing such research. For a start, it would be useful to examine whether intuitive features other than the 22 that we have considered here might improve the performance of approximate policy improvement. The interested reader is encouraged to review the summary of features provided by Thierry and Scherrer (2010a, see Table 1). We think of it as an open challenge to demonstrate that given a reasonable representation, VF methods can learn policies that clear millions of lines per episode in Tetris. To this end, it might be convenient to first experiment within a more constrained environment such as SZ-Tetris.

The results in this chapter also lend support to a complementary perspective that constitutes the basis of this dissertation. If a researcher implementing, say, a VF method on some problem, fails to make progress, then how might he/she proceed? Working on improving the representation is one possibility; the other option is to change the learning algorithm itself. Surely the spectacular results achieved by Szita and Lőrincz (2006) by using the cross-entropy method on the same representation used by Bertsekas and Tsitsiklis validate the merit in the latter alternative. We hope that our work in this chapter prompts further research on intelligently picking learning methods when provided a representation.

Our experiments in this chapter provide some insight on how representation handicaps VF methods in Tetris. It would be equally interesting to probe more deeply into the properties of Tetris that allow PS methods to succeed even with fairly simple representations.

Along with Chapter 3, this chapter has presented experiments and analysis to understand the role of representations in learning. The next two chapters in the dissertation act on the lessons we have thereby learned. One of the observations we made in Chapter 3 was that even if PS methods are able to perform well with poor representations, they are relatively sample-inefficient. In Chapter 5, we investigate a mechanism for improving the sample-efficiency of PS methods such as CMA-ES, which rely only on identifying the best subset among a population of policies in order to determine the next generation. We formalize subset selection in a multi-armed bandit setting, devise sampling algorithms for this setting, and derive theoretical bounds on the sample complexity. In Chapter 6, we explore possibilities for integrating the strengths of VF and PS methods. We proceed by way of two case studies, and demonstrate the effectiveness of our learning strategies both on the suite of parameterized learning problems developed in Chapter 3, and on the more complex task of robot soccer Keepaway (Stone et al., 2005).

Chapter 5

Subset Selection and Efficient Policy Search

This chapter presents the first of the two major algorithmic contributions of this dissertation, which were introduced in Section 1.3.3. Specifically this chapter considers the general, widely applicable problem of selecting, from n real-valued random variables, a subset of size m of those with the highest means, based on efficiently sampling the random variables. This "subset selection" problem is relevant as a subroutine within ranking-based policy search methods such as CMA-ES and the cross-entropy method (CEM). Recall from Chapter 3 that these methods tend to be less sample-efficient than value function-based methods. When fitness evaluations are noisy (as they are while evaluating policies in stochastic MDPs), efficient subset selection can significantly improve the sample-efficiency of policy search. Subset selection also finds application independently in a variety of other areas, such as simulation, industrial engineering, and on-line advertising.

In this chapter, we present a detailed study of the subset selection problem. We formalize the problem using stochastic multi-armed bandits, where each arm corresponds to a random variable (sections 5.1 and 5.2). We consider three relevant operational settings: probably approximately correct (PAC) selection (Section 5.3), simple regret (SR) minimization (Section 5.4), and cumulative regret (CR) minimization (Section 5.5). Under all three settings, we generalize existing problem formulations that focus on the selection of a single arm (that *is, $m = 1$) to instead select a subset of arms. In the PAC setting, we introduce a new algorithmic framework, using which we derive sample complexity bounds that improve existing ones when instantiated with $m = 1$. In the SR and CR settings, our algorithms and regret bounds essentially generalize existing algorithms and bounds. We demonstrate that our algorithm for the SR setting indeed improves the performance of CMA-ES and CEM on the suite of parameterized learning problems introduced in Chapter 3. The algorithms developed in this chapter apply well beyond the realm of this dissertation.*

In chapters 3 and 4, we presented experimental studies investigating the effect of representations on VF and PS methods. In this chapter, we present an algorithmic contribution that is directly motivated by the results from the previous chapters. Specifically we noticed in Chapter 3 that PS methods tend to be significantly less sample-efficient than VF methods, even if they are relatively more robust towards severely deficient representations. Can PS methods such as CMA-ES and the cross-entropy method (CEM) be made more sample-efficient?

A main source of inefficiency in ranking-based PS methods such as CMA-ES and CEM is in their selection routines. Recall that these methods, and several other evolutionary strategies, loop over the following steps:

1. Generate a population of individuals.

2. Evaluate the fitness of each individual.

3. Select a fixed-size subset of individuals with the highest fitness, and use the selected individuals to seed the next generation.

In the context of RL, each “individual” corresponds to a policy, and “fitness” equates with the expected long-term reward accrued by policies. In general since there is stochasticity in MDPs, fitness estimation requires averaging over a sufficient number of rollouts (or episodes) to reduce noise. Herein arises the question of how many rollouts must be performed with each policy in order to achieve reliable selection of the best subset of policies. In Chapter 3, under all the PS methods we implemented, we performed the same number of rollouts on each individual in every generation. However, surely we could be more judicious in allocating fitness evaluations. Suppose we notice after only a few evaluations that some individuals are clearly superior (or inferior) to others, it seems natural that we must devote fewer evaluations (or *samples*) to such individuals in the future, when compared to individuals whose rank appears more uncertain.

This chapter presents an extensive investigation into the subset selection problem. While our motivations stem from the relevance of the problem to policy search, we treat the problem in isolation and study it within the formal framework of multi-armed bandits. In particular we consider three problem settings: probably approximately correct (PAC) subset selection, simple regret (SR) minimization, and cumulative regret (CR) minimization. Under each of these settings, we design algorithms for subset selection that generalize previous work devoted to identifying just the single best arm (a subset of size $m = 1$) in a an n-armed bandit. Our algorithms under the PAC setting are qualitatively distinct from the “elimination”-based approaches pursued in previous work, and indeed achieve better bounds when instantiated in the $m = 1$ case. Under the SR and CR settings, we generalize previous work. When we deploy our algorithm for the SR setting as a selection subroutine within

CMA-ES and CEM, we find that the performance of these PS methods indeed improves on the parameterized learning problems introduced in Chapter 3.

The remainder of this chapter is organized as follows. In Section 5.1, we provide a brief background of the subset selection problem and its formulation within the PAC, SR, and CR frameworks. In Section 5.2, we provide formal definitions of the PAC-m, SR-m, and CR-m problems. Each of sections 5.3, 5.4, and 5.5 is devoted to one of these settings, presenting our algorithms along with accompanying analysis. Additionally Section 5.4 includes experimental results applying our algorithms for SR-m within CMA-ES and CEM. While we present the statements of our main formal results in sections 5.3, 5.4, and 5.5, we defer the proofs to Section 5.6, where we present the proofs in aggregate. We conclude the chapter with a summary and discussion in Section 5.7.

5.1 Subset Selection: Background

The problem of *efficient subset selection* is as follows: given n real-valued random variables, identify m among them with the highest means, based on efficiently sampling the random variables. This basic problem finds application in a variety of contexts, and over the years, has been studied under various guises. In an early study, Becker (1961) considers the problem of confidently identifying the stocks of poultry laying the highest-quality eggs, based on samples of eggs. Dalal and Srinivasan (1977) devise sampling strategies for advertising "pretests", to help determine which subset among a pool of advertisements would be the most profitable to ultimately publish in some magazine or newspaper. The advent of the Internet has resulted in a similar interest within on-line advertising (Kale et al., 2010). Owing to these and other applications in systems simulation, the subset selection problem has received a significant amount of attention from the operations research and industrial engineering community (Koenig and Law, 1985; Sullivan and Wilson, 1989; Kim and Nelson, 2007). As mentioned earlier, our study of subset selection within this dissertation is motivated by its occurrence as a crucial subroutine within certain evolutionary (policy search) algorithms, in which the fittest individuals in a population must be identified based on noisy samples of fitness (Schmidt et al., 2006).

In this chapter, we adopt the framework of stochastic multi-armed bandits (Robbins, 1952; Berry and Fristedt, 1985) to ground our pursuit of algorithms for subset selection. Multi-armed bandits are concise abstractions of the problem of sequentially exploring a number of alternatives and picking desirable ones. A classical example is the case of a physician who has several drugs for treating a certain illness. Over time spent testing the effects of these drugs on patients, the physician must learn to administer the more effective ones (Chernoff, 1967). Whereas this physician might ultimately identify the single best drug for the illness, in our applications of interest, the final solution corresponds to a fixed-size *subset* from the set of alternatives. To model subset selection, we consider an n-armed bandit (each of whose arms corresponds to a random variable), in which our task is to select m arms that yield the highest expected rewards. Now, suppose indeed we design an algorithm for this subset selection task, how might we evaluate it? Depending on the application, several natural alternatives arise in terms of defining the precise objectives for efficient subset selection. We consider three such objectives: probably approximately correct (PAC) selection, simple regret (SR) minimization, and cumulative regret (CR) minimization. We proceed to introduce these settings against a backdrop of related work.

5.1.1 PAC Setting

Under the PAC setting, an algorithm is expected to return a subset of m arms that, with high *probability*, is of a high *quality*. For given specifications of the (mistake) probability and the quality, we desire algorithms that satisfy the specifications based on a minimal amount of sampling. The theoretical basis for our work in this setting is provided by recent research from Even-Dar et al. (2006). In their formulation, the aim is to minimize the total number of samples needed to reliably identify an arm with an expected reward that is within ϵ of the maximum achievable (an "ϵ-optimal" arm). The parameter ϵ serves to specify tolerance. Casting the problem into a PAC framework, Even-Dar et al. (2006) provide an algorithm that identifies an ϵ-optimal arm in an n-armed bandit with probability at least $1 - \delta$, incurring a sample complexity that is $O(\frac{n}{\epsilon^2} \log(\frac{1}{\delta}))$. This sample complexity *matches*, up to constants, a lower bound derived by Mannor and Tsitsiklis (2004).

In this chapter, we generalize the problem formulated by Even-Dar et al. (2006): our aim is to identify m arms of an n-armed bandit such that the expected reward of each chosen arm is at least $p_m - \epsilon$, where p_m is the m^{th} highest expected reward among the bandit's arms. We denote this problem "PAC-m" (thus, the special case studied by Even-Dar et al. (2006) is PAC-1). We devise algorithms for PAC-m and analyze their sample complexity. First we consider "fixed-sample-complexity" (FSC) algorithms, whose sample complexity is determined by n, m, and relevant PAC parameters ϵ and δ. Among such algorithms, we show that our "Halving" algorithm enjoys a sample complexity that is $O(\frac{n}{\epsilon^2} \log(\frac{m}{\delta}))$. The main practical drawback of FSC algorithms is that they are designed for the worst case: their *expected* sample complexity on a problem does not reduce even if the actual "degree of separation" between the highest m and the lowest $n-m$ arms is large. In response we design "variable-sample-complexity" (VSC) algorithms for PAC-m, whose termination depends on the actual rewards obtained during the sampling procedure. We show that the expected sample complexity of these "fully sequential" algorithms (Girshick, 1946a,b; Wald, 1947) can be bounded in terms of a natural parameter measuring problem hardness. Indeed the sample complexity bounds of our VSC algorithms for PAC-1 improve the best ones existing in the literature (Even-Dar et al., 2006). Another novel feature of our VSC algorithms is that unlike elimination-based algorithms, they have a clear decoupling between the sampling strategy and the stopping criterion. Consequently we can apply our stopping criterion with any arbitrary sampling strategy in order to yield a PAC algorithm.

Among the settings we consider in this chapter, the PAC setting is philosophically the closest to the study of subset selection within the operations research community. "Ranking and selection" is today an established area of study within operations research, as evinced by the existence of several textbooks on the topic (Bechhofer et al., 1995; Gibbons et al., 1999; Gupta and Panchapakesan, 2002). As with our PAC setting, typical problem formulations in the ranking and selection literature involve specifying quality requirements on the returned solution, and attempting to minimize the number of samples needed to achieve the desired quality. The "indifference zone" formulation (Bechhofer et al., 1995, see chapter 2) is very similar to our PAC setting: solutions are deemed correct if the worst among the selected arms do not fall below some indifference range relative to the best (as determined by ϵ in PAC-m). The objective is to maximize the probability of selecting correct solutions. Under the "subset selection" framework (Bechhofer et al., 1995, see chapter 3), the size of the subset returned is itself a random variable: the objective is to return a "small" subset that has a high probability of containing the best arms.[1] A third problem formulation involves "multiple comparison" approaches (Bechhofer et al., 1995, see chapter 4), wherein confidence intervals are obtained at a specified level of significance to *simultaneously* cover, say, pairwise differences between the means of the arms.

Predominantly, the ranking and selection literature has focused on devising rules to set sample sizes under specific assumptions on the distributions of the bandit arms. For example, the textbook by Gibbons et al. (1999) provides tables for normal, multinomial, and gamma distributions. By contrast, in this chapter, we remain agnostic towards the specific parametric form of the arms' distributions: we only assume that the distributions have a bounded support, in order that we may apply Hoeffding's inequality. Our approach is in line with the objective of obtaining fundamental distribution-independent bounds on the sample complexity. In practice we believe that it could be profitable to meld the distribution-dependent rules devised in the ranking and selection literature with the economical sampling strategies in our VSC algorithms. We hope to investigate this possibility in future research.

5.1.2 Simple Regret Setting

Obtaining samples from bandit arms is typically expensive, and in some situations, we may not be able to afford enough samples to achieve a preset PAC criterion. Rather, the more meaningful modeling approach in some cases

[1]Thus, our use of the term "subset selection" to describe the work in this chapter (in which we always assume that the size of the returned subset is fixed to be m) is not to be confused with the technical interpretation of the term in the ranking and selection literature (in which the size of the returned subset is random).

would be to assume that we are given a fixed number of samples, T, and consider how we may best allocate these samples to the arms such that *Selected*, the m-sized subset returned after T rounds, is as close as possible to optimal. If p_a is the mean of the distribution corresponding to arm a in a bandit instance, and Top is an m-sized subset containing the arms with the m highest means, then a natural measure of the sub-optimality of an algorithm on the bandit instance is its "simple regret", defined as: $\mathbb{E}\left[\sum_{i \in Top} p_i - \sum_{a \in Selected} p_a\right]$.

The simple regret setting is an instance of budgeted learning, wherein sampling each arm is associated with a cost, and there is a predefined limit for the total sampling cost. Certain *computational* problems associated with budgeted learning—such as computing sampling strategies for identifying the best arm, when prior distributions on the means of the arms are *known*—have been shown to be hard (Madani et al., 2004). Nevertheless, recent work has come up with approximate strategies for such computational problems (Guha and Munagala, 2007) and heuristic treatments for related learning problems (Tran-Thanh et al., 2010).

The definition of simple regret we have provided above generalizes an existing definition for $m = 1$ (Bubeck et al., 2009; Audibert et al., 2010). Indeed, in this chapter, we generalize the problem formulation introduced by Audibert et al. (2010) for the case of $m = 1$, referring to our generalized problem as SR-m. Correspondingly we also generalize their sampling algorithms, definitions of problem hardness, analyses, and bounds on the simple regret. We then apply the generalized algorithm as a subset selection routine within CMA-ES and CEM.

5.1.3 Cumulative Regret Setting

Under both PAC-m and SR-m settings, algorithms are evaluated solely based on the subset of arms they ultimately return—the actual sequence of payoffs an algorithm receives during the course of its run is ignored by the performance measures. In this sense, PAC-m and SR-m can be considered "off-line" settings. Now, consider an advertising agency that has a pool of advertisements to display on a web page, but can only display a small subset of them on any given day. The profits earned by the agency depend on the number of clicks being generated through the advertisements. In such a scenario, it would not be in the agency's interest to ignore the rewards (clicks) that are being accrued as the algorithm is learning. Rather, the most appropriate evaluation measure of algorithms would be the amount of clicks they generate even while they adapt.

To formalize the above requirement, let us consider a bandit instance that will be sampled for T rounds. Unlike in the PAC-m and SR-m settings, now we assume that an m-sized subset of arms, A^t, will be sampled during every round t. If so, the expected rewards that will be accrued by the algorithm, through round T, would amount to $\mathbb{E}\left[\sum_{t=1}^{T}\sum_{a\in A^t} p_a\right]$. Maximizing this quantity becomes equivalent to minimizing the "cumulative regret", which is defined to be $\mathbb{E}\left[\sum_{t=1}^{T}\left(\sum_{i\in Top} p_i - \sum_{a\in A^t} p_a\right)\right]$ (note that the variable m is implicit in this definition, as the size of the sets Top and A^t).

Traditionally, cumulative regret has been the focus of studies involving multi-armed bandits (Robbins, 1952): in order to minimize cumulative regret, algorithms have to achieve an optimal balance between *exploration* (sampling those arms about whose means there is uncertainty) and *exploitation* (sampling seemingly optimal arms to accumulate greater payoffs). By contrast note that our PAC-m and SR-m problems are akin to "pure exploration" settings, where the rewards earned while learning are inconsequential (Bubeck et al., 2009).

For the case of $m = 1$, Lai and Robbins (1985) provide sampling algorithms that asymptotically achieve the optimal cumulative regret. Auer et al. (2002a) show that optimal cumulative regret can be achieved in finite time using the UCB algorithm. In this chapter, we generalize the UCB algorithm appropriately for the CR-m setting. Thereby we obtain bounds on the cumulative regret that generalize the bounds for the $m = 1$ case.

It is worth noting here that in this chapter, our attention is restricted to stochastic bandits; that is, the rewards from each arm are assumed to be drawn i.i.d. from a fixed distribution. By contrast, several studies consider adversarial settings, in which no such statistical assumptions are made (Auer et al., 2002b). Specifically relevant to our focus on subset selection is the "non-stochastic bandit slate" formulation of Kale et al. (2010), wherein a *slate* (a fixed number of distinct arms) must be selected at every time step. The authors consider both ordered slate problems (under which the selected arms must be ordered) and unordered slate problems (like our setting, where the selected arms are unordered). In turn, they devise algorithms for these problems both under the "bandit" assumption (every round, only the reward from the sampled arm is seen) and the "experts" assumption (rewards corresponding to all the arms recommended by a set of experts are visible). In all cases, cumulative regret is defined with respect to having taken the best action(s) in hindsight against some adversary.

In most bandit formulations (including ours and that of Kale et al. (2010)), an arbitrary arm (or subset of arms) may be sampled during every round. However, in the "sleeping experts and bandits" setting Kleinberg et al.

(2010), an adversary determines a subset of arms at every round—only arms from this subset may be sampled. Technically, this setting has little in common with CR-m, where it is the learning agent that has to select a subset of arms every round.

5.2 Problem Definitions and Related Terminology

In this section, we provide a formal definition of stochastic multi-armed bandits, and specify templates for algorithms under each of our three problem settings. We also introduce useful notation and define terms related to problem hardness.

5.2.1 Stochastic Multi-armed Bandits

We consider an arbitrary instance of an n-armed bandit, $n \geq 2$; let its arms be numbered $1, 2, \ldots, n$. Each sample (or "pull") of arm a yields a reward of either 0 or 1, generated randomly from a fixed Bernoulli distribution with mean $p_a \in [0, 1]$. Indeed each bandit instance is completely determined by the distributions corresponding to its arms. Without loss of generality, we assume that the indices of the arms are in non-increasing order of their means, with ties broken deterministically using some fixed rule. Thus:

$$p_1 \geq p_2 \geq \ldots \geq p_n. \tag{5.1}$$

The random variables generating rewards for the arms are mutually independent.[2] Arm a is defined to be (ϵ, m)-optimal, $\forall \epsilon \in (0, 1)$, $\forall m \in$

[2]The algorithms and analysis in this paper easily extend to the case where the distributions of the arms have *bounded* ranges with width not exceeding some fixed, known quantity.

$\{1, 2, \ldots, n-1\}$, iff

$$p_a \geq p_m - \epsilon. \tag{5.2}$$

We find it convenient to denote as *Arms* the set of all arms in our n-armed bandit instance:

$$Arms \overset{\text{def}}{=} \{1, 2, \ldots, n\}.$$

We also define two sets, *Top* and *Bot*, as follows:

$$\begin{aligned} Top &\overset{\text{def}}{=} \{1, 2, \ldots, m\}, \text{ and} \\ Bot &\overset{\text{def}}{=} \{m+1, m+2, \ldots, n\}. \end{aligned}$$

We see from (5.1) and (5.2) that every arm in *Top* is (ϵ, m)-optimal.[3] Hence, there are *at least* m (ϵ, m)-optimal arms. Let *Good* be the set of all (ϵ, m)-optimal arms, and let the set *Bad* contain all the remaining arms. In general: $m \leq |Good| \leq n$ and $0 \leq |Bad| \leq (n - m)$.

[3]Note that the set *Arms* depends on n, the set *Top* depends on m, and the set *Bot* depends on both n and m. Many other variables we subsequently define in this chapter also depend on n and m, which are the primary inputs in our subset selection problem. To reduce notational burden, we leave these dependencies on n and m implicit unless the context demands otherwise. Thus, we stick with *Arms*, *Top*, and *Bot* rather than use, say, $Arms(n)$, $Top(m)$, and $Bot(n, m)$. However, we find it appropriate to denote our problem settings PAC-m, SR-m, and CR-m (rather than PAC, SR, and CR), as we specifically compare these problem settings with PAC-1, SR-1, and CR-1, respectively.

5.2.2 Algorithms

A bandit instance may be sampled sequentially in rounds. Under PAC-m and SR-m, during each round $t \in \{1, 2, \dots\}$, some arm, a^t, is selected to be sampled. By contrast, under CR-m, during each round $t \in \{1, 2, \dots\}$, some m-sized subset of arms, A^t, is selected to be sampled. The outcome of the sampling is a pair of the form (arm, reward) under PAC-m and SR-m, and m pairs of the form (arm, reward) under CR-m. In each case, the reward is drawn as an independent sample from the distribution associated with the arm. We refer to the sequence of outcomes up to (and *excluding*) round t as the *history* at round t. Below we provide full specifications of algorithms under the PAC-m, SR-m, and CR-m settings.

PAC-m. Under PAC-m, on each round t, an algorithm must either (1) select an arm a^t to sample, or (2) terminate and return an m-sized subset of $Arms$. Problem parameters n, m, ϵ, and δ are provided as input to the algorithm; also, during round t, the history at round t is available to the algorithm. Note that we do not require the m arms returned by the algorithm to be in any particular order.

For $\delta \in (0, 1)$, an algorithm $\mathcal{A}$ is defined to be (ϵ, m, δ)-optimal, iff *for every bandit instance*: (1) every run of $\mathcal{A}$ terminates with probability 1, and (2) with probability at least $1 - \delta$, every arm returned by $\mathcal{A}$ is (ϵ, m)-optimal. Note that the PAC-1 setting (for which $(\epsilon, 1, \delta)$-optimal algorithms are to be designed) matches the problem formulation of Even-Dar et al. (2006).

The *sample complexity* of algorithm $\mathcal{A}$ during a terminating run is the total number of pulls it performs before termination. Our objective in this chapter is to design (ϵ, m, δ)-optimal algorithms and to bound the sample complexity of their runs. First, in Section 5.3.1, we propose "fixed-sample-complexity" (FSC) algorithms, which, for given problem parameters (n, m, ϵ, and δ), incur exactly the same sample complexity on every run of every bandit instance. Thus, in the context of FSC algorithms, we may refer to the sample complexity of each run as the sample complexity of the algorithm itself; we bound this sample complexity in terms of n, m, ϵ, and δ. In Section 5.3.2, we design "variable-sample-complexity" (VSC) algorithms for PAC-m, which, for given problem parameters and a given bandit instance $\mathcal{B}$, can lead to runs with different sample complexities. Thus, for a given bandit instance $\mathcal{B}$, a VSC algorithm may have different expected and worst case sample complexities. In this case, again we bound the worst case complexity in terms of n, m, ϵ, and δ, but importantly, we bound the expected sample complexity by a smaller quantity that depends on the hardness of $\mathcal{B}$. We also obtain high-probability bounds on the sample complexity of runs on $\mathcal{B}$.

Algorithms under PAC-m are governed by four parameters: n, m, ϵ, and δ. In our problem definition above, we have purposefully set the ranges for these parameters—$n \geq 2$, $m \in \{1, 2, \ldots n-1\}$, $\epsilon \in (0, 1)$, and $\delta \in (0, 1)$—such that they do not lead to trivial or impractical solutions. In particular, setting $n = 1$; $m = n$; $\epsilon = 1$; or $\delta = 1$ would facilitate (ϵ, m, δ)-optimal algorithms that do not need to sample the arms at all. Setting $\epsilon = 0$ or $\delta = 0$ would not allow for (ϵ, m, δ)-optimal algorithms that can always terminate after a finite number of samples.[4]

[4]If we only consider bandit instances in which $p_m > p_{m+1}$ (strict inequality), termination can indeed be achieved in finite time while meeting the relevant PAC requirement even with $\epsilon = 0$. A similar possibility arises with $\delta = 0$ for the set of bandit instances in which every arm a is such that $p_a = 0$ or $p_a = 1$.

SR-m. Under SR-m, on each round t, an algorithm must select an arm a^t to sample. The algorithm is run for T rounds, where $T \geq n$. After T rounds, the algorithm must return a subset of size m; let us denote this subset *Selected*. Problem parameters n, m, and T are provided as input to the algorithm, and as under PAC-m, the history is available at each round. For a given bandit instance, the simple regret of an algorithm is defined to be

$$\mathbb{E}\left[\sum_{i \in Top} p_i - \sum_{a \in Selected} p_a\right],$$

where the expectation is over the probability space of the distributions of its arms, and possible randomization of the algorithm. We aim to develop algorithms that will have a low simple regret on every bandit instance.

CR-m. Under CR-m, algorithms are run for T rounds, where $T \geq \left\lceil \frac{n}{m} \right\rceil$. On each round $t \in \{1, 2, \ldots, T\}$, an algorithm must select an m-sized subset of arms A^t to sample (however, the algorithm itself does not know T). Observe that under CR-m, the total number of samples will therefore be mT, whereas under SR-m, it is T. To start with, the algorithm is provided n and m as input parameters, and at each round t, it can use the current history to inform its selection of A^t. Unlike in PAC-m and SR-m, the algorithm is evaluated continuously, rather than solely based on the subset of arms selected at termination. The cumulative regret of an algorithm for a given bandit instance is defined as

$$\mathbb{E}\left[\sum_{t=1}^{T}\left(\sum_{i \in Top} p_i - \sum_{a \in A^t} p_a\right)\right], \tag{5.3}$$

where the expectation is over the probability space of the distribution of the arms and possible randomization of the algorithm. We aim to develop algorithms that have minimal cumulative regret on every bandit instance.

The three settings above share the ultimate goal of identifying m-sized subsets of arms with high rewards. The main distinction between PAC-m and SR-m on the one hand, and CR-m on the other, is that the latter needs to achieve a profitable balance between exploration and exploitation, whereas the former may focus solely on exploration. PAC-m and SR-m are philosophically complementary: under PAC-m, algorithms must try to minimize the number of samples needed to achieve a desired quality of selection, whereas under SR-m, algorithms must maximize the quality of selection when specified a quota of samples. These contrasts among the various settings lead to significant technical differences in the algorithms we develop for them. Table 5.1 summarizes the characteristics of the three problem settings and algorithm definitions under each.

5.2.3 Hardness Quantities

Whereas the input parameters to our various algorithms are n, m, ϵ, δ, and T, the performance bounds we obtain for the algorithms additionally depend on the "hardness" of the specific bandit instances on which they are applied. The one exception is that the sample complexity of our FSC algorithms for PAC-m only depend on n, m, ϵ, and δ. A bandit instance is fixed by the means corresponding to its arms $(p_1, p_2, \ldots, p_n)$; recall from (5.1) that the means are in non-increasing order. Below we define additional problem-specific constants that occur in our bounds.

Table 5.1: Summary of problem settings and associated algorithms. Under LUCB, which is one of our algorithms for PAC-m, <u>two</u> arms get sampled every round, rather than one.

Property	**PAC-m**	**SR-m**	**CR-m**
Generalizes:	PAC-1 (Even-Dar et al., 2006)	SR-1 (Audibert et al., 2010)	CR-1 (Auer et al. 2002)
Input	n, m, ϵ, δ	n, m, T	n, m
Number of arms sampled per round	1	1	m
Termination	Determined by algorithm	After T rounds	After T rounds
Requirement of algorithm	Must be (ϵ, m, δ)-optimal	None	None
To minimize:	Sample complexity	Simple regret	Cumulative regret

We denote by $\Delta_{i,j}$ the separation between the means of arms i and j:

$$\Delta_{i,j} \stackrel{\text{def}}{=} p_i - p_j.$$

In all three of our problem settings, note that it would be optimal to select arms 1 through m (and discard arms $m+1$ through n). Hence, arms m and $m+1$ assume a special significance in our analysis. We find it convenient to use shorthand for the separation of arms in Top from arm $m+1$, and the separation of arms in *Bot* from arm m. The quantity Δ_a defines this separation for arm a:

$$\Delta_a \overset{\text{def}}{=} \begin{cases} \Delta_{a,m+1} & \text{if } 1 \leq a \leq m, \\ \Delta_{m,a} & \text{if } m+1 \leq a \leq n. \end{cases}$$

Observe that $\Delta_m = \Delta_{m+1} = p_m - p_{m+1}$. Let c denote the mid-point of the means of arms m and $m+1$:

$$c = \frac{p_m + p_{m+1}}{2}.$$

We have:

$$\forall a \in Arms : \frac{\Delta_a}{2} \leq |p_a - c| \leq \Delta_a. \tag{5.4}$$

To see the inequality above, consider, without loss of generality, that arm a is in Top. Then,

$$\Delta_a = p_a - p_{m+1} \geq p_a - c = p_a - \frac{p_m + p_{m+1}}{2} = \frac{\Delta_a}{2} + \frac{p_a - p_m}{2} \geq \frac{\Delta_a}{2}.$$

For $\gamma \in [0,1]$, we denote the larger of Δ_a and γ as $[\Delta_a \vee \gamma]$ (the term $[\Delta_a \vee \frac{\epsilon}{2}]$ occurs in our bounds for PAC-m).

$$[\Delta_a \vee \gamma] \overset{\text{def}}{=} \max\{\Delta_a, \gamma\}.$$

Larger values of Δ_a in a bandit instance indicate a higher degree of separation between the highest m and lowest $n-m$ means corresponding to the arms; the quantities H_1 and H_1^γ defined below aggregate the individual Δ_a values to yield natural measures of problem hardness.

$$H_1 \stackrel{\text{def}}{=} \sum_{a=1}^{n} \frac{1}{\Delta_a^2}.$$

$$H_1^\gamma \stackrel{\text{def}}{=} \sum_{a=1}^{n} \frac{1}{[\Delta_a \vee \gamma]^2}.$$

Note that our definitions of Δ_a and H_1 generalize the same terms employed by Audibert et al. (2010) for the case of $m = 1$. The quantity H_1^γ arises in our analysis of VSC algorithms for PAC-m: note that H_1^γ is well-defined for $\gamma > 0$, and is upper-bounded by $\frac{n}{\gamma^2}$. Also note that H_1 and H_1^γ are at least n.

Audibert et al. introduce a second hardness quantity, H_2 in their analysis of SR-1. For general m, let us define $H_2(i)$, for every arm $i \in Top$, as follows:

$$H_2(i) \stackrel{\text{def}}{=} \max_{j \in Bot} \frac{j - m + 1}{\Delta_{i,j}^2}.$$

Notice that unlike H_1, which is defined for the whole bandit instance, H_2 is defined only for arms i in Top. We may view $H_2(i)$ as an indicator of the hardness of separating arm i from arms in Bot. The simple regret bounds we derive in Section 5.4 contain terms of the form $H_2(i)$ for all $i \in Top$. Apart from its occurrence as a term in our analysis, it is not clear whether $H_2(i)$ has any independent significance. For $m = 1$, the quantity $H_2(1)$ is the same as the quantity H_2 defined by Audibert et al.

Observe that $\forall i_1, i_2 \in Top : i_1 < i_2 \implies H_2(i_1) \le H_2(i_2)$; thus, $H_2(m)$ is the largest among these terms. It it useful to bear in mind that H_1 and

$H_2(m)$ are within only a small factor of each other. In particular the following relationship between H_1 and $H_2(m)$ generalizes a result shown for $m = 1$ by Audibert et al. (2010); we include a proof in Appendix B.1.

$$H_2(m) \leq H_1 \leq \left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k} \right) H_2(m). \tag{5.5}$$

Note that $\sum_{k=2}^{n-m+1} \frac{1}{k} \leq \ln(n - m + 1)$.[5] In the next section, we present algorithms for PAC-m. Sections 5.4 and 5.5 respectively address SR-m and CR-m.

5.3 PAC Setting

In this section, we propose and analyze algorithms for PAC-m. First, in Section 5.3.1, we consider FSC algorithms. In Section 5.3.2.1, we propose "stopping rules" as a novel mechanism to decouple the sampling strategy of an algorithm from the analysis of the algorithm's mistake probability. Using this principle, we can apply our stopping rules to any sampling strategy and derive an (ϵ, m, δ)-optimal algorithm. This insight about stopping rules motivates our VSC algorithms for PAC-m, which we present in sections 5.3.2.2 and 5.3.2.3.

5.3.1 Fixed-Sample-Complexity Algorithms

Recall that fixed-sample-complexity (FSC) algorithms are those whose sample complexity (on every run on every n-armed bandit instance) is determined by problem parameters n, m, ϵ, and δ. Even-Dar et al. (2006)

[5]Throughout this chapter, we use ln to denote the natural logarithm and log to denote the logarithm to the base 2.

present two FSC, $(\epsilon, 1, \delta)$-optimal algorithms for PAC-1: a naïve algorithm that achieves a sample complexity of $O\left(\frac{n}{\epsilon^2}\log\left(\frac{n}{\delta}\right)\right)$, and a "median elimination" algorithm that improves the sample complexity to $O\left(\frac{n}{\epsilon^2}\log\left(\frac{1}{\delta}\right)\right)$. We generalize these existing algorithms to construct three (ϵ, m, δ)-optimal algorithms for PAC-m. Our first algorithm, Direct, essentially implements the naïve strategy of pulling each arm a fixed number of times. The second algorithm, Incremental, uses the median elimination algorithm as a subroutine. The sample complexities of both methods are improved by our third algorithm, Halving, which modifies the median elimination algorithm to account for m arms instead of 1.

5.3.1.1 DIRECT Algorithm

Under Direct (Algorithm 5.1), arms are sampled for a fixed number of rounds (line 2) such that with high probability, the m arms with the highest *empirical* averages (denoted $\hat{p}_a$ for arm a) are all (ϵ, m)-optimal.

Algorithm 5.1 Direct(n, m, ϵ, δ)

```
1: for all $a \in Arms$ do
2:     Sample arm $a$ for $\left\lceil \frac{2}{\epsilon^2}\ln\left(\frac{n}{\delta}\right)\right\rceil$ rounds; let $\hat{p}_a$ be its average reward.
3: Find $S \subset Arms$ such that $|S| = m$, and $\forall a_1 \in S\ \forall a_2 \in Arms \setminus S$:
   $\hat{p}_{a_1} \geq \hat{p}_{a_2}$.
4: Return $S$.
```

Theorem 5.1. Direct(n, m, ϵ, δ) *is* (ϵ, m, δ)*-optimal with sample complexity* $O\left(\frac{n}{\epsilon^2}\log\left(\frac{n}{\delta}\right)\right)$.

Proof. See page 184. □

5.3.1.2 INCREMENTAL Algorithm

Unlike the Direct algorithm, Incremental (Algorithm 5.2) proceeds through m phases, attempting to select an (ϵ, m)-optimal arm in each phase with high probability. At the beginning of phase l, S_l is the set of arms that have been selected, and R_l the set of arms remaining (line 1). During phase l, an $(\epsilon, 1)$-optimal arm in R_l is selected with high probability by invoking the median elimination algorithm (Even-Dar et al., 2006) (line 3). We can show that an $(\epsilon, 1)$-optimal arm in R_l is necessarily (ϵ, m)-optimal in $Arms$.

Algorithm 5.2 Incremental(n, m, ϵ, δ)

```
1: $S_1 \leftarrow \{\}$; $R_1 \leftarrow Arms$.
2: for $l = 1$ to $m$ do
3:     $a' \leftarrow$Median-Elimination$\left(R_l, \epsilon, \frac{\delta}{m}\right)$.
4:     $S_{l+1} \leftarrow S_l \cup \{a'\}$; $R_{l+1} \leftarrow R_l \setminus \{a'\}$.
5: Return $S_{m+1}$.
```

Theorem 5.2. Incremental(n, m, ϵ, δ) *is* (ϵ, m, δ)*-optimal with sample complexity* $O\left(\frac{mn}{\epsilon^2} \log\left(\frac{m}{\delta}\right)\right)$.

Proof. See page 185. ☐

5.3.1.3 HALVING Algorithm

While Incremental *selects* an arm during every phase, Halving (Algorithm 5.3) *eliminates* multiple arms every phase based on their inferior empirical averages. From R_l, the set of arms remaining at the beginning of phase l, half proceed to phase $l + 1$ (except that m proceed to the last phase). Arms are sampled for enough rounds during each phase (line 5) such that at least m (ϵ, m)-optimal arms are likely to survive elimination. Specifically, phase l is associated with parameters ϵ_l and δ_l, and we ensure that with probability at least $1-\delta_l$, the m^{th} highest true mean in R_{l+1} is not lower than the m^{th} highest true mean in R_l by more than ϵ_l. The sequences (ϵ_l) and (δ_l) (lines 2 and 9) are designed such that HALVING is (ϵ, m, δ)-optimal with sample complexity $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$.

Algorithm 5.3 HALVING(n, m, ϵ, δ)

```
1: $R_1 \leftarrow Arms$.
2: $\epsilon_1 \leftarrow \frac{\epsilon}{4}$; $\delta_1 \leftarrow \frac{\delta}{2}$.
3: for $l = 1$ to $\left\lceil \log\left(\frac{n}{m}\right)\right\rceil$ do
4:     for all $a \in R_l$ do
5:         Sample arm $a$ for $\left\lceil \frac{2}{\epsilon_l^2}\ln\left(\frac{3m}{\delta_l}\right)\right\rceil$ rounds; let $\hat{p}_a$ be its average re-
           ward.
6:     Find $R'_l \subset R_l$ such that $|R'_l| = \max\left(\left\lceil\frac{|R_l|}{2}\right\rceil, m\right)$, and $\forall a_1 \in R_l\ \forall a_2 \in$
       $R_l \setminus R'_l$: $\hat{p}_{a_1} \geq \hat{p}_{a_2}$.
7:     $R_{l+1} \leftarrow R'_l$.
8:     $\epsilon_{l+1} \leftarrow \frac{3}{4}\epsilon_l$; $\delta_{l+1} \leftarrow \frac{1}{2}\delta_l$.
9: Return $R_{\lceil \log(\frac{n}{m})\rceil + 1}$.
```

Theorem 5.3. HALVING(n, m, ϵ, δ) *is* (ϵ, m, δ)*-optimal with sample complexity* $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$.

Proof. See page 185. □

Let $T_{\text{HALVING}}(n, m, \epsilon, \delta)$ denote the exact sample complexity of the HALVING algorithm, when run with parameters n, m, ϵ, and δ. Our proof shows that $T_{\text{HALVING}}(n, m, \epsilon, \delta)$ is $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$. We conjecture that this sample complexity is indeed the lowest (up to a constant factor) that an (ϵ, m, δ)-optimal algorithm can achieve.

Conjecture 5.4. *There exist constants $C > 0$, $\epsilon_0 \in (0, 1)$, $\delta_0 \in (0, 1)$, $m_0 > 0$, and $f > 1$, such that $\forall \epsilon \in (0, \epsilon_0), \forall \delta \in (0, \delta_0), \forall m > m_0, \forall n > fm$, and for every (ϵ, m, δ)-optimal algorithm $\mathcal{A}$, there exists an n-armed bandit instance $\mathcal{B}$ on which the worst case sample complexity of $\mathcal{A}$ is at least $C \frac{n}{\epsilon^2} \log \left(\frac{m}{\delta}\right)$.*

Note that Mannor and Tsitsiklis (2004) prove such a sample complexity lower bound for the $m = 1$ case. We believe that their proof can be generalized to prove Conjecture 5.4.

5.3.2 Variable-Sample-Complexity Algorithms

Intuition would suggest that in bandit instances where the highest m and the lowest $n - m$ means of the arms are separated by a relatively large margin, or when arm distributions have low variances, fewer samples should suffice to reliably identify m (ϵ, m)-optimal arms. However, for given n, m, ϵ, and δ, note that each of the algorithms presented in Section 5.3.1 will perform exactly the same number of pulls, regardless of the arm distributions. These algorithms are designed to be *sufficient* for achieving a PAC guarantee in the *worst* case (when $p_1 - p_n$ is in the order of ϵ). Can we devise algorithms that incur fewer samples on "easy" bandit instances?

The situation we confront is akin to the one addressed by Schuurmans and Greiner (1995) in the general context of PAC learning. Correctness and efficiency are the two main requirements of a PAC algorithm. Schuurmans and Greiner show that while preserving the correctness guarantee over an entire class of problems, often it is possible to improve efficiency on a given problem instance by sequentially adapting based on the samples observed. Specifically they derive algorithms with such a property for the PAC-learning of concept classes, where a problem instance is defined by the distribution from which samples are drawn. In this setting, which involves supervised learning, the primary question facing the algorithm at every stage is whether to stop or to sample further. The authors apply the sequential probability ratio test (Wald, 1947) to inform this choice.

In our bandit setting, too, an algorithm has to decide whether or not to terminate after seeing some number of samples. However, if the algorithm decides to continue sampling, an additional question it faces is: "which arm to sample next?" Naturally, both the stopping criterion and the sampling strategy would affect the sample complexity of the algorithm. In this section, we design an algorithm for subset selection in the spirit of Schuurmans and Greiner. Our algorithm enjoys the following properties:

1. It meets the PAC correctness requirement.
2. It has the best-known worst case sample complexity.
3. Its expected sample complexity on a bandit instance depends on the hardness of that instance, and can be substantially lower than the worst case sample complexity. In particular when we instantiate our "high-probability bound" on the sample complexity for $m = 1$, the resulting bound improves the previously best one provided by Even-Dar et al. (2006) for PAC-1.

Interestingly our algorithm indeed maintains a clear separation between its stopping rule and sampling strategy. This approach contrasts with previous elimination-based algorithms for exploration problems (Even-Dar et al., 2006; Mnih et al., 2008), wherein these aspects are conflated. Maintaining a clear distinction allows us to develop a proof of correctness solely based on the stopping rule: this proof does not depend on the sampling strategy. The stopping rule itself assumes an intuitive form: (1) at every stage, we draw *lower* confidence bounds on the means of the m arms with the highest empirical averages, and *upper* confidence bounds on the $n - m$ arms with the lowest empirical averages; (2) the algorithm terminates if the difference between the highest upper confidence bound and the the lowest lower confidence bounds is less than ϵ. While this stopping rule can be coupled with any sampling strategy to realize an (ϵ, m, δ)-optimal algorithm, we show that it is economical to use a sampling strategy that is greedy with respect to the stopping rule. Under such a greedy sampling strategy, two arms are sampled at every stage: one with the lowest lower confidence bound (among the m arms with the highest empirical means), and one with the highest upper confidence bound (among the $n - m$ arms with the lowest empirical means).

Given the centrality of lower and upper confidence bounds in our stopping rule and sampling strategy, we denote our family of algorithms LUCB. After showing the correctness of using our stopping rule, we analyze the expected sample complexity of LUCB1, an algorithm that samples arms greedily for a specific choice of confidence bound. We conjecture that a small modification to this confidence bound can result in a more efficient sampling algorithm LUCB2. It is interesting to note the similarity between our LUCB algorithms, which achieve the best-known PAC sample complexity bounds, and the UCB algorithm (Auer et al., 2002a), which achieves optimal *regret*. Our proofs for the sample complexity bounds for LUCB involve many of the same techniques used by Auer et al. (2002a). In particular we bound the probability of events specific to each arm, and show that the sampling strategy induces a logical relationship between events corresponding to different arms.

5.3.2.1 Stopping Rule

Let us assume that there is some sampling strategy $\mathcal{S}$ according to which arms are being sampled: $\mathcal{S}$ is therefore a mapping from the set of histories to the set of arms. Our stopping rule does not depend on $\mathcal{S}$; rather, it is a mapping from the set of histories to the set $\{\texttt{STOP}, \texttt{CONTINUE}\}$.

The specific stopping rule we consider below does not need to maintain the entire history at every stage, but rather, only the number of times each arm has been sampled, and the arms' empirical means. During round t, let u_a^t denote the number of times arm a has been sampled, and let $\hat{p}_a^t$ be the empirical mean of the rewards from arm a. The key element of our stopping rule is a confidence bound $\beta(u_a^t, t)$, which is a positive number interpreted to be a bound on the deviation of the empirical mean of arm a from its true mean, for some given probability. In particular the lower confidence bound for arm a during round t is given by $\hat{p}_a^t - \beta(u_a^t, t)$, and the upper confidence bound is given by $\hat{p}_a^t + \beta(u_a^t, t)$.

During round t, let $High^t$ be the set of m arms with the highest empirical averages, and Low^t be the set of $n - m$ arms with the lowest empirical averages. Among the arms in $High^t$, let h_*^t be the arm with the lowest lower confidence bound; among the arms in Low^t, let l_*^t be the arm with the highest upper confidence bound (with ties broken arbitrarily):

$$\begin{aligned} h_*^t &\overset{\text{def}}{=} \operatorname*{argmin}_{h \in High^t} \hat{p}_h^t - \beta(u_h^t, t), \text{ and} \\ l_*^t &\overset{\text{def}}{=} \operatorname*{argmin}_{l \in Low^t} \hat{p}_l^t + \beta(u_l^t, t). \end{aligned}$$

Our stopping rule is to terminate iff

$$\left(\hat{p}^t_{l^t_*} + \beta(u^t_{l^t_*}, t)\right) - \left(\hat{p}^t_{h^t_*} - \beta(u^t_{h^t_*}, t)\right) < \epsilon.$$

Figure 5.1 provides an illustration. Together, the sampling strategy $\mathcal{S}$ and our stopping rule (determined by the confidence bound β) give rise to an algorithm for the PAC-m problem. We denote this algorithm LUCB (specified in Algorithm 5.4). Whereas the idea behind our stopping rule is to terminate early on easy problem instances, the stopping rule by itself does not ensure that every run on every bandit instance will terminate after a finite number of rounds. To achieve this latter requirement, we run an outer loop within LUCB to ensure that it will never perform more than $O(\frac{n}{\epsilon^2} \log\left(\frac{m}{\delta}\right))$ rounds of sampling.

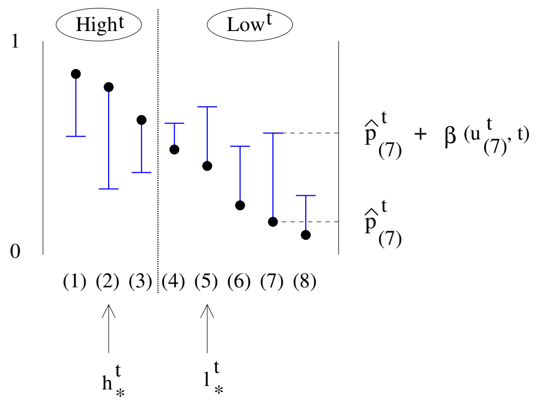


Figure 5.1: Illustration of stopping rule, with $n = 8$ and $m = 3$. In the figure, the arms are sorted in non-increasing order of the *empirical* means during round t. $High^t$ is the set of three arms with the highest empirical means, and Low^t the set of five arms with the lowest empirical means. In the example, arm (2) has the lowest lower confidence bound (LLCB) among arms in $High^t$, and arm (5) has the highest upper confidence bound (HUCB) among the arms in Low^t. The rule is to stop iff HUCB $-$ LLCB $< \epsilon$.

Algorithm 5.4 LUCB(n, m, ϵ, δ, $\mathcal{S}$, β)

```
1: // Initialization.
2: Sample each arm once.
3: // Adaptive monitoring of returns; possible early termination.
4: for $t = n + 1$ to $T_{\text{HALVING}}(n, m, \epsilon, \frac{\delta}{2}) + 1$ do
5:     $h_*^t \leftarrow \text{argmin}_{h \in High^t}\, \hat{p}_h^t - \beta\,(u_h^t, t)$.
6:     $l_*^t \leftarrow \text{argmax}_{l \in Low^t}\, \hat{p}_h^t + \beta\,(u_l^t, t)$.
7:     if $\left(\hat{p}_{l_*^t}^t + \beta\left(u_{l_*^t}^t, t\right)\right) - \left(\hat{p}_{h_*^t}^t - \beta\left(u_{h_*^t}^t, t\right)\right) < \epsilon$ then
8:         Return $High^t$ and terminate.
9:     Sample using sampling strategy $\mathcal{S}$.
10: // Safeguard: worst case solution strategy.
11: Clear the history of pulls. Execute HALVING($n$, $m$, $\epsilon$, $\frac{\delta}{2}$); return its output
    and terminate.
```

Theorem 5.5 shows that if $\beta(u, t)$ is sufficiently large, then LUCB necessarily achieves the PAC correctness guarantee. Theorem 5.6 then formally bounds the worst case sample complexity of the algorithm. Note that these results apply for a large number of choices of β, and for every sampling strategy $\mathcal{S}$. We provide specific choices for β and $\mathcal{S}$ later in this section.

Theorem 5.5 (Correctness)**.** *Let*

1. *$\mathcal{S}$: Set of histories $\rightarrow$ Arms be an arbitrary sampling strategy, and*

2. *$\beta : \{1, 2, 3, \dots\} \times \{1, 2, 3, \dots\} \rightarrow (0, \infty)$ be a function such that*

$$\sum_{t=1}^{\infty} \sum_{u=1}^{t} \exp\left(-2u\beta\,(u, t)^2\right) \leq \frac{\delta}{2n}. \tag{5.6}$$

Then, LUCB*(n, m, ϵ, δ, $\mathcal{S}$, β) is (ϵ, m, δ)-optimal.*

Proof. See page 187. □

Theorem 5.6 (Worst case sample complexity)**.** *For every sampling strategy $\mathcal{S}$: Set of histories $\rightarrow$ Arms and every function $\beta : \{1, 2, 3, \dots\} \times \{1, 2, 3, \dots\} \rightarrow (0, \infty)$,* LUCB*(n, m, ϵ, δ, $\mathcal{S}$, β) has a worst case sample complexity of $O\left(\frac{n}{\epsilon^2} \log\left(\frac{m}{\delta}\right)\right)$.*

Proof. See page 191. □

5.3.2.2 Greedy Sampling Strategy

During round t, the arms h_t^* and l_t^* can be construed as the arms that are most likely to lead to a mistake: naturally it would then be advisable to sample these arms instead of others. Indeed our greedy sampling strategy implements this very intuition. Our sampling strategy is as follows.

$$\underline{\text{During round } t\text{: sample arms } h_*^t \text{ and } l_*^t.}$$

Since we sample two arms every round (from rounds $n + 1$ through $T_{\text{Halving}}(n, m, \epsilon, \frac{\delta}{2})$), note that the sample complexity of our algorithm is at most twice the number of rounds. If we couple our greedy sampling strategy with the stopping rule described in Section 5.3.2.1, how sample-efficient is the resulting algorithm? In Section 5.3.2.3, we analyze the sample complexity of LUCB1, which uses a specific confidence bound β_1. First we show that regardless of the confidence bound β, our choice to sample arms h_t^* and l_t^* achieves the intended effect of sampling "deserving" arms. Below we introduce the infrastructure used for our analysis.

Recall that c is the mid-point of the true means of arms m and $m + 1$. During round t, let us partition the set of arms into three sets: *Above*^{t}, which comprises arms whose lower confidence bounds fall above c; $Below^t$, which comprises arms whose upper confidence bounds fall below c; and $Middle^t$, the remaining set of arms. Thus, c lies between the lower and upper confidence bounds of every arm in $Middle^t$, as visible in the illustration in Figure 5.2.

$$\begin{aligned} Above^t &\stackrel{\text{def}}{=} \{a \in Arms : \hat{p}_a^t - \beta(u_a^t, t) > c\}. \\ Below^t &\stackrel{\text{def}}{=} \{a \in Arms : \hat{p}_a^t + \beta(u_a^t, t) < c\}. \\ Middle^t &\stackrel{\text{def}}{=} Arms \setminus \left(Above^t \cup Below^t\right). \end{aligned}$$

Note that $Above^t$, $Below^t$ and $Middle^t$ are sets we have defined for the purpose of analysis: the algorithm itself does not know the value of c and the contents of these sets. We expect that by and large, arms in Top must stay in $Above^t$ or $Middle^t$, while arms in Bot must stay in $Below^t$ or $Middle^t$. However, let $CROSS_a^t$ denote the event that arm a does not obey such an expectation (and let $CROSS^t$ denote the event that some arm has "crossed").

$$CROSS_a^t \stackrel{\text{def}}{=} \begin{cases} a \in Below^t, \text{ if } a \in Top, \\ a \in Above^t, \text{ if } a \in Bot. \end{cases}$$

$$CROSS^t \stackrel{\text{def}}{=} \exists a \in Arms : CROSS_a^t.$$

Let us now define a "needy" arm as one that is in $Middle^t$, and has a confidence bound that is longer than $\frac{\epsilon}{2}$: let $NEEDY_a^t$ be the event that arm a is needy during round t. As the name suggests, a needy arm is expected to need additional rounds of sampling to subsequently force it into $Above^t$ if it is in Top, and into $Below^t$ if it is in Bot.

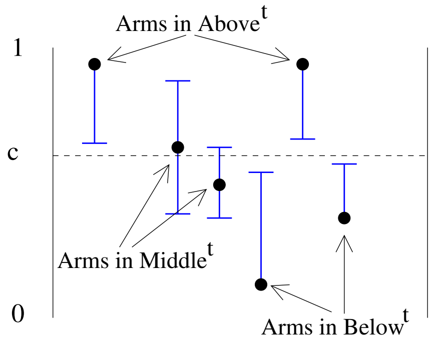


Figure 5.2: Illustration of sets $Above^t$, $Below^t$, and $Middle^t$.

$$NEEDY_a^t \overset{\text{def}}{=} \left(a \in Middle^t\right) \wedge \left(\beta\left(u_a^t, t\right) > \frac{\epsilon}{2}\right).$$

Additionally let $TERMINATE^t$ denote the event that during round t, the stopping rule will lead to termination:

$$TERMINATE^t \overset{\text{def}}{=} \left(\hat{p}_{l_*^t}^t + \beta\left(u_{l_*^t}, t\right)\right) - \left(\hat{p}_{h_*^t}^t - \beta\left(u_{h_*^t}, t\right)\right) < \epsilon.$$

The following lemma shows that if $CROSS^t$ does not occur, and the algorithm does not terminate during round t, then either h_*^t or l_*^t is a needy arm.

Lemma 5.7. $\neg CROSS^t \wedge \neg TERMINATE^t \implies NEEDY_{h_*^t}^t \vee NEEDY_{l_*^t}^t$.

Proof. See page 191. $\square$

Observe that if no arm has crossed and no arm is needy, then the LUCB algorithm must stop. In the remainder of this section, we consider a specific confidence bound, β_1, under which we bound the probability of arms crossing or staying needy for too long. Combining these probability bounds, we arrive at a bound on the expected sample complexity of the algorithm.

5.3.2.3 LUCB1 Algorithm

Algorithm LUCB1 follows a greedy sampling strategy, and uses a stopping rule based on the following confidence bound:

$$\beta_1(u,t) \overset{\text{def}}{=} \sqrt{\frac{1}{2u}\ln\left(\frac{k_1 n t^4}{\delta}\right)},$$

where $k_1 = 2.5$.

First, it is easy to verify that LUCB1 achieves the PAC correctness requirement encapsulated in (5.6):

$$\sum_{t=1}^{\infty}\sum_{u=1}^{t}\exp\left(-2u\beta_1\left(u,t\right)^2\right) = \sum_{t=1}^{\infty}\frac{\delta}{k_1 n t^3} < \frac{\delta}{2n}.$$

The following lemma establishes that the probability that arms cross under LUCB1 is small.

Lemma 5.8. *Under* LUCB1*:*

$$\mathbb{P}\{CROSS^t\} \leq \frac{\delta}{k_1 t^3}.$$

Proof. See page 194. □

How many rounds of sampling does an arm a need before it can stop being needy with high probability? For sufficiently large t, we define $u_1^*(a,t)$ as a sufficient number of samples of arm a such that $\beta(u_1^*(a,t),t)$ is guaranteed to be smaller than $[\Delta_a \vee \frac{\epsilon}{2}]$:

$$u_1^*(a,t) \overset{\text{def}}{=} \left\lceil \frac{1}{2\left[\Delta_a \vee \frac{\epsilon}{2}\right]^2} \ln\left(\frac{k_1 n t^4}{\delta}\right) \right\rceil .$$

The following lemma then shows that the probability that arm a remains needy despite being sampled for more than $4u_1^*(a,t)$ rounds is small.

Lemma 5.9. *Under* LUCB1*:*

$$\mathbb{P}\{\exists a \in Arms : \left(u_a^t > 4u_1^*(a,t)\right) \wedge NEEDY_a^t\} \leq \frac{3\delta H_1^{\frac{\epsilon}{2}}}{4k_1 n t^4}.$$

Proof. See page 195. □

We are now ready to combine the results of lemmas 5.8 and 5.9. Let T be a number no less than $\left\lceil 146 H_1^{\epsilon/2} \ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\right\rceil$. We show that beyond $\left\lceil \frac{T}{2} \right\rceil$ rounds of sampling, the probability that some arm crossed, or some arm remained needy beyond a certain number of samples, is small, but nevertheless, one of these events must occur for the algorithm to last more than T rounds. Thereby we show that the probability of LUCB1 not terminating after T rounds of sampling is small.

Lemma 5.10. *Let* $T_1^* = \left\lceil 146 H_1^{\epsilon/2} \ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\right\rceil$*. For every* $T \geq T_1^*$*, the probability that* LUCB1 *has not terminated after* T *rounds of sampling is at most* $\frac{2\delta}{T^2}$*.*

Proof. See page 197. □

Lemma 5.10 directly leads to a bound on the expected sample complexity, and a related high-probability bound.

Theorem 5.11 (Expected sample complexity)**.** *The expected sample complexity of* LUCB1 *is* $\min\left\{O\left(H_1^{\epsilon/2}\log\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\right), O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)\right\}$.

Corollary 5.12 (High-probability bound)**.** *With probability at least* $1-\delta$*,* LUCB1 *terminates after completing* $O\left(H_1^{\epsilon/2}\log\left(\frac{1}{\delta H_1^{\epsilon/2}}\right)\right)$ *rounds.*

Proof. See page 199. □

Observe that the high-probability bound improves upon the one provided by Even-Dar et al. (2006, see Remark 9). We conjecture that the expected sample complexity of LUCB1 can be further improved through a refinement to the confidence bound. In particular consider

$$\beta_2(u,t) \stackrel{\text{def}}{=} \sqrt{\frac{1}{2u}\ln\left(\frac{k_2 n t^{k_3}\widetilde{\Delta}^2}{\delta}\right)},$$

where $\widetilde{\Delta}$ is the smallest non-negative integer power of $\frac{1}{2}$ such that $\beta_2(u,t) < \widetilde{\Delta}$, and k_2 and k_3 are suitably large constants. Indeed $\beta_2(u,t)$ is well-defined; the design of the $\widetilde{\Delta}$ term within the log is such that it will quickly reach the order of the true Δ_a for each arm a. As a result, arms with higher Δ's will get sampled relatively fewer times than they do under LUCB1. The idea of using $\widetilde{\Delta}$ as a guess of each arm's true Δ is borrowed from Auer and Ortner (2010), who employ a similar scheme to reduce the cumulative regret of the UCB algorithm.

Let LUCB2 be an algorithm that follows a greedy sampling strategy with respect to β_2, and also uses the corresponding stopping rule. We make the following conjecture about LUCB2.

Conjecture 5.13. LUCB2 *is* (ϵ, m, δ)*-optimal, with an expected sample complexity that is:*

$$\min\left\{O\left(\sum_{a\in Arms}\frac{1}{\left[\Delta_a \vee \frac{\epsilon}{2}\right]^2}\log\left(\frac{H_1^{\epsilon/2}\left[\Delta_a \vee \frac{\epsilon}{2}\right]}{\delta}\right)\right), O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)\right\}.$$

With probability at least $1-\delta$*,* LUCB2 *terminates after completing*

$$O\left(\sum_{a\in Arms}\frac{1}{\left[\Delta_a \vee \frac{\epsilon}{2}\right]^2}\log\left(\frac{\left[\Delta_a \vee \frac{\epsilon}{2}\right]}{\delta H_1^{\epsilon/2}}\right)\right)$$

rounds.

We leave the proof of this conjecture for future work. In the next section, we consider subset selection under the simple regret setting.

5.4 Simple Regret Setting

Recall that the formulation of SR-m, the problem of simple regret minimization, is complementary to that of PAC-m. Under PAC-m, our aim is to minimize the number of samples needed to achieve a desired probability of correct selection. Under SR-m, the number of sample available, T, is fixed beforehand: the problem is to use these T samples most judiciously in order to minimize the simple regret, given by $\mathbb{E}\left[\sum_{i\in Top} p_i - \sum_{a\in Selected} p_a\right]$, where the set *Selected* is the m-sized subset returned after T rounds.

Audibert et al. (2010) consider SR-1, where one arm needs to be selected after T rounds. They provide a lower bound on the simple regret in this setting, and highlight the difficulty of devising an algorithm with a matching upper bound unless the algorithm is given the hardness of the problem, H_1, as an input.[6] Indeed if H_1 is known, a simple modification to LUCB1, our algorithm for PAC-m, can yield an effective algorithm for SR-m. We could run LUCB1 with parameters $\epsilon = 0$ and $\delta = H_1 \exp\left(-\frac{kT}{H_1}\right)$, for suitably small k, and (1) if LUCB1 terminates by T rounds, return its solution, and (2) if it does not terminate by T rounds, return a random m-sized subset. Such an algorithm would yield a simple regret that is $O\left(H_1 \exp\left(-\frac{T}{H_1}\right)\right)$.

In general we may not assume knowledge of H_1. Below we present a generalization of the "successive rejects" algorithm of Audibert et al. (2010); we denote our algorithm Elimination. This algorithm comprises $n - m$ phases; at the end of each phase, one arm with the lowest empirical mean is eliminated. The m arms remaining at the end of the $n - m$ phases are returned as the solution. Key to minimizing the simple regret of this scheme is properly tuning the sequence $(b_l), \forall l = \{1, 2, n - m\}$: b_l is the number of times that an arm eliminated at the end of phase l has been sampled. Algorithm 5.5 provides a full specification of Elimination, and Theorem 5.14 provides a bound on its simple regret.

Theorem 5.14. Elimination(n, m, T) *has a simple regret that is at most*

$$\left(\sum_{i=1}^{m} \Delta_{i,n-i}\right) \min\left\{1, \frac{(n-m)(n-m+1)}{2} \sum_{i=1}^{m} \exp\left(-\frac{T-n}{2\left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right) H_2(i)}\right)\right\}.$$

Proof. See page 200. □

Consider the selection subroutine within PS methods such as CMA-ES and CEM. Given n policies and a total of T evaluation episodes, how might

[6]In the SR-m and CR-m settings, we assume that H_1 is well-defined; that is, $p_m > p_{m+1}$.

Algorithm 5.5 ELIMINATION(n, m, T)

```
1:  $R_1 \leftarrow Arms$.
2:  $b_0 \leftarrow 0$.
3:  for $l = 1$ to $n - m$ do
4:      $b_l \leftarrow \left\lceil \frac{T-n}{\left(\frac{m}{2}+\sum_{k=2}^{n-m+1}\frac{1}{k}\right)(n-m+2-l)} \right\rceil$.
5:      for all $a \in R_l$ do
6:          Sample arm $a$ for $(b_l - b_{l-1})$ additional rounds.
7:      In the last phase (that is, when $l = n - m$), if the total number of
        samples is less than $T$, arbitrarily sample arms for additional rounds
        until the total number of samples is $T$.
8:      Let $\hat{p}_a$ be the average reward of arm a, $\forall a \in Arms$.
9:      Let $a' = \text{argmin}_{a \in R_l} \hat{p}_a$ (with ties broken arbitrarily).
10:     $R_{l+1} \leftarrow R_l \setminus \{a'\}$.
11: $Selected \overset{\text{def}}{=} R_{n-m+1}$. Return $Selected$.
```

we allocate episodes to the policies in order to select a relatively high-fitness m-sized subset of policies? Under our implementations of these methods in Chapter 3, we allocated each policy the same number of allocations, $\frac{T}{n}$. By contrast, the ELIMINATION algorithm provides us the means to achieve better simple regret after T evaluations. In experiments in this chapter, we implement the ELIMINATION algorithm as the subset selection procedure within CMA-ES and CEM; we refer to the resulting "optimized" versions of these PS methods as OPT-CMA-ES and OPT-CEM, respectively.

Figure 5.3 compares the four methods on benchmark problem instances considered in Chapter 3. Under each of problem instances I_1, I_2, and I_3, we find that the optimized versions of CMA-ES and CEM outperform their counterparts after 50,000 episodes of training (p-value < 0.0005). Recall that these

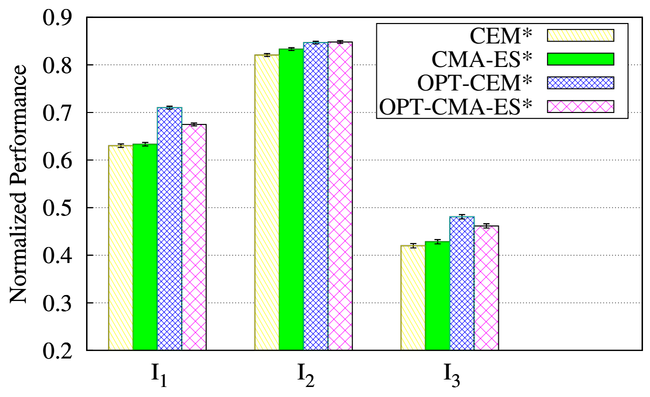


Figure 5.3: Performance of OPT-CMA-ES* and OPT-CEM* on problem instances I_1, I_2, and I_3 (see Table 3.4 on page 68).

methods are suffixed "$*$" to indicate that their method-specific parameters (number of generations, and total trials per generation) have themselves been tuned for each problem instance. We do not observe any marked differences between the optimized and non-optimized versions of the methods in terms of the settings found by our search procedure for their method-specific parameters. We posit that for similar population sizes and total trials in a generation, the better quality of the subsets found by the optimized versions of the methods leads to their improved performance. A closer investigation is necessary to obtain a more fine-grained understanding of the effect of better subset selection on the dynamics of PS methods.

In our next section, we consider the CR setting, wherein unlike PAC and SR, exploration must be balanced with exploitation in order to minimize cumulative (on-line) regret.

5.5 Cumulative Regret Setting

In this section, we present our algorithm for CR-m, wherein a subset of m arms is selected to be sampled at every stage. We denote our algorithm UCBS. Under UCBS, we draw upper confidence bounds on each arm based on the number of times it has been sampled, and the total number of rounds conducted so far. At every stage, the subset selected for sampling comprises m arms with the highest upper confidence bounds. Algorithm 5.6 provides a description of UCBS, and Theorem 5.15 establishes a bound on the algorithm's cumulative regret. The algorithm and analysis are both generalizations of those provided by Auer et al. (2002a) for CR-1.

Algorithm 5.6 UCBS(n, m)

```
1: //Initialization
2: For the first $\lceil \frac{n}{m} \rceil$ rounds, select subsets of arms such that every arm gets
   sampled at least once.
3: for $t = \lceil \frac{n}{m} \rceil + 1$ onwards do
4:     $\forall a \in Arms$, let $u_a^t$ be the number of times arm $a$ has been sampled,
       and let $\hat{p_a^t}$ be the empirical mean of the rewards from that arm. Define
       $b_a^t \overset{\text{def}}{=} \hat{p_a^t} + \sqrt{\frac{1}{2u_a^t} \ln(t^4)}$.
5:     Select $A^t$, an $m$-sized subset of arms such that
       $\forall a_1 \in A^t \; \forall a_2 \in Arms \setminus A^t : \hat{p_{a_1}^t} + \sqrt{\frac{1}{2u_{a_1}^t} \ln(t^4)} \geq \hat{p_{a_2}^t} + \sqrt{\frac{1}{2u_{a_2}^t} \ln(t^4)}$.
```

Theorem 5.15. *The cumulative regret of* UCBS*(n, m) after T rounds of sampling is at most*

$$\min\left\{\left(\sum_{i=1}^{m}\Delta_{i,n-i}\right)T, \sum_{i=1}^{m}\sum_{j=m+1}^{n}\frac{8\ln(T)}{\Delta_{i,j}} + \frac{1+\pi^2}{3}\sum_{i=1}^{m}\sum_{j=m+1}^{n}\Delta_{i,j}\right\}.$$

Proof. See page 203. □

5.6 Proofs

This section provides proofs for the various lemmas and theorems presented in the preceding three chapters. The reader may only peruse selected proofs, or skip this section altogether, without missing any important details in the chapter.

Theorem 5.1 (page 164)

Statement. Direct(n, m, ϵ, δ) is (ϵ, m, δ)-optimal with sample complexity $O\left(\frac{n}{\epsilon^2}\log\left(\frac{n}{\delta}\right)\right)$.

Proof. Since each arm is pulled exactly $\left\lceil \frac{2}{\epsilon^2}\ln\left(\frac{n}{\delta}\right)\right\rceil$ times (line 2), every run of Direct terminates, and the sample complexity of Direct is $O\left(\frac{n}{\epsilon^2}\log\left(\frac{n}{\delta}\right)\right)$. To show that Direct achieves the PAC guarantee, recall that *Bad* is the set of arms that are not (ϵ, m)-optimal. From (5.1) and (5.2), we can relate *Top* and *Bad* as follows:

$$\forall i \in Top\ \forall b \in Bad : p_i - p_b > \epsilon. \tag{5.7}$$

Since $|S| = m$, an arm b in *Bad* can occur in S only if there is some arm i in *Top* such that $\hat{p}_i \leq \hat{p}_b$ (line 4). In turn, (5.7) implies that the latter event can only occur if $\hat{p}_i \leq p_i - \frac{\epsilon}{2}$ or $\hat{p}_b \geq p_b + \frac{\epsilon}{2}$. Switching to probabilities, applying the union bound and Hoeffding's inequality, we get:

$$\begin{aligned} & \mathbb{P}\left\{\exists b \in Bad : b \in S\right\} \\ \leq\ & \mathbb{P}\left\{\exists i \in Top : \left(\hat{p}_i \leq p_i - \frac{\epsilon}{2}\right)\right\} + \mathbb{P}\left\{\exists b \in Bad : \left(\hat{p}_b \geq p_b + \frac{\epsilon}{2}\right)\right\} \\ \leq\ & \sum_{i \in Top} \mathbb{P}\left\{\hat{p}_i \leq p_i - \frac{\epsilon}{2}\right\} + \sum_{b \in Bad} \mathbb{P}\left\{\hat{p}_b \geq p_b + \frac{\epsilon}{2}\right\} \end{aligned}$$

$$\leq |Top| \exp\left(-\frac{\epsilon^2}{2}\left\lceil\frac{2}{\epsilon^2}\ln\left(\frac{n}{\delta}\right)\right\rceil\right) + |Bad| \exp\left(-\frac{\epsilon^2}{2}\left\lceil\frac{2}{\epsilon^2}\ln\left(\frac{n}{\delta}\right)\right\rceil\right)$$
$$\leq (|Top| + |Bad|)\left(\frac{\delta}{n}\right) \leq \delta.$$

□

Theorem 5.2 (page 165)

Statement. Incremental(n, m, ϵ, δ) is (ϵ, m, δ)-optimal with sample complexity $O\left(\frac{mn}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$.

Proof. Incremental makes a call to the Median-Elimination algorithm during each phase l, in order to select a single $(\epsilon, 1)$-optimal arm in R_l with probability at least $\frac{\delta}{m}$. Since $|R_l| \leq n$, each such call performs at most $O\left(\frac{n}{\epsilon^2}\log\left(\frac{1}{\left(\frac{\delta}{m}\right)}\right)\right)$ pulls (Even-Dar et al., 2006, see Lemma 12). Thus, after m phases, Incremental will terminate with a sample complexity that is $O\left(\frac{mn}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$.

Since $|R_l| = n - l + 1$, and $l \leq m$, R_l necessarily contains an arm a in Top. Among the true means of the arms in R_l, let p^* be the highest. It follows from (5.1) and (5.2) that for any arm a' that is $(\epsilon, 1)$-optimal with respect to R_l, $p_{a'} \geq p^* - \epsilon \geq p_a - \epsilon \geq p_m - \epsilon$: that is, a' is (ϵ, m)-optimal with respect to $Arms$. On phase l, since Median-Elimination (line 3) returns an arm that is *not* $(\epsilon, 1)$-optimal in R_l with probability at most $\frac{\delta}{m}$, the probability that Incremental selects an arm that is not (ϵ, m)-optimal is at most δ. □

Theorem 5.3 (page 166)

Statement. Halving(n, m, ϵ, δ) is (ϵ, m, δ)-optimal with sample complexity $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$.

Proof. In Algorithm 5.3 (lines 3–5), we can see that the number of pulls in phase l is $|R_l| \left\lceil \frac{2}{\epsilon_l^2} \ln\left(\frac{3m}{\delta_l}\right) \right\rceil$. Thus, the total number of pulls across all phases is $\sum_{l=1}^{\lceil \log(\frac{n}{m}) \rceil} |R_l| \left\lceil \frac{2}{\epsilon_l^2} \ln\left(\frac{3m}{\delta_l}\right) \right\rceil \overset{\text{def}}{=} T_{\text{Halving}}(n, m, \epsilon, \delta)$. Observe that this sum, $T_{\text{Halving}}(n, m, \epsilon, \delta)$, can be computed exactly for given n, m, ϵ, and δ. Indeed we will use $T_{\text{Halving}}(n, m, \epsilon, \delta)$ in designing the termination condition for our VSC algorithms in Section 5.3.2. Slightly modifying the calculation for the $m = 1$ case (Even-Dar et al., 2006, see Lemma 12), we obtain that $T_{\text{Halving}}(n, m, \epsilon, \delta)$ is $O(\frac{n}{\epsilon^2} \log(\frac{m}{\delta}))$.

Our proof that Halving achieves the PAC requirement also closely follows that of Even-Dar et al. (2006) for the case of $m = 1$. Let us sort the arms in R_l in non-increasing order of their *true* means, breaking ties using the same deterministic rule applied in (5.1). Let a_i^l be the i^{th} arm in the sorted list and let p_i^l be its true mean. We say a "mistake" is made in phase l iff $p_m^l - p_m^{l+1} > \epsilon_l$. Note that (1) $p_m^1 = p_m$, (2) $\sum_{l=1}^{\lceil \log(\frac{n}{m}) \rceil} \epsilon_l < \epsilon$, and (3) $\sum_{l=1}^{\lceil \log(\frac{n}{m}) \rceil} \delta_l < \delta$. In effect, it suffices to show that the probability of making a mistake in phase l is at most δ_l: this would establish that $\mathbb{P}\left\{ p_m - p_m^{\lceil \log(\frac{n}{m}) \rceil + 1} > \epsilon \right\} < \delta$, or in other words, that Halving is (ϵ, m, δ)-optimal. Our proof strategy is to show that for a mistake to occur on phase l, at least one of two events, E_1 and E_2, must occur; however, $\mathbb{P}\{E_1\} + \mathbb{P}\{E_2 | \neg E_1\} \leq \delta_l$.

Let $Top^l = \{a_i^l, i \in 1, 2, \ldots, m\}$: Top^l contains the m arms from R_l with the highest true means (after deterministic tie-breaking as in (5.1)). E_1 denotes the event $\exists a \in Top^l : \hat{p}_a \leq p_a - \frac{\epsilon_l}{2}$. By applying Hoeffding's inequality and the union bound, we obtain:

$$\mathbb{P}\{E_1\} \leq m \, \exp\left(-\frac{\epsilon_l^2}{2} \left\lceil \frac{2}{\epsilon_l^2} \ln\left(\frac{3m}{\delta_l}\right) \right\rceil \right) \leq \frac{\delta_l}{3}. \tag{5.8}$$

Let $Bad^l = \{b \in R_l, p_b < p^l_m - \epsilon_l\}$; that is, Bad^l is the set of arms that are not (ϵ_l, m)-optimal in R_l. We call an arm b in Bad^l "deceptive" if its empirical average equals or exceeds that of some arm in Top^l. If E_1 does not occur, note that b can be deceptive only if $\hat{p}_b \geq p_b + \frac{\epsilon_l}{2}$; a similar application of Hoeffding's inequality shows that the probability of the latter event is at most $\frac{\delta_l}{3m}$. Let the number of deceptive arms in Bad^l be $\#deceptive$; we obtain that $\mathbb{E}[\#deceptive|\neg E_1]$ is at most $|Bad^l|\frac{\delta_l}{3m}$.

E_2 denotes the event that $\#deceptive \geq |R_{l+1}| - m + 1$. A moment's reflection informs us that if E_1 does not occur, a mistake can be made in phase l only if E_2 occurs. Markov's inequality establishes that

$$\begin{aligned}
\mathbb{P}\{E_2|\neg E_1\} &= \mathbb{P}\{\#deceptive \geq (|R_{l+1}| - m + 1)|\neg E_1\} \\
&\leq \frac{\mathbb{E}[\#deceptive|\neg E_1]}{|R_{l+1}| - m + 1} \\
&\leq \frac{|Bad^l|}{|R_{l+1}| - m + 1}\left(\frac{\delta_l}{3m}\right) \\
&\leq \frac{|R_l| - m}{|R_{l+1}| - m + 1}\left(\frac{\delta_l}{3m}\right) \\
&\leq \frac{2}{3}\delta_l. \qquad (5.9)
\end{aligned}$$

The inequality in the last step follows from the observation that $|R_l| \leq 2|R_{l+1}|$ (lines 7 and 8). Together, (5.8) and (5.9) complete our proof. □

Theorem 5.5 (page 172)

Statement. Let

1. $\mathbb{S}$: Set of histories $\rightarrow$ *Arms* be an arbitrary sampling strategy, and

2. $\beta : \{1, 2, 3, \ldots\} \times \{1, 2, 3, \ldots\} \to (0, \infty)$ be a function such that

$$\sum_{t=1}^{\infty} \sum_{u=1}^{t} \exp\left(-2u\beta\left(u, t\right)^2\right) \leq \frac{\delta}{2n}.$$

Then, LUCB(n, m, ϵ, δ, $\mathcal{S}$, β) is (ϵ, m, δ)-optimal.

Proof. LUCB necessarily terminates in one of two ways. In the first case, termination during round $n + 1 \leq t \leq T_{\text{HALVING}}\left(n, m, \epsilon, \frac{\delta}{2}\right) + 1$, if the upper confidence bound of l_*^t does not exceed the lower confidence bound of h_*^t by more than ϵ (line 7). In the second case, termination occurs at the end of a call to the HALVING algorithm (line 11). If the second case occurs, the mistake probability of LUCB is equal to the mistake probability of HALVING(n, m, ϵ, $\frac{\delta}{2}$), which is at most $\frac{\delta}{2}$ (see Theorem 5.3). It suffices to show that in the first case, the probability that a non-(ϵ, m)-optimal arm is returned is at most $\frac{\delta}{2}$. The remainder of this proof considers this first case: that is, when the algorithm terminates during round $n + 1 \leq t \leq T_{\text{HALVING}}\left(n, m, \epsilon, \frac{\delta}{2}\right) + 1$.

Our proof involves a standard argument using confidence bounds. We show that with high probability, the true means of arms stay within their respective confidence bounds; in other words, that the arms remain "well-behaved". We then show that if all the arms are well-behaved, then a mistake cannot occur. Specifically let WB_a^t denote the event that arm a is well-behaved during round t:

$$WB_a^t \stackrel{\text{def}}{=} \begin{cases} \hat{p}_a^t \geq p_a - \beta\left(u_a^t, t\right), & \text{if } a \in Top, \text{ and} \\ \hat{p}_a^t \leq p_a + \beta\left(u_a^t, t\right), & \text{if } a \in Bot. \end{cases}$$

Note that u_a^t (the number of rounds for which arm a has been sampled before round t) is a random variable. The notation used in defining WB_a^t

above (and also adopted in subsequent derivations) is a convenient shorthand for quantifying the union of events corresponding to every value of u_a^t. Regardless of the sampling strategy used, note that u_a^t has to be between 1 and $t-1$. Applying the union bound and Hoeffding's inequality, we bound the probability that arm a is *not* well-behaved during round t:

$$\mathbb{P}\{\neg WB_a^t\} \leq \sum_{u=1}^{t-1} \exp\left(-2u\beta\left(u,t\right)^2\right). \tag{5.10}$$

This union bound is the key step allowing us to keep our correctness proof independent of the sampling strategy. The bound allows us to think in terms of "the state of arm a during round t," without worrying about how many times the arm has been sampled in that state. Indeed our subsequent proofs define other events whose probabilities are bounded in a similar manner.

Let $MISTAKE^t$ denote the event that LUCB terminates during rounds t and returns a non-(ϵ, m)-optimal arm. In the case we are considering, the mistake event corresponds to $\cup_{t=n+1}^{T_{\text{HALVING}}+1}(MISTAKE^t)$.

$$\begin{aligned} MISTAKE^t \overset{\text{def}}{=} & \text{ (Algorithm reaches round t)} \\ & \wedge \left(\left(\hat{p}_{l_*^t}^t + \beta\left(u_{l_*^t}^t, t\right)\right) - \left(\hat{p}_{h_*^t}^t - \beta\left(u_{h_*^t}^t, t\right)\right) < \epsilon\right) \\ & \wedge \left(\exists b \in Bad : b \in High^t\right). \end{aligned}$$

High^t is of size m: if it contains an arm $b \in Bad$, then there must be some arm $i \in Top$ that falls into *Low*^{t}. Further, the upper confidence bound of i cannot exceed the lower confidence bound of b by ϵ or more. We get (bounding ϵ by applying (5.1)):

$$\begin{aligned}
& MISTAKE^t \\
\Longrightarrow \quad & \exists i \in Top\ \exists b \in Bad : \left(\hat{p}_i^t + \beta\left(u_i^t, t\right)\right) - \left(\hat{p}_b^t - \beta\left(u_b^t, t\right)\right) < \epsilon \\
\Longrightarrow \quad & \exists i \in Top\ \exists b \in Bad : \left(\hat{p}_i^t + \beta\left(u_i^t, t\right)\right) - \left(\hat{p}_b^t - \beta\left(u_b^t, t\right)\right) < (p_i - p_b) \\
\Longrightarrow \quad & \left(\exists i \in Top : \hat{p}_i^t < p_i - \beta(u_i^t, t)\right) \vee \left(\exists b \in Bot : \hat{p}_b^t > p_b + \beta(u_b^t, t)\right) \\
\Longrightarrow \quad & \exists a \in Arms : \neg WB_a^t. \qquad (5.11)
\end{aligned}$$

Thus, a mistake occurs during round t only if some arm is not well-behaved during round t. For a mistake never to occur, there must be no round t and no arm a such that a is not well-behaved during round t. The second important step in this proof is to invert the order between rounds and arms, as below, where we apply (5.11) to get:

$$\begin{aligned}
\cup_{t=1}^{T_{\text{Halving}}} \left(MISTAKE^t\right) \quad & \Longrightarrow \quad \cup_{t=1}^{T_{\text{Halving}}} \left(\exists a \in Arms : \neg WB_a^t\right) \\
& \Longleftrightarrow \quad \cup_{t=1}^{T_{\text{Halving}}} \cup_{a \in Arms} \left(\neg WB_a^t\right) \\
& \Longleftrightarrow \quad \cup_{a \in Arms} \cup_{t=1}^{T_{\text{Halving}}} \left(\neg WB_a^t\right). \qquad (5.12)
\end{aligned}$$

We complete our proof by switching to probabilities and invoking (5.12), (5.10), and (5.6):

$$\begin{aligned}
\mathbb{P}\{\cup_{t=1}^{T_{\text{Halving}}} \left(MISTAKE^t\right)\} \quad & \leq \quad \mathbb{P}\{\cup_{a \in Arms} \cup_{t=1}^{T_{\text{Halving}}} \left(\neg WB_a^t\right)\} \\
& \leq \quad \sum_{a \in Arms} \sum_{t=1}^{T_{\text{Halving}}} \mathbb{P}\{\neg WB_a^t\} \\
& \leq \quad \sum_{a \in Arms} \sum_{t=1}^{T_{\text{Halving}}} \sum_{u=1}^{t} \exp\left(-2u\beta\left(u, t\right)^2\right)
\end{aligned}$$

$$\begin{aligned} &\leq n \sum_{t=1}^{\infty} \sum_{u=1}^{t} \exp\left(-2u\beta\left(u,t\right)^2\right) \\ &\leq \frac{\delta}{2}. \end{aligned}$$

□

Theorem 5.6 (page 173)

Statement. For every sampling strategy $\mathcal{S}$: Set of histories $\rightarrow$ $Arms$ and every function $\beta : \{1, 2, 3, \dots\} \times \{1, 2, 3, \dots\} \rightarrow (0, \infty)$, LUCB($n$, m, ϵ, δ, $\mathcal{S}$, β) has a worst case sample complexity of $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$.

Proof. Regardless of the sampling strategy $\mathcal{S}$ and the confidence bound β, LUCB(n, m, ϵ, δ, $\mathcal{S}$, β) samples for at most $1 + 2T_{\text{Halving}}\left(n, m, \epsilon, \frac{\delta}{2}\right)$ rounds. Recall from Theorem 5.3 that $T_{\text{Halving}}\left(n, m, \epsilon, \delta\right)$ is $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$. It follows that the worst case sample complexity of LUCB(n, m, ϵ, δ, $\mathcal{S}$, β) is also $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$. □

Lemma 5.7 (page 175)

Statement. $\neg CROSS^t \wedge \neg TERMINATE^t \implies NEEDY^t_{h^t_*} \vee NEEDY^t_{l^t_*}$.

Proof. In our proof below, we reduce notational clutter by dropping the suffix t in our variables. Additionally we use the shorthand $\beta[a]$ for $\beta(u^t_a, t)$. To prove the lemma, we prove the following statements.

$$\neg CROSS \wedge \neg TERMINATE \implies (h_* \in Middle) \vee (l_* \in Middle)\,. \quad (5.13)$$

$$\neg TERMINATE \wedge (h_* \in Middle) \wedge (l_* \notin Middle) \implies \beta[h_*] > \frac{\epsilon}{2}. \quad (5.14)$$

$$\neg TERMINATE \wedge (h_* \notin Middle) \wedge (l_* \in Middle) \implies \beta[l_*] > \frac{\epsilon}{2}. \quad (5.15)$$

$$\begin{aligned} &\neg TERMINATE \wedge (h_* \in Middle) \wedge (l_* \in Middle) \\ &\qquad \implies \left(\beta[h_*] > \frac{\epsilon}{2}\right) \vee \left(\beta[l_*] > \frac{\epsilon}{2}\right). \end{aligned} \quad (5.16)$$

We prove (5.13) by proving

$$\neg CROSS \implies (h_* \in Middle) \vee (l_* \in Middle) \vee TERMINATE.$$

If neither of h_* and l_* is in $Middle$, then these arms have to be in *Above* or *Below*. Thus, the contrapositive of the statement above can be written as a disjunction of four mutually exclusive cases. Below we consider each of these cases. Recall that (1) h_* has the lowest lower confidence bound among arms in $High$, (2) l_* has the highest upper confidence bound among arms in *Low*, and (3) $\hat{p}_{h_*} \geq \hat{p}_{l_*}$.

$$\begin{array}{ll} (\textbf{Case 1}) & (h_* \in Above) \wedge (l_* \in Above) \wedge \neg TERMINATE \\ \implies & (h_* \in Above) \wedge (l_* \in Above) \\ \implies & (\forall h \in High : h \in Above) \wedge (l_* \in Above) \end{array}$$

$$\begin{aligned}
&\Longrightarrow && |\{a \in Arms : a \in Above\}| \geq m+1 \\
&\Longrightarrow && \exists j \in Bot : j \in Above \\
&\Longleftrightarrow && \exists j \in Bot : CROSS_j \\
&\Longrightarrow && CROSS.
\end{aligned}$$

$$\begin{aligned}
&(\textbf{Case 2}) && (h_* \in Above) \wedge (l_* \in Below) \wedge \neg TERMINATE \\
&\Longrightarrow && (\hat{p}_{h_*} - \beta[h_*] > c) \wedge (\hat{p}_{l_*} + \beta[l_*] < c) \\
& && \wedge (\hat{p}_{l_*} + \beta[l_*] - \hat{p}_{h_*} + \beta[h_*] > \epsilon) \\
&\Longrightarrow && (\hat{p}_{l_*} + \beta[l_*] - \hat{p}_{h_*} + \beta[h_*] < 0) \wedge (\hat{p}_{l_*} + \beta[l_*] - \hat{p}_{h_*} + \beta[h_*] > \epsilon) \\
&\Longleftrightarrow && \phi.
\end{aligned}$$

$$\begin{aligned}
&(\textbf{Case 3}) && (h_* \in Below) \wedge (l_* \in Above) \wedge \neg TERMINATE \\
&\Longrightarrow && (h_* \in Below) \wedge (l_* \in Above) \\
&\Longrightarrow && (\hat{p}_{h_*} + \beta[h_*] < c) \wedge (\hat{p}_{l_*} - \beta[l_*] > c) \\
&\Longrightarrow && \hat{p}_{h_*} < \hat{p}_{l_*} \\
&\Longleftrightarrow && \phi.
\end{aligned}$$

$$\begin{aligned}
&(\textbf{Case 4}) && (h_* \in Below) \wedge (l_* \in Below) \wedge \neg TERMINATE \\
&\Longrightarrow && CROSS. \ \{\text{Similar to Case 1.}\}
\end{aligned}$$

Similarly, we prove (5.14) by proving two disjoint cases.

$$(\textbf{Case 1}) \quad \neg TERMINATE \wedge (h_* \in Middle) \wedge (l_* \in Above)$$

$$
\begin{aligned}
\Longrightarrow \quad & (\hat{p}_{l_*} + \beta[l_*] - \hat{p}_{h_*} + \beta[h_*] > \epsilon) \wedge (\hat{p}_{h_*} - \beta[h_*] < c) \\
& \wedge (\hat{p}_{l_*} - \beta[l_*] > c) \\
\Longrightarrow \quad & (\hat{p}_{h_*} - \beta[h_*] < c) \wedge (\hat{p}_{h_*} + \beta[h_*] > c + \epsilon) \\
\Longrightarrow \quad & \beta[h_*] > \frac{\epsilon}{2}.
\end{aligned}
$$

$$
\begin{aligned}
(\textbf{Case 2}) \quad & \neg TERMINATE \wedge (h_* \in Middle) \wedge (l_* \in Below) \\
\Longrightarrow \quad & (\hat{p}_{l_*} + \beta[l_*] - \hat{p}_{h_*} + \beta[h_*] > \epsilon) \wedge (\hat{p}_{h_*} + \beta[h_*] > c) \\
& \wedge (\hat{p}_{l_*} + \beta[l_*] < c) \\
\Longrightarrow \quad & (\hat{p}_{h_*} + \beta[h_*] > c) \wedge (\hat{p}_{h_*} - \beta[h_*] < c - \epsilon) \\
\Longrightarrow \quad & \beta[h_*] > \frac{\epsilon}{2}.
\end{aligned}
$$

(5.15) can be proven similarly. To complete our proof of the lemma, we prove (5.16):

$$
\begin{aligned}
& \neg TERMINATE \wedge (h_* \in Middle) \wedge (l_* \in Middle) \\
\Longrightarrow \quad & \neg TERMINATE \\
\Longrightarrow \quad & \hat{p}_{l_*} + \beta[l_*] - \hat{p}_{h_*} + \beta[h_*] > \epsilon \\
\Longrightarrow \quad & \beta[h_*] + \beta[l_*] > \epsilon. \\
\Longrightarrow \quad & \left(\beta[h_*] > \frac{\epsilon}{2}\right) \vee \left(\beta[l_*] > \frac{\epsilon}{2}\right).
\end{aligned}
$$

□

Lemma 5.8 (page 176)

Statement. Under LUCB1:

$$\mathbb{P}\{CROSS^t\} \leq \frac{\delta}{k_1 t^3}.$$

Proof. For every arm i in Top, we get:

$$\begin{aligned}
\mathbb{P}\{CROSS_i^t\} &= \mathbb{P}\left\{\hat{p}_i^t + \beta_1\left(u_i^t, t\right) < c\right\} \\
&\leq \sum_{u=1}^{t} \exp\left(-2u\left(p - c + \beta_1\right)^2\right) \\
&\leq \sum_{u=1}^{t} \exp\left(-2u\beta_1^2\right) \\
&= \sum_{u=1}^{t} \frac{\delta}{k_1 n t^4} \\
&= \frac{\delta}{k_1 n t^3}.
\end{aligned}$$

The same bound applies for every arm j in *Bot*. In aggregate, we get:

$$\mathbb{P}\{CROSS^t\} \leq \sum_{a \in Arms} \mathbb{P}\{CROSS_a^t\} \leq n\frac{\delta}{k_1 n t^3} = \frac{\delta}{k_1 t^3}.$$

□

Lemma 5.9 (page 177)

Statement. Under LUCB1:

$$\mathbb{P}\{\exists a \in Arms : \left(u_a^t > 4u_1^*(a,t)\right) \wedge NEEDY_a^t\} \leq \frac{3\delta H_1^{\frac{\epsilon}{2}}}{4k_1 n t^4}.$$

Proof. Consider an arm a in *Arms*. If $\Delta_a \leq \frac{\epsilon}{2}$, we obtain:

$$\begin{aligned}
&\mathbb{P}\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge NEEDY_a^t\} \\
= \;& \mathbb{P}\left\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge \left(a \in Middle^t\right) \wedge \left(\beta\left(u_a^t, t\right) > \frac{\epsilon}{2}\right)\right\}
\end{aligned}$$

$$
\begin{aligned}
&\leq \mathbb{P}\left\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge \left(\sqrt{\frac{1}{2u_a^t}\ln\left(\frac{k_1 n t^4}{\delta}\right)} > \frac{\epsilon}{2}\right)\right\} \\
&\leq \sum_{u=4u_1^*(a,t)+1}^{\infty} \mathbb{P}\left\{\sqrt{\frac{1}{8u_1^*(a,t)}\ln\left(\frac{k_1 n t^4}{\delta}\right)} > \frac{\epsilon}{2}\right\} \\
&\leq \sum_{u=4u_1^*(a,t)+1}^{\infty} \mathbb{P}\left\{\frac{\epsilon}{4} > \frac{\epsilon}{2}\right\} = 0. \qquad (5.17)
\end{aligned}
$$

Now, consider the case that $\Delta_a > \frac{\epsilon}{2}$. Without loss of generality, we may assume that $a \in Top$. Then, we have:

$$
\begin{aligned}
&\mathbb{P}\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge NEEDY_a^t\} \\
&= \mathbb{P}\left\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge \left(a \in Middle^t\right) \wedge \left(\beta\left(u_a^t, t\right) > \frac{\epsilon}{2}\right)\right\} \\
&\leq \mathbb{P}\left\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge \left(a \in Middle^t\right)\right\} \\
&\leq \mathbb{P}\left\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge \left(\hat{p}_a^t - \beta_a\left(u_a^t, t\right) < c\right)\right\} \\
&\leq \sum_{u=4u_1^*(a,t)+1}^{\infty} \exp\left(-2u\left(p_a - c - \beta_1\left(u_a^t, t\right)\right)^2\right).
\end{aligned}
$$

Substituting for β_1, and using (5.4), we get

$$
\begin{aligned}
&\mathbb{P}\{\left(u_a^t > 4u_1^*(a,t)\right) \wedge NEEDY_a^t\} \\
&\leq \sum_{u=4u_1^*(a,t)+1}^{\infty} \exp\left(-2u\left(\frac{\Delta_a}{2} - \sqrt{\frac{1}{2u}\ln\left(\frac{k_1 n t^4}{\delta}\right)}\right)^2\right) \\
&= \sum_{u=4u_1^*(a,t)+1}^{\infty} \exp\left(-2\Delta_a^2\left(\sqrt{u} - \sqrt{\frac{1}{2\Delta_a^2}\ln\left(\frac{k_1 n t^4}{\delta}\right)}\right)^2\right) \\
&\leq \sum_{u=4u_1^*(a,t)+1}^{\infty} \exp\left(-2\Delta_a^2\left(\sqrt{u} - \sqrt{u_1^*(a,t)}\right)^2\right)
\end{aligned}
$$

$$\leq \quad \frac{3\delta}{4\Delta_a^2 k_1 n t^4}. \tag{5.18}$$

The derivation for the last step is shown in Appendix B.2. From (5.17) and (5.18), we get

$$\begin{aligned}
& \mathbb{P}\{\exists a \in Arms : \big(u_a^t > 4u_1^*(a,t)\big) \wedge NEEDY_a^t\} \\
\leq \quad & \sum_{a \in Arms} \mathbb{P}\{\big(u_a^t > 4u_1^*(a,t)\big) \wedge NEEDY_a^t\} \\
\leq \quad & \frac{3\delta}{4k_1 n t^4} \sum_{a \in Arms, \Delta_a > \frac{\epsilon}{2}} \frac{1}{\Delta_a^2} \\
\leq \quad & \frac{3\delta H_1^{\frac{\epsilon}{2}}}{4k_1 n t^4}.
\end{aligned}$$

□

Lemma 5.10 (page 177)

Statement. Let $T_1^* = \left\lceil 146 H_1^{\epsilon/2} \ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\right\rceil$. For every $T \geq T_1^*$, the probability that LUCB1 has not terminated after T rounds of sampling is at most $\frac{2\delta}{T^2}$.

Proof. Let $\overline{T} = \left\lceil \frac{T}{2} \right\rceil$. We define two events, E_1 and E_2, over the interval $\{\overline{T}, \overline{T}+1, \ldots, T-1\}$:

$$E_1 \overset{\text{def}}{=} \exists t \in \{\overline{T}, \overline{T}+1, \ldots, T-1\} : CROSS^t, \text{ and}$$

$$E_2 \overset{\text{def}}{=} \exists t \in \{\overline{T}, \overline{T}+1, \ldots, T-1\} \; \exists a \in Arms : \big(u_a^t \geq 4u_1^*(a,t)\big) \wedge NEEDY_a^t.$$

We show that if neither E_1 nor E_2 occurs, then LUCB1 must necessarily terminate after at most T rounds of sampling. If the algorithm terminates after some $t \leq \overline{T}$ rounds of sampling, there is nothing left to prove. On the other hand, consider the case that the algorithm has not terminated after $\overline{T}$ rounds of sampling, and neither E_1 nor E_2 occurs. In this case, let $\#rounds$ be the number of additional rounds for which sampling occurs, up to round T. Applying Lemma 5.7, we get:

$$\begin{aligned}
\#rounds &= \sum_{t=\overline{T}}^{T-1} \mathbf{1}\left[NEEDY^t_{h^t_*} \vee NEEDY^t_{l^t_*}\right] \\
&\leq \sum_{t=\overline{T}}^{T-1} \sum_{a \in Arms} \mathbf{1}\left[\left(a = h^t_* \vee a = l^t_*\right) \wedge NEEDY^t_a\right] \\
&= \sum_{t=\overline{T}}^{T-1} \sum_{a \in Arms} \mathbf{1}\left[\left(a = h^t_* \vee a = l^t_*\right) \wedge \left(u^t_a \leq 4u^*_1(a,t)\right)\right] \\
&\leq \sum_{t=\overline{T}}^{T-1} \sum_{a \in Arms} \mathbf{1}\left[\left(a = h^t_* \vee a = l^t_*\right) \wedge \left(u^t_a \leq 4u^*_1(a,T)\right)\right] \\
&= \sum_{a \in Arms} \sum_{t=\overline{T}}^{T-1} \mathbf{1}\left[\left(a = h^t_* \vee a = l^t_*\right) \wedge \left(u^t_a \leq 4u^*_1(a,T)\right)\right] \\
&\leq \sum_{a \in Arms} 4u^*_1(a,T).
\end{aligned}$$

Appendix B.3 verifies that $T \geq T^*_1 \implies T > 2 + 8\sum_{a \in Arms} u^*_1(a,T)$. Thus, if neither E_1 nor E_2 occurs, the total number of rounds for which LUCB1 lasts is at most

$$\overline{T} + \#rounds \leq \left\lceil \frac{T}{2} \right\rceil + \sum_{a \in Arms} 4u^*_1(a,T) < \left\lceil \frac{T}{2} \right\rceil + \frac{T-2}{2} \leq T.$$

Consequently the probability that LUCB1 has not terminated after T rounds can be upper-bounded by $\mathbb{P}\{E_1 \vee E_2\}$.

$$\begin{aligned}
& \mathbb{P}\{E_1 \vee E_2\} \\
\leq\ & \mathbb{P}\{E_1\} + \mathbb{P}\{E_2\} \\
\leq\ & \mathbb{P}\left\{\exists t \in \{\overline{T}, \overline{T}+1, \ldots, T-1\} : CROSS^t\right\} \\
& + \mathbb{P}\left\{\exists t \in \{\overline{T}, \overline{T}+1, \ldots, T-1\}\ \exists a \in Arms : \left(u_a^t > 4u_1^*(a,T)\right) \wedge NEEDY_a^t\right\} \\
\leq\ & \sum_{t=\overline{T}}^{T-1} \left(\mathbb{P}\left\{CROSS^t\right\} + \mathbb{P}\left\{\exists a \in Arms : \left(u_a^t > 4u_1^*(a,T)\right) \wedge NEEDY_a^t\right\}\right).
\end{aligned}$$

Applying lemmas 5.8 and 5.9, we obtain:

$$\begin{aligned}
\mathbb{P}\{E_1 \vee E_2\} &\leq \sum_{t=T_1}^{T-1} \left(\frac{\delta}{k_1 t^3} + \frac{3\delta H_1^{\epsilon/2}}{4k_1 n t^4}\right) \\
&\leq \sum_{t=T_1}^{T-1} \frac{\delta}{k_1 T_1^3}\left(1 + \frac{3H_1^{\epsilon/2}}{4nT_1}\right) \\
&\leq \left(\frac{T}{2}\right)\left(\frac{8\delta}{k_1 T^3}\right)\left(1 + \frac{3H_1^{\epsilon/2}}{2nT}\right) \\
&< \frac{2\delta}{T^2}.
\end{aligned}$$

□

Theorem 5.11 and Corollary 5.12 (page 178)

Statement of theorem. The expected sample complexity of LUCB1 is

$$\min\left\{O\left(H_1^{\epsilon/2}\log\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\right), O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)\right\}.$$

Statement of corollary. With probability at least $1-\delta$, LUCB1 terminates after completing $O\left(H_1^{\epsilon/2}\log\left(\frac{1}{\delta H_1^{\epsilon/2}}\right)\right)$ rounds.

Proof. From Lemma 5.10, it follows that the expected sample complexity of LUCB1 is at most

$$2\left(T_1^* + \sum_{t=T_1^*+1}^{\infty} \frac{2\delta}{t^2}\right) < 292 H_1^{\epsilon/2} \ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right) + 10.$$

The expected sample complexity cannot exceed the worst case sample complexity, which we know from Theorem 5.11 to be $O\left(\frac{n}{\epsilon^2}\log\left(\frac{m}{\delta}\right)\right)$.

To prove the corollary, let us take $T = 2T_1^*$, and let $\delta' = \frac{2\delta}{T^2}$. From Lemma 5.10, we see that with probability at least $1-\delta'$, LUCB1 terminates before $\left\lceil 292 H_1^{\epsilon/2} \ln\left(\frac{2H_1^{\epsilon/2}}{\delta' T^2}\right)\right\rceil$ rounds. Since $T > H_1^{\epsilon/2}$, we obtain that with probability at least $1-\delta'$, LUCB1 terminates in $O\left(H_1^{\epsilon/2} \ln\left(\frac{1}{\delta' H_1^{\epsilon/2}}\right)\right)$ rounds. $\square$

Theorem 5.14 (page 180)

Statement. ELIMINATION(n, m, T) has a simple regret that is at most

$$\left(\sum_{i=1}^{m} \Delta_{i,n-i}\right) \min\left\{1, \frac{(n-m)(n-m+1)}{2} \sum_{i=1}^{m} \exp\left(-\frac{T-n}{2\left(\frac{m}{2} + \sum_{k=2}^{n-m+1}\frac{1}{k}\right) H_2(i)}\right)\right\}.$$

Proof. First observe that our allocation of samples to the $n-m$ phases in the ELIMINATION algorithm are such that they will not exceed T. An arm that is eliminated at the end of phase l (line 11) would have been sampled for exactly b_l rounds, $\forall l \in \{1, 2, \ldots, n-m\}$. The m arms that remain after $n-m$ phases would have been sampled for exactly b_{n-m} rounds before line 8 is encountered and any additional rounds are conducted, if necessary, to use up all T of the available samples.

$$
\begin{aligned}
& \sum_{l=1}^{n-m} b_l + m b_{n-m} \\
= \quad & \sum_{l=1}^{n-m} \left\lceil \frac{T-n}{\left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right)(n-m+2-l)} \right\rceil + m \left\lceil \frac{T-n}{\left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right)(2)} \right\rceil \\
\leq \quad & n + \sum_{l=1}^{n-m} \frac{T-n}{\left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right)(n-m+2-l)} + \frac{m(T-n)}{\left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right)(2)} \\
= \quad & n + \frac{T-n}{\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}} \left(\frac{m}{2} + \sum_{l=1}^{n-m} \frac{1}{n-m+2-l} \right) = T.
\end{aligned}
$$

First we upper-bound the simple regret in terms of the probability that *Selected* is different from *Top*. In the following, let M denote the set of all m-sized subsets of *Arms*. We have:

$$
\begin{aligned}
& \mathbb{E}\left[\sum_{i \in Top} p_i - \sum_{a \in Selected} p_a \right] \\
= \quad & \sum_{S \in M} \mathbb{P}\{Selected = S\} \left(\sum_{i \in Top} p_i - \sum_{a \in S} p_a \right) \\
= \quad & \sum_{S \in M \setminus Top} \mathbb{P}\{Selected = S\} \left(\sum_{i \in Top} p_i - \sum_{a \in S} p_a \right) \\
\leq \quad & \sum_{S \in M \setminus Top} \mathbb{P}\{Selected = S\} \left(\sum_{i \in Top} p_i - \sum_{a=n-m+1}^{n} p_a \right) \\
= \quad & \mathbb{P}\{Selected \neq Top\} \left(\sum_{i=1}^{m} \Delta_{i,n-i} \right). \qquad (5.19)
\end{aligned}
$$

If indeed *Selected* $\neq$ *Top*, then some arm $i \in Top$ must have been eliminated; therefore:

$$\begin{aligned}
&\mathbb{P}\{Selected \neq Top\} \\
\leq\ & \sum_{i \in Top} \mathbb{P}\{i \notin Selected\} \\
=\ & \sum_{i \in Top} \sum_{l=1}^{n-m} \mathbb{P}\{\text{Arm } i \text{ is eliminated in phase } l\}. \qquad (5.20)
\end{aligned}$$

Since exactly one arm is eliminated in each phase, at least one of the l arms in the set $\{n-l+1, n-l+2, \ldots, n\}$ reaches phase l (that is, is present in R_l), for $1 \leq l \leq n-m$. If arm i is eliminated in phase l, it implies that (1) arm i and some arm l' in the set above reach phase l, and (2) at the end of phase l, $\hat{p}_{l'} \geq \hat{p}_i$. If indeed (1) is true, note that arms i and k would each have been sampled at least for b_l rounds before their empirical means are compared. Applying Hoeffding's inequality to the difference between the Bernoulli random variables corresponding to arms i and l', we would get: $\mathbb{P}\{\hat{p}_{l'} \geq \hat{p}_i | \text{ arms } l' \text{ and } i \text{ reach phase } l\} \leq \exp\left(-2b_l \frac{(p_i - p_{l'})^2}{4}\right) = \exp\left(-b_l \frac{\Delta^2_{i,l'}}{2}\right)$. As a consequence,

$$\begin{aligned}
&\mathbb{P}\{\text{Arm } i \text{ is eliminated in phase } l\} \\
\leq\ & \sum_{l'=n-l+1}^{n} \exp\left(-b_l \frac{\Delta^2_{i,l'}}{2}\right) \\
\leq\ & \sum_{l'=n-l+1}^{n} \exp\left(-b_l \frac{\Delta^2_{i,n-l+1}}{2}\right) \\
=\ & l \exp\left(-\left\lceil \frac{T-n}{\left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right)(n-m+2-l)} \right\rceil \frac{\Delta^2_{i,n-l+1}}{2}\right)
\end{aligned}$$

$$\leq \quad l \exp\left(-\frac{T-n}{2\left(\frac{m}{2}+\sum_{k=2}^{n-m+1}\frac{1}{k}\right)H_2(i)}\right). \tag{5.21}$$

Combining (5.19), (5.20), and (5.21) yields the required bound on the simple regret. A further simplified bound based on the first-order terms is $\frac{m^2(n-m)(n-m+1)}{2}\exp\left(-\frac{T-n}{2\left(\frac{m}{2}+\sum_{k=2}^{n-m+1}\frac{1}{k}\right)H_2(m)}\right)$. This bound essentially generalizes the bound provided for $m = 1$ by Audibert et al. (2010). □

Theorem 5.15 (page 183)

Statement. The cumulative regret of UCBS(n, m) after T rounds of sampling is at most

$$\min\left\{\left(\sum_{i=1}^{m}\Delta_{i,n-i}\right)T, \sum_{i=1}^{m}\sum_{j=m+1}^{n}\frac{8\ln(T)}{\Delta_{i,j}}+\frac{1+\pi^2}{3}\sum_{i=1}^{m}\sum_{j=m+1}^{n}\Delta_{i,j}\right\}.$$

Proof. The cumulative regret after T rounds is given by

$$\mathbb{E}\left[\sum_{t=1}^{T}R^t\right], \text{ where}$$

$$R^t \overset{\text{def}}{=} \sum_{i\in Top}p_i - \sum_{a\in A^t}p_a.$$

We bound R^t in terms of the number of times some arm i from Top is not selected, but some arm j in Bot is selected:

$$R^t = \sum_{i\in Top}p_i - \sum_{a\in A^t}p_a$$

$$
\begin{aligned}
&= \sum_{i \in Top \setminus A^t} p_i - \sum_{a \in A^t \setminus Top} p_a \\
&= \mathbf{1}\left[Top \neq A^t\right] \left(\sum_{i \in Top \setminus A^t} p_i - \sum_{a \in A^t \setminus Top} p_a \right) \\
&= \frac{1}{|Top \setminus A^t|} \sum_{i \in Top \setminus A^t} \sum_{a \in A^t \setminus Top} (p_i - p_a) \\
&\leq \sum_{i \in Top \setminus A^t} \sum_{a \in A^t \setminus Top} (p_i - p_a) \\
&\leq \sum_{i \in Top} \sum_{j \in Bot} \Delta_{i,j} \mathbf{1}\left[i \notin A^t, j \in A^t\right].
\end{aligned}
$$

This bound is necessarily loose if A^t contains more than one arm from Bot. Now, let us define, $\forall i \in Top, \forall j \in Bot$:

$$
T_{i,j} \stackrel{\text{def}}{=} \sum_{t=1}^{T} \mathbf{1}\left[i \notin A^t, j \in A^t\right].
$$

Thus, the cumulative regret can be bounded as:

$$
\mathbb{E}\left[\sum_{t=1}^{T} R^t\right] \leq \sum_{i \in Top} \sum_{j \in Bot} \Delta_{i,j} \mathbb{E}[T_{i,j}]. \tag{5.22}
$$

We will show that $\mathbb{E}[T_{i,j}]$ does not substantially exceed $T^*_{i,j}$, where

$$
T^*_{i,j} \stackrel{\text{def}}{=} \left\lceil \frac{2}{\Delta^2_{i,j}} \ln(T^4) \right\rceil.
$$

Consider arm i in Top, and arm j in Bot. Under UCBS, after t rounds, j can be selected and i not selected only if $\hat{p}_i^t + \sqrt{\frac{1}{2u_i^t} \ln\left(t^4\right)} \leq \hat{p}_j^t + \sqrt{\frac{1}{2u_j^t} \ln\left(t^4\right)}$. Consider two cases:

1. $\sum_{\tau=1}^{t-1} \mathbf{1}\left[i \notin A^\tau, j \in A^\tau\right] < T^*_{i,j}$, and

2. $\sum_{\tau=1}^{t-1} \mathbf{1}\left[i \notin A^\tau, j \in A^\tau\right] \geq T^*_{i,j}$.

In the first case, we simply bound $\mathbb{P}\left\{\hat{p_i^t} + \sqrt{\frac{1}{2u_i^t}\ln\left(t^4\right)} \leq \hat{p_j^t} + \sqrt{\frac{1}{2u_j^t}\ln\left(t^4\right)}\right\}$ by 1. In the second case, observe that each of u_i^t and u_j^t must be at least $T^*_{i,j}$, and so, the lengths of the confidence bounds of arms i and j cannot exceed $\frac{\Delta_{i,j}}{2}$. Thus, in this case, the event $i \notin A^\tau, j \in A^\tau$ implies that either $\hat{p_i^t} \leq p_i - \frac{\Delta_{i,j}}{2}$, or $\hat{p_j^t} \geq p_j + \frac{\Delta_{i,j}}{2}$. By Hoeffding's inequality, the probabilities of these events are at most $\exp\left(-\frac{u_i^t \Delta_{i,j}^2}{2}\right)$ and $\exp\left(-\frac{u_j^t \Delta_{i,j}^2}{2}\right)$, respectively. We can therefore bound $\mathbb{E}[T_{i,j}]$ as follows.

$$\begin{aligned}
\mathbb{E}[T_{i,j}] &\leq T^*_{i,j} + \sum_{t=T^*_{i,j}}^{T-1} \sum_{u=T^*_{i,j}}^{t} \sum_{u_2=T^*_{i,j}}^{t} \left(\exp\left(-\frac{u_1\Delta_{i,j}^2}{2}\right) + \exp\left(-\frac{u_2\Delta_{i,j}^2}{2}\right)\right) \\
&\leq T^*_{i,j} + \sum_{t=T^*_{i,j}}^{T-1} 2t^2 \exp\left(-\frac{T^*_{i,j}\Delta_{i,j}^2}{2}\right) \\
&\leq T^*_{i,j} + \sum_{t=T^*_{i,j}}^{T-1} \frac{2T^2}{T^4} \\
&\leq T^*_{i,j} + \frac{\pi^2}{3}.
\end{aligned}$$

Substituting the bound for $\mathbb{E}[T_{i,j}]$ into (5.22) completes our proof. □

5.7 Summary and Discussion

In this chapter, we have undertaken a detailed investigation into the problem of subset selection, wherein the objective is to select, from n real-valued random variables, a subset of size m of those with relatively high means, based on efficiently sampling the random variables. This problem, which finds application in numerous areas, is particularly relevant to this dissertation through its connection with the selection routines of ranking-based PS methods such as CMA-ES and CEM. We have formalized subset selection using the framework of stochastic multi-armed bandits, specifically considering three operational settings: probably approximately correct (PAC) selection, simple regret (SR) minimization, and cumulative regret (CR) minimization. Under each of these settings, we have generalized previous work devoted to identifying just the single best arm in an n-armed bandit.

Our most novel results are under the PAC framework, where, in addition to generalizing the formulation of Even-Dar et al. (2006), we improve their high-probability sample complexity bounds. Indeed our LUCB algorithm for the PAC setting is qualitatively novel, and analogous to the well-known UCB algorithm for regret minimization. Additionally the algorithm is naturally decoupled into a stopping rule and a sampling strategy, of which the stopping rule alone is shown to determine the correctness of the algorithm. Under the SR and CR settings, our algorithms and analyses generalize previous work. Table 5.2 provides a summary of our results.

A recent line of work related to subset selection involves the OCBA-m algorithm (Chen et al., 2008), which is devised under a Bayesian framework. Indeed this method has been applied for subset selection within CEM (He et al., 2010). An important direction for future work is to experimentally compare our Elimination method with OCBA-m and related methods. Another related family of methods are racing algorithms (Mnih et al., 2008; Heidrich-Meisner and Igel, 2009a). Racing algorithms trace back to the seminal work of Maron and Moore (1997), which was originally developed to address model

Table 5.2: Subset selection: summary of results (see Section 5.2.3 for definitions of terms).

Setting	Our result and its relation to previous work
PAC-m	Expected sample complexity $= O\left(H_1^{\epsilon/2}\ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\right)$; generalizes and improves results of Even-Dar et al. (2006).
SR-m	Simple regret $\leq \frac{m^2(n-m)(n-m+1)}{2}\exp\left(-\frac{T-n}{2\left(\frac{m}{2}+\sum_{k=2}^{n-m+1}\frac{1}{k}\right)H_2(m)}\right)$; generalizes results of Audibert et al. (2010).
CR-m	Cumulative regret $\leq \sum_{i=1}^{m}\sum_{j=m+1}^{n}\left(\frac{8\ln(T)}{\Delta_{i,j}}+\frac{1+\pi^2}{3}\Delta_{i,j}\right)$; generalizes results of Auer et al. (2002a).

selection in machine learning. Racing algorithms fall in between our PAC and SR settings. In these algorithms, the number of available samples, T, and the mistake probability, δ, are both specified beforehand. Given these constraints, the objective is to identify the best individuals (or subset of individuals), *if at all possible*. Thus, if T or δ is relatively small, racing algorithms could terminate without returning a feasible subset.

Several improvements can be made to the algorithms we have presented in this chapter. Foremost, our algorithms and bounds are distribution-agnostic, depending only on the means of the bandit arms. In general further efficiency can be obtained by also keeping track of the empirical variances of arms (Mnih et al., 2008; Audibert et al., 2009), or entire empirical distributions (Honda and Takemura, 2010). Yet another fundamental assumption we might revisit concerns (the lack of) dependencies between bandit arms. In several practical applications, it would be profitable to explicitly model dependencies across arms (Frazier et al., 2009) and even across populations of bandit instances (Li et al., 2010).

Whereas we have only focused in this chapter on subset selection, in practice, several other selection schemes, such as tournament selection and proportionate selection (Miller and Goldberg, 1996), are used in tandem with evolutionary methods. Also, some evolutionary algorithms seek to maintain good *on-line* performance (Whiteson and Stone, 2006b), which the "pure exploration" nature of PAC-m and SR-m does not match. It would be useful to investigate whether our UCBS algorithm (under CR-m) could be used effectively in tandem with on-line evolutionary algorithms.

The subset selection algorithms developed in this chapter apply beyond the ambit of this dissertation. Yet, they are specifically relevant to this dissertation as we have shown them to improve the sample-efficiency of certain PS methods. In the next chapter, we consider developing hybrid learning methods for sequential decision making, wherein VF and PS methods are combined in effective ways.

Chapter 6

Hybrid Learning Methods: Two Case Studies

Experimental results from Chapter 3 indicate a strong advantage in favor of VF methods with respect to learning speed; yet it is PS methods that perform better when working with significantly deficient representations. As introduced in Section 1.3.3, this chapter presents two case studies wherein we synthesize the strengths of VF and PS methods via hybrid learning architectures.

Case Study 1 *involves a conceptually simple, yet surprisingly effective algorithmic contribution arising from this dissertation. We show that the straightforward scheme of applying VF and PS methods in sequence—initially VF for some duration, followed by a transfer of the learned weights to be refined using PS—indeed inherits the strengths of its VF and PS constituents, and is often more successful than either of them taken in isolation. We arrive at this conclusion based on results both from the suite of parameterized learning problems introduced in Chapter 3, and from the more complex and realistic RL benchmark task of robot soccer Keepaway.*

Case Study 2 *undertakes a deeper investigation involving the Keepaway task. Whereas Case Study 1, and indeed most previous work on Keepaway, limit learning in the task to an isolated, infrequent decision that amounts to a turn-taking behavior ("*Pass*" behavior), here we expand the agents' learning capability to include a much more ubiquitous action (moving without the ball, or "*GetOpen*" behavior). We introduce a policy search method for learning*

GetOpen*; policy search is indeed more aptly suited for learning* GetOpen *than the TD learning approach previous studies have employed for learning* Pass*. Results establish the success of learning* GetOpen *both separately and together with* Pass*. We consider our demonstration of the autonomous learning of a fairly substantial fraction of a complex task, achieved only by putting together distinct learning techniques, a relatively novel and crucial element in scaling RL to even more complex problems.*

The two case studies presented in this chapter both contribute to learning methodology oriented towards practical applications. After providing a detailed description of the Keepaway task in Section 6.1, we present the two case studies in Sections 6.2 and 6.3. A discussion of related work follows in Section 6.4.

This chapter, like the previous one, draws inspiration from the experiments and analyses in chapters 3 and 4 in order to design learning methods that are effective in practice. Whereas the previous chapter undertook a focused, theoretical study of subset selection and its application to policy search, this chapter examines the possibility of developing *hybrid* learning methods that combine the strengths of VF and PS methods.

Results from chapters 3 and 4 highlight that VF methods display superior sample efficiency when compared with PS methods; on the other hand, VF methods are affected to a greater degree by poor representations than are PS methods. Problems encountered in practice are bound to have representations varying from "very good" to "very poor". Ideally, we desire learning methods that for any given representation, are able to quickly learn the best policies achievable. Can we devise such learning methods? In this chapter, we consider two promising learning frameworks, each through a case study.

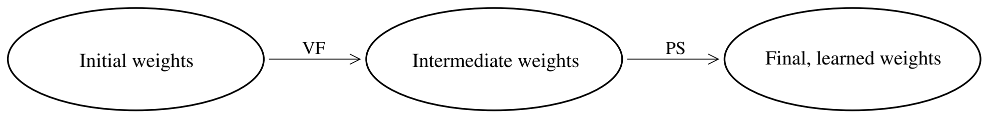
Under Case Study 1, we introduce a sequencing algorithm under which a VF method is applied for an initial phase during learning, following which the learned behavior is refined using a PS method. Experiments show that this conceptually simple scheme is quite successful on the suite of parameterized learning problems designed in Chapter 3. Indeed its success also extends to the more complex RL benchmark of robot soccer Keepaway (Stone et al., 2005), which becomes the focus of our second case study.

Keepaway is a subtask of robot soccer in which one team of players must keep possession of the ball, while resisting the attempts of an opposing team. Whereas successful ball possession depends both on (1) the strategy of the player with the ball (in deciding whether and where to pass) and (2) of its teammates (in deciding where to position themselves), prior work in Keepaway has focused only on learning the former behavior, treating the latter behavior as fixed and manually programmed. Under Case Study 1, too, we only learn the former behavior, "Pass", while following a standard hand-coded policy for the latter behavior, "GetOpen".

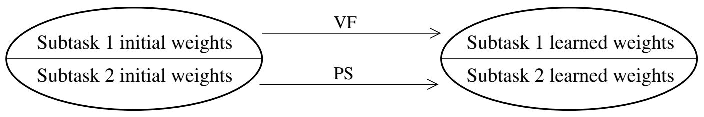
Under Case Study 2, we show that the complementary behaviors of Pass and GetOpen can indeed be learned together. Incidentally the structure of these tasks is such that whereas Pass is well-suited to be learned using a VF method, it is far more effective to set up a parameterized policy for GetOpen and to optimize it using policy search. In effect, if we consider "Pass+GetOpen" as a compound sequential decision making problem, we find that it can be learned most efficiently by decomposing it into interrelated subtasks, which are themselves learned in tandem using qualitatively different learning methods.

The contrasting learning approaches under case studies 1 and 2 are summarized in Figure 6.1. Whereas VF and PS methods are applied in sequence under Case Study 1, and in parallel under Case Study 2, importantly, both schemes are instances of multiple learning methods being applied to the same task. We believe that such integrated approaches are necessary to extend the reach of RL to more complex applications, and that the case studies presented here can offer useful guidance for future research in this area.

The remainder of this chapter is organized as follows. First, in Section 6.1, we give a detailed description of the Keepaway task, which is used as a testbed in both the case studies that follow. Section 6.2 is devoted to Case Study 1, and Section 6.3 to Case Study 2. A discussion of related work follows in Section 6.4, and finally, Section 6.5 summarizes the chapter.



(a) Case Study 1


Subtask 1 initial weights
Subtask 2 initial weights
VF
PS
Subtask 1 learned weights
Subtask 2 learned weights

(b) Case Study 2

Figure 6.1: Summary of learning schemes under (a) Case Study 1 and (b) Case Study 2. In Case Study 1, the weights to be learned are first adapted using a VF method, such that they approximate the value function being learned. Once learning begins to plateau, a PS method is employed to tune the weights such that the value of the resulting policy is increased. Under Case Study 2, the task is itself decomposed into subtasks (PASS and GETOPEN), each with its own policy structure and weights. The weights of the different subtasks are worked upon by different learning methods. In our implementation (see Section 6.3.1), the weights under PASS and GETOPEN are adapted in an interleaved manner.

6.1 Task Description: Robot Soccer Keepaway

This section provides a detailed description of the Keepaway task, specifically its decomposition into the Pass and GetOpen behaviors. Recall that the learning methodology developed in Case Study 1 is evaluated on the Pass subtask, while keeping GetOpen fixed. However, learning Pass and GetOpen in tandem becomes the focus of Case Study 2.

The RoboCup 2D simulation soccer domain (Chen et al., 2003) models several difficulties that agents must cope with in the real world. Soccer is necessarily a multiagent enterprise, in which agents have both teammates and opponents. The RoboCup 2D simulation platform only provides agents partial and noisy perceptions; the agents have noisy actuators. The agents' sensing and acting routines are not synchronized, and in the interest of keeping real time, do not admit extensive deliberation. The atomic actions available to an agent are Turn, Turn-Neck, Dash, Kick, and Catch; skills such as passing to a teammate or going to a point must be composed of a string of these low-level actions executed in sequence. For all these reasons, RoboCup simulation soccer becomes a challenging domain for machine learning.

Keepaway (Stone et al., 2005) is a subtask of soccer in which a team of three *keepers* aims to keep possession of the ball[1] away from the opposing team of two *takers*. The game is played within a square region of side $20m$.[2] Each episode begins with some keeper having possession of the ball, and ends when some taker claims possession or the ball overshoots the region of play. It is the objective of the keepers to maximize the expected length of the episode,

[1]A player is deemed to have *possession* of the ball if the ball is close enough to be kicked.

[2]Keepaway can be generalized to varying numbers of keepers and takers, as well as different field sizes (Stone et al., 2005).

referred to as the episodic **hold time**. The keepers must *cooperate* with each other in order to realize this objective; they *compete* with the team of takers, which seeks to minimize the hold time. Figure 6.2 shows a snapshot of a Keepaway episode in progress.

In order to make the Keepaway task amenable to learning, it becomes necessary to constrain the scope of decision making by the keepers. Figure 6.3 outlines the policy followed by *each* keeper in the scheme employed by Stone *et al.* (2005). The keeper closest to the ball intercepts the ball until it has possession. Once it has possession, it must execute the Pass behavior (not to be confused with a pass action), by way of which it may retain ball possession or pass to a teammate. Keepers other than the one closest to the ball move to a position conducive for receiving a pass by executing GetOpen behavior.

Pass and GetOpen, by offering a *choice* of high-level actions based on the keeper's state, are candidates for the application of learning. Most prior work involving learning in Keepaway is only concerned with learning Pass (as

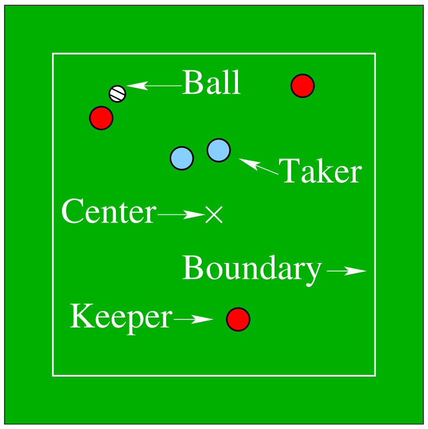


Figure 6.2: A snapshot of Keepaway.

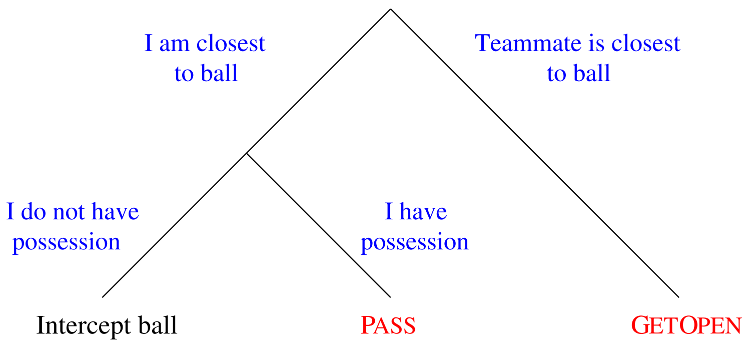


Figure 6.3: Policy followed by each keeper in Keepaway

we will be under Case Study 1 for testing a sequencing algorithm). In such work, fixed, hand-coded strategies are assigned for GetOpen, and indeed for the behavior followed by the takers. In other words, the teammates and opponents of the keeper with the ball do not *adapt* to the specific characteristics of that keeper, as they might in real soccer. As a step in the direction of furthering team adaptation, under Case Study 2, we extend the frontier of learning in Keepaway to include GetOpen. Thus, we treat Keepaway as a composite of two distinct behaviors to be learned: Pass and GetOpen. As in previous work (Stone et al., 2005), we restrict the takers to the fixed policy of moving towards the ball.[3] In the remainder of this section, we provide detailed specifications of the Pass and GetOpen behaviors and describe how they interact.

[3]Note that in recent work, Iscen and Erogul (2008) do explore learning taker behavior, thereby complementing the work in this chapter. Section 6.4 includes a brief description of their work.

6.1.1 Keepaway PASS

Here we revisit the problem of PASS defined by Stone *et al.* (2005). The keepers and takers assume roles that are indexed based on their distances to the ball: K_i is the i^{th} closest keeper to the ball, and T_j the j^{th} closest taker. From Figure 6.3, we see that the keeper executing PASS must be K_1. The three high-level actions available to K_1 are HoldBall, which is composed of a series of kicks close to itself, but away from any approaching takers; and PassBall-i, $i \in \{2, 3\}$, a direct pass to K_i. Each player processes its low-level perceptual information to construct a world model, which constitutes a continuous state space. This space is represented through a vector of 13 state variables, comprising distances and angles among the players and the center C of the field. These state variables are marked in Figure 6.4, and enumerated in Table 6.1.

A policy for PASS maps a 13-dimensional vector representing the state variables to one of the high-level actions: HoldBall, PassBall-2, and PassBall-3.

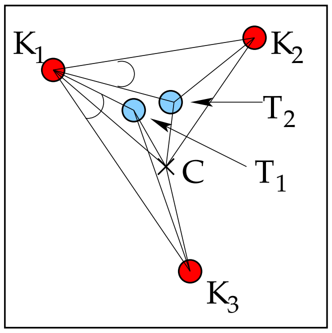


Figure 6.4: Illustration of PASS state variables (as specified in Table 6.1).

Table 6.1: PASS state variables.

$dist(K_1, K_2)$	$\min_{j \in 1,2} dist(K_2, T_j)$	$dist(K_1, C)$
$dist(K_1, K_3)$	$\min_{j \in 1,2} ang(K_2, K_1, T_j)$	$dist(K_2, C)$
$dist(K_1, T_1)$	$\min_{j \in 1,2} dist(K_3, T_j)$	$dist(K_3, C)$
$dist(K_2, T_2)$	$\min_{j \in 1,2} ang(K_3, K_1, T_j)$	$dist(T_1, C)$
		$dist(T_2, C)$

An example of such a policy is PASS:HAND-CODED (Algorithm 6.1), which implements a well-tuned manually programmed strategy (Stone et al., 2005). Under this policy, K_1 executes HoldBall until the takers get within a certain range, after which distances and angles involving its teammates and opponents are used to decide whether (and to which teammate) to pass. Yet another policy for PASS is PASS:RANDOM, under which K_1 chooses one of the three

Algorithm 6.1 PASS:HAND-CODED

```
Input: PASS state variables (13).
Output: Action ∈ {HoldBall, PassBall-2, PassBall-3}.

if $dist(K_1, T_1) > C_1$ then
    Return HoldBall.
for $i \in \{2, 3\}$ do
    $valAng_i \leftarrow \min_{j \in \{1,2\}} ang(K_i, K_1, T_j)$.
    $valDist_i \leftarrow \min_{j \in \{1,2\}} dist(K_i, T_j)$.
    $val_i \leftarrow C_2 \cdot valAng_i + valDist_i$.
if $\max_{i \in \{2,3\}} val_i > C_3$ then
    $passIndex \leftarrow \text{argmax}_{i \in \{2,3\}} val_i$.
    Return PassBall-$passIndex$.
else
    Return HoldBall.

{$C_1 = 5.0, C_2 = 0.25, C_3 = 22.5$; distances are taken to be in meters and angles in degrees.}
```

available actions with equal likelihood. Pass:Learned denotes a learned Pass policy, of which we will encounter examples in Section 6.2.2.2 (under Case Study 1) and Section 6.3.1 (under Case Study 2).

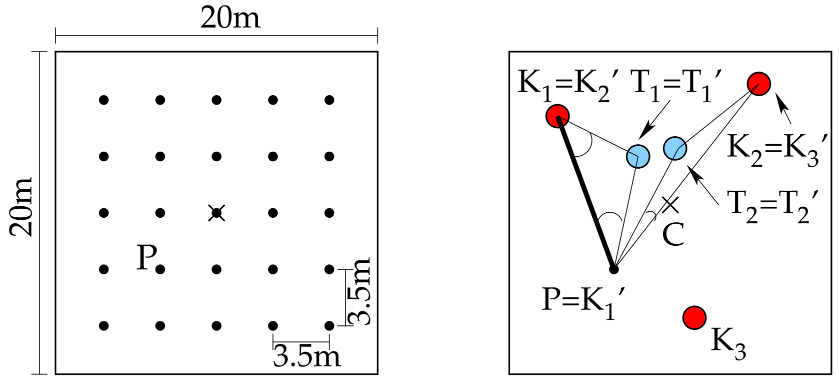
6.1.2 Keepaway GETOPEN

Whereas learning Pass has been studied quite extensively in the literature (Jung and Polani, 2007; Metzen et al., 2008), predominantly, all previous work has used the hand-coded GetOpen policy originally defined by Stone *et al.* (2005), to which we refer here as GetOpen:Hand-coded. Thus, while previous work on Keepaway has considered multiple agents learning, the agents have never been executing their learned behaviors concurrently (only one player executes Pass at any given time). By introducing a learned GetOpen behavior in Case Study 2, we significantly expand the scope of multiagent learning in Keepaway. Below we describe our formulation of GetOpen.

In principle there are infinitely many positions on the square playing field that K_2 and K_3 can occupy. However, they only get a small amount of time to pick a target. Since points close to each other are likely to be of similar value, an effective strategy is to evaluate only a small, finite set of points spread out across the field and choose the most promising. Figure 6.5(a) shows a uniform grid of 25 points overlaid on the field, with a 15% margin on the sides. GetOpen is implemented by evaluating each grid point P and moving to the one with the highest value. Indeed we define the learning problem under GetOpen to be that of learning an evaluation function assigning a value to every target point P (given the configuration of the players).

As with Pass, it becomes necessary to define a set of state variables for learning GetOpen. In Figure 6.5(b), K_3 is shown seeking to evaluate a potential target point P at some instant of time t. The distances and angles marked in the figure correspond to the GetOpen state variables used for the purpose, which we identify based on informal experimentation. None of the state variables involves K_3, as K_3 is examining a situation at time t' in the future when it would itself be at P. At time t', K_3 expects to have possession of the ball, and re-orders the other players based on their distances to it. Thus K_3 becomes K'_1, and in the configuration in Figure 6.5(b), K_1 becomes K'_2, T_1 becomes T'_1, and so on.

Conceptually the evaluation of the target point P should consider both the likelihood of receiving a pass at P and the value of being at P with the



(a) (b)

Figure 6.5: (a) Uniformly-spaced target points under GetOpen. Each point is evaluated independently by K_2 and K_3, and one with the highest evaluation is chosen by each of those keepers as its target point. (b) For example, if some point P is being evaluated by K_3, state variables (listed in Table 6.2) are derived based on P and the positions of the players, and used in computing P's evaluation. Dashed names (such as K'_2 and T'_1) index the players based on their distances to P. In this example, state variables $dist(K'_1, K'_2)$ and $dist(K_1, K'_1)$ (darkened) overlap.

ball afterwards. This leads to two logical groups within the state variables for GetOpen. One group contains two variables that influence the success of a pass from K_1 to K'_1, the latter being at P. These variables are the distance between K_1 and K'_1, and the minimum angle between K_1, K'_1, and any taker. State variables in the other group bear direct correspondences with those used for learning Pass, but computed under the re-ordering at t'. Of the 13 state variables used for Pass, we leave out the five distances between the players and the center of the field, as they do not seem to benefit the learning of GetOpen. The resulting ten state variables for GetOpen are listed in Table 6.2.

In defining state variables for GetOpen, it is implicitly assumed that players other than K'_1 do not change their positions between instants t and t'. This clearly imperfect assumption does not have too adverse an effect since GetOpen is executed *every* cycle, always with the *current* positions of all players. Revising the target point every cycle, however, has an interesting effect on a random GetOpen policy. In order to get from point A to point B, a player must first turn towards B, which takes 1-2 cycles. When a random target point is chosen each cycle, K'_1 constantly keeps turning, achieving little net dis-

Table 6.2: GetOpen state variables.

$dist(K'_1, K'_2)$	$\min_{j\in 1,2} dist(K'_2, T'_j)$	$dist(K_1, K'_1)$
$dist(K'_1, K'_3)$	$\min_{j\in 1,2} ang(K'_2, K'_1, T'_j)$	$\min_{j\in 1,2} ang(K'_1, K_1, T_j)$
$dist(K'_1, T'_1)$	$\min_{j\in 1,2} dist(K'_3, T'_j)$	
$dist(K'_2, T'_2)$	$\min_{j\in 1,2} ang(K'_3, K'_1, T'_j)$	

placement. To redress this effect, our implementation of GetOpen:Random only allows K_1' to revise its target when it reaches its current target. Such a measure is not necessary when the targets remain reasonably stable, as they do for GetOpen:Learned, the learned GetOpen policy under Case Study 2, and GetOpen:Hand-coded, which we describe next.

Under GetOpen:Hand-coded (Stone et al., 2005), specified in Algorithm 6.2, the value of a point P is inversely related to its *congestion*, a measure of its distances to the keepers and takers. Additionally, assuming that K_1 will pass the ball from *predictedBallPos*, P is deemed an inadmissible target (given a value of $-\infty$) if any taker comes within a threshold angle of the line joining *predictedBallPos* and P. The effective use of variables such as congestion and the predicted position of the ball make GetOpen:Hand-coded a fairly sophisticated policy. These abstract variables are not among the state variables used by GetOpen:Learned (listed in Table 6.2). In Section 6.3.2.1, we compare GetOpen:Hand-coded with GetOpen:Learned to verify if indeed the distances and angles used by the latter suffice for describing competent GetOpen behavior.

Algorithm 6.2 GetOpen:Hand-coded

```
Input: Evaluation point $P$, World State.
Output: Value at $P$.
$teamCongestion \leftarrow \sum_{i \in \{1,2,3\}, i \neq myIndex} \frac{1}{dist(K_i,P)}$.
$oppCongestion \leftarrow \sum_{j \in \{1,2\}} \frac{1}{dist(T_j,P)}$.
$congestion \leftarrow teamCongestion + oppCongestion$.
$value \leftarrow -congestion$.
$safety \leftarrow \min_{j \in \{1,2\}} ang(P, predictedBallPos, T_j)$.
if $safety < C_1$ then
    $value \leftarrow -\infty$.
Return $value$.

{$C_1 = 18.4$; angles are taken to be in degrees.}
```

6.1.3 Keepaway PASS+GETOPEN

Pass and GetOpen are separate behaviors of the keepers, which, together, may be viewed as "distinct populations with coupled fitness landscapes" (Rosin and Belew, 1995). At any instant, there are two keepers executing GetOpen; their teammate, if it has intercepted the ball, executes Pass. Specifically each keeper executes GetOpen when it assumes the role of K_2 or K_3, and executes Pass when it has possession of the ball, as K_1. The extended sequence of actions that results as a combination each keeper's Pass and GetOpen policies determines the team's performance. Indeed the episodic hold time is precisely the temporal length of that sequence.

Pass has been the subject of many previous studies, in which it is modeled as a semi-Markov Decision Problem (SMDP) (Puterman, 1994) and tackled using TD learning methods (Stone et al., 2005; Jung and Polani, 2007). In Pass, exactly one keeper takes an action at a time (one of HoldBall, PassBall-1, and PassBall-2). Hence, only the keeper that takes an action needs to get rewarded for it. Indeed if the reward is the time elapsed until the keeper takes its next action (or the episode ends), the episodic hold time gets maximized if each keeper maximizes its own long-term reward.

Unfortunately GetOpen does not admit a similar credit assignment scheme, because at any instant, two keepers (K_2 and K_3) take GetOpen actions to move to their target points. If K_1 executes the HoldBall action, neither K_2 nor K_3 will receive a pass. If K_1 passes to K_2, how must K_3 be rewarded? In principle the sequence of *joint actions* taken by K_2 and K_3 up to the successful pass must be rewarded. Yet a joint action is taken *every* cycle (in contrast with Pass actions, which last 4-5 cycles on average), and the large number of atomic GetOpen actions (25, compared to 3 for Pass) leads to a very large joint action space for GetOpen. In short, GetOpen induces a far more complex MDP than Pass, and clearly one that is qualitatively dissimilar. An additional obstacle to be surmounted while learning Pass and GetOpen together is the non-stationarity introduced by each of these behavioral components into the other's environment. All these reasons, combined with the inherent complexity of RoboCup 2D simulation soccer, make Pass+GetOpen a demanding problem for machine learning.

Having specified Keepaway Pass and GetOpen in detail, we now proceed to describe the algorithmic contributions of this chapter. Case Study 1 uses the suite of parameterized learning problems from Chapter 3, along with Keepaway Pass, in its experiments. Case Study 2 specifically investigates the learning of Pass+GetOpen (after learning Pass and GetOpen separately).

6.2 Case Study 1: Sequencing Learning Methods

In our experiments in Chapter 3, we found that CMA-ES (and other PS methods) are typically much slower to learn than Sarsa (and other VF methods). For example, the disparity between the sample-efficiency of VF and PS methods is clearly visible in Figure 3.8 (page 74), in which their performance on three separate problem instances is measured after various amounts of training time. Over shorter training durations, Sarsa and Q-learning are easily the most effective learning methods on all three problems. However, as more training samples become available, we observe that on problem instance I_2, the most successful methods are PS methods such as CMA-ES and CEM. Recall that I_2 is characterized by its relatively low setting for χ (0.5), corresponding to a generalization scheme with limited expressiveness.

In our parameterized learning problem from Chapter 3, the state aliasing parameter σ is somewhat less regular than χ in separating the performance of VF and PS methods. For example, in Figure 3.10 (page 80), we see that for a given σ setting, the order between Sarsa* and CMA-ES* can differ based on the setting of w, the generalization width. Nevertheless, under all three settings of w in Figure 3.10, we still see that the difference between the performance of Sarsa* and CMA-ES* diminishes as σ is increased. Under $w = 5$, indeed we observe a threshold for σ (between 2 and 4) *below* which Sarsa* outperforms CMA-ES*, and *above* which the opposite happens. In summary, we might conclude based on Chapter 3 that

1. VF methods are more sample-efficient than PS methods, but
2. PS methods can often achieve better performance under deficient representations (low χ and high σ settings).

Surely it is desirable that any practical learning method possess both attributes: sample-efficient learning and robustness to poor representations. In this case study, we devise one such a method, whose effectiveness we then demonstrate both on the suite of parameterized learning problems from Chapter 3, and on Keepaway Pass, which was described in Section 6.1.1.

6.2.1 Sequencing Algorithm

VF methods tend to "plateau" much earlier than PS methods: that is, they approach close to their maximal performance based on far fewer samples. Consider a task in which we have 50,000 episodes for training. Suppose a VF method such as Sarsa begins to plateau on this task, say, after a mere 5,000 episodes; it then seems wasteful to devote the remaining 45,000 episodes towards the very minor improvements Sarsa might yet achieve. Could we put the remaining 45,000 episodes to better use?

In problems with poor representations, our results from Chapter 3 suggest that the asymptote reached by Sarsa is often significantly inferior compared to that of CMA-ES. A natural possibility for combining the strengths of these methods is therefore to execute Sarsa for, say, 5000 episodes, and then to use CMA-ES over the remaining 45,000 episodes to refine the policy learned by Sarsa. One way to interpret such a scheme is that CMA-ES is essentially used to improve the policy learned by Sarsa; another interpretation is that we use Sarsa to quickly identify a better-than-random initial setting for CMA-ES. Since in our parameterized learning problem, Sarsa and CMA-ES are constrained to share a common representation (Section 3.1, page 40), a straightforward way to initialize CMA-ES with a policy learned using Sarsa is to set its initial weights to those learned by Sarsa. Although a raw transfer of weights is not always applicable across different representations, we conjecture that the resulting technique can still offer insights about synthesizing the merits of VF and PS methods.

We refer to the resulting sequencing method—first running a VF method until it plateaus, and then transferring weights to a PS method—as "Seq", which is summarized in Algorithm 6.3. Note that Seq is more akin to an algorithmic framework where different constituent methods from the VF and PS classes can be employed, and different mechanisms implemented for the transfer of policy between them. In the next section, we discuss experimental results with two different instantiations of Seq.

Algorithm 6.3 Seq

```
1. Execute a VF method (such as Sarsa or Q-learning) until its performance
   no longer improves significantly, or the total training time has been reached.
   Save the resulting vector of weights $\mathbf{w}_{VF}$.
2. Execute a PS method (such as CMA-ES or CEM), starting with $\mathbf{w}_{VF}$
   its initial weights, until the total training time has been reached. Save the
   resulting vector of weights $\mathbf{w}_{PS}$.
3. Return $\mathbf{w}_{PS}$.
```

6.2.2 Results

We apply Seq both to the suite of parameterized learning problems introduced in Section 3.1 (page 40), which facilitates extensive testing, and to Keepaway PASS, which was described in Section 6.1.1. We find that Seq is successful in both cases.

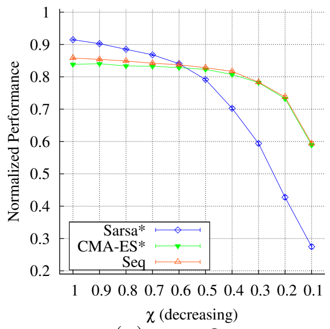
6.2.2.1 Parameterized Learning Problem

To test Seq in our parameterized learning problem, we use Sarsa and CMA-ES as its constituent methods. Recall from Chapter 3 that these methods achieve the best performance among the VF and PS classes, respectively. In principle, the method-specific parameters of Seq therefore include all of the method-specific parameters of Sarsa (λ, α_0, ϵ_0, θ_0) and CMA-ES (#*trials*, #*gens*), along with the number of episodes at which the transfer of weights is effected from Sarsa to CMA-ES.

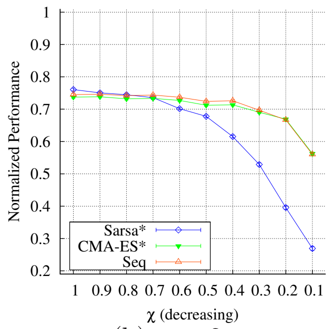
For a given problem setting, we may conduct a search over all seven of these method-specific parameters to find the most effective configuration of Seq, and denote that configuration Seq*. If we did so, it is clear that Seq* would always perform at least as well as the better of Sarsa* and CMA-ES*

(by running either Sarsa* or CMA-ES* for the entire training period). In the experiments to follow, rather than search for and evaluate this best instance of Seq, we constrain Seq to (1) use Sarsa* and CMA-ES*, each optimized independently for 50,000 episodes, as its constituents; and (2) transfer weights from Sarsa* to CMA-ES* after 2,500 episodes of training. Thus, Seq essentially amounts to running CMA-ES*, but starting from a potentially useful initialization. Our choice of transferring weights after 2,500 episodes arises from the judgment that when learning runs are conducted for 50,000 episodes, the performance of Sarsa typically does not increase much after 2,500 episodes. Of course, slightly better results could be achieved on a given problem instance by tuning this "transfer point" between Sarsa and CMA-ES more carefully—for our current experiments, we consider the fixed setting of 2,500 episodes a reasonable choice. In our experiments, we set the initial variance corresponding to each parameter to be optimized by CMA-ES* (after the switch) to the overall variance of the weights themselves.

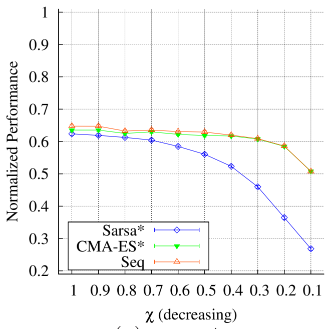
Figure 6.6 compares Seq with Sarsa* and CMA-ES* under problem settings in which σ and χ are varied. Under all settings, we find that Seq performs at least as well as CMA-ES, if not marginally better. Thorough "head-to-head" comparisons between each pair of methods are plotted in the bottom row of the figure. Each plot therein compares two of the methods. For specified settings of σ and χ, the method registering higher performance is marked if the evidence is statistically significant (p-value < 0.01). Figure 6.6(d) identifies regions of the problem space suiting Sarsa* and CMA-ES*. From Figure 6.6(e), we observe that Seq marginally extends the territory claimed by CMA-ES*. Indeed Seq outperforms CMA-ES* in regions where σ is low and χ is high, and performs at least as well in the remainder of the problem configurations



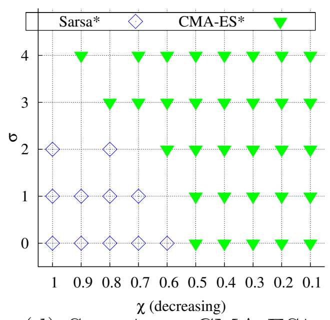
(a) $\sigma = 0$



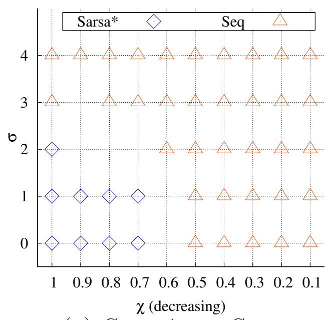
(b) $\sigma = 2$



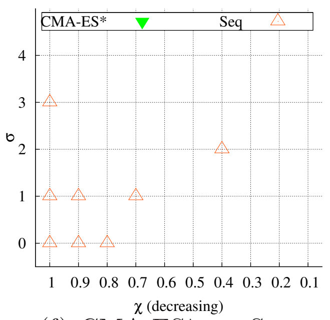
(c) $\sigma = 4$



(d) Sarsa* vs. CMA-ES*



(e) Sarsa* vs. Seq



(f) CMA-ES* vs. Seq

Figure 6.6: [$s = 10$, $p = 0.2$, $w = 5$.] **Top row** (a, b, and c): Sarsa*, CMA-ES*, and Seq compared at different settings of σ and χ in our suite of parameterized learning problems from Chapter 3. The performance plotted is based on 50,000 episodes of training. Sarsa* and CMA-ES* are optimized independently at each problem setting; Seq combines the methods thus tuned (with no further optimization), transferring weights from Sarsa to CMA-ES after 2,500 training episodes. **Bottom row** (d, e, and f): Plots showing *pairwise* comparisons between Sarsa*, CMA-ES*, and Seq at different settings of σ and χ. At each reported setting, one of the methods is indicated if with statistical significance ($p < 0.01$), it achieves a higher performance than the other. At some settings, the methods cannot be thus separated; note that in plot (f), CMA-ES* does not outperform Seq at *any* of the reported settings of σ and χ.

(Figure 6.6(f)). Thus, in summary we note that Seq consistently outperforms CMA-ES* in the section of problems covered in Figure 6.6; on some of these problems, Seq also outperforms Sarsa*. We find further evidence in strong support of Seq when we evaluate it on Keepaway Pass. On this more complex task, Seq indeed outperforms both its VF and PS constituents.

6.2.2.2 Keepaway Pass

When we evaluate our learning methods on the Keepaway Pass task, we pair it with GetOpen:Hand-coded (described earlier in Algorithm 6.2), which is a well-tuned manually designed policy for GetOpen (Stone et al., 2005). We consider learning GetOpen under Case Study 2 (in Section 6.3).

Unlike with our parameterized learning problem, a linear representation with a small number of features is not sufficient for representing competent Keepaway Pass policies, and nor is it clear if optimality is achievable. Note that each keeper learns autonomously, using only its own stream of experiences: this multiagent aspect of Keepaway Pass is another aspect distinguishing it from the parameterized learning problems considered in Section 6.2.2.1.

In the experiments reported here, our Pass policy is represented through three 13-20-1 neural networks, computing activations for each action. Such an architecture has been used successfully in the past with Sarsa(0) as the learning method (Stone et al., 2006). Indeed we retain Sarsa(0) as the VF method to work with our neural network architecture, using constant values of $\epsilon = 0.01$ and $\alpha = 0.0001$. We fix these exploration and learning rates based on manual tuning; we do not find it necessary to anneal either of them.

Note that other function approximators have also been used successfully in the literature for learning Pass: in particular, tile coding and radial basis functions. One of the reasons we employ neural networks here for representing the value function and policy is their compactness (a total of 903 weights in this case, rather than the tens of thousands needed with tile coding and radial basis functions for obtaining good performance). The relatively small number of weights to adapt makes the architecture readily usable by PS methods, and indeed enables us to implement Seq using a direct transfer of weights from VF to PS. Under Case Study 2, we adopt the tile coding scheme introduced by Stone et al. (2005) for learning PASS using Sarsa(0): tile coding achieves slightly better results than the present neural network implementation.

As a representative method from the PS class, we run the cross-entropy method (CEM), earlier described in Section 3.2.2. We tune this method in order to maximize performance after 40,000 episodes of training, which we achieve by maintaining a population of 20 solutions, each evaluated for 125 episodes. The initial Gaussian distribution used for generating the weights of the neural net is $N(0,1)^{903}$ (each neural net has 301 weights, including biases). After each generation (2500 episodes), the 5 best performing policies are used to determine the mean and variance for the subsequent generation.

Keepaway is significantly more complex than our parameterized learning problem. Owing to time constraints, we decide to stick with Sarsa(0) and CEM as representative VF and PS methods to test on this task, rather than conduct extensive within-class comparisons with other methods (as in Section 3.3.2). To implement Seq, we combine Sarsa(0) and CEM, where we transfer the weights of the neural networks after 15,000 episodes of Sarsa training (and set each variance to 1). Figure 6.7 shows that on PASS, Seq dominates both Sarsa(0) and CEM beyond 20,000 episodes of training (p-value < 0.01). The "Random" policy shown in the graph picks from the three actions in

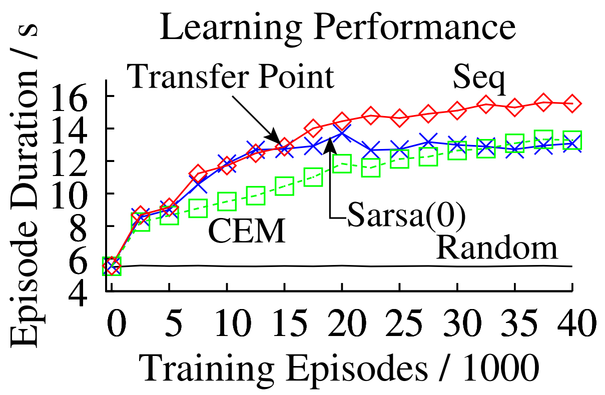


Figure 6.7: Learning curves corresponding to four methods on the Keepaway Pass task: Random, Sarsa(0), CEM, and Seq. The x axis marks the number of episodes of training, and the y axis shows the performance registered by the learned policy, calculated by averaging at least 14 independent testing trials. At each reported point, the learned policy is executed for 500 episodes with learning switched off. Under Seq, the transfer of weights from Sarsa(0) to CEM is effected after 15,000 episodes of training.

Pass uniformly at random. While we cannot "dissect" Keepaway like we do our parameterized learning problem—in order to ascertain the quality of the representation employed—it is clear that state aliasing and generalization affect learning in Keepaway Pass to some degree. That Seq outperforms both Sarsa and CEM on this task suggests that Seq might be a useful algorithmic strategy to adopt in other realistic applications of RL.

While here we have applied VF and PS methods in sequence for learning Pass, in our next case study, we apply a VF method to learn Pass and a PS method to learn GetOpen in an interleaved manner. The next section is devoted to Case Study 2. Section 6.4 discusses related work, and Section 6.5 summarizes the chapter.

6.3 Case Study 2: Learning Multiple Task Components

Consider a team of robots that must learn to play soccer. Learning to play soccer can be framed quite elegantly as a multiagent RL problem, for example, using a construct such as a decentralized POMDP (DEC-POMDP) (Bernstein et al., 2002). However, general learning algorithms proven to work in DEC-POMDPs can scarcely be expected to cope in practice with the demands of a task as complex and large. Rather, any successful learning strategy would have to carefully exploit the inherent regularities in the domain. Towards expediting multiagent RL, a number of formal models have been proposed to exploit task-specific regularities such as coordination of actions (Guestrin et al., 2002), state abstraction (Ghavamzadeh et al., 2006), and information sharing (Rosin and Belew, 1995). While such formulations all pave the way towards learning increasingly complex behavior, they still assume that the task being considered is "monolithic": capable of being learned using a *single* learning algorithm. Yet complex tasks such as soccer comprise *multiple* distinct skills and behaviors, the *combination* of which determines the overall performance on the task. In order to surmount the challenge of learning to play soccer, surely it is necessary to appropriately decompose the task and tackle its components in specialized ways (Stone, 1998).

The possibility of applying different learning algorithms to various conjunctive components within a task has not received much attention from a theoretical standpoint; successful practical demonstrations of this strategy are also few. We identify this aspect of learning as a crucial direction for research: if a complex task naturally decomposes into qualitatively disparate components, can we successfully learn policies for these components in conjunction? In this section, we consider this question through a case study involving Keepaway (Stone et al., 2005), which was earlier described in Section 6.1. Keepaway is significantly more complex than synthetic, discrete tasks that have been used in the past for studying multiagent scenarios. For example, the Predator-Prey task (Benda et al., 1986) has been used in early studies investigating agent cooperation (Haynes et al., 1995); small board games such as Tic-Tac-Toe and Nim have pivoted studies involving agent competition (Rosin and Belew, 1995). In contrast with these tasks, Keepaway has a continuous, high-dimensional state space; noisy and asynchronous actions; and occluded, noisy perception.

Even if Keepaway is realistic in several respects, previous studies related to Keepaway have only considered learning the PASS behavior, assuming that GETOPEN is hand-coded and fixed (that is, the GETOPEN:HAND-CODED policy, specified in Algorithm 6.2, is followed). Indeed this was also our approach under Case Study 1. In this section, under Case Study 2, we extend learning in Keepaway from PASS to PASS+GETOPEN. Consequently Keepaway becomes an instance of a learning problem composed of highly interdependent behaviors executing simultaneously. Each player executes multiple behaviors (PASS and GETOPEN) that affect the outcome of its teammates' behaviors, and in the long run, also interact with one another. Such a scenario poses a significant challenge for designing a credit assignment scheme that both reflects the intended objectives in the underlying task and guides learning in a natural, incremental manner.

Although GETOPEN can be *described* accurately as a sequential decision making problem, the presence of multiple agents, their large action spaces, and the high frequency of decision making render GETOPEN ill-suited to be *learned* using model-based or value function-based methods. In response we devise a solution for learning GetOpen using policy search, which contrasts with the TD approach used for learning Pass. Results show that our learned GetOpen policy matches the best performing hand-coded policy for this task. Further experiments illustrate that learning these complementary behaviors results in a tight coupling between them, and indeed that Pass and GetOpen can be learned simultaneously. These results demonstrate the effectiveness of applying separate learning algorithms to distinct components of a significantly complex task. We provide detailed analyses to guide the design and evaluation of similar solutions in the future, and hope that this exploratory research will also motivate theoretical advances towards the methodology of combining separately learned behaviors. As here, such behaviors might be learned using very different learning algorithms.

The remainder of this section presents our learning framework (Section 6.3.1) and relevant experimental results (Section 6.3.2).

6.3.1 Learning Framework

Each of the 3 keepers must learn one Pass and one GetOpen policy: an array of choices therefore arises in deciding whether the keepers learn separate policies or learn policies in common. The total number of policies learned may range from 2 (1 Pass, 1 GetOpen) to 6 (3 Pass, 3 GetOpen). Different configurations have different advantages in terms of the size of the overall search space, communication constraints, and the ability to learn specialized behaviors. It falls beyond the scope of this work to systematically comb the space of possible solutions for learning Pass and GetOpen. As an exploratory study, our emphasis in this chapter is rather on verifying the *feasibility* of learning these behaviors, guided by intuition, trial and error. In the learning scheme we adopt, each keeper learns a unique Pass policy, while the three keepers share a common GetOpen policy. We proceed to describe the relevant learning schemes. As in Section 6.1, we furnish pseudo-code and parameter settings to ensure that the presentation is complete and our experiments are reproducible.

6.3.1.1 Learning PASS

For learning Pass, we apply the same algorithm and parameter values employed by Stone et al. (2005), under which each keeper uses Sarsa(0) to make TD learning updates. The reward for a keeper's action is the time elapsed until the next action is taken from the next state by that keeper (or the episode ends). Assuming that the keepers follow stationary GetOpen policies, this scheme seeks to directly maximize the episodic hold time by separately improving each keeper's Pass policy. A one-dimensional tile coding scheme is used for function approximation; tile widths are $3.0m$ along state variables corresponding to distances, and $10°$ along those representing angles. An ϵ-greedy behavior policy is employed, with the exploration parameter ϵ set to 0.01. The learning rate α is set to 0.125. As mentioned in Section 6.2.2.2, tile coding leads to slightly better results on Keepaway Pass than neural network-based function approximation.

6.3.1.2 Learning GETOPEN

Recall from Section 6.1.2 that the solution to be learned under Keepaway GetOpen is an evaluation function over its ten state variables. As described therein, a keeper executing GetOpen will apply this evaluation function to each of 25 regularly spaced points on the field and head towards the one with the highest evaluation. We desire an evaluation function that leads to maximization of the episodic hold time. Whereas TD learning is a natural choice for learning PASS, difficulties in exploiting the sequential nature of GETOPEN (as outlined in Section 6.1.3) make direct policy search a more promising alternative for tackling it. Thus, we represent the evaluation function as a parameterized function and search for parameter values that lead to the highest episodic hold time.

Our learned GETOPEN policy is implicitly represented through a neural network that computes a value for a target location, given the 10-dimensional feature vector. Note that unlike with PASS, these values do not have the same semantics as action values computed through TD learning; rather, they merely serve as action *preferences*, whose relative order determines which action is chosen. After manually experimenting with roughly 20 different network topologies, we find that the best results are achieved using a 10-5-5-1 network, with a total of 91 parameters (including biases at each hidden node). The parameters are initialized to values drawn uniformly at random from the range $[-0.5, 0.5]$; each hidden node implements the sigmoid function $f(x) = 1.7159 \cdot tanh(\frac{2}{3}x)$, as recommended by Haykin (1998, see pages 179–181).

A variety of policy search methods are applicable for optimizing the resulting 91-dimensional GETOPEN policy. We verify informally that methods such as hill climbing, genetic algorithms, and policy gradient methods all achieve qualitatively similar results. The experiments reported here are conducted using the cross-entropy method (de Boer et al., 2005), which evaluates a population of candidate solutions drawn from a distribution, and progressively refines the distribution based on a selection the fittest candidates. We use a population size of 20 drawn initially from $N(0, 1)^{91}$, picking the fittest five candidates after each evaluation of the population. Each keeper follows a fixed, stationary PASS policy across all evaluations in a generation; within each evaluation, all the keepers follow that same GETOPEN policy (the one being evaluated). The fitness function used is the average hold time over 125 episodes, which blunts the high variance in the lengths of Keepaway episodes.

6.3.1.3 Learning PASS+GETOPEN

Algorithm 6.4 outlines our method for learning PASS+GETOPEN. We bootstrap the learning process by optimizing a GETOPEN policy for a random PASS policy. The best GETOPEN policy found after two generations (a total of $2 \times 20 \times 125 = 5000$ episodes) is fixed, and followed while learning PASS using Sarsa(0) for the next 5000 episodes. The PASS policy is now frozen, and GETOPEN is improved once again. Thus, inside the outermost loop, either PASS or GETOPEN is fixed and stationary, while the other is improved, starting from its current value. Note that π_{PASS} and π_{GETOPEN} are still *executed* concurrently during each Keepaway episode when *learnPass*() and *learnGetOpen*() are called.

Whereas Algorithm 6.4 describes a general learning routine for each keeper to follow, in our specific implementation, the keepers execute the al-

Algorithm 6.4 Learning PASS+GETOPEN

```
Output: Policies $\pi_{\text{PASS}}$ and $\pi_{\text{GETOPEN}}$.

$\pi_{\text{PASS}} \leftarrow$ PASS:RANDOM.
$\pi_{\text{GETOPEN}} \leftarrow$ GETOPEN:RANDOM.
repeat
    $\pi_{\text{GETOPEN}} \leftarrow learnGetOpen(\pi_{\text{PASS}}, \pi_{\text{GETOPEN}})$.
    $\pi_{\text{PASS}} \leftarrow learnPass(\pi_{\text{PASS}}, \pi_{\text{GETOPEN}})$.
until convergence
Return $\pi_{\text{PASS}}, \pi_{\text{GETOPEN}}$.
```

gorithm in phase, and indeed share a common GetOpen policy (π_{GetOpen}). Also, we obtain slightly better performance in learning Pass+GetOpen by spending more episodes on learning GetOpen than on learning Pass (the next section provides details).

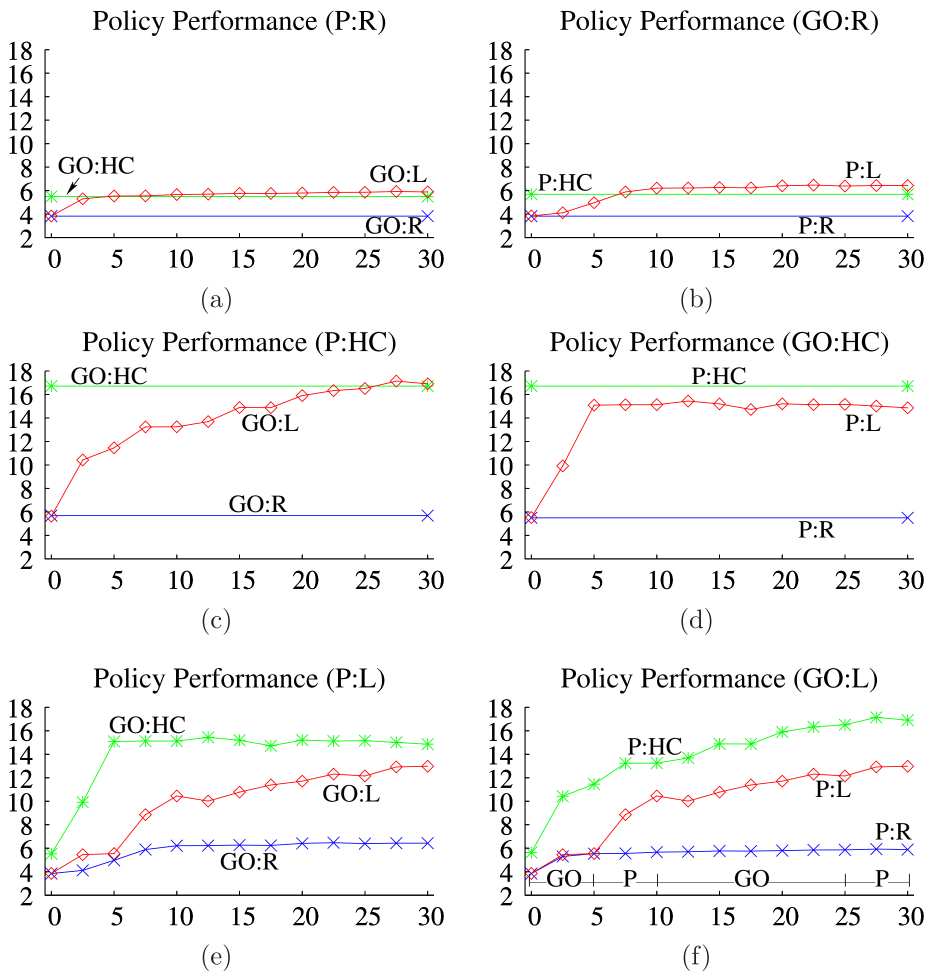
6.3.2 Results

In this section, we report the results of a systematic study pairing three Pass policies: Pass:Random, Pass:Hand-coded, and Pass:Learned; with three GetOpen policies: GetOpen:Random, GetOpen:Hand-coded, and GetOpen:Learned. For the sake of notational convenience, we use abbreviations: thus, for example, Pass:Random is denoted P:R, GetOpen: Learned is denoted GO:L, and their conjunction P:R-GO:L. Nine configurations arise in total. Figure 6.8 shows the performance of each Pass policy when paired with different GetOpen policies, and vice versa.[4] Policies in which both Pass and GetOpen are either random or hand-coded are static, while the others display learning.

6.3.2.1 Learning Performance

Figure 6.8(e) shows the performance of P:L. P:L-GO:HC corresponds to the experiment conducted by Stone et al. (2005), and we see similar results. After 30,000 episodes of training, the hold time achieved is about 14.9 seconds, which falls well short of the 16.7 seconds registered by the static P:HC-GO:HC policy (Figure 6.8(c)). Although P:L-GO:HC is trained in these experiments with a constant learning rate of $\alpha = 0.125$, we posit that anneal-

[4]Videos of policies are posted on the following web page: `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway-getopen/`.



Key: x axis: Training episodes / 1000 y axis: Hold time / s

Figure 6.8: Learning curves corresponding to conjunctions of various PASS and GETOPEN policies. Each curve represents an average of at least 20 independent trials. Each reported point corresponds to an evaluation (non-learning) for 500 episodes; points are reported every 2500 episodes. Note that each of the nine experiments appears once in the left column, where experiments are grouped by common PASS policies, and once in the right column, where they are grouped by GETOPEN.

ing α will improve its performance by avoiding the gradual dip in hold time we observe between episodes 12,500 and 30,000. In the absence of any guarantees about convergence to optimality, we consider the well-tuned P:HC-GO:HC to serve as a near-optimal benchmark for the learning methods. Interestingly, under the random GetOpen policy GO:R (Figure 6.8(b)), P:HC is overtaken by P:L after 30,000 episodes ($p < 0.0001$). This result highlights the ability of learning methods to adapt to different settings, for which hand-coded approaches would demand tedious manual attention.

Figure 6.8(f) confirms the viability of our policy search method for learning GetOpen, and its robustness in adapting to different Pass policies. Practical considerations force us to terminate experiments after 30,000 episodes of learning, which corresponds roughly to one day of real training time. After 30,000 episodes, P:HC-GO:L achieves a hold time of 16.9 seconds, which indeed exceeds the hold time of P:HC-GO:HC (Figure 6.8(c)). Yet despite running 20 independent trials of each variant, this result is not statistically significant. Thus, we may only conclude that when coupled with P:HC, learning GetOpen, a novel contribution of this work, matches the hand-coded GetOpen policy that has been used in all previous studies on the Keepaway task. This result also highlights that well-crafted state variables such as *congestion* and *predictedBallPos*, which are used by P:HC-GO:HC, are not necessary for describing good GetOpen behavior. Interestingly the hold time of P:HC-GO:L is significantly higher than that of P:L-GO:HC ($p < 0.001$). In other words, our GetOpen learning approach outperforms the previously studied Pass learning when each is paired with a hand-coded counterpart, underscoring the relevance of *learning* GetOpen.

An important result we observe from Figures 6.8(e) and 6.8(f) is that not only can Pass and GetOpen be learned when paired with static policies, they can indeed be learned in tandem. In our implementation of Algorithm 6.4, we achieve the best results by first learning GetOpen using policy search for 5000 episodes, followed by 5000 episodes of learning Pass using Sarsa(0). Subsequently we conduct six generations of learning GetOpen (episodes 10,000 to 25,000), followed by another 5000 episodes of Sarsa(0), as depicted along the x axis in Figure 6.8(f). The hold time of P-L:GO-L (13.0 seconds after 30,000 episodes) is significantly lower than those of P:L-GO:HC, P:HC-GO:L, and P:HC-GO:HC ($p < 0.001$), reflecting the additional challenges encountered while learning Pass and GetOpen simultaneously.

Indeed we notice several *negative* results with other variant methods for learning Pass+GetOpen. In one approach, we represent both Pass and GetOpen as parameterized policies and evolve their weights concurrently to maximize hold time. In another approach, GetOpen uses the value function being learned by Pass as the evaluation function for target points. In both these cases, the performance never rises significantly above random. We interpret the failure of these relatively natural solution strategies as an indication of the hardness of learning Pass+GetOpen. While the reported P:L-GO:L results confirm that Pass and GetOpen can indeed be learned in tandem (note that the learning curve in Figure 6.8(f) is still rising after 30,000 episodes), there appears to be significant room for improving this result. In Section 6.4, we consider multiple channels of related work that may apply.

6.3.2.2 Specialization of Learned Policies

We conduct a further experiment in order to ascertain the degree of specialization achieved by learned Pass and GetOpen policies; that is, whether it is beneficial to learn Pass specifically for a given GetOpen policy (and vice versa). In Table 6.3, we summarize the performances of learned Pass and GetOpen policies trained and tested with different counterparts. Each column corresponds to a test pairing. We notice that the best performing Pass policy for a given GetOpen policy is one that was trained with the same GetOpen policy (and vice versa); the maximal sample mean in each column in Table 6.3 coincides with the diagonal. It must be noted, however, that despite conducting at least 20 trials of each experiment, some comparisons are not statistically significant. A possible reason for this small number of inconclusive comparisons is the high variance caused by the stochasticity of the domain. Yet it is predominantly the case that learned behaviors adapt to work best alongside the counterpart behavior with which they are trained. Several practical problems demand specialized solutions for specific situations; by automatically gravitating towards tightly-coupled behaviors that maximize performance, learning can offer a significant advantage.

The results reported above conclude our technical description of Case Study 2. In summary, both the case studies presented in this chapter provide encouraging results for the strategy of combining different learning methods to combat the complexity of tasks. With the experimental validation that they have received, we hope that the ideas presented in the case studies will inspire further research on developing hybrid learning algorithms. In the next section, we discuss related work and possibilities for extending the work described in this chapter.

Table 6.3: Performance of learned Pass and GetOpen policies when trained and tested with different pairings. In the top table, Pass learned while paired with different GetOpen policies is tested with different GetOpen pairings. Each entry shows the mean hold time and one standard error of at least 20 independent runs, conducted for 500 episodes. Each *column* corresponds to a test GetOpen policy. The largest entry in each column is in boldface; entries in the same column are marked with "-" if not lower with statistical significance ($p < 0.05$). The cell GO:L-GO:L shows two entries: when the learned Pass policy is tested against the same ("s") learned GetOpen policy as used in training, and when tested against a different ("d") learned GetOpen policy. The bottom table is constructed similarly for GetOpen, and uses the same experiments as Pass for the cell P:L-P:L.

Pass:Learned

Train	**Test**		
	GO:R	GO:HC	GO:L
GO:R	**6.37**$\pm$.05	11.73$\pm$.25	10.54$\pm$.26
GO:HC	6.34$\pm$.06$^-$	**15.27**$\pm$.26	12.25$\pm$.32
GO:L	5.96$\pm$.07	13.39$\pm$.35	**13.08**$\pm$.26 (s) 12.32$\pm$.32 (d)$^-$

GetOpen:Learned

Train	**Test**		
	P:R	P:HC	P:L
P:R	**5.89**$\pm$.05	10.40$\pm$.39	11.15$\pm$.43
P:HC	5.48$\pm$.04	**16.89**$\pm$.39	12.99$\pm$.43$^-$
P:L	5.57$\pm$.06	11.78$\pm$.56	**13.08**$\pm$.26 (s) 12.32$\pm$.32 (d)$^-$

6.4 Related Work and Discussion

Below we review literature within the overall context of hybrid learning methods for sequential decision making, and also specifically discuss work that is related to the case studies presented in this chapter.

6.4.1 Hybrid Learning Methods

Moriarty et al. (1999, see page 271) apply a suite of Evolutionary Algorithms (EA) for Reinforcement Learning (EARL) to a simple grid-world MDP. They reflect on their work as follows (by "TD" they imply temporal difference learning methods):

> *"It is not useful to view the path towards practical RL systems as a choice between EA and TD methods. We have tried to highlight some of the strengths of the evolutionary approach, but we have also shown that EARL and TD, while complementary approaches, are by no means mutually exclusive. We have cited examples of successful EARL systems such as SAMUEL and ALECSYS that explicitly incorporate TD elements into their multi-level credit assignment methods. It is likely that many practical applications will depend on these kinds of multi-strategy approaches to machine learning."*

Of the multi-strategy approaches therein cited, SAMUEL (Grefenstette et al., 1990), an evolutionary algorithm, stores a reserve of individual state transitions, which is used to inform the genetic operations of specialization and deletion of genes. ALECSYS (Dorigo and Colombetti, 1998) is yet another evolutionary algorithm that allows a human programmer to specify problem subtasks. Whereas Moriarty et al. specifically comment on the combination of evolutionary algorithms with TD learning, their perspective is equally relevant to other possibilities for combining the strengths of various RL methods.

Actor-critic algorithms (Konda and Tsitsiklis, 2003; Bhatnagar et al., 2008), a relatively old conceptual framework within RL, can be construed as an instance of the strategy of integrating qualitatively different learning methods. Under actor-critic methods, the actor is akin to a PS algorithm, which directly optimizes policy parameters in order to reap high long-term reward. Whereas the PS algorithms presented in this thesis use a Monte Carlo estimate of the long-term return as the objective function to maximize, under typical actor-critic architectures, it is the critic's remit to provide the actor an evaluation of its current policy. The critic usually does so by maintaining a value function that tracks the actor's policy: thus, the critic is akin to a VF method. Evaluating the current policy based on the critic's stored value function, rather than through Monte Carlo estimates, has the advantage of a reduced variance. Actor-critic architectures are an important algorithmic class that this thesis does not evaluate. In Chapter 7, though, we formulate a course for future work, in which they are considered (also see Appendix C).

"Value and Policy Search" (VAPS) (Baird and Moore, 1999) is a group of methods that employ "general gradient descent" for RL. The gradient specified can vary from (1) a term based on the value function (which would lead to a TD-like algorithm) to (2) an aggregate quantity such as the cumulative reward (which would make the algorithm akin to a policy gradient method). An attractive feature of VAPS is that a smooth balance can be achieved between these two extremes by setting a parameter $\beta \in [0, 1]$. As a gradient descent algorithm, VAPS is guaranteed convergence even in POMDPs.

The famous application of RL to helicopter control (Ng et al., 2004) indeed involves an innovative combination of model learning with policy search. To arrive at a good control policy, first a model of the reward and transition dynamics is learned based on data logged during a human operator's control of the helicopter. Subsequently a parameterized policy is optimized using the PEGASUS algorithm (Ng and Jordan, 2000), with trajectories simulated and evaluated on the learned model.

6.4.2 Case Study 1

Recall that our Seq algorithm involves a transfer of weights from a VF to a PS method after the former method's performance begins to saturate. Guestrin et al. (2002) design a closely related scheme in a multiagent learning setting to transfer weights learned using LSPI to initialize a policy gradient method. In their experiments—on the SysAdmin (Guestrin et al., 2001b) and power grid (Schneider et al., 1999) tasks—the inherited policy is "softmax" over the learned action values. In our experiments, the PS method (CMA-ES or CEM) inherits a greedy policy.

In both our applications under Case Study 1 (parameterized learning problems from Chapter 3 and Keepaway Pass), the number of episodes after which the switch is made from the VF method to the PS method under Seq is set based on our own (human) judgment of an effective transfer point. In future work, we might attempt to determine this transfer point in an intelligent, automated manner. Indicators such as the Bellman error could be used to gauge whether the VF method has begun to plateau. Also, we note that in general, VF and PS methods could achieve their best performance under very different representations, and so, another consideration for future work is to enable a transfer of policies, rather than a transfer of weights, between the VF and PS methods constituting Seq. Relevant work in transfer learning (Taylor and Stone, 2007) addresses this issue. For example, Sarsa could learn using tile coding as a function approximator, and a PS method could employ a neural network-based representation of the policy to improve the policy learned by Sarsa. The transfer itself might be effected by using supervised learning to train the neural network based on samples from the policy learned by Sarsa.

6.4.3 Case Study 2

Multiple learning methods are used in the "layered learning" architecture applied by Stone (1998) to RoboCup 2D simulation soccer. These methods include neural networks for learning to intercept the ball, decision trees for evaluating passes, and TPOT-RL, a TD learning method, for high-level strategy learning. The idea of layered learning shares our motivation that different sub-problems in a complex multiagent learning problem can benefit from specialized solutions. Yet a key difference is that in Stone's architecture, skills learned using supervised learning are employed in higher-level sequential decision making, to which RL is then applied; in our work, the two learning problems we consider, PASS and GETOPEN, are themselves both sequential decision making problems. Also, in our learning method for PASS+GETOPEN, the PASS and GETOPEN behaviors are learned in an interleaved manner; under the layered learning paradigm, lower behaviors are frozen before higher ones are learned. In this respect, our interleaved learning approach better matches the "concurrent layered learning" idea proposed by Whiteson et al. (2005), under which learned behaviors are allowed to continually adapt. On a relatively simple implementation of the Keepaway task, Whiteson et al. (2005)

demonstrate that by being able to fine-tune behaviors (such as interception, passing, and getting open) towards each other, concurrent layered learning indeed improves upon the performance of traditional layered learning.

In their survey of the field, Panait and Luke (2005) divide cooperative multiagent learning into two broad categories: under *team learning*, a single learner develops the behavior for the entire team; under *concurrent learning*, each agent follows a separate learning processes. Interestingly our method for learning Pass+GetOpen occupies both categories: GetOpen uses team learning, while Pass uses concurrent learning. Further, these two processes are themselves interleaved.

The policy search approach we use with GetOpen is similar in spirit to the method used by Haynes et al. (1995) for evolving cooperative behavior among four predators that must collude in order to catch a prey. The predators share a common policy, represented as a strongly-typed LISP S-expression, in contrast with the neural representation we engage for computing a real-valued evaluation function. The Predator-Prey domain (Benda et al., 1986) that they use, which is small and discrete, is much simpler compared to Keepaway. Whereas we employ qualitatively different methods for learning Pass and GetOpen in a cooperative setting, Rosin and Belew (1995) consider evolving opponents in *competitive* scenarios using a genetic algorithm. On games such as Tic-Tac-Toe and Nim, they demonstrate that coupled fitness functions present opportunities for members of opposing populations to share evaluations, thereby expediting learning.

Several learning methods that have been applied to Keepaway Pass also find relevance in the context of GetOpen. For instance, Metzen et al. (2008) introduce EANT, a method to evolve *both* the structure and the weights of a neural network representing a policy for Pass. While we use a fixed neural network topology in this work for representing the GetOpen policy, EANT can potentially evolve topologies that yield higher performance. Keepaway can be extended to more keepers and takers: Taylor et al. (2007) study transferring knowledge obtained in simpler configurations to more complex ones. Although the experiments in this chapter all involve three keepers and two takers, a promising idea for future work is to implement GetOpen with more players, possibly incorporating the knowledge transfer methods of Taylor et al.

Iscen and Erogul (2008) consider applying TD learning to the behavior of the takers. The actions available to the takers are ball interception and player marking. Their experiments show that taker behavior can be successfully learned to compete against a wide array of keepers. Whereas Pass+GetOpen models cooperation, extending Keepaway to include taker behavior would also incorporate agent competition.

6.5 Summary

This chapter continues in the vein of Chapter 5, making contributions to learning methodology to address the practical needs of sequential decision making tasks. The chapter is organized around two case studies involving hybrid learning methods.

Under Case Study 1, we provide Seq, an algorithmic framework for combining the strengths of VF and PS methods. Seq is motivated by the observation that whereas VF methods are able to learn relatively quickly to reach their asymptotic performance, the performance thereby achieved on problems with deficient representations falls short of the performance PS methods can achieve on the same problems when provided enough training time. Seq combines the sample efficiency of VF methods with the robustness of PS methods towards imperfect representations: under Seq, a VF method is executed until it plateaus, and then the learned policy is transferred to the PS method for further improvement. Experimental results affirm that this strategy is effective both on the parameterized learning problems introduced in this thesis (in Chapter 3) and in the more complex task of Keepaway PASS.

Keepaway encapsulates the challenges faced in a large number of realistic domains: high-dimensional spaces, noisy perceptions and actions, and real-time constraints. The focus of Case Study 2 is the extension of learning within Keepaway from solely learning PASS, a turn-taking behavior, to also learning GETOPEN, which is executed by multiple players every cycle. PASS+GETOPEN is successfully learned through yet another scheme combining VF and PS methods. In particular we provide a policy search method for learning GETOPEN, which compares on par with the well-tuned hand-coded GETOPEN policy used in previous work, and indeed performs better when paired with a random PASS policy. Learning GETOPEN with a hand-coded PASS policy outperforms results from earlier studies in which PASS is learned and GETOPEN is hand-coded. Our algorithm for learning PASS and GETOPEN in an interleaved manner confirms the feasibility of learning them together, but also shows significant scope for improvement. We discuss several ideas from related work that may aid progress in this direction. Our experiments investigate the interdependence between learned PASS and GETOPEN policies, exposing their tightly-coupled nature. Our demonstration of the autonomous learning of a significant fraction of a complex task extends the reach of RL in practical applications. While this work showcases the richness of the PASS+GETOPEN learning problem, it also takes the important step of putting together distinct learning techniques that apply to sequential decision making, which we consider a crucial element in scaling RL to even more complex problems.

Chapters 5 and 6 have focused on specific methodological contributions inspired by the experimental and analytical work undertaken in chapters 3 and 4. The next chapter (Chapter 7) reflects on the lessons learned from these previous chapters, identifies a variety of avenues for future work, and concludes the dissertation.

Chapter 7

Conclusion and Future Work

In this final chapter, we reflect on the findings of the previous chapters. Section 7.1 presents a summary of the dissertation. In Section 7.2, we identify several directions for continuing the work presented in this dissertation. Section 7.3 provides the conclusion.

In this final chapter, we review the main topics covered in this dissertation. Section 7.1 summarizes the dissertation. In Section 7.2, we outline four broad questions that emerge from the work presented in the preceding chapters. With the hope that future work will address these questions, we discuss relevant perspectives, provide references, and identify promising directions. Section 7.3 concludes the dissertation.

7.1 Summary of Thesis

Finite MDPs have traditionally been at the heart of reasoning about sequential decision making tasks. The formulation of the RL problem in terms of finite MDPs paves they way for developing learning algorithms with the capacity to achieve optimality, to converge, and to be sample-efficient. Unfortunately, RL methods seldom enjoy these attributes when they get deployed in practice. One key reason for this disparity between "theory" and "practice" is state aliasing: the inability to unambiguously identify environmental states based on observations alone. The literature provides several principled methods to deal with state aliasing; however, most of these methods do not scale in practice to tasks with more than a few hundreds of states. The large, possibly continuous state spaces encountered in real-world tasks are a second challenge to the assumption that tasks can be treated as finite MDPs. In most practical tasks, there is a need to generalize over the state-action space. Although some formal assurances can be provided for learning with generalization (such as guarantees of convergence), it remains that fundamentally, the performance that can be achieved is limited by the generalization scheme used.

This dissertation takes the philosophical position that whereas we might mitigate the adverse consequences of state aliasing and generalization, it is impractical to assume that we can eliminate them. As violations of the finite-MDP assumption, these factors lead to imperfect representations for learning. As a response to this observation, this dissertation considers the question of designing learning methods specifically to work with imperfect representations.

To begin, in chapters 1 and 2, we argued that even if a learning method is paired with an imperfect representation, it is possible to precisely define an objective for the learning method to achieve. We defined the objective to be that of learning a policy with relatively high expected long-term reward, while using a minimal number of samples. While such a goal appears to be a very natural one, note that the existing literature only provides us the means to achieve it when working with perfect representations. How might we achieve the same goal when working with imperfect representations? This question, which strikes at the heart of developing practically-effective RL algorithms, does not have readily forthcoming answers. It is the question that motivates this dissertation. The dissertation makes four contributions towards developing learning methods that can work successfully with imperfect representations.

1. Experimental comparison of learning methods. First, in Chapter 3, we introduced "parameterized learning problems", a novel experimental methodology facilitating the systematic control of representational aspects such as state aliasing and generalization. We applied this methodology to compare two qualitatively distinct classes of RL methods: on-line value function-based (VF) methods, and policy search (PS) methods. After extensive comparisons of various methods within each class, we picked Sarsa as a representative method from the VF class, and CMA-ES from PS. Comparing Sarsa and CMA-ES, we found that the former enjoys superior sample-efficiency, and also better asymptotic performance when the learner is provided an expressive representation. On the other hand, CMA-ES is significantly more robust to severely deficient representations. Both methods suffer noticeably when state noise is added; the order between the methods is additionally determined by the width of generalization.

2. Limits of representation. Following the results of our experimental study, we conducted a deeper analysis of the reasons VF methods perform relatively poorly with inexpressive generalization schemes. Specifically, in Chapter 4, we examined this question in the context of Tetris, the popular video game. With a linear representation scheme introduced by Bertsekas and Tsitsiklis (1996), we tested the results of approximate policy iteration on this task. We designed a hand-coded policy that clears roughly 1,000 lines per episode, and used Monte Carlo simulation to estimate the best approximations of its value function under three natural definitions of approximation error. We found that none of the resulting approximate value functions induces a policy that clears more than 70 lines per episode, indicating that approximate policy improvement with this commonly-used linear architecture results in a performance decrease. We surmise it is likely that in several practical tasks, the failure of VF methods can be attributed to a similar phenomenon.

3. Subset selection and efficient policy search. The third major contribution of this dissertation is a formal study of the "subset selection" problem in multi-armed bandits. This problem, which finds application in numerous areas, is particularly relevant to this dissertation through its connection with the selection routines of ranking-based PS methods such as CMA-ES and CEM. In Chapter 5, we formalized subset selection and devised algorithms under three operational settings: probably approximately correct (PAC) selection, simple regret (SR) minimization, and cumulative regret (CR) minimization. Under each of these settings, we generalized previous work devoted to identifying just the single best arm in an n-armed bandit. Our most novel results are under the PAC framework, where our high-probability sample complexity bounds improve upon the best existing ones for the single-arm case. Additionally we showed that our elimination algorithm under the SR setting indeed improves the performance of policy search methods such as CMA-ES and CEM.

4. Hybrid learning methods. The fourth and final contribution of this thesis is in the form of two case studies, which investigate techniques for combining the strengths of different classes of learning methods. These case studies were presented in Chapter 6. Under Case Study 1, we showed that the strategy of applying VF and PS methods in sequence—initially VF for some duration, followed by a transfer of the learned weights to be refined using PS—indeed inherits the strengths of its VF and PS constituents, and is often more successful than either of them taken in isolation. We verified these results both on the suite of parameterized learning problems introduced in this thesis, and on the more complex and realistic RL benchmark task of robot soccer Keepaway. Case Study 2 undertook a deeper investigation involving the Keepaway task. Whereas Case Study 1, and indeed most previous work on Keepaway, limit learning in the task to an isolated, infrequent decision that amounts to a turn-taking behavior ("PASS" behavior), under Case Study 2, we expanded the agents' learning capability to include a much more frequent action (moving without the ball, or "GETOPEN" behavior). We introduced a policy search method for learning GETOPEN; results established the success of learning GETOPEN both separately and together with PASS. Our demonstrations of successfully learning complex tasks by applying hybrid learning methods are a step towards scaling RL to even more complex problems.

We hope that the various contributions of this dissertation, described above, will motivate and serve future research on several important topics related to sequential decision making in practice. In the next section, we present a number of ideas for future work to examine.

7.2 Future Work

In this section, we devote some attention to the broad questions raised, but not fully answered, in chapters 3, 4, 5, and 6.

7.2.1 Inductive Bias of Reinforcement Learning Methods

Inductive bias is a central concept in machine learning. In the context of supervised learning, Mitchell (1980, see page 185) defines inductive bias as *"... any basis for choosing one generalization over another, other than strict consistency with the observed training instances."* In rough terms, we may construe the inductive bias of a method as the assumptions it makes in order to infer a *general* pattern based on the *specific* instances that the data provide. Naturally the concept of inductive bias also applies to RL. However, whereas it is possible to formally define inductive bias in settings such as supervised learning (Haussler, 1988; Bishop, 2006, see Section 3.2), the complex nature of the RL problem is less easy to negotiate. How might we proceed?

For a start, we might consider the characterization given by Cobb (1992), under which the inductive bias in a reinforcement learner is further divided into "language" (or representational) and "procedural" biases (Utgoff (1986) enumerates similar types of biases under inductive concept learning). The division corresponds closely with the distinction this dissertation makes between "representations" and "learning methods". In our experiments in Chapter 3, we enforced that VF and PS methods shared the same representation; this approach facilitated a direct comparison between the procedural components characterizing VF and PS methods.

Unfortunately it is not possible to standardize representations across all the different classes of RL methods. For example, a model-based method fundamentally requires that a model (a mapping from state-action pairs to distributions over states and rewards) be represented. Likewise, policy gradient algorithms typically require the representation to provide an analytically differentiable policy. In the absence of a common representation for different

RL methods, how might we compare their procedural biases?

As an added complication in defining inductive bias, consider yet another distinguishing trait of RL. Whereas in the supervised learning scenario, a learning algorithm is provided a set of training examples as its input, under RL, the learning agent is itself the entity in charge of exploring the world and gathering experience. How does exploration factor into inductive bias?

Even if answers to these questions are not forthcoming at present, we believe that the practice of RL can greatly benefit from a more formal characterization of the relationships between learning methods, representations, and tasks—as encapsulated within the notion of inductive bias. To motivate research along this direction, we present a qualitative argument for ordering different classes of RL methods based on their "sample-sensitivity", a term we define to loosely imply the extent to which individual samples influence the outcome of a method. We hypothesize that sample-sensitivity might correspond closely with the procedural bias of different RL methods. Our speculative essay on this hypothesis is presented in Appendix D.

7.2.2 Empirical Analysis of Value Function-based Methods

In Chapter 4, we adopted an experimental approach to investigate the reasons VF methods performed significantly worse than PS methods on the Tetris task. Through a carefully-engineered Monte Carlo simulation, we established that approximate policy iteration (using the linear representation of Bertsekas and Tsitsiklis (1996)) could, in fact, lead to a *worsening* of the policy in Tetris (under three separate weighting distributions for the error term). We believe that analytical studies of this nature are essential for understanding the strengths and weakness of VF methods when they get applied in complex sequential decision making problems.

If working with a tabular representation, we have a clear theoretical understanding of why a VF method should find an optimal policy (Bellman, 1957). Unfortunately the same cannot be said of learning with imperfect representations. Unlike PS methods, which directly traverse the space of policies, VF methods learn value functions as intermediate structures, and only derive policies indirectly. If a VF method fails on some practical application, it could be due to several reasons, including deficient features, state aliasing, inappropriate learning rate, and insufficient exploration. By and large, the literature does not describe instances where VF methods have failed, and provide explanations for such failure in terms of the elements listed above. We believe that studies similar to the one we undertook in Chapter 4 could begin contributing useful insights for practitioners of RL. To test the capability of a representation to support value function-based learning on some task, one could employ approximate policy improvement (approximate policy evaluation followed by greedy policy improvement) as a diagnostic test, as we did under Tetris.

Analytical studies explaining the *successes* of VF methods could also be informative. A sizeable number of the popular successes of RL have involved the application of VF methods. Application domains include backgammon (Tesauro, 1992), elevator dispatching (Crites and Barto, 1996), robot soccer Keepaway (Stone et al., 2005), optimized trade execution (Nevmyvaka et al., 2006), blimp control (Rottmann et al., 2007), 9×9 Go (Silver et al., 2007), and computer memory scheduling (İpek et al., 2008) (Table 1.1 lists several others). How does the best policy learned in these tasks compare with the optimal policy, or some approximation thereof? What is the error in approximating of the value function? How is the error distributed over the state-action space? We admit that Tetris has several convenient attributes (such as a discrete state space and a concise forward model) for answering such questions; ingenious experimental techniques might be required in more complex RL tasks.

7.2.3 Efficient Exploration for Policy Search

Our work in Chapter 5, on subset selection and its role in policy search, raises many interesting possibilities for future work. Some relevant problems were discussed in Section 5.7; below we present some broader questions.

Our work generalizes existing formulations under the PAC, SR, and CR settings (from selecting one arm to selecting a subset of arms). From the point of view of the bandits literature, it could be fruitful to explore whether our formulations can be generalized further. Also to examine are the fundamental relationships between the PAC, SR, and CR settings themselves. Apart from the improved bounds it delivers, our LUCB algorithm under the PAC setting is remarkable for its similarity to the UCB algorithm under the CR setting (Auer et al., 2002a). How does a simple switch in the directions of confidence bounds effect a change from "only-explore" to "explore-exploit"?

In terms of the application of our work to policy search, it is worth reiterating that evolutionary and other algorithms for stochastic optimization implement varying selection strategies: subset selection is only one among these strategies. In particular, proportionate selection (selecting candidates with a probability proportional to their fitness) and tournament selection (selecting candidates based on "head-to-head" comparisons) are other common selection mechanisms (Miller and Goldberg, 1996). Also, while our work treats fitness as an independent attribute of each individual in the population, in some cases, fitness could additionally depend on individuals' likeness to other members of the population (Stanley, 2004). The evolutionary computation community has long wrestled with the issue of noise in fitness evaluations (Arnold, 2001): mapping relevant problems to the framework of multi-armed bandits could result in both theoretical and practical advances.

7.2.4 Principles for Integrating Learning Methods

Over the years, RL research has developed several distinct paradigms for learning (we provide a brief survey of five different classes of RL algorithms in Appendix C). However, there has not been much research on mechanisms for combining these methods in effective ways. Surely this aspect demands greater attention if we aim to scale RL to more complex tasks. Our efforts in Chapter 6 are a step in this direction. Recall that we presented two case studies involving hybrid learning architectures, in which VF and PS methods were combined for learning complex tasks. In the first case study, we applied these methods in sequence; in the second, they were applied to qualitatively distinct components of a complex subtask of robot soccer. We believe that a significant amount of work needs to be done towards identifying the principles behind tying together different learning methods.

Neuroscience provides evidence of multiple control modes working in tandem in animal brains (Niv, 2009, see Section 2.3). These mechanisms include goal-directed, habitual, episodic, and Pavlovian modes. Dayan (2008) notes that not only do these different modes find application in different tasks in animal behavior, often they compete on the same task. As an example of interaction between different control mechanisms, Dayan cites the architecture of Deep Blue (Campbell et al., 2002), the successful Chess program. The success of the program owes to the adoption of qualitatively distinct control regimes in different parts of the state space. In particular, early-game and end-game moves are drawn from fixed databases, while large portions of middle game are tackled by forward modeling and the application of a well-tuned evaluation function. The mid-game search, which is perhaps the most crucial component, benefits from the use of specialized hardware and massive parallelization. The success of multi-strategy approaches in tackling a complex game like Chess serves as an incentive for developing similar schemes within the context of RL.

The interaction between evolution and learning (Ackley and Littman, 1992) is yet another biological phenomenon that bears relevance to the design of hybrid learning methods. Evolution, even if it happens at a much slower rate than learning, can ring in definitive adaptations in species. The NEAT+Q algorithm proposed by Whiteson and Stone (2006a) is a computational analogue of this phenomenon. The outer loop in this "two timescale" algorithm corresponds to running NEAT, an evolutionary algorithm that searches the space of topologies for neural networks. Within the inner loop, each individual individual in the population (a neural network) is trained using Q-learning. The authors show that NEAT+Q is successful on the Mountain Car task, which has traditionally been problematic to learn with VF methods that use neural networks for function approximation. NEAT+Q also performs well on a job scheduling task.

To the extent that multi-strategy approaches can cope with greater complexity in tasks, it also becomes difficult to learn from experience under such approaches. Identifying general principles for the development hybrid learning methods is an important challenge for future work.

7.3 Conclusion

This dissertation is founded by the observation that whereas the "theory" of sequential decision making is largely dependent on the assumption of perfect representations (enumerable state-action spaces), the "practice" of sequential decision making almost always has to cope with imperfect representations. This dissertation calls for a "theory of practice": what principles might guide the design of learning methods that can work effectively with imperfect representations? It is impossible—at any rate, in the span of one Ph.D. dissertation!—to provide complete answers to a question so enormous, even if basic. Yet, this dissertation makes progress through philosophical, experimental, analytical, and theoretical contributions to various elements involved with learning in the presence of imperfect representations.

We look forward to future research to advance both the individual contributions of this thesis, and the overarching ideas behind it. We view this dissertation as a purposeful step towards developing more robust and practically-effective learning methods for sequential decision making.

Appendices

Appendix A

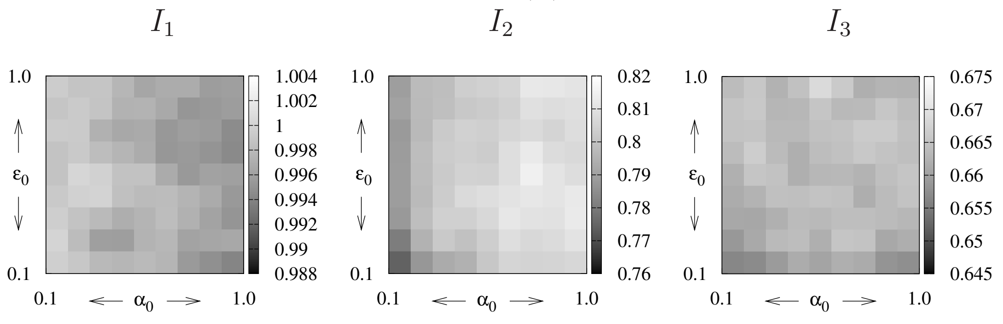
Additional Experiments from Chapter 3

This appendix provides additional sets of graphs pertaining to experiments in Chapter 3.

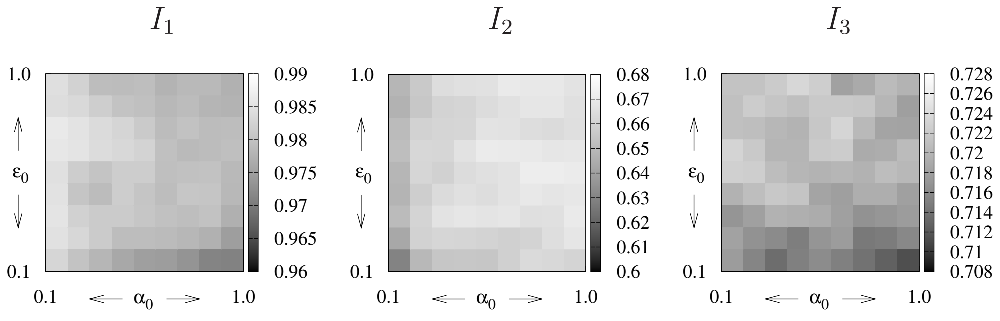
A.1 Effect of α_0 and ϵ_0 on Methods in VF

The plots below show the effect of the initial learning rate α_0 and exploration rate ϵ_0 on the learned performance of different methods in VF. Intensity ranges are indicated to the right of each plot. Under I_1, all the methods use $\theta_0 = 10$; under I_2, they use $\theta_0 = 0.5$; under I_3, they use $\theta_0 = 5$. We observe that under instance I_1, Sarsa(0), Q-learning(0), and ExpSarsa(0) all achieve normalized performance values close to 1. Under I_2, ExpSarsa(0) and ExpSarsa(1) perform best at low values of α_0 and ϵ_0.

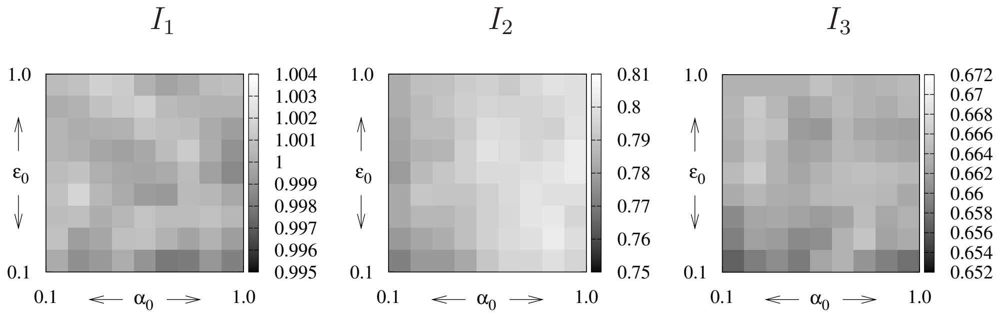
Sarsa(0)



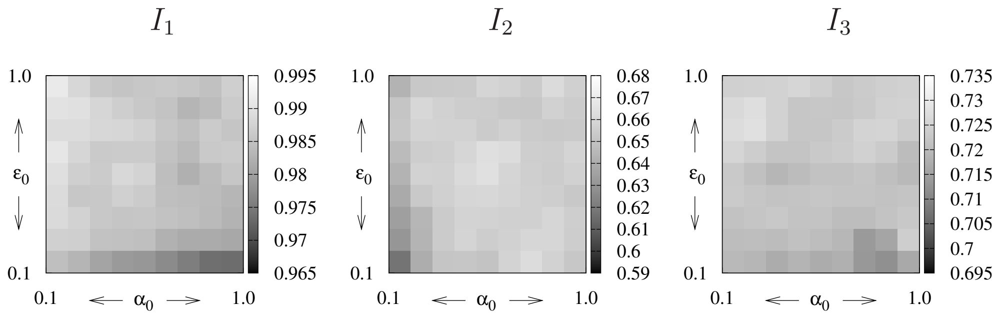
Sarsa(1)



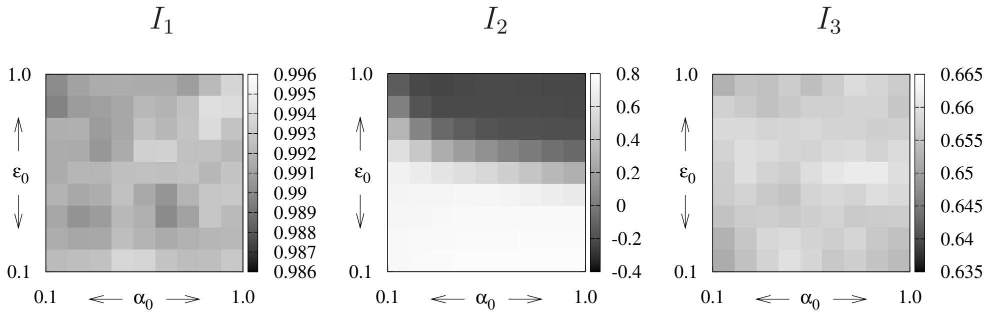
Q-learning(0)



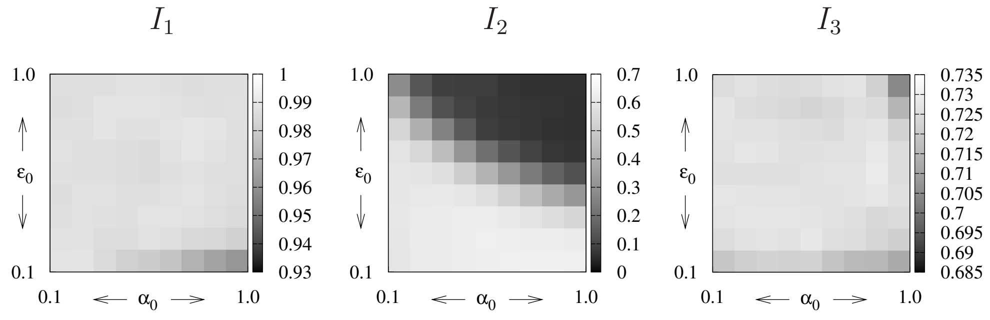
Q-learning(1)



ExpSarsa(0)



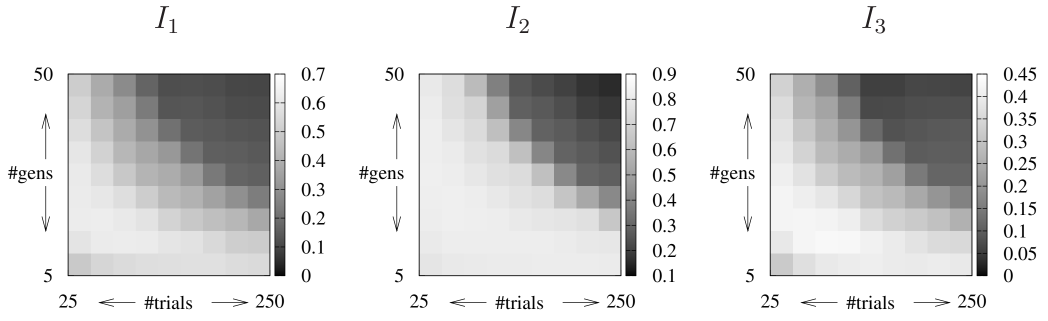
ExpSarsa(1)



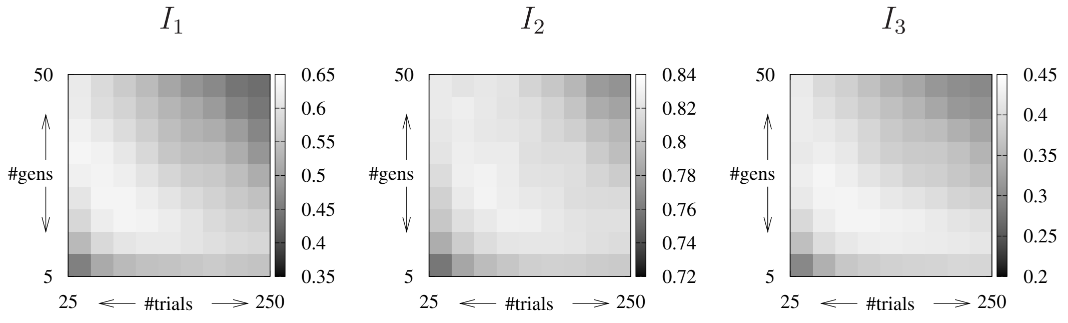
A.2 Effect of $\#trials$ and $\#gens$ on Methods in PS

The plots below show the effect of $\#trials$ and $\#gens$ on the performance of different policy search methods. Intensity ranges are indicated to the right of each plot. Under RWG, only $\#trials$ is varied ($\#gens = \frac{50{,}000}{\#trials}$); the mean performance is plotted with one standard error. The methods all show noticeable variance in performance over the ranges plotted, underscoring the need for careful tuning. For all methods, the highest performance is under problem instance I_2. We see that CMA-ES performs better on average, over the parameter ranges plotted, than both CEM and GA.

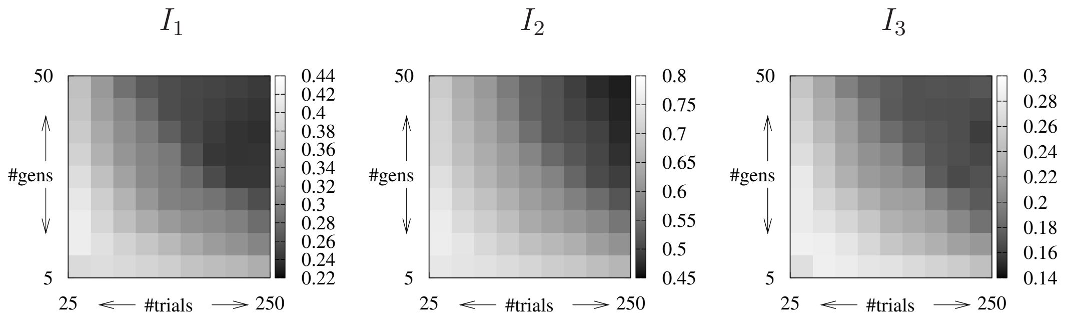
CEM



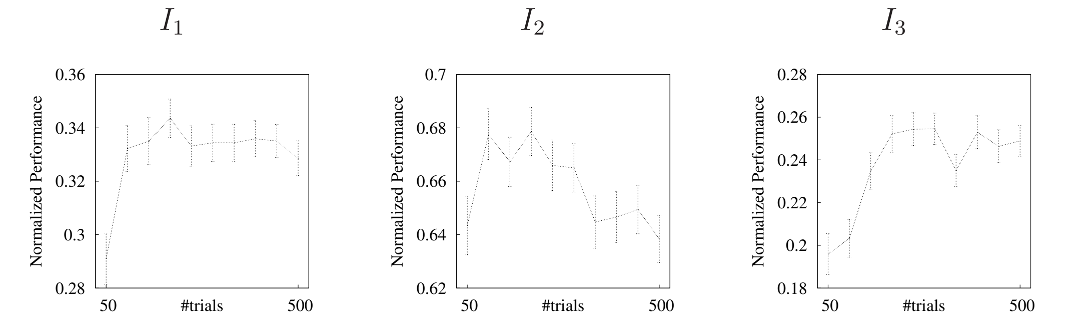
CMAES


I_1
I_2
I_3
50
5
#gens
25
#trials
250
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.84
0.82
0.8
0.78
0.76
0.74
0.72
0.45
0.4
0.35
0.3
0.25
0.2

GA


I_1
I_2
I_3
50
5
#gens
25
#trials
250
0.44
0.42
0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.3
0.28
0.26
0.24
0.22
0.2
0.18
0.16
0.14

RWG


I_1
I_2
I_3
Normalized Performance
#trials
50
500
0.36
0.34
0.32
0.3
0.28
0.7
0.68
0.66
0.64
0.62
0.28
0.26
0.24
0.22
0.2
0.18

Appendix B

Derivations from Chapter 5

This appendix provides derivations for results used in Chapter 5.

B.1 Relationship between H_1 and $H_2(m)$

We prove the statement of (5.5); that is:

$$H_2(m) \leq H_1 \leq \left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k} \right) H_2(m).$$

Inequalities relating the various Δ terms all follow from (5.1).

1. First we prove the left inequality. Let $j^* = \text{argmax}_{j \in Bot} \frac{j-m+1}{\Delta^2_{m,j}}$, with ties broken arbitrarily. Then,

$$\begin{aligned} H_2(m) &= \frac{j^* - m + 1}{\Delta^2_{m,j^*}} = \frac{1}{\Delta^2_{m,j^*}} + \sum_{k=m+1}^{j^*} \frac{1}{\Delta^2_{m,j^*}} \leq \frac{1}{\Delta^2_{m,m+1}} + \sum_{k=m+1}^{j^*} \frac{1}{\Delta^2_{m,k}} \\ &= \sum_{a=m}^{j^*} \frac{1}{\Delta^2_a} \leq \sum_{a=1}^{n} \frac{1}{\Delta^2_a} = H_1. \end{aligned}$$

When $m = 1$, we obtain $H_2(m) = H_1$ if $p_2 = p_3 = \cdots = p_n$, which yields $\Delta_1 = \Delta_2 = \cdots = \Delta_n$. For $m \geq 2$, H_1 exceeds $H_2(m)$ by at least $\sum_{i=1}^{m-1} \frac{1}{\Delta^2_i}$.

2. The right inequality is proven as follows.

$$\begin{aligned}
H_1 &= \sum_{a=1}^{n} \frac{1}{\Delta_a^2} = \sum_{i=1}^{m} \frac{1}{\Delta_i^2} + \sum_{j=m+1}^{n} \frac{1}{\Delta_j^2} \\
&\leq \sum_{i=1}^{m} \frac{1}{\Delta_m^2} + \sum_{j=m+1}^{n} \frac{1}{j-m+1} \max_{l \in Bot} \frac{l-m+1}{\Delta_l^2} \\
&= \left(\frac{m}{2}\right)\left(\frac{2}{\Delta_{m+1}^2}\right) + \sum_{k=2}^{n-m+1} \frac{1}{k} \max_{l \in Bot} \frac{l-m+1}{\Delta_l^2} \\
&\leq \left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right) \max_{l \in Bot} \frac{l-m+1}{\Delta_l^2} \\
&= \left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right) H_2(m).
\end{aligned}$$

Suppose $p_1 = p_2 = \ldots p_m$, and $\forall j \in Bot : p_j = p_m - c\sqrt{j-m+1}$, for a suitably small constant c; then we get $H_1 = \left(\frac{m}{2} + \sum_{k=2}^{n-m+1} \frac{1}{k}\right) H_2(m)$.

B.2 Derivation in Lemma 5.9

$$\begin{aligned}
&\sum_{u=4u_1^*(a,t)+1}^{\infty} \exp\left(-2\Delta_a^2 \left(\sqrt{u} - \sqrt{u_1^*(a,t)}\right)^2\right) \\
\leq\ & \int_{x_1=4u_1^*(a,t)}^{\infty} \exp\left(-2\Delta_a^2 \left(\sqrt{x_1} - \sqrt{u_1^*(a,t)}\right)^2\right) dx_1 \\
=\ & 2\int_{x_2=\sqrt{u_1^*(a,t)}}^{\infty} x_2 \exp\left(-2\Delta_a^2 x_2^2\right) dx_2 \\
& +2\sqrt{u_1^*(a,t)} \int_{x_2=\sqrt{u_1^*(a,t)}}^{\infty} \exp\left(-2\Delta_a^2 x_2^2\right) dx_2 \\
=\ & \frac{1}{2\Delta_a^2} \int_{x_3=2\Delta^2 u_1^*(a,t)}^{\infty} \exp\left(-x_3\right) dx_3
\end{aligned}$$

$$
\begin{aligned}
&+\frac{\sqrt{2\pi u_1^*(a,t)}}{\Delta_a}\int_{x_4=2\Delta_a\sqrt{u_1^*(a,t)}}^{\infty}\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x_4^2}{2}\right)dx_4\\
\leq\ & \frac{1}{2\Delta_a^2}\exp\left(-2\Delta_a^2u_1^*(a,t)\right)+\frac{1}{4\Delta_a^2}\exp\left(-2\Delta_a^2u_1^*(a,t)\right)\\
=\ & \frac{3}{4\Delta_a^2}\exp\left(-2\Delta_a^2\left\lceil\frac{1}{2\Delta_a^2}\ln\left(\frac{k_1nt^4}{\delta}\right)\right\rceil\right)\\
\leq\ & \frac{3\delta}{4\Delta_a^2k_1nt^4}.
\end{aligned}
$$

B.3 Calculation in Lemma 5.10

Let $T = CH_1^{\epsilon/2}\ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right)$, where $C \geq 146$. Then:

$$
\begin{aligned}
&2+8\sum_{a\in Arms}u_1^*(a,T)\\
=\ & 2+8\sum_{a\in Arms}\left\lceil\frac{1}{2\left[\Delta_a|\frac{\epsilon}{2}\right]^2}\ln\left(\frac{k_1nT^4}{\delta}\right)\right\rceil\\
\leq\ & 2+8n+4H_1^{\epsilon/2}ln\left(\frac{k_1nT^4}{\delta}\right)\\
\leq\ & (10+4\ln(k_1))\,H_1^{\epsilon/2}+4H_1^{\epsilon/2}ln\left(\frac{n}{\delta}\right)+16H_1^{\epsilon/2}\ln(T)\\
<\ & 14H_1^{\epsilon/2}+4H_1^{\epsilon/2}ln\left(\frac{n}{\delta}\right)+16H_1^{\epsilon/2}\left(\ln(C)+2\ln\left(H_1^{\epsilon/2}\right)+\ln\left(\frac{n}{\delta}\right)\right)\\
=\ & 14H_1^{\epsilon/2}+20H_1^{\epsilon/2}ln\left(\frac{n}{\delta}\right)+32H_1^{\epsilon/2}\ln\left(H_1^{\epsilon/2}\right)+16H_1^{\epsilon/2}\ln(C)\\
\leq\ & (66+16\ln(C))\,H_1^{\epsilon/2}\ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\\
<\ & CH_1^{\epsilon/2}\ln\left(\frac{H_1^{\epsilon/2}}{\delta}\right)\\
=\ & T.
\end{aligned}
$$

Appendix C

Classes of Learning Methods: Brief Survey

In this appendix, we provide a brief survey of different classes of RL methods. We identify five classes of learning methods that are qualitatively distinct, and together account for a large number of the standard algorithms for sequential decision making. Yet, our list is not exhaustive, as methods from these classes can be innovatively combined to yield new methods. The five classes we consider (in appendices C.1 through C.5) are listed below.

C.1:	Model-free on-line value function-based (VF) methods
C.2:	Model-based and batch (MB) methods
C.3:	Policy gradient (PG) methods
C.4:	Actor-critic (AC) methods
C.5:	Policy search (PS) methods

We find the order above the most naturally suited to the purpose of exposition. However, this order is different from the one hypothesized in Appendix D (page 288) to be consistent with the procedural bias of these classes of methods.

In our survey, we lay particular emphasis on literature that addresses the effects of state aliasing and generalization on different learning methods. Even so, our survey remains necessarily brief. For more in-depth descriptions of RL algorithms, we refer the reader to the excellent surveys undertaken by Geist and Pietquin (2010) and Szepesvári (2010). In addition the textbook by

Sutton and Barto (1998) provides detailed descriptions of many fundamental RL methods.

This appendix may be read independently from the main chapters in the dissertation. Some portions from this appendix appear as fragments in Chapter 3.

C.1 Model-free On-line Value Function-based Methods

Model-free value function-based (VF) methods are conceptually simple RL methods. In these methods, the basic idea is to maintain an approximate action value function Q_t, and progressively refine it as new transition samples become available. In particular each transition sample (s_t, a_t, r_t, s_{t+1}) is used to update Q_t through a temporal difference (TD) update rule. The update is meant to minimize a temporal difference error TD_Error_t, and typically assumes the form:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t TD\_Error_t,$$

where α_t is a learning rate. TD_Error_t essentially captures the difference between the current estimate of the action value, $Q_t(s_t, a_t)$, and a potentially better estimate that relies on having observed r_t and s_{t+1}. Evidence gathered by neuroscientists suggests that TD-like learning mechanisms operate in the human brain (Seymour et al., 2004).

The precise definition of TD_Error_t gives rise to different instances of learning methods. For example, under Q-learning: $TD\_Error_t \overset{\text{def}}{=} r_{t+1} + \gamma \max_{a \in A} Q_t(s_{t+1}, a) - Q_t(s_t, a_t)$. Under Sarsa(0), we have: $TD\_Error_t \overset{\text{def}}{=} r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$ (computed after action a_{t+1} is selected). For finite

MDPs, in which Q_t can be maintained in a table that enumerates all state-action pairs, it has be shown that both Q-learning (Watkins and Dayan, 1992) and Sarsa(0) (Singh et al., 2000) can be made to converge to the optimal action value function Q^*, from which an optimal policy π^* can be obtained as $\pi(s) = \operatorname{argmax}_{a \in A} Q^*(s, a), \forall s \in S$.

The only data structure a VF method needs to keep in memory is the action value function Q_t, whose size remains constant. We consider this characteristic the defining aspect of VF methods, which are model-free. Model-based and batch methods, described in Section C.2, sometimes also compute value functions, but value functions are not the sole data structure they store in memory. It is typical for such methods to store an entire model of the transition and rewards dynamics, or a batch of previously collected samples. Consequently the learning updates performed by model-based and batch methods tend to be more computationally intensive than those made by VF methods.

VF methods are simple to describe and easy to implement. However, they encounter difficulties as soon as Q_t is no longer an exact tabular representation, but is a "compact" representation incorporating some form of generalization. The generalization (or *function approximation*) architecture most commonly considered for theoretical analysis involves representing Q_t as a linear combination of basis functions $\boldsymbol{\phi}$: that is, $Q_t = \mathbf{w}_t^T \boldsymbol{\phi}$. Here the parameters $\mathbf{w}_t$ are updated based on experience.

Whereas linear architectures, and in particular, sparse representations such as tile coding (Sutton, 1996), have found some success in practice with VF methods, it remains that in general, approximate architectures are incapable of representing the optimal action value function Q^*. With architectures capable of representing Q^*, it is still not guaranteed that an optimal vector of parameters $\mathbf{w}^*$ will be found, as *intermediate* value functions encountered during learning might prove problematic for function approximation (Boyan and Moore, 1995). Further, even if a value function is approximated relatively well (in terms of some suitably defined error), greedy action selection might yet pick suboptimal actions in regions of inaccurate approximation (Koller and Parr, 2000; Baxter and Bartlett, 2001; Kalyanakrishnan and Stone, 2007). Chapter 4 in this dissertation demonstrates the occurrence of such a phenomenon in Tetris, the popular video game.

A bulk of the research in RL with linear function approximation has been in the context of prediction: estimating the value function of a fixed policy (without policy improvement). An early result due to Sutton (1988) establishes that TD(0) with linear function approximation converges when the features used are linearly independent. Dayan (1992), and Dayan and Sejnowski (1994), extend this result to TD(λ), $\forall \lambda \in [0, 1]$, while Tsitsiklis and Van Roy (1997) show convergence for the more realistic case of infinite state spaces and linearly dependent features. Although most results for the convergence of linear TD learning are for estimating values of the policy that is used to gather experiences, the more general (and useful) case of off-policy learning has also been addressed (Precup et al., 2001; Sutton et al., 2009).

Residual gradient algorithms (Williams and Baird, III, 1994; Baird, 1995) are VF methods specifically designed for the function approximation scenario. These algorithms are distinguished from other TD methods in the objective function they seek to minimize. Whereas methods such as TD(λ) seek convergence to “the least squares fixed point approximation”, residual (gradient) methods minimize the “squared Bellman residual” (Lagoudakis and Parr, 2003, see Section 5). This alternative formulation imparts residual gradient algorithms stability while performing off-policy bootstrapping, which can potentially cause conventional TD methods to diverge (Baird, 1995; Sutton and Barto, 1998). In a recent comparative study of residual gradient algorithms and TD learning algorithms, Li (2008) finds that while the former tend to result in smaller TD errors, the latter make better predictions of value.

The problems in learning approximate value functions on-line primarily arise due to the nonstationarity and bias *in the targets* provided to the function approximator (Thrun and Schwartz, 1993). The best theoretical guarantees for learning control policies with approximate schemes come with several restrictions. Most results are limited to linear function approximation schemes; in addition some methods make demands such as Lipschitz continuity of the policy being learned (Perkins and Precup, 2003), and favorable initial conditions (Melo et al., 2008). Certain updating schemes enjoy guarantees of convergence, but typically lack guarantees about the long-term reward that will be accrued at convergence (Sabes, 1993; Perkins and Pendrith, 2002; Perkins and Precup, 2003). In recent work, Maei et al. (2010) introduce the Greedy-GQ algorithm, which provably converges while making off-policy learning updates to a linear function approximator. Unfortunately Greedy-GQ requires that the policy followed *while learning* stay fixed, preventing the agent from actively exploring based on the experiences it gathers. Thus, for example, ϵ-greedy exploration with $\epsilon < 1$ violates the conditions needed for Greedy-GQ to converge.

It is quite well-known that bootstrapping—estimating values based on existing estimates of values—is a prime cause of instability while learning on-line with function approximation. Indeed it has been shown that doing away with bootstrapping, and instead making Monte Carlo updates, can result in more stable learning (Boyan and Moore, 1995). Another strategy is to control the extent of bootstrapping through eligibility traces (Sutton and Barto, 1998, see Chapter 7). The eligibility trace parameter λ is such that setting it to 0 yields a fully bootstrapping update, as in Sarsa(0); setting it to 1 yields Monte Carlo updates that involve no bootstrapping; intermediate values implement varying degrees of bootstrapping. Singh and Sutton (1996) show that Sarsa(λ) with $\lambda > 0$ (and with "replacing" traces) promotes quicker learning and better asymptotic behavior than Sarsa(0) on the Mountain Car task. They use tile coding for function approximation.

Not only do Monte Carlo and partially bootstrapping methods benefit learning with function approximation; research shows that they also help cope with state aliasing (Singh et al., 1994; Jaakkola et al., 1995; Pendrith and McGarity, 1998; Perkins, 2002). Both state aliasing and function approximation have the effect of corrupting estimates of the value function—using true returns in place of these estimates reduces bias, even if it means increasing the variance in the updates. In general the best memoryless policies for POMDPs can be stochastic (Singh et al., 1994). However, Loch and Singh (1998) demonstrate that deterministic policies learned using Sarsa(λ) with ample exploration still perform relatively well on a suite of benchmark POMDPs. Key to this success is the high values of λ used (between 0.8 and 0.975). It must be noted from these results that *some* amount of bootstrapping still appears to be useful (that is, the best setting of λ is not 1) even under severe state aliasing. A more recent variant of Sarsa(λ) applied to POMDPs is SarsaLandmark (James and Singh, 2009), in which λ is set to 0 when special "landmark" states (which are perfectly observable) are visited, but remains 1 at all other times.

Whereas most theoretical analyses of VF methods with function approximation pertain to the use of linear architectures, there have been efforts to generalize these results to more expressive representations (Papavassiliou and Russell, 1999). However, it is seldom the case that a practical application of RL exactly matches the function approximation formulations that have been theoretically analyzed. This mismatch has not curtailed the possibility of realizing successful learning algorithms in practice. On the other hand, remarkable RL successes such as in Backgammon (Tesauro, 1992) have employed VF methods using non-linear function approximators in highly stochastic and nonstationary environments.

C.2 Model-based and Batch Methods

Rather than directly learn a value function based on experience, model-based methods first approximate the transition function T and the reward function R of the task MDP. Together, T and R constitute a *model*, which can effectively serve as a surrogate for the environment; that is, they can be used to predict next states and rewards. With access to a model, several alternatives present themselves for deriving a control policy. One possibility is to run forward simulations using the model at action selection time, and to pick actions based on their predicted long-term effects (Kocsis and Szepesvári, 2006). Otherwise, a value function can be computed beforehand using the model, and used to implicitly represent the policy (Kalyanakrishnan et al., 2008). Yet another approach, adopted, for instance, in the popular helicopter control application, is to perform policy search by using the model for evaluating policies (Ng et al., 2004).

The Dyna architecture proposed by Sutton (1990) encapsulates the essential aspects of model-based RL. In the set of discrete maze tasks Sutton uses to demonstrate Dyna, T and R are estimated simply from the frequency counts of state visits. Under Dyna, the action value function Q can be improved both based on direct experience gathered from environmental interaction, and based on indirect experience *simulated* by the model. *Planning*, or learning from simulation in the model, can considerably speed up the convergence rate. Several variants and improvements to Dyna, such as prioritized sweeping, have subsequently been demonstrated to further speed up learning on tasks such as navigation and rod maneuvering through mazes (Peng and Williams, 1993; Moore and Atkeson, 1993; Sutton and Barto, 1998, see Chapter 9).

While model-based methods summarize the information available from transition samples through estimates of T and R, batch methods preserve this information by storing the transitions themselves in memory. Thus, they maintain a data structure $D = \{(s_i, a_i, r_i, s'_i)\}$ comprising transition samples observed in the past. In general the size of D can grow with time; it is typical in practice to exercise some rule to manage its size. Like model-based methods, batch methods also allow for sophisticated updates to Q every time step, going beyond the simple on-line updates that VF methods perform.

Experience replay (Lin, 1992) is a conceptually simple form of batch updating, under which stored transitions are simply replayed multiple times and "on-line" learning updates performed. Stored batches of experiences can be used in more ingenious ways to impart speed and stability to learning. For example, function approximators such as neural networks are often easier to train under sweeps of batch updating, rather than under on-line updating. Fitted Q Iteration (FQI), which is derived from a related method called Fitted Value Iteration (Gordon, 1995), is one such method that uses regression as a subroutine to fit action values to a batch of samples. For a given batch, FQI is guaranteed to converge for "averaging" function approximators such as trees (Ernst et al., 2005) and kernel methods (Ormoneit and Sen, 2002). Even in the absence of such guarantees, FQI has proven effective using neural networks for function approximation in some challenging tasks (Riedmiller, 2005; Kalyanakrishnan and Stone, 2007).

As with VF methods, a significant body of literature devoted to formal analysis of the function approximation case in model-based and batch methods assumes that the model (and value function) are represented as linear combinations of basis functions. For instance, recent work has provided convergence guarantees under certain restrictions for a linear implementation of Dyna (Sutton et al., 2008). In the context of batch RL, the most popular approaches with linear architectures are least squares methods. Least Squares Temporal Difference (LSTD) learning (Bradtke and Barto, 1996) is an efficient procedure to approximate the value function of a policy by processing a batch of samples collected from the policy. LSTD converges to the same set of weights as TD(0), but at a significantly faster rate (Lazaric et al., 2010). LSTD can also be extended to different values of λ (Boyan, 2002). Least Squares Policy Evaluation (LSPE) (Nedić and Bertsekas, 2003) is yet another evaluation algorithm akin to LSTD. Least Squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003) extends LSTD to control problems, computing a sequence of policies from a *fixed* set of data, such that successive iterations are likely to yield better policies. An advantage of batch least squares approaches when compared to on-line methods is that they involve little parameter tuning (they do not need a learning rate parameter), and so they tend to be more robust. This robustness comes at the price of overhead in memory and computation.

While it is natural to imagine model-based and batch RL methods as qualitatively distinct, there is also reason to treat them alike (as we do in Appendix D, addressing these methods together under the abbreviation "MB"). In a formal sense, Strehl et al. (2006) differentiate model-based methods from model-free methods based on the size of the agent's memory. Batch methods that store a significant amount of experience can thus be considered implicitly model-based. One can imagine batch methods to be model-based methods that incidentally simulate only those experiences that have actually occurred while interacting with the environment. Indeed this bias towards "direct" data often helps batch methods such as experience replay achieve better results than model-based approaches that simulate experiences based on imperfect models (Lin, 1992). However, in some cases, such as when model-based and batch methods share the same data and features in a linear architecture, they can be expected to yield identical results (Boyan, 2002; Schoknecht, 2003; Parr et al., 2008; Sutton et al., 2008).[1]

"Best-match learning" (van Seijen et al., 2011) is a recent framework that combines the space-efficiency of VF methods with the sample-efficiency of MB methods. Under this framework, rather than updating the value function as and when samples are gathered, updates are postponed (and samples stored in memory) for as long as possible. The advantage of such a postponement is that updates can reflect any changes that have occurred in the targets for a sample *after* the sample was collected (and before the update is made). In their experiments, van Seijen et al. (2011) show that best-match learning can outperform both experience replay (a batch method) and TD(λ) (a VF method).

[1]Also see: `http://www.cs.duke.edu/~parr/icml08-addendum.html`.

C.3 Policy Gradient Methods

Unlike VF and MB methods, policy gradient (PG) methods do not attempt to compute the effects or expected utilities of actions taken from individual states; instead, they reason about the aggregate performance of policies. In tasks where a policy π can be represented using a vector of real-valued parameters $\mathbf{w}$, PG methods estimate the gradient of the policy value $V(\pi_{\mathbf{w}})$ with respect to $\mathbf{w}$, that is, $\nabla_{\mathbf{w}} V(\pi_{\mathbf{w}})$. In general an unbiased estimate of this gradient can be obtained from transition data (Sutton et al., 2000; Kakade, 2001). Performing ascent on this gradient then leads to local optima in the space of parameters.

PG methods for RL date at least as far back as the REINFORCE suite of algorithms proposed by Williams (1992). Whereas the REINFORCE algorithms themselves do not enjoy provable convergence guarantees, subsequent work has considered multiple formulations of policy gradients that are amenable to theoretical treatment. A key requirement in all these approaches is that the policy be differentiable with respect to its parameters, so that a gradient can be obtained (and this, in turn, requires that policies be stochastic). The thrust, then, is in extracting an unbiased estimate of the gradient from the transition data. An illustrative example is Kakade's natural gradient, which has the additional benefit of being invariant to linear transformations of the parameter space, thereby reducing the burden of tuning step sizes along different parameter axes.

Finite MDPs with small state and action spaces are well-suited to be solved by VF methods, as they require no generalization; the appeal of PG methods lies in tasks where generalization by way of the parameterized policy is effective. Often policies can be easier to represent than value functions;

PG methods become particularly attractive when value functions are hard to represent or learn. Some PG approaches apply to partially observable environments (Baxter and Bartlett, 2001), and to policies that maintain a memory (Aberdeen and Baxter, 2002). Ghavamzadeh and Engel (2007b) show that PG methods can be derived within a Bayesian framework.

C.4 Actor-Critic Methods

Actor-Critic (AC) methods (Barto et al., 1983) encompass a broad range of algorithms. The essence of these algorithms is a division of the learning agent into an *actor* part, which is responsible for implementing a policy, and a *critic*, which evaluates this policy and thereby directs adaptation. In general multiple realizations are possible of both actor and critic, but under the most common interpretation, the actor executes a parameterized policy, which it adapts using policy gradient updates; the actor depends on the critic to compute the gradient (Konda and Tsitsiklis, 2003). The main difference between AC and PG methods is in the computation of the gradient itself.

The gradient of a policy depends on the long-term rewards accrued from the states it visits. Canonical PG approaches would depend on actual returns to compute this gradient (as would Monte Carlo methods), whereas in actor-critic methods, the critic typically maintains a value function that can readily provide the gradient. Suppose that at some instant, the critic has exactly computed the value function V_1 for the actor's policy π_1. Now, say, the actor adapts its policy to π_2. Since the value function V_2 of π_2 is likely to be close to V_1, a few updates to V_1 based on observed transitions under π_2 should suffice to converge to V_2. Estimating V_2 solely based on returns under π_2 is likely to suffer from much higher variance, and would take more transitions to produce comparable gradient estimates. Naturally, the bias introduced by the critic's parameterized representation of the value function could affect the actor's policy improvement. However, theoretical work has addressed the design of "compatible" function approximation for the critic to match the actor's policy representation, such that no bias is introduced (Sutton et al., 2000; Konda and Tsitsiklis, 2003).

The theory of natural gradients has been extended to the actor-critic context (Bhatnagar et al., 2008) and has been shown effective on control tasks such as Cart Pole and hitting a baseball (Peters and Schaal, 2008a). Ghavamzadeh and Engel (2007a) extend the theory of Bayesian reasoning to actor-critic architectures by modeling the policy gradient as a Gaussian Process. In recent times, a number of AC and PG methods have specifically been proposed for learning robotic control, including "Policy learning by Weighting Exploration with the Returns" (PoWER) (Kober and Peters, 2009), "Relative Entropy policy Search" (REPS) (Peters et al., 2010), and "Policy Improvement with Path Integrals" (PI^2) (Theodorou et al., 2010).

C.5 Policy Search Methods

Policy search (PS) methods are much like PG methods: rather than computing values of actions, they search the space of policy parameters for settings that yield high expected long-term reward. Unlike PG methods, PS methods do not necessarily assume that gradients can be computed analytically. Consequently they can operate on deterministic policies. Also, there is no restriction that their parameters be real numbers, and so PS methods can work with a variety of representations. An apt example is the PS framework implemented by Kohl and Stone (2004) to optimize the forward walking speed of an Aibo robot. The robot's gait is described by a manually designed rule, which contains parameters corresponding to positions and timings in the trajectories described by the legs. This policy can be optimized using several methods, including hill climbing, genetic algorithms, Amoeba, and also a "policy gradient" technique that employs finite differences to approximate gradients. A number of PS methods (for example, hill climbing and genetic algorithms) only rely on discerning *ranks* among the values of different policies: these methods are likely to be more robust than PG methods in the presence of stochasticity.

Evolutionary computation has been a particularly popular choice for PS; in particular several neuroevolutionary techniques have been tested on control tasks (Gomez and Miikkulainen, 1999; Stanley, 2004; Metzen et al., 2008). Typically the policy is represented using a neural network, whose topology and weights are evolved to yield policies with higher values. Ignoring the Markovian relationship between observed states and next states allows these methods to cope to some extent with partial observability. Glickman and Sycara (2001) show that even in severely occluded maze tasks, a small neural network with recurrent connections, interpreted as a stochastic policy, can achieve excellent performance.

Just as AC methods aim to reduce the variance in estimates of gradients, techniques to reduce the variance in estimates of policy values have been employed in tandem with PS methods. PEGASUS (Ng and Jordan, 2000) is one such technique. Assuming that an environmental model is available, and further, that different "deterministic" realizations of this model can be obtained by setting different random seeds, PEGASUS effectively controls stochasticity in comparisons between policies. On simple tasks such as grid worlds and a bicycle controller, PEGASUS is demonstrated to significantly improve the performance of a hill climbing algorithm. PEGASUS is also used in the complex helicopter control task (Ng et al., 2004), in which the model is itself learned from traces obtained when an expert controls the helicopter. In yet another model-based approach to PS, Ng et al. (2000) use indirect means to estimate the policy gradient and value, after first estimating the density induced by the policy over the state space. With the aim of increasing the sample-efficiency of PS, Peshkin and Shelton (2002) apply importance sampling techniques to evaluate a policy from samples obtained by following a different policy.

Recent advances in the field of stochastic optimization have had a positive effect on PS methods. The cross-entropy method (de Boer et al., 2005) is among a family of optimization techniques that generate candidate solutions (in the case of RL, policy parameters) from a parametric distribution. This distribution is progressively updated such that with time, its variance diminishes and its mean gravitates towards local optima in the parameter space. Szita and Lőrincz (2006) demonstrate that the cross-entropy method achieves orders of magnitude improvement over the best VF approaches for Tetris (see Chapter 4 in this dissertation for a detailed discussion). Another method in a similar vein is CMA-ES (Covariance Matrix Adaptation Evolution Strategy) (Hansen, 2009), which, like natural policy gradient methods, has the attractive property of being immune to linear transformations of the fitness landscape. Note that we use CMA-ES as a representative PS method in our comparative study in Chapter 3.

Appendix D

An Order Among Classes of Learning Methods

In this appendix, we present a speculative essay inspired by results from chapters 3 and 4. We put forth a hypothesis for ordering different classes of RL methods (specifically the five classes surveyed in Appendix C) based on their procedural bias (discussed in Section 7.2.1, on page 257). We arrive at our hypothesis based on a qualitative argument involving the "sample-sensitivity" of different RL methods. Our hypothesis is to be treated as a guide to organize future research on defining and evaluating the inductive bias of RL methods; such research may validate, refine, or even entirely reject the hypothesis.

We define the sample-sensitivity of an RL method as the extent to which individual samples influence its outcome. If learning method L_1 is more sample-sensitive than another method L_2, then L_1 would "extract more information" from samples than L_2. Thus, L_1 is likely to learn "more quickly" than L_2. On the other hand, if L_1 and L_2 are applied with the same imperfect representation R, then L_1 will achieve a lower asymptotic performance than L_2 (because R corrupts the information extraction process of L_1 more than that of L_2). Thus, the sample-sensitivity of a method is expected to relate directly with its sample-efficiency, but inversely with the method's robustness to poor representations (henceforward, simply referred to as a method's *robustness*).

Figure D.1 summarizes our ordering of different classes of RL methods based on their sample-sensitivity. We consider five classes of RL methods.

In decreasing order of sample-sensitivity, these classes are: model-based and batch (MB) methods, model-free on-line value function-based (VF) methods, actor-critic (AC) methods, policy gradient (PG) methods, and policy search (PS) methods. For the interested reader, we provide a brief survey of these different classes of methods in Appendix C.

Model-based methods (Sutton and Barto, 1998, see chapter 9) rely on simulated experiences generated by an environmental model that is itself inferred from transition samples. Hence, the assumption on which model-based methods bank is that an accurate model of the transition and reward functions can be represented and learned based on observed experiences. This is a stronger assumption than that made by (model-free) on-line value function-

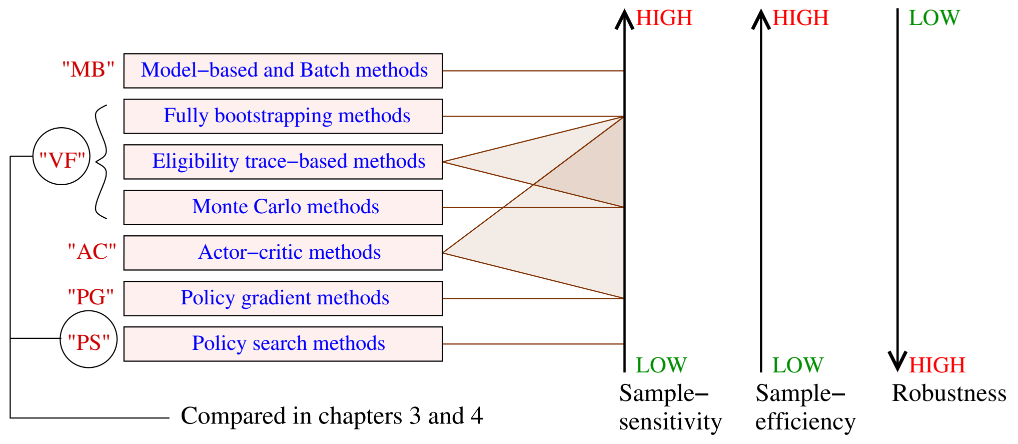


Figure D.1: A proposed ordering of classes of learning methods based on their "sample-sensitivity" (hypothesized to be consistent with the procedural bias of RL methods). Appendix C provides a brief survey of the different classes of methods included in the figure. Eligibility trace methods span a segment of the spectrum between fully bootstrapping methods ($\lambda = 0$) and Monte Carlo methods ($\lambda = 1$). The implementations of the actor and critic in actor-critic methods also give rise to several points on the spectrum.

based (VF) methods (Watkins and Dayan, 1992; Rummery and Niranjan, 1994), which only assume that observed samples can be used in computing state-action *values*. We consider batch RL methods (Lagoudakis and Parr, 2003; Lin, 1992) similar to model-based methods in that they perform multiple passes, or aggregate updates, based on a set of experiences. Note that VF methods make only a single update based on each transition. In line with the view that the data used by batch methods themselves constitute an implicit model (Strehl et al., 2006; Boyan, 2002), we treat model-based and batch methods as one category of methods (abbreviated "MB"): the one with the highest sample-sensitivity.

VF methods make progress towards the goal of approximating the value function by constantly shifting their current estimates towards "better" estimates derived from transition samples. Among this class itself, the extent of bootstrapping could determine the sample-efficiency and robustness of different methods. Fully bootstrapping methods such as Sarsa(0) (Rummery and Niranjan, 1994) estimate values of states using the estimated values of next states, while Monte Carlo methods such as Sarsa(1) estimate state values based on samples of long-term reward. Thus, Monte Carlo methods are less dependent on the state signal and value function representation than fully bootstrapping methods. In general, controlling the eligibility trace parameter λ yields intermediate methods, such as Sarsa(λ), which implement varying extents of bootstrapping.

While VF methods still try to estimate state-action values, policy gradient (PG) methods (Sutton et al., 2000; Kakade, 2001) only estimate the *gradient of the value of a policy* with respect to its parameters, which they achieve by summing actual returns, as under Monte Carlo methods. Policies contain less information than their action value functions, and consequently, can be expressed with a smaller number of parameters. The aggregation of information gathered from multiple transitions to perform an update on a small number of parameters has the effect of making PG methods less sensitive to each individual transition. Actor-critic (AC) methods (Konda and Tsitsiklis, 2003) are much like PG methods, but they rely on a "critic" learning values (to reduce the variance in gradient estimates), as under VF methods. The values learned by the critic do not directly yield a control policy; rather, they guide the improvement of the "actor's" control policy. Thus, we place AC methods in between VF and PG methods in the spectrum shown in Figure D.1.

By "policy search" (PS) methods, we refer to a generic class of optimization algorithms (for example, genetic and evolutionary algorithms (Stanley, 2004), hill climbing, and CMA-ES (Hansen, 2009)) that primarily rely on estimating the *ranks* among a population of policies with respect to their values. Ranks are typically easier to estimate than gradients; indeed PS methods can work perfectly well on policies that do not have analytically computable gradients, which PG methods usually require. Their disregard for everything but the relative order among the set of policies being considered makes PS methods the least dependent on atomic state transitions for the purpose of learning. Thus, PS methods have the lowest sample-sensitivity in Figure D.1.

The scope of the RL problem is broad, and it conjoins numerous dimensions: memory and representation, learning, computation, exploration, and so on. Given this complexity, we believe it unlikely that different learning algorithms will all fall neatly into the pockets presented in Figure D.1. The proposed order is merely intended as a pivot around which future experimental and theoretical investigations can be undertaken. A plausible target for the near term is to extend our experiments using parameterized learning problems to classes of methods other than VF and PS. To develop theoretical frameworks for understanding the inductive bias of RL methods, we may look to build on the information-theoretic constructs developed by Lucas (2010), or turn to standard learning-theoretic formulations such as the VC dimension (Kearns and Vazirani, 1994, see Section 3.2).

As a related exercise, one might draw an analogy between the "VF versus PS methods" comparison we make in chapters 3 and 4, in the context of RL, and the "generative versus discriminative" comparison that has long been studied in relation to supervised learning (Ng and Jordan, 2001). Although the output of an RL method is a policy, VF methods learn policies by first learning a "deeper" structure, the value function (and similarly, model-based methods first learn a model). In this sense, these methods are akin to generative supervised learning methods, which first account for dependencies in the data before applying themselves to the task of predicting labels. On the other hand, PS methods and discriminative supervised methods directly work on the final "deliverable"—a policy or a classifier, respectively. Ng and Jordan (2001) demonstrate through an example that generative methods can perform better when the training data size is small, but discriminative methods typically outperform them on larger data sets. Jebara (2002) unifies these contrasting classes of methods under the umbrella of "maximum entropy discrimination", a more general class. As yet, our understanding of the "VF versus PS" distinction in RL is nowhere as clear as our understanding of the "generative versus discriminative" contrast in supervised learning. We hope future work will tackle this question, which the experiments in this dissertation highlight.

Bibliography

Aberdeen, D. and Baxter, J. (2002). Scaling internal-state policy-gradient methods for POMDPs. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 3–10. Morgan Kauffman.

Ackley, D. and Littman, M. (1992). Interactions between learning and evolution. In *Artificial Life II, SFI Studies in the Sciences of Complexity*, pages 487–509. Addison-Wesley.

Albus, J. S. (1981). *Brains, Behavior and Robotics*. McGraw-Hill.

Arnold, D. V. (2001). Evolution strategies in noisy environments–a survey of existing work. In *Theoretical aspects of evolutionary computing*, pages 239–250. Springer.

Åström, K. J. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10:174–205.

Audibert, J.-Y., Bubeck, S., and Munos, R. (2010). Best arm identification in multi-armed bandits. In *Proceedings of the Twenty-third Conference on Learning Theory (COLT 2010)*, pages 41–53. Omnipress.

Audibert, J.-Y., Munos, R., and Szepesvári, C. (2009). Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2002b). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77.

Auer, P. and Ortner, R. (2010). UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1–2):55–65.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 30–37. Morgan Kauffman.

Baird, L. and Moore, A. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11 (NIPS 1998)*, pages 968–974. MIT Press.

Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, pages 430–435. IEEE.

Banko, M. and Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. In *Proceedings of Thirty-ninth Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, pages 26–33. Association for Computational Linguistics.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):835–846.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1–2):105–139.

Baxter, J. and Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350.

Bechhofer, R. E., Santner, T. J., and Goldsman, D. M. (1995). *Design and analysis of experiments for statistical selection, screening, and multiple comparisons*. Wiley.

Becker, W. A. (1961). Comparing entries in random sample tests. *Poultry Science*, 40(6):1507–1514.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, first edition.

Benda, M., Jagannathan, V., and Dodhiawala, R. (1986). On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, WA, USA.

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840.

Berry, D. A. and Fristedt, B. (1985). *Bandit problems*. Chapman and Hall.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Beyer, H.-G. (2000). Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. *Computer methods in applied mechanics and engineering*, 186(2–4):239–267.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. (2008). Incremental natural actor-critic algorithms. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 105–112. MIT Press.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Böhm, N., Kókai, G., and Mandl, S. (2005). An evolutionary approach to Tetris. In *Proceedings of the Sixth Metaheuristics International Conference (MIC 2005)*. Available at `http://www2.informatik.uni-erlangen.de/publication/download/mic.pdf`.

Boumaza, A. (2009). On the evolution of artificial Tetris players. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, pages 387–393. IEEE.

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2–3):233–246.

Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, pages 369–376. MIT Press.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57.

Brafman, R. I. and Tennenholtz, M. (2003). R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breukelaar, R., Demaine, E. D., Hohenberger, S., Hoogeboom, H. J., Kosters, W. A., and Liben-Nowell, D. (2004). Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, 14(1–2):41–68.

Brodley, C. E. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, 20(1–2):63–94.

Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure exploration in multi-armed bandits problems. In *Proceedings of the Twentieth International Conference on Algorithmic Learning Theory (ALT 2009)*, pages 23–37. Springer.

Burgiel, H. (1997). How to lose at Tetris. *The Mathematical Gazette*, 81(491):194–200.

Campbell, M., Hoane Jr., A. J., and Hsu, F.-h. (2002). Deep Blue. *Artificial Intelligence*, 134(1–2):57–83.

Caruana, R., Karampatziakis, N., and Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, pages 96–103. ACM.

Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, pages 161–168. ACM.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth national conference on Artificial intelligence (AAAI 1994)*, pages 1023–1028. AAAI Press.

Chen, C.-H., He, D., Fu, M., and Lee, L. H. (2008). Efficient simulation budget allocation for selecting an optimal subset. *INFORMS Journal on Computing*, 20(4):579–595.

Chen, M., Dorer, K., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Murray, J., Noda, I., Obst, O., Riley, P., Steffens, T., Wang, Y., and Yin, X. (2003). *Users Manual: RoboCup Soccer Server — for Soccer Server Version 7.07 and Later*. The RoboCup Federation.

Chernoff, H. (1967). Sequential models for clinical trials. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 805–812. University of California Press.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI 1992)*, pages 183–188. AAAI Press.

Cobb, H. G. (1992). Inductive biases in a reinforcement learner. Technical Report AIC-92-013, Navy Center for Applied Research in Artificial Intelligence, Washington DC, USA.

Cobb, H. G. and Bock, P. (1994). Using a genetic algorithm to search for the representational bias of a collective reinforcement learner. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature (PPSN III)*, pages 576–587. Springer.

Cohen, P. R. and Howe, A. E. (1988). How evaluation guides AI research: The message still counts more than the medium. *AI Magazine*, 9(4):35–43.

Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pages 1017–1023. MIT Press.

Dalal, S. R. and Srinivasan, V. (1977). Determining sample size for pretesting comparative effectiveness of advertising copies. *Management Science*, 23(12):1284–1294.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362.

Dayan, P. (2008). The role of value systems in decision making. In *Better than Conscious? Decision Making, the Human Mind, and Implications for Institutions*, pages 51–70. MIT Press.

Dayan, P. and Sejnowski, T. J. (1994). TD(λ) converges with probability 1. *Machine Learning*, 14:295–301.

de Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67.

Degris, T., Sigaud, O., and Wuillemin, P.-H. (2006). Learning the structure of factored Markov decision processes in reinforcement learning problems. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, pages 257–264. ACM.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Digney, B. L. (1998). Learning hierarchical control structures for multiple tasks and changing environments. In *From animals to animats 5*, pages 321–330. MIT Press.

Diuk, C., Li, L., and Leffler, B. R. (2009). The Adaptive k-Meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML 2009)*, pages 249–256. ACM.

Dorigo, M. and Colombetti, M. (1998). *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press.

Downey, C. and Sanner, S. (2010). Temporal difference Bayesian model averaging: A Bayesian perspective on adapting lambda. In *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, pages 311–318. Omnipress.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556.

Even-Dar, E., Mannor, S., and Mansour, Y. (2006). Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, 7:1079–1105.

Even-Dar, E. and Mansour, Y. (2001). Convergence of optimistic and incremental Q-Learning. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 1499–1506. MIT Press.

Farias, V. F. and Van Roy, B. (2006). Tetris: A study of randomized constraint sampling. In *Probabilistic and Randomized Methods for Design under Uncertainty*, chapter 6, pages 189–202. Springer.

Finnsson, H. and Björnsson, Y. (2008). Simulation-based approach to General Game Playing. In *Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 259–264. AAAI Press.

Frazier, P., Powell, W., and Dayanik, S. (2009). The knowledge-gradient policy for correlated normal beliefs. *INFORMS Journal on Computing*, 21(4):599–613.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996)*, pages 148–156. Morgan Kaufmann.

Gabel, T., Riedmiller, M., and Trost, F. (2009). A case study on improving defense behavior in soccer simulation 2D: The NeuroHassle approach. In *RoboCup 2008: Robot Soccer World Cup XII*, pages 61–72. Springer.

Geist, M. and Pietquin, O. (2010). A brief survey of parametric value function approximation. Technical report, Supélec. Available at `http://www.metz.supelec.fr/metz/personnel/geist_mat/pdfs/Supelec644.pdf`.

George, A. P. and Powell, W. B. (2006). Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65(1):167–198.

Ghavamzadeh, M. and Engel, Y. (2007a). Bayesian actor-critic algorithms. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML 2007)*, pages 297–304. ACM.

Ghavamzadeh, M. and Engel, Y. (2007b). Bayesian policy gradient algorithms. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, pages 457–464. MIT Press.

Ghavamzadeh, M., Mahadevan, S., and Makar, R. (2006). Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13(2):197–229.

Gibbons, J. D., Olkin, I., and Sobel, M. (1999). *Selecting and Ordering Populations: A New Statistical Methodology*. SIAM. First published in 1977 by Wiley.

Girgin, S. and Preux, P. (2008). Feature discovery in reinforcement learning using genetic programming. In *Proceedings of the Eleventh European Conference on Genetic Programming (EuroGP 2008)*, pages 218–229. Springer.

Girshick, M. A. (1946a). Contributions to the theory of sequential analysis. I. *The Annals of Mathematical Statistics*, 17(2):123–143.

Girshick, M. A. (1946b). Contributions to the theory of sequential analysis, II, III. *The Annals of Mathematical Statistics*, 17(3):282–298.

Glickman, M. R. and Sycara, K. (2001). Evolutionary search, stochastic policies with memory, and reinforcement learning with hidden state. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 194–201. Morgan Kauffman.

Gomes, C. P. and Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, 126(1–2):43–62.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9:937–965.

Gomez, F. J. and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuro-evolution. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 1356–1362. Morgan Kauffman.

Gomez, F. J. and Miikkulainen, R. (2003). Active guidance for a finless rocket using neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 2084–2095. Springer.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 261–268. Morgan Kauffman.

Grefenstette, J. J., Ramsey, C. L., and Schultz, A. C. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4):355–381.

Grześ, M. and Kudenko, D. (2009). Improving optimistic exploration in model-free reinforcement learning. In *Proceedings of the Ninth International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009)*, pages 360–369. Springer.

Guestrin, C., Koller, D., and Parr, R. (2001a). Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 673–680. Morgan Kaufmann.

Guestrin, C., Koller, D., and Parr, R. (2001b). Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 1523–1530. MIT Press.

Guestrin, C., Lagoudakis, M. G., and Parr, R. (2002). Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 227–234. Morgan Kauffman.

Guez, A., Vincent, R. D., Avoli, M., and Pineau, J. (2008). Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 1671–1678. AAAI Press.

Guha, S. and Munagala, K. (2007). Approximation algorithms for budgeted learning problems. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 104–113. ACM.

Gupta, S. S. and Panchapakesan, S. (2002). *Multiple Decision Procedures: Theory and Methodology of Selecting and Ranking Populations*. SIAM. First published in 1979 by Wiley.

Hansen, N. (2009). *The CMA Evolution Strategy: A Tutorial.* Available at `http://www.lri.fr/~hansen/cmatutorial.pdf`.

Hansen, N., Niederberger, A. S., Guzzella, L., and Koumoutsakos, P. (2009). A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177–221.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation.* Prentice Hall, second edition.

Haynes, T., Wainwright, R. L., Sen, S., and Schoenefeld, D. A. (1995). Strongly typed genetic programming in evolving cooperation strategies. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA 1995)*, pages 271–278. Morgan Kaufmann.

He, D., Lee, L. H., Chen, C.-H., Fu, M. C., and Wasserkrug, S. (2010). Simulation optimization using the cross-entropy method with optimal computing budget allocation. *ACM Transactions on Modeling and Computer Simulation*, 20(1):4:1–4:22.

Heidrich-Meisner, V. and Igel, C. (2008a). Similarities and differences between policy gradient methods and evolution strategies. In *Proceedings of the Sixteenth European Symposium on Artificial Neural Networks (ESANN 2008)*, pages 149–154. D-side Publication.

Heidrich-Meisner, V. and Igel, C. (2008b). Variable metric reinforcement learning methods applied to the noisy mountain car problem. In *Recent*

Advances in Reinforcement Learning: Eighth European Workshop (EWRL 2008), pages 136–150. Springer-Verlag.

Heidrich-Meisner, V. and Igel, C. (2009a). Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML 2009)*, pages 401–408. ACM.

Heidrich-Meisner, V. and Igel, C. (2009b). Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 64(4):152–168.

Holmes, M. P. and Isbell, Jr, C. L. (2006). Looping suffix tree-based inference of partially observable hidden state. In *Proceedings of the twenty-third international conference on Machine learning (ICML 2006)*, pages 409–416. ACM.

Honda, J. and Takemura, A. (2010). An asymptotically optimal bandit algorithm for bounded support models. In *Proceedings of the Twenty-third Conference on Learning Theory (COLT 2010)*, pages 67–79. Omnipress.

Hutter, M. and Legg, S. (2008). Temporal difference updating without a learning rate. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 705–712. MIT Press.

İpek, E., Mutlu, O., Martínez, J., and Caruana, R. (2008). Self-optimizing memory controllers: A reinforcement learning approach. In *Proceedings of the Thirty-fifth International Symposium on Computer Architecture (ISCA 2008)*, pages 39–50. IEEE Press.

Iscen, A. and Erogul, U. (2008). A new perspective to the keepaway soccer: the takers. In *Proceedings of the seventh International Joint Conference on*

Autonomous Agents and Multiagent Systems (AAMAS 2008), pages 1341–1344. IFAAMAS.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov problems. In *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, pages 345–352. MIT Press.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600.

James, M. R. and Singh, S. (2009). SarsaLandmark: an algorithm for learning in POMDPs with landmarks. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 585–591. IFAAMAS.

Jebara, T. (2002). *Discriminative, Generative and Imitative learning.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Jung, T. and Polani, D. (2007). Learning RoboCup-Keepaway with kernels. *Journal of Machine Learning Research - Proceedings Track*, 1:33–57.

Kakade, S. (2001). A natural policy gradient. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 1531–1538. MIT Press.

Kale, S., Reyzin, L., and Schapire, R. E. (2010). Non-stochastic bandit slate problems. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 1054–1062. MIT Press.

Kalyanakrishnan, S. and Stone, P. (2007). Batch reinforcement learning in a complex domain. In *Proceedings of the Sixth International Joint Conference* *on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 650–657. IFAAMAS.

Kalyanakrishnan, S., Stone, P., and Liu, Y. (2008). Model-based reinforcement learning in a complex domain. In *RoboCup 2007: Robot Soccer World Cup XI*, pages 171–183. Springer.

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

Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

Kim, S.-H. and Nelson, B. L. (2007). Recent advances in ranking and selection. In *Proceedings of the Thirty-ninth Winter Simulation Conference (WSC 2007)*, pages 162–172. IEEE Press.

Kistemaker, S. (2008). Cross-entropy method for reinforcement learning. Bachelor's Thesis, University of Amsterdam, Amsterdam, The Netherlands.

Kleinberg, R., Niculescu-Mizil, A., and Sharma, Y. (2010). Regret bounds for sleeping experts and bandits. *Machine Learning*, 80(2–3):245–272.

Kober, J. and Peters, J. (2009). Learning motor primitives for robotics. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 2112–2118. IEEE.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML 2006)*, pages 282–293. Springer.

Koenig, L. W. and Law, A. M. (1985). A procedure for selecting a subset of size m containing the l best of k independent normal populations, with applications to simulation. *Communications in statistics. Simulation and computation*, 14(3):719–734.

Kohl, N. and Stone, P. (2004). Machine learning for fast quadrupedal locomotion. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 611–616. AAAI Press.

Koller, D. and Parr, R. (2000). Policy iteration for factored MDPs. In *Proceedings of the Sixteenth Conference in Uncertainty in Artificial Intelligence (UAI 2000)*, pages 326–334. Morgan Kauffman.

Kolter, J. Z. and Ng, A. Y. (2009). Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the Twenty-sixth Annual International Conference on Machine Learning (ICML 2009)*, pages 521–528. ACM.

Konda, V. R. and Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166.

Kwok, C. and Fox, D. (2004). Reinforcement learning for sensing strategies. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (ICRA 2004)*, pages 3158–3163. IEEE Press.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

Lagoudakis, M. G., Parr, R., and Littman, M. L. (2002). Least-squares methods in reinforcement learning for control. In *Proceedings of the Second Hellenic Conference on AI (SETN 2002)*, pages 249–260. Springer.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22.

Langley, P. (1988). Machine learning as an experimental science. *Machine Learning*, 3(1):5–8.

Langley, P. and Pendrith, M. (1998). Symposium on applications of reinforcement learning: Final report for NSF Grant IIS-9810208. Technical report, Institute for the Study of Learning and Expertise.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2010). Finite-sample analysis of LSTD. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pages 615–622. Omnipress.

Lee, H., Shen, Y., Yu, C.-H., Singh, G., and Ng, A. Y. (2006). Quadruped robot obstacle negotiation via reinforcement learning. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 3003–3010. IEEE Press.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., and Shoham, Y. (2003). Boosting as a metaphor for algorithm design. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003)*, pages 899–903. Springer.

Li, L. (2008). A worst-case comparison between temporal difference and residual gradient with linear function approximation. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML 2008)*, pages 560–567. ACM.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of* *the Nineteenth International Conference on the World Wide Web (WWW 2010)*, pages 661–670. ACM.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321.

Lin, L.-J. and Mitchell, T. M. (1993). Reinforcement learning with hidden states. In *From animals to animats 2*, pages 271–280. MIT Press.

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318.

Littman, M. L. (1993). An optimization-based categorization of reinforcement learning environments. In *From Animals to Animats 2*, pages 262–270. MIT Press.

Loch, J. and Singh, S. (1998). Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 323–331. Morgan Kauffman.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

Lucas, S. M. (2010). Estimating learning rates in evolution and TDL: Results on a simple grid-world problem. In *Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games (CIG 2010)*, pages 372–379. IEEE Press.

Madani, O., Lizotte, D. J., and Greiner, R. (2004). Active model selection. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI 2004)*, pages 357–365. AUAI Press.

Maei, H. R., Szepesvári, C., Bhatnagar, S., and Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, pages 719–726. Omnipress.

Mahadevan, S. (2009). Learning representation and control in Markov decision processes: New frontiers. *Foundations and Trends in Machine Learning*, 1(4):403–565.

Mannor, S. and Tsitsiklis, J. N. (2004). The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5:623–648.

Maron, O. and Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1–5):193–225.

McCallum, A. K. (1996). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Computer Science Department, University of Rochester, Rochester, NY, USA.

McCallum, R. A. (1993). Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning (ICML 1993)*, pages 190–196. Morgan Kauffman.

Melo, F. S., Meyn, S. P., and Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the Twenty-*

fifth international conference on Machine learning (ICML 2008), pages 664–671. ACM.

Metzen, J. H., Edgington, M., Kassahun, Y., and Kirchner, F. (2008). Analysis of an evolutionary reinforcement learning method in a multiagent domain. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 291–298. IFAAMAS.

Miller, B. L. and Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131.

Mitchell, T. M. (1980). The need for biases in learning generalizations. In Shavlik, J. W. and Dietterich, T. G., editors, *Readings in Machine Learning*, pages 184–191. Morgan Kauffman. Book published in 1990.

Mnih, V., Szepesvári, C., and Audibert, J.-Y. (2008). Empirical Bernstein stopping. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, pages 672–679. ACM.

Monahan, G. E. (1982). A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130.

Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artifical Intelligence Research*, 11:241–276.

Munos, R. and Moore, A. W. (2002). Variable resolution discretization in optimal control. *Machine Learning*, 49(2–3):291–323.

Nedić, A. and Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1–2):79–110.

Nevmyvaka, Y., Feng, Y., and Kearns, M. (2006). Reinforcement learning for optimized trade execution. In *Proceedings of the Twenty-third international conference on Machine learning (ICML 2006)*, pages 673–680. ACM.

Ng, A. Y. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference in Uncertainty in Artificial Intelligence (UAI 2000)*, pages 406–415. Morgan Kaufmann.

Ng, A. Y. and Jordan, M. I. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 841–848. MIT Press.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16 (NIPS 2003)*. MIT Press.

Ng, A. Y., Parr, R., and Koller, D. (2000). Policy search via density estimation. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 1022–1028. MIT Press.

Nissen, S. (2007). Large scale reinforcement learning using Q-Sarsa(λ) and cascading neural networks. Master's thesis, Department of Computer Science, University of Copenhagen, Copenhagen, Denmark.

Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, 49(2–3):161–178.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Papavassiliou, V. A. and Russell, S. (1999). Convergence of reinforcement learning with general function approximators. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 748–757. Morgan Kaufmann.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the Twenty-fifth International Conference on Machine learning (ICML 2008)*, pages 752–759. ACM.

Parr, R. E. (1998). *Hierarchical Control and Learning for Markov Decision Processes*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA.

Pendrith, M. D. and McGarity, M. J. (1998). An analysis of direct reinforcement learning in non-Markovian domains. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 421–429. Morgan Kaufmann.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1(4):437–454.

Peng, J. and Williams, R. J. (1996). Incremental multi-step Q-learning. *Machine Learning*, 22(1–3):283–290.

Perkins, T. J. (2002). Reinforcement learning for POMDPs based on action values and stochastic optimization. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002)*, pages 199–204. AAAI Press.

Perkins, T. J. and Pendrith, M. D. (2002). On the existence of fixed points for Q-learning and Sarsa in partially observable domains. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 490–497. Morgan Kauffman.

Perkins, T. J. and Precup, D. (2003). A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 1595–1602. MIT Press.

Peshkin, L. and Shelton, C. R. (2002). Learning from scarce experience. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002)*, pages 498–505. Morgan Kauffman.

Peters, J., Mülling, K., and Altün, Y. (2010). Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 1607–1612. AAAI Press.

Peters, J. and Schaal, S. (2008a). Natural actor-critic. *Neurocomputing*, 71(7–9):1180–1190.

Peters, J. and Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Petrik, M. and Scherrer, B. (2009). Biasing approximate dynamic programming with a lower discount factor. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1265–1272. MIT Press.

Petrik, M., Taylor, G., Parr, R., and Zilberstein, S. (2010). Feature selection using regularization in approximate linear programs for Markov decision processes. In *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, pages 871–878. Omnipress.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 743–750. Morgan Kaufmann.

Pineau, J., Gordon, G. J., and Thrun, S. (2006). Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27:335–380.

Precup, D., Sutton, R. S., and Dasgupta, S. (2001). Off-policy temporal difference learning with function approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 417–424. Morgan Kaufmann.

Puterman, M. L. (1994). *Markov Decision Processes.* Wiley.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI 1996)*, pages 725–730. AAAI Press.

Ramon, J. and Driessens, K. (2004). On the numeric stability of Gaussian processes regression for relational reinforcement learning. In Tadepalli, P., Givan, R., and Driessens, K., editors, *ICML 2004 Workshop on Relational Reinforcement Learning*. Available as `https://lirias.kuleuven.be/bitstream/123456789/131147/1/2004_rrlws_ramon.pdf`.

Ratitch, B. and Precup, D. (2003). Using MDP characteristics to guide exploration in reinforcement learning. In *Proceedings of the Fourteenth European Conference on Machine Learning (ECML 2003)*, pages 313–324. Springer.

Riedmiller, M. (2005). Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML 2005)*, pages 317 – 328. Springer.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535.

Rodríguez, A., Parr, R., and Koller, D. (1999). Reinforcement learning using approximate belief states. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 1036–1042. MIT Press.

Rosin, C. D. and Belew, R. K. (1995). Methods for competitive co-evolution: Finding opponents worth beating. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA 1995)*, pages 373–381. Morgan Kaufmann.

Rottmann, A., Plagemann, C., Hilgers, P., and Burgard, W. (2007). Autonomous blimp control using model-free reinforcement learning in a continuous state and action space. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, pages 1895–1900. IEEE.

Rummery, G. A. (1995). *Problem Solving with Reinforcement Learning*. PhD thesis, Cambridge University Engineering Department, Cambridge, UK.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

Sabes, P. (1993). Approximating Q-values with basis function representations. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 264–271. Lawrence Erlbaum.

Schmidt, C., Branke, J., and Chick, S. E. (2006). Integrating techniques from statistical ranking into evolutionary algorithms. In *Applications of Evolutionary Computing*, pages 752–763. Springer.

Schneider, J., Wong, W.-K., Moore, A., and Riedmiller, M. (1999). Distributed value functions. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, pages 371–378. Morgan Kaufmann.

Schoknecht, R. (2003). Optimality of reinforcement learning algorithms with linear function approximation. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 1555–1562. MIT Press.

Schuurmans, D. and Greiner, R. (1995). Sequential PAC learning. In *Proceedings of the Eigth Annual Conference on Computational Learning Theory (COLT 1995)*, pages 377–384. ACM.

Seymour, B., O'Doherty, J. P., Dayan, P., Koltzenburg, M., Jones, A. K., Dolan, R. J., Friston, K. J., and Frackowiak, R. S. (2004). Temporal difference models describe higher-order learning in humans. *Nature*, 429:664–667.

Sherstov, A. A. and Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In *Proceedings of the Sixth International Symposium on Abstraction, Reformulation and Approximation (SARA 2005)*, pages 194–205. Springer.

Silver, D., Sutton, R. S., and Müller, M. (2007). Reinforcement learning of local shape in the game of Go. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1053–1058. IJCAI.

Singh, S. and Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, pages 974–980. MIT Press.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML 1994)*, pages 284–292. Morgan Kauffman.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158.

Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304.

Spall, J. C. (2003). *Introduction to Stochastic Search and Optimization*. Wiley.

Stanley, K. O. (2004). *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, University of Texas at Austin, Austin, TX, USA.

Stone, P. (1998). *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon Univerity, Pittsburgh, PA, USA.

Stone, P., Kuhlmann, G., Taylor, M. E., and Liu, Y. (2006). Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 93–105. Springer.

Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). PAC model-free reinforcement learning. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, pages 881–888. ACM.

Strehl, A. L. and Littman, M. L. (2005). A theoretical analysis of model-based interval estimation. In *Proceedings of the Twenty-second International Conference on Machine Learning (ICML 2005)*, pages 856–863. ACM.

Sullivan, D. W. and Wilson, J. R. (1989). Restricted subset selection procedures for simulation. *Operations Research*, 37(1):52–71.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning (ICML 1990)*, pages 216–224. Morgan Kauffman.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pages 1038–1044. MIT Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the Twenty-sixth Annual International Conference on Machine Learning (ICML 2009)*, pages 993–1000. ACM.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 1057–1063. MIT Press.

Sutton, R. S., Precup, D., and Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211.

Sutton, R. S. and Singh, S. P. (1994). On bias and step size in temporal-difference learning. In *Proceedings of the Eighth Yale Workshop on Adap-*

tive and Learning Systems, pages 91–96. Center for Systems Science, Yale University.

Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. (2008). Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the Twenty-fourth Conference in Uncertainty in Artificial Intelligence (UAI 2008)*, pages 528–536. AAAI Press.

Suttorp, T., Hansen, N., and Igel, C. (2009). Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, 75(2):167–197.

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning.* Morgan & Claypool.

Szita, I. and Lőrincz, A. (2006). Learning Tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941.

Szita, I. and Lőrincz, A. (2007). Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man. *Journal of Artificial Intelligence Research*, 30:659–684.

Szita, I. and Lőrincz, A. (2008). The many faces of optimism: a unifying approach. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML 2008)*, pages 1048–1055. ACM.

Szita, I. and Szepesvári, C. (2010a). Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, pages 1031–1038. Omnipress.

Szita, I. and Szepesvári, C. (2010b). SZ-Tetris as a benchmark for studying key problems of reinforcement learning. In *Proceedings of the ICML 2010 Workshop on Machine Learning and Games*. Available as `http://www-kd.iai.uni-bonn.de/icml2010mlg/papers/SzitaSzepesvari.pdf`.

Taylor, M. E. and Stone, P. (2007). Representation transfer for reinforcement learning. In *Proceedings of the AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*. AAAI Press.

Taylor, M. E., Stone, P., and Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167.

Tedrake, R., Zhang, T. W., and Seung, H. S. (2004). Stochastic policy gradient reinforcement learning on a simple 3D biped. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, pages 2849–2854. IEEE.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4):257–277.

Tesauro, G., Jong, N. K., Das, R., and Bennani, M. N. (2007). On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299.

Theodorou, E., Buchli, J., and Schaal, S. (2010). Reinforcement learning of motor skills in high dimensions: A path integral approach. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010)*, pages 2397–2403. IEEE Press.

Thierry, C. and Scherrer, B. (2010a). Building controllers for Tetris. *International Computer Games Association Journal*, 32(1):3–11.

Thierry, C. and Scherrer, B. (2010b). Improvements on learning Tetris with cross-entropy. *International Computer Games Association Journal*, 32(1):23–33.

Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263. Lawrence Erlbaum.

Todd, M. T., Niv, Y., and Cohen, J. D. (2009). Learning to use working memory in partially observable environments through dopaminergic reinforcement. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1689–1696. MIT Press.

Togelius, J., Schaul, T., Wierstra, D., Igel, C., Gomez, F., and Schmidhuber, J. (2009). Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, pages 30–33.

Tran-Thanh, L., Chapman, A., Munoz de Cote, E., Rogers, A., and Jennings, N. R. (2010). ϵ-first policies for budget-limited multi-armed bandits. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 1211–1216. AAAI Press.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42:674–690.

Tukey, J. W. (1962). The future of data analysis. *Annals of Mathematical Statistics*, 33(1):1–67.

Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 107–148. Morgan Kauffman.

van Hasselt, H. P. (2011). *Insights in Reinforcement Learning: formal analysis and empirical evaluation of temporal-difference learning algorithms*. PhD thesis, Universiteit Utrecht, Utrecht, The Netherlands.

Van Roy, B. (1998). *Learning and Value Function Approximation in Complex Decision Processes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

van Seijen, H., van Hasselt, H., Whiteson, S., and Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa. In *Proceedings of the 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2009)*, pages 177–184. IEEE.

van Seijen, H., Whiteson, S., van Hasselt, H., and Wiering, M. (2011). Exploiting best-match equations for efficient reinforcement learning. *Journal of Machine Learning Research*, 12:2045–2094.

Vilalta, R. and Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95.

Wald, A. (1947). *Sequential Analysis*. Wiley, first edition.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3–4):279–292.

Whitehead, S. D. and Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83.

Whiteson, S., Kohl, N., Miikkulainen, R., and Stone, P. (2005). Evolving soccer keepaway players through task decomposition. *Machine Learning*, 59(1–2):5–30.

Whiteson, S. and Stone, P. (2004). Adaptive job routing and scheduling. *Engineering Applications of Artificial Intelligence*, 17(7):855–869.

Whiteson, S. and Stone, P. (2006a). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917.

Whiteson, S. and Stone, P. (2006b). On-line evolutionary computation for reinforcement learning in stochastic domains. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference (GECCO 2006)*, pages 1577–1584. ACM.

Whiteson, S., Tanner, B., Taylor, M. E., and Stone, P. (2011). Protecting against evaluation overfitting in empirical reinforcement learning. In *Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2011)*, pages 120–127. IEEE.

Whiteson, S., Taylor, M. E., and Stone, P. (2010). Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 21(1):1–35.

Whiteson, S. A. (2007). *Adaptive Representations for Reinforcement Learning*. PhD thesis, Department of Computer Sciences, University of Texas at Austin, Austin, TX, USA.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Williams, R. J. and Baird, III, L. C. (1994). Tight performance bounds on greedy policies based on imperfect value functions. In *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*. Available at `http://leemon.com/papers/1994wb.pdf`.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606.

Zhang, W. and Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995)*, pages 1114–1120. Morgan Kauffman.

Vita

Shivaram Kalyanakrishnan was born in Madras (now Chennai), India, where he grew up and did his schooling at Vidya Mandir Adyar. He pursued his undergraduate studies at the Indian Institute of Technology Madras, obtaining a B.Tech. degree in Computer Science and Engineering. In the Fall of 2004, he joined the Ph.D. program in the Computer Science department at the University of Texas at Austin. He has served several terms as teaching assistant and graduate research assistant in the department. As part of his graduate studies, he has also undertaken a summer internship at the Honda Research Institute, Mountain View. He has been an active participant at the RoboCup competitions, where his team, UT Austin Villa, has won several prizes in the simulation league, and his work has merited two Best Student Paper awards.

Permanent address: Department of Computer Science
The University of Texas at Austin
1616 Guadalupe St Suite 2.408
Austin TX 78701 USA
`shivaram@cs.utexas.edu`

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.