Practical Reinforcement Learning Using Representation Learning and Safe Exploration for Large Scale Markov Decision Processes

by
Alborz Geramifard
M.Sc., Computer Science
University of Alberta (2007)
B.Sc., Computer Engineering
Sharif University of Technology (2003)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
January 2012

Author ....
Department of Aeronautics and Astronautics
January 5, 2012

Certified by ....
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by ....
Nicholas Roy
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by ....
Leslie Kaelbling
Panasonic Professor of Electrical Engineering and Computer Science

Accepted by ....
Eytan H. Modiano
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Practical Reinforcement Learning Using Representation Learning and Safe Exploration for Large Scale Markov Decision Processes

by

Alborz Geramifard

Submitted to the Department of Aeronautics and Astronautics
on January 19, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

While creating intelligent agents who can solve stochastic sequential decision making problems through interacting with the environment is the promise of Reinforcement Learning (RL), scaling existing RL methods to realistic domains such as planning for multiple unmanned aerial vehicles (UAVs) has remained a challenge due to three main factors: 1) RL methods often require a plethora of data to find reasonable policies, 2) the agent has limited computation time between interactions, and 3) while exploration is necessary to avoid convergence to the local optima, in sensitive domains visiting all parts of the planning space may lead to catastrophic outcomes.

To address the first two challenges, this thesis introduces incremental Feature Dependency Discovery (iFDD) as a representation expansion method with cheap per-time-step computational complexity that can be combined with any online, value-based reinforcement learning using binary features. In addition to convergence and computational complexity guarantees, when coupled with SARSA, iFDD achieves much faster learning (*i.e.,* requires much less data samples) in planning domains including two multi-UAV mission planning scenarios with hundreds of millions of state-action pairs. In particular, in a UAV mission planning domain, iFDD performed more than 12 times better than the best competitor given the same number of samples. The third challenge is addressed through a constructive relationship between a planner and a learner in order to mitigate the learning risk while boosting the asymptotic performance and safety of an agent's behavior. The framework is an instance of the intelligent cooperative control architecture where a learner initially follows a safe policy generated by a planner. The learner incrementally improves this baseline policy through interaction, while avoiding behaviors believed to be risky. The new approach is demonstrated to be superior in two multi-UAV task assignment scenarios. For example in one case, the proposed method reduced the risk by 8%, while improving the performance of the planner up to 30%.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Thesis Supervisor: Nicholas Roy
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

Without the help of the following people, this thesis would not have reached this level of maturity. I am truly fortunate to be surrounded by such magnificent individuals.

- My family, especially my parents Monir and Mehdi, who inspired me to love science and raised me with the passion to understand the unknown. Even after 4 years of separation, you kept rejuvenating my will to make it this far by your phone calls. Thank you for being so supportive with my life decisions and enduring such a long separation. You are wonderful!

- My committee members, Prof. Jonathan How, Prof. Nicholas Roy, and Prof. Leslie Kaelbling. Jon taught me how to see the big picture and stay away from research downfalls. Despite his busy schedule, he always found time to make precise and constructive feedback on my research work. Nick has been my role model of a young successful scientist, who has always reminded me that both intelligence and commitment are necessary for high quality research. He helped me refine my vision on Artificial Intelligence (AI) through countless hours of discussions in front of his whiteboard. Leslie aided me to unify the shared concepts in the Aero/Astro and the Computer Science communities. Every time that I met her I realized how little I know about Machine Learning!

- All of the MIT faculty members with whom I had the chance to interact, namely: Prof. Patrick Winston who elevated my presentation skills and showed me how amazing human brains are; Prof. Josh Tenenbaum who advised me in brain and cognitive sciences minor and motivated me to see AI from the biological prospective; Prof. John Tsitsiklis who kindly helped me through the theoretical development of my thesis; Prof. Emillio Frazzoli for doing such a great job in teaching the ”Principles of Autonomy and Decision Making”; Prof. Russell Tedrake for your magnificent ”Underactuated Robotics” class; Prof. Demitri Bertsekas for his invaluable feedbacks on my research work; and Prof. Brian Williams for helping me carry on my early research at MIT.

- My former advisors at University of Alberta namely: Prof. Richard Sutton and Prof. Michael Bowling to whom I am still grateful for inspiring me to learn about Reinforcement Learning and teaching me how to aim for perfection in research. I am also indebted to Prof. Csaba Szepesvári for his early help on the theoretical development of this work and intellectual discussions on Skype, and Prof. Ronald Parr for his great research that motivated substantial parts of both my M.Sc. and Ph.D. theses.

- My thesis readers, Stefanie Tellex and Brett Bethke, who patiently went through my thesis draft, endured countless typos and mistakes, and provided me with thoughtful feedbacks. Your questions played a major role in solidifying my grasp on my thesis.

- My MIT and University of Alberta colleagues, namely: David Silver who encouraged me on the early steps of this research; Finale Doshi who closely worked with me on the theoretical analysis of the work; Josh Redding for countless fruitful brainstorming sessions and for being such a descent collaborator; Josh Joseph for all the insightful discussions; Amir Massoud Farahmand for his constructive feedback on my paper submissions; Kemal Üre, Aditya Undurti, and Javier Velez for helping me speak my thoughts whenever you where around; and all other members of Aerospace Control Laboratory (ACL), Robust Robotic Group (RRG), and Model-Based Embedded & Robotic Systems (MERS) for being so friendly and resourceful.

This research was sponsored by Air Force Office of Scientific Research (AFOSR, PM Fariba Fahroo) and the Natural Sciences and Engineering Research Council of Canada (NSERC). I would also like to thank Boeing Research and Technology (PM, John Vian) for funding the experimental facility.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

x	Scalar (any lowercase variable)	N/A
$\mathbf{X}$	Matrix or vector (any bolded uppercase variable)	N/A

a	Action	25
$\mathcal{A}$	Action space	25
s	State	25
$\mathcal{S}$	State space	25
r	Reward	25
$\mathcal{R}$	Reward function	25
$\mathcal{P}$	Transition function	25
π	Policy	25
γ	Discount factor	25
Q	State-action value function	26
V	State value function	26
$\boldsymbol{V}$	State value vector of policy π	29
$\boldsymbol{P}$	The transition matrix under policy π	29
$\boldsymbol{R}$	The reward vector under policy π	29
$\mathbf{T}$	The Bellman Operator	29
$\boldsymbol{I}$	Identity matrix	29
ϕ	Feature function for approximating values	31
f	Feature	31
$\boldsymbol{\theta}$	Weight vector	31
$\tilde{\boldsymbol{V}}$	Approximated state value vector	32
$\boldsymbol{\Phi}$	Feature Matrix ($\lvert\mathcal{S}\rvert \times n$)	32
$\mathbf{d}$	Steady state distribution vector	33
$\boldsymbol{D}$	Steady state distribution Matrix = $diag(\mathbf{d})$	33
$\boldsymbol{\Pi}$	The orthogonal projection operator	35
n	Total number of features	40
L_1	Number of samples to approximate Bellman equation	41
φ	Expected feature function	41
ϵ	Probability for random selection in ϵ-greedy mechanism	44

L_2	Number of samples to approximate φ	45
α	Learning rate	46
$\boldsymbol{e}_i$	A zero $n \times 1$ vector with a 1 in its ith index	46
Q^+	Temporary State-action value function	46
δ	TD error	51
ρ	Preference of state-action pair	52
τ	Temperature parameter in the Gibbs Softmax Distribution	52
ψ	Feature function for approximating preferences	53
d	Total number of dimensions of a metric state space	60
k	Maximum number of active features	63
$\wedge$	The mathematical AND operator	69
$\mathcal{B}_n$	Indices (initial features) = $\{1, \cdots, n\}$	72
B_n	$\mathcal{B}_n \cup \{\varnothing\}$	72
$\wp$	Power set function: returns the set of all subsets	73
$\boldsymbol{\chi}$	Well-ordered set of features	73
$\mho$	Sparsifying feature matrix operator	89
$\boldsymbol{\Xi}$	Relevance of each basic feature for split	117
$\boldsymbol{\mu}$	Weighted mean of samples in a basic tile	117
$\boldsymbol{\sigma}$	Weighted variance of of samples in a basic tile	117
T	The true model of the MDP: $(\mathcal{P}, \mathcal{R})$	128
$\hat{T}$	The approximated model of the MDP: $(\hat{\mathcal{P}}, \hat{\mathcal{R}})$	128
λ	The parameter used to bias the preferences of the actor	129
$\mathcal{K}$	The knownness parameter used for the risk analyzer	136
$\mathcal{M}$	The number of Monte-Carlo Simulations to estimate the risk	141
ε	The maximum acceptable acceptable risk	141
$\mathcal{H}$	The maximum length of Monte-Carlo simulations	141
ΔT	The minimum norm change for replanning	142

Chapter 1

Introduction

1.1 Vision and the Existing Gaps

Planning for teams of heterogeneous autonomous mobile agents in stochastic systems is a challenging problem arising in many domains such as robotics, aviation, and military applications. While cooperative planners provide fast and reliable solutions to these problems (Alighanbari, 2004; Alighanbari et al., 2003; Beard et al., 2002; Casal, 2002; Choi et al., 2009; Ryan et al., 2004; Saligrama & Castañón, 2006; Wang et al., 2007; Xu & Ozguner, 9-12 Dec. 2003), their solutions are typically suboptimal due to the inevitable inaccuracy of the model (*e.g.,* the exact model is approximated by a linear system), or violation of assumptions (*e.g.,* the underlying noise in practice does not obey a Gaussian distribution). Moreover, the output of cooperative planners are often nominal trajectories through a narrow part of the state space. The limited applicability of the solution requires recalculation of the plan after a small deviation from the nominal path. By relaxing most assumptions made by cooperative planners, online reinforcement learning (RL) techniques operate in a more realistic framework where the system reasons about the open loop consequences of its own actions and improves its future performance without the need of a third party. Additionally, RL techniques provide global policies executable from anywhere in the state space where not all planners are capable of doing so. However, applying RL methods to multi-agent domains involves three important challenges:

(I) Sample Complexity For multi-agent domains, the size of the state space is often very large as it is exponential in the number of agents. For example, a domain with 10 agents where each agent can take 2 modes and 50 positions has 10^{20} possible states. On the other hand, the number of interactions required by RL methods to achieve reasonable performance often grows with the size of the state space. Hence, scaling RL methods to multi-agent domains is challenging.

(II) Limited Computation As this thesis focuses on lifelong learning which requires embedding learning within the normal operation of the system (*i.e.,* online setting), the time allowed for learning on each interaction is limited. RL methods that have good sample complexities often require expensive computation per-time-step that is not feasible for the online setting.

(III) Safe Exploration While catastrophic outcomes provide learners with valuable information, sometime the loss involved in such feedback is not sustainable. For example, crashing a fuel-limited unmanned aerial vehicle (UAV), while carrying the message that running out of fuel is bad, costs a lot of money and potentially endangers the life of nearby civilians. Avoiding such fatal states requires a robust learning scheme where the range of exploration is bounded within some safe region.

Value-based RL techniques, a popular family of RL, calculate the optimal control policy by estimating the long term advantage of each action from every state and then acting greedily with respect to those values. Approximation techniques have scaled RL methods to large domains by reducing their sample complexity (Stone et al., 2005; Sutton, 1996), allowing RL methods to exceed the human level of expertise in several domains (Silver et al., 2008; Tesauro, 2002; Zhang & Dietterich, 1995). In particular, practitioners have favored the linear family of approximators (Bowling & Veloso, 2003; Li et al., 2009; Petrik et al., 2010; Ratitch & Precup, 2004; Sutton, 1996; Waldock & Carse, 2008) due to desirable properties such as theoretical analysis (Tsitsiklis & Van Roy, 1997) and cheap computational complexity (Geramifard et al., 2006). A fundamental open problem in the realm of RL is how to pick features (*i.e.,* basis functions) for the linear function approximator in order to make the task learnable with small number of samples. While, for most applications, the domain expert selects the set of features through manual tuning (Silver et al., 2008), in recent years a substantial body of research has been dedicated towards automating this process, resulting in adaptive function approximators (AFAs) (Lin & Wright, 2010; Petrik et al., 2010; Ratitch & Precup, 2004; Reynolds, 2000; Sherstov & Stone, 2005; Waldock & Carse, 2008; Whiteson et al., 2007). Finding a computationally cheap AFA which scales to large domains, provides convergence guarantees when combined with RL techniques, and requires minimal design skill is still an open problem.

1.2 The *Right* Features

Many practitioners have focused their attention on feature expansion techniques (Lin & Wright, 2010; Ratitch & Precup, 2004; Whiteson et al., 2007), as there is biological evidence that humans tend to start from a coarse representation and move towards a more detailed representation as they gain more experience (Goodman et al., 2008). For example, human subjects were tested in a task of categorization of objects specified by a set of individual features (*e.g.,* is the object round? is the object colored?). In primary stages, subjects relied mainly on individual features to assign objects to categories. Later, subjects started to develop more detailed features built on top of individual features (*e.g.,* is the object both round and colored?) to correctly categorize objects which were miscategorized previously (Goodman et al., 2008). Similarly, Ullman showed that a hierarchical feature representation with intermediate complexity built on top of small fragments is optimal in the sense of correctly identifying faces in the task of visual classification (Ullman et al., 2002). Motivated by this idea, Finlayson and Winston introduced the Goldilocks hypothesis (Finlayson & Winston, 2006) in the analogical retrieval setting. They argued that on average intermediate features are best as they maximize mutual information. While these cognitive experiments are promising, they do not answer the following fundamental question:

Question #1: What are the mathematical properties of good features?

The theoretical RL research investigating the answer to Question #1 is still young. Recent studies have taken basic steps towards identifying features that will help reduce the approximation error (Parr et al., 2007). This thesis elevates the current understanding about feature properties by mathematically introducing the notation of feature coverage which plays a critical role in identifying good features for approximating the value function. Coverage for a feature is the portion of states for which a feature is active (*i.e.,* the value of the feature is not zero).

For example, consider a simple planning scenario where a fuel-limited UAV performs a mission. The UAV can be in 3 locations (A,B,C) and have 3 fuel levels (low, medium, high) amounting to 9 states. The feature (fuel = low), has the coverage of $^1/_3$, because this feature is active regardless of the UAV's location. On the other hand, the feature (fuel = low AND location = B) has coverage of $^1/_9$, since it is active only in one state. It is clear that changing the weight of features with more coverage affects the value function for larger parts of the state space. This may suggest that having high coverage is a useful property for features, as the value function can be approximated with fewer data points. Unfortunately, while features with high coverage provide a fast approximation to the value function, the resulting approximation is not necessarily useful. Going back to our example, suppose that having low fuel translates into negative values except when the UAV is at location B (Base). If the weight corresponding to feature (fuel = low) is set to a negative value, the value function is pushed down for all parts of the state space where (fuel = low), including location B. Hence providing a good approximation of the value function requires features with less coverage that captures such dependencies (*i.e.,* fuel = low AND location = B).

One may ask then, why not take an opposite approach and include only features with the narrowest coverage so that any improper information transfer in the value function is avoided? While asymptotically such a representation provides a perfect approximation, it requires a plethora of training samples to capture the shape of the function, because the number of parameters (*i.e.,* weights) equals the number of possible states, which is exponential in the number of state dimensions. So the next immediate questions are:

> *Question #2: What is the right amount of coverage for features and how can an algorithm expand such features?*

This thesis answers both these questions by first providing a theorem that identifies features with guaranteed convergence rate in terms of approximation error reduction. Based on these results, incremental feature dependency discovery (iFDD) is introduced as a novel feature expansion technique that has both theoretical and empirical support. The core benefit of iFDD over related feature expansion techniques is its ability to consider features with better coverage values, while other methods reduce the coverage of features they consider in a faster rate, eliminating many promising features along the way. This part of the thesis addresses the **sample complexity** challenge. Using sparsification techniques, iFDD is simplified and applied to the online setting, where the per-time-step computational complexity of online iFDD (*i.e.,* the complexity of executing iFDD on every interaction with the world) is independent of the total number of features. This property hedges against the **limited computation** challenge. Empirical results in several domains, including two UAV mission planning scenarios, demonstrate the sample complexity advantage of online iFDD over two fixed representations and two state-of-the-art feature expansion techniques. The extra computation time consumed by the C++ implementation of online iFDD to add new features was less than 4 milliseconds in all experiments run on an Intel Xeon 2.40 Ghz with 4 GB of RAM and Linux Debian 5.0.

1.3 Safety

Exploring the environment while avoiding catastrophic states is critical for learning in domains involving expensive hardware. This constraint is another hurdle that practitioners come across when applying RL to realistic domains. Existing methods for safe exploration either behave too pessimistically (Heger, 1994), lack convergence guarantees (Geibel & Wysotzki, 2005), and/or provide no safety guarantees (Abbeel & Ng, 2005). While RL methods assume no prior knowledge about the model, for most practical domains, an approximate model can be obtained through domain experts or model estimation. Moreover, as mentioned earlier, there is a rich body of literature on planning methods that provide sub-optimal solutions for large control problems. The solutions generated by such planners are often safe because they take conservative measures. Now the question is:

> *Question #3: How can learning methods take advantage of existing planners and approximated models to improve the safety of the system during the learning phase?*

In the presence of a safe sub-optimal policy (mentor) built on top of an estimated model, the learner (protégé) can take advantage of the mentor's wisdom in two ways. First, the mentor can guide the protégé in promising parts of the state space where suggested by mentor's sub-optimal policy. This guidance often reduces the sample complexity of learning techniques (Knox & Stone, 2010) which is important when dealing with large state spaces. Secondly, the protégé can probe the mentor before taking any actions in order to ensure the safety of the action it is about to take. This safety checking can be done by projecting the action through the estimated model and verifying the existence of a safe policy from one step ahead using the mentor policy. The resulting methods are cooperative planners introduced in this thesis that address both the **safety** and **sample complexity** challenges. Empirical results in UAV mission planning scenarios with more then 9 billion state-action pairs are presented to demonstrate the applicability of cooperative planners in large domains.

1.4 Thesis Statement

This thesis introduces iFDD as a novel feature expansion technique with cheap per-time-step complexity to provide RL practitioners using linear function approximation with a useful tool for tackling large MDPs. This thesis also demonstrates how inaccurate models and existing planning methods can be integrated with learning approaches to reduce the risk involved in the learning process.

1.5 Thesis Structure

The structure of this thesis is as follows: Chapter 2 reviews Markov Decision Processes as the framework that formulates the sequential decision making problem under uncertainty. It then provides a solution trend starting from Dynamic Programming methods to Reinforcement Learning (RL) methods including the use of linear function approximation. Chapter 3 defines good features for linear function approximation in the context of RL, and takes a theoretical approach to finding such features with a guaranteed rate of convergence in terms of the error reduction in value approximation. Chapter 4 takes a practical approach to finding good features by approximating theoretical algorithms using online techniques. It provides an extensive set of experiments showing the advantage of iFDD over related approaches. While Chapters 3 and 4 focus on sample complexity and computational complexity, Chapter 5 switches the main focus to safety. In particular this chapter explores how prior knowledge and existing planners can mitigate the risk, and reduce the sample complexity involved in learning, while boosting the performance of planners. Highlighting the main contributions and pointing out promising future works, Chapter 6 concludes the thesis.

Chapter 2

Background

This chapter reviews the preliminaries of this thesis consisting of Markov Decision Processes (MDPs), linear function approximation, model-based MDP solvers, and model-free MDP solvers. Readers seeking more detail are encouraged to read fundamental text books on these topics (Bertsekas & Tsitsiklis, 1995; Buşoniu et al., 2010; Sutton & Barto, 1998)

2.1 Markov Decision Processes (MDPs)

A Markov Decision Process (MDP) (Sutton & Barto, 1998) is a tuple defined by $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}^a_{ss'}$ is the probability of getting to state s' by taking action a in state s, $\mathcal{R}^a_{ss'}$ is the corresponding reward, and $\gamma \in [0, 1]$ is a discount factor that balances current and future rewards . A *trajectory* is a sequence $s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots$, where the action $a_t \in \mathcal{A}$ is chosen probabilistically according to a *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ mapping each state-action pair to a probability. The agent selects the action in each specified state using its policy. Every consequent state is generated by the environment according to the transition model (*i.e.,* $i \geq 1, s_{i+1} \sim \mathcal{P}^{a_i}_{s_i}$). For every state $s \in S$, $\pi(s, .)$ forms a probability distribution:

$$\forall s \in \mathcal{S}, \sum_{a \in \mathcal{A}} \pi(s, a) = 1.$$

This thesis focuses on deterministic policies $\pi : \mathcal{S} \to \mathcal{A}$, which is a special case of general policies with binary probabilities:

$$\pi(s) = \bar{a} \Rightarrow \pi(s, a) = \begin{cases} 1 & \text{if } a = \bar{a} \\ 0 & \text{otherwise} \end{cases} .$$

Given a policy π, the state-action value function, $Q^\pi(s,a)$ of each state-action pair, is the expected sum of the discounted rewards for an agent starting at state s, taking action a, and then following policy π thereafter:

$$Q^\pi(s,a) = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t \middle| s_0 = s, a_0 = a\right]. \tag{2.1}$$

In finite state spaces, $Q^\pi(s,a)$ can be stored in a table. The goal of solving an MDP is to find the optimal policy which maximizes the expected cumulative discounted rewards in all states. In particular, the optimal policy π^* is defined as:

$$\forall s, \pi^*(s) \quad = \quad \operatorname*{argmax}_{a\in\mathcal{A}} Q^{\pi^*}(s,a). \tag{2.2}$$

The state value function, called the value function, for a given policy π is defined as:

$$\begin{aligned} V^\pi(s) \quad &\triangleq \quad \max_{a\in\mathcal{A}} Q^\pi(s,a) && (2.3)\\ &= \quad E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t \middle| s_0 = s\right]. && (2.4)\end{aligned}$$

The optimal value function is defined accordingly as:

$$V^*(s) \triangleq V^{\pi^*}(s) = \max_{a\in\mathcal{A}} Q^{\pi^*}(s,a) = Q^{\pi^*}(s,\pi^*(s)).$$

The optimal value function satisfies the Bellman Equation:

$$\begin{aligned} \forall s \in \mathcal{S} \quad V^*(s) \quad &= \quad \max_{a\in\mathcal{A}} E_{s'}\left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \middle| s' \sim \mathcal{P}^a_s\right] \\ &= \quad \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right]. && (2.5)\end{aligned}$$

2.2 MDP Solvers at a Glance

This section provides an overview of existing MDP solvers. The goal of this section is to show how the complexity reduction of MDP solvers provides a path from dynamic programming methods towards reinforcement learning algorithms.

MDPs are generally tackled by two approaches (Bertsekas & Tsitsiklis, 1995): 1) value-based methods and 2) policy search techniques. Value-based methods solve MDPs by finding the state-action (Q) values defined in Equation 2.1. The optimal policy for each state is

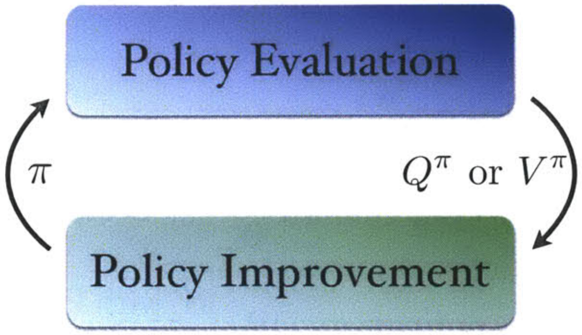


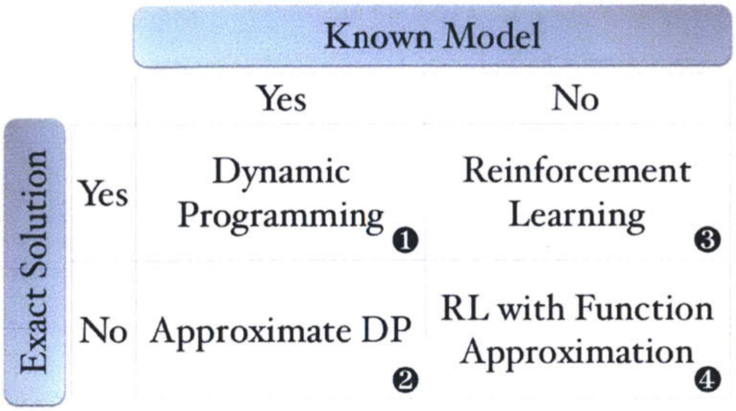
Figure 2-1: Policy Evaluation - Policy Improvement Loop: The convergent policy is guaranteed to be optimal, if the Q or V value function is represented exactly.

then calculated by taking the action which maximizes the value over all possible actions as shown in Equation 2.2. Policy search methods avoid the calculation of the value function all together by searching directly for policies with high cumulative discounted rewards. To do so, they assume a parametric form for the policy function with an initialization point. On each iteration, policy search methods improve the policy parameters using gradient descent methods, ensuring convergence to a local optimum.

The strength of value-based methods is in their policy representation power, meaning that they can capture any form of optimal policy as long as the ranking of state-action values are captured correctly through the value function. On the other hand, if an expert knows a parametric form of the policy function in which good performing policies lie, policy search methods can in practice find reasonable policies faster than value-based methods (Kohl & Stone, 2004; Peters & Schaal, 2008; Sutton et al., 1999; Tedrake et al., 2004). This thesis mainly focuses on the value-based methods, as 1) they are more general because the policy is not assumed to have a parametric form and 2) calculating state values provides direct insight into why the solver believes a particular policy is good. This section also covers the actor-critic family of methods (Bhatnagar et al., 2007) which is the fusion of value-based and policy search techniques.

Value-based solvers tackle an MDP in two phases shown in Figure 2-1: 1) policy evaluation and 2) policy improvement. In the policy evaluation phase, the solver calculates the value function for all states given the fixed policy. In the policy improvement step, the algorithm improves the previous policy by considering values obtained in the policy evaluation step. This process continues until either the policy function remains unchanged, a time limit has been reached, or a certain accuracy for the value function is reached.

Figure 2-2 depicts the family of value-based MDP solvers, categorized based on their knowledge of the MDP models (*i.e.,* $\mathcal{R}$ and $\mathcal{P}$), and the use of function approximation to



		Known Model	
		Yes	No
Exact Solution	Yes	Dynamic Programming ❶	Reinforcement Learning ❸
	No	Approximate DP ❷	RL with Function Approximation ❹

Figure 2-2: Four types of value-based MDP solvers categorized based on their assumption about the MDP model and the use of approximation methods.

represent the value function. The next four sections cover each family of the value-based MDP solvers. The order in which this thesis covers these methods is highlighted by the circled number in Figure 2-2.

2.3 Dynamic Programming

Dynamic Programing (DP) refers to a set of algorithms that solve a known MDP by finding the optimal value function and its corresponding optimal policy (Bellman, 1958; Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998). First, let us observe that given an MDP, policy evaluation (*i.e.,* finding the value function under a fixed policy) can be done in closed form. Looking back at the Equation 2.4, the value function can be derived recursively as:

$$\begin{aligned} V^\pi(s) &= E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t \middle| s_0 = s\right] \\ &= E_\pi\left[r_0 + \sum_{t=1}^{\infty}\gamma^t r_t \middle| s_0 = s\right] \\ &= E_\pi\left[r_0 + \gamma\sum_{t=1}^{\infty}\gamma^{t-1} r_t \middle| s_0 = s\right] \\ &= E_\pi\left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma\sum_{t=0}^{\infty}\gamma^t r_{t+1} \middle| s_{-1} = s, s_0 = s'\right] \\ &= E_\pi\left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V^\pi(s') \middle| s_{-1} = s, s_0 = s'\right] \end{aligned}$$

$$= \sum_{s' \in \mathcal{S}} \mathcal{P}^{\pi(s)}_{ss'} \left[\mathcal{R}^{\pi(s)}_{ss'} + \gamma V^{\pi}(s') \right]. \quad (2.6)$$

Notice the difference between Equations 2.5 and 2.6. The former is the Bellman equation which is independent of the policy while the latter is the recursive form of the value function depending on a fixed policy. The state values can be calculated by solving $|\mathcal{S}|$ linear equations each specified by writing Equation 2.6 for every state of the MDP. The solution for a finite state MDP with $\mathcal{S} = \{s_1, s_2, ..., s_{|\mathcal{S}|}\}$ for which the vector $\boldsymbol{V}^{\pi}{}_{|\mathcal{S}|\times 1}$ represents the value function of the policy π, can be calculated in closed form (notations may exclude the π superscript for brevity, yet the policy dependency is always assumed implicitly). The $\boldsymbol{P}$ matrix represents the transition model with $\boldsymbol{P}_{ij} = \mathcal{P}^{\pi(s_i)}_{s_i s_j}$, and the $\boldsymbol{R}$ vector is the reward model, where $\boldsymbol{R}_i = \sum_j \mathcal{P}^{\pi(s_i)}_{s_i s_j} \mathcal{R}^{\pi(s_i)}_{s_i s_j}$. Hence Equation 2.6 can be written in the matrix form as:

$$\begin{aligned} \boldsymbol{V} &= \boldsymbol{R} + \gamma \boldsymbol{P} \boldsymbol{V}, \\ &= \mathrm{T}(\boldsymbol{V}), \end{aligned} \quad (2.7)$$

where T, defined here, is the Bellman operator applied to the value function. The output of T is a vector with the same size as the input vector (*i.e.,* $\mathrm{T} : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$). It can be seen that each element of $\mathrm{T}(\boldsymbol{V})$ is calculated by the Equation 2.6. The problem of policy evaluation translates into finding the fixed point of the Bellman operator:

$$\begin{aligned} \boldsymbol{V} &= \mathrm{T}(\boldsymbol{V}) \\ \Rightarrow \boldsymbol{V} &= (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} \boldsymbol{R}, \end{aligned} \quad (2.8)$$

where $\boldsymbol{I}$ is the identity matrix. Following the Neumann series as suggested by Barto and Duff (1994), it can be shown that the inverse always exists and can be calculated as:

$$(\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} = \sum_{k=0}^{\infty} (\gamma \boldsymbol{P})^k.$$

Equation 2.8 summarizes the policy evaluation step. As for the policy improvement, the new policy is updated by selecting the action which is greedy with respect to the calculated values. Because the MDP is known, the best action can be found by maximizing the expected one step look ahead estimation of values:

$$\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]. \quad (2.9)$$

```
Algorithm 1: Policy Iteration                                               Complexity
  Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
  Output: $\pi$
1 $\pi(s) \leftarrow Random(\mathcal{A})$ for $s \in \mathcal{S}$
2 changed $\leftarrow$ True
3 while changed do
4     $\boldsymbol{V}^\pi \leftarrow (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi)^{-1} \boldsymbol{R}^\pi$                                       $\Theta(|\mathcal{S}|^3)$
5     for $s \in \mathcal{S}$ do
6         $\pi^+(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$          $\Theta(|\mathcal{A}||\mathcal{S}|)$
7     changed $\leftarrow (\pi^+ \neq \pi)$
8     $\pi \leftarrow \pi^+$
9 return $\pi$
```

Putting these two phases together, we arrive at the Policy Iteration method shown in Algorithm 1. The algorithm also shows the computational complexity of selected lines for each iteration on the right side. The output of the algorithm is guaranteed to be the optimal policy (Sutton & Barto, 1998). Note that the algorithm requires $\Theta(|\mathcal{S}|^3)$ calculations per iteration to evaluate the policy (line 4) and $\Theta(|\mathcal{A}||\mathcal{S}|^2)$ to calculate the new policy (lines $5, 6$). However, as long as the value function is getting closer to the exact solution, the loop shown in Figure 2-1 still converges to the optimal solution (Sutton & Barto, 1998). This idea leads to a process that saves substantial computation for finding a good policy, because as long as the agent gets the ranking of actions in a state correct, the result of policy improvement yields optimal action selection. To accommodate this idea, line 4 can be changed to the following update rule, known as the Bellman update:

$$\forall s \in \mathcal{S}, V^\pi(s) \quad = \quad \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right], \tag{2.10}$$

where the value of each state is improved by one step lookahead. This change brings us to the Value Iteration method shown in Algorithm 2. When the loop is finished, the resulting value function satisfies the Bellman Equation (*i.e.,* Equation 2.5). Coping with computer precision limitations, practitioners often replace line 8 of Algorithm 2 with a checkpoint verifying that the maximum change applied to the value function is less than a small positive number (Chapter 4 of Sutton & Barto, 1998). The value iteration algorithm eliminates the matrix inversion step that required $\Theta(|\mathcal{A}||\mathcal{S}|^2)$ computation per iteration. Note that the inner loop requires a loop over all states.

For large state spaces, representing the value function using a lookup table is not feasible due to memory and computation restrictions. The next section describes DP methods

```
Algorithm 2: Value Iteration                                                      Complexity
Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
Output: $\pi$
$V(s) \leftarrow Random()$ for $s \in \mathcal{S}$
changed $\leftarrow$ False
repeat
    for $s \in \mathcal{S}$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$           $\Theta(|\mathcal{A}||\mathcal{S}|)$
        $\pi(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$        $\Theta(|\mathcal{A}||\mathcal{S}|)$
        changed $\leftarrow$ changed or $v \neq V(s)$
until not changed
return $\pi$
```

applied to such domains by trading off solution quality for lower memory and computation requirements.

2.4 Linear Function Approximation

Holding a tabular representation of the Q function (*i.e.,* storing a unique value for each state-action pair) is impractical for large state spaces. A common approach to coping with large state spaces is to use a linear approximation of the form $Q^\pi(s, a) = \boldsymbol{\theta}^T \phi(s, a)$. The feature function $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$ maps each state-action pair to a vector of scalar values. Each element of the feature function $\phi(s, a)$ is called a *feature*; $\phi_f(s, a) = c \in \mathbb{R}$ denotes that feature f has scalar value c for state-action pair (s, a), where $f \in \chi = \{1, \cdots, n\}$. χ represents the set of features[1] ; the vector $\boldsymbol{\theta} \in \mathbb{R}^n$ holds weights. As defined in Equation 2.2, finding the optimal policy requires the correct ranking of Q values in a given state. Hence for any state s, practitioners often avoid approximating the value of $Q(s, a)$ based on $Q(s, a')$, where $a' \neq a$ (Geramifard et al., 2006; Lagoudakis & Parr, 2003; Sutton, 1996). This is done by first mapping each state to a set of features $\phi : \mathcal{S} \to \mathbb{R}^n$ and then copying the resulting feature vector in the corresponding action slot, while setting the features for the rest of the actions to zero. This process separates the approximation for each individual action, providing a more accurate ranking among various actions in a certain state. The following example shows this mechanism for an MDP with 2 actions, where $3 \times 2 = 6$

[1] Properties such as being close to a wall or having low fuel can be considered as features. In our setting, unless specified, we assume for each domain, all such properties are labeled with numbers and are addressed with their corresponding number.

features are used for linear function approximation.

$$\phi(s) = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} \Rightarrow \phi(s,a_1) = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \ \phi(s,a_2) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} \tag{2.11}$$

For the rest of the thesis, $\phi(s,a)$ is assumed to be generated from $\phi(s)$ following the above process.[2] Chapter 3 will describe useful properties of features and describe popular function approximators that have been favored by RL practitioners (Stone & Sutton, 2001; Stone et al., 2005; Sutton, 1996). For this chapter the ϕ function is assumed to be given.

2.5 Approximate Dynamic Programming

So far, for the policy evaluation phase $\boldsymbol{V}$ represented the value function of a fixed policy as a point in an $|\mathcal{S}|$ dimensional space using $|\mathcal{S}|$ parameters (elements of the $\boldsymbol{V}$ vector). The idea of approximate DP (ADP) methods is to approximate the value function by representing it in a lower dimensional space using $n \ll |\mathcal{S}|$ parameters. In particular, this thesis focuses on the linear family of approximators explained in Section 2.4, where:

$$\boldsymbol{V} \approx \tilde{\boldsymbol{V}} = \begin{bmatrix} \text{---} \, \phi^{\top}(s_1) \, \text{---} \\ \text{---} \, \phi^{\top}(s_2) \, \text{---} \\ \vdots \\ \text{---} \, \phi^{\top}(s_{|\mathcal{S}|}) \, \text{---} \end{bmatrix} \times \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \triangleq \boldsymbol{\Phi}_{|\mathcal{S}| \times n} \boldsymbol{\theta}_{n \times 1}.$$

When the value function is approximated using linear function approximation, the true value function might not lie in the space spanned by the feature function (*i.e.,* column space of $\boldsymbol{\Phi}$). Hence a metric is required to define the best approximate value function in the span of $\boldsymbol{\Phi}$. There are two major metrics used in the literature (Bradtke & Barto, 1996; Farahmand et al., 2008; Lagoudakis & Parr, 2003; Scherrer, 2010; Sutton et al., 2009) to define the best approximated value function, namely: 1) Bellman residual minimization (BRM), and 2) Projected Bellman Residual Minimization, also known as the least-squares Temporal Difference solution (LSTD e the value of each state is being approximated, a

[2] The size of the action space (*i.e.,* $|\mathcal{A}|$) is assumed to be small enough so that $n|\mathcal{A}|$ is always maintainable in memory.

weight vector is also required to highlight our concerns with respect to the resulting error in each state. Intuitively, states that are visited more often should have higher weights, penalizing the error correspondingly. This intuition is captured through the *steady state probability distribution* for a fixed policy defined as a vector $\mathbf{d}_{1\times n}$, where

$$\begin{aligned} & \mathbf{d}\boldsymbol{P} = \mathbf{d} && (2.12)\\ \text{s.t.} \quad & ||\mathbf{d}||_{\ell_1} = 1, && (2.13)\\ & \forall i \in \{1, \cdots, |\mathcal{S}|\}, \mathbf{d}_i \geq 0, && (2.14) \end{aligned}$$

where $\mathbf{d}_i$ is the i^{th} element of $\mathbf{d}$. Essentially, the steady state probability distribution is the eigenvector of $\boldsymbol{P}^\top$, representing a probability distribution over states that is invariant to the transition model.[3] For mathematical purposes $\boldsymbol{D}_{|\mathcal{S}|\times|\mathcal{S}|}$ is defined as a matrix, with $\mathbf{d}$ on its diagonal ($\boldsymbol{D} = diag(\mathbf{d})$). Figure 2-3 shows a simple example of an MDP with two states (shown as white circles) and the fixed policy shown as arrows exiting each state. The outcome of actions (filled black circles) are stochastic and shown as probabilities on the top of each resulting transition. For this domain:

$$\begin{aligned} \boldsymbol{P} &= \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix} \\ \mathbf{d}\boldsymbol{P} = \mathbf{d} &\Rightarrow \begin{bmatrix} d_1 & d_2 \end{bmatrix} \begin{bmatrix} 0.7 & 0.3 \\ 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} d_1 & d_2 \end{bmatrix} \\ ||\mathbf{d}||_{\ell_1} = 1 &\Rightarrow \begin{cases} 3d_1 - 5d_2 = 0 \\ d_1 + d_2 = 1 \end{cases} \\ &\Rightarrow \mathbf{d} = \begin{bmatrix} 0.625 & 0.375 \end{bmatrix}, \boldsymbol{D} = \begin{bmatrix} 0.625 & 0 \\ 0 & 0.375 \end{bmatrix} \end{aligned}$$

This means that in the limit of following the above policy in this MDP, regardless of the starting distribution, the probabilities of being at states 1 and 2 are 0.625 and 0.375, correspondingly.

Figure 2-4 depicts a geometric view of the optimal value function ($\boldsymbol{V}$) and its projection into the span of $\boldsymbol{\Phi}$, using the orthogonal projection operator, $\boldsymbol{\Pi}$. Unfortunately $\boldsymbol{V}$ is not known, but the value function which has zero Bellman error for all states (*i.e.*, $\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}} = 0$) is guaranteed to be optimal. Hence both the BRM and LSTD approaches minimize a term defined with respect to the Bellman error with the desired goal that the resulting solution is close to $\boldsymbol{\Pi V}$. In particular BRM minimizes the norm of the Bellman error

[3] Notice that the transition model, $\boldsymbol{P}$, already included the fixed policy π.

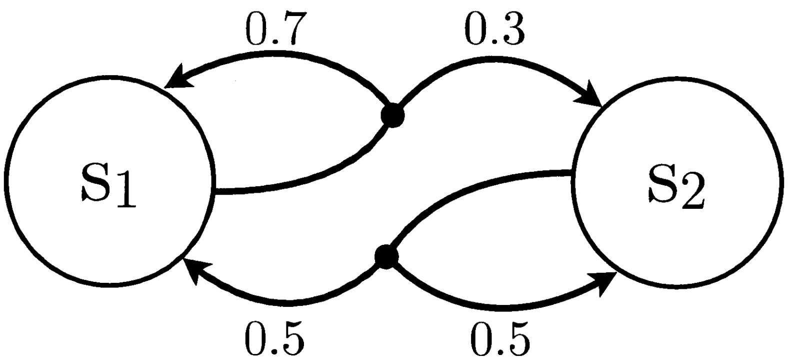


Figure 2-3: A simple MDP with two states. The fixed deterministic policy is shown as arrows exiting each node. The outcome of actions (filled black circles) are stochastic shown as multiple arrows exiting each action.

shown as the solid green vector, while LSTD minimizes the projected Bellman error shown as the dashed blue line.

2.5.1 Bellman Residual Minimization

As the name suggests, the Bellman residual minimization (BRM) approach to solving for a policy minimizes the cost function C, the ℓ_2-norm of the Bellman Residual error weighted by the steady state distribution. From this point, the term "norm" will be used for ℓ_2-norm weighted by the steady state distribution. Hence, $||\mathrm{x}|| = ||\mathrm{x}||^{\mathrm{D}}_{\ell_2}$. The solution of the BRM method is calculated by:

$$\begin{aligned} C(\boldsymbol{\theta}) &= ||\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}|| \\ &= (\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}})^{\top}\boldsymbol{D}(\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}) \\ &= (\boldsymbol{R} + (\gamma \boldsymbol{P\Phi} - \boldsymbol{\Phi})\boldsymbol{\theta})^{\top}\boldsymbol{D}(\boldsymbol{R} + (\gamma \boldsymbol{P\Phi} - \boldsymbol{\Phi})\boldsymbol{\theta}) \\ &\triangleq \boldsymbol{E}^{\top}\boldsymbol{D}\boldsymbol{E}. \end{aligned}$$

Taking the derivative with respect to $\boldsymbol{\theta}$ and noting that $\boldsymbol{D}^{\top} = \boldsymbol{D}$, results in:

$$\begin{aligned} \frac{\partial C}{\partial \boldsymbol{\theta}} &= \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{\theta}} \frac{\partial C}{\partial \boldsymbol{E}} \\ &= 2(\gamma \boldsymbol{P\Phi} - \boldsymbol{\Phi})^{\top}\boldsymbol{D}(\boldsymbol{R} + (\gamma \boldsymbol{P\Phi} - \boldsymbol{\Phi})\boldsymbol{\theta}) \\ \frac{\partial C}{\partial \boldsymbol{\theta}} = \mathbf{0} &\Rightarrow (\gamma \boldsymbol{P\Phi} - \boldsymbol{\Phi})^{\top}\boldsymbol{D}(\boldsymbol{R} + (\gamma \boldsymbol{P\Phi} - \boldsymbol{\Phi})\boldsymbol{\theta}) = \mathbf{0} \end{aligned}$$

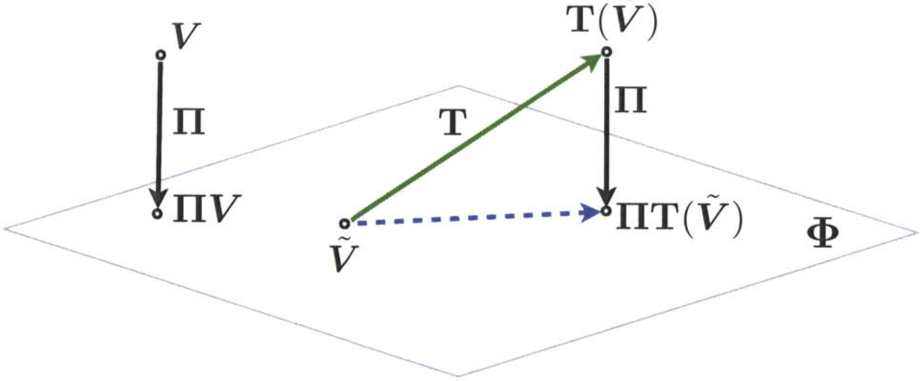


Figure 2-4: A geometric interpretation of what Bellman Residual Minimization (solid green) and Projected Bellman Residual Minimization (dashed blue) minimize (akin to Lagoudakis & Parr, 2003). $\boldsymbol{\Pi}$ is the projection operator, mapping every point to its orthogonal projection on the span of $\boldsymbol{\Phi}$ shown as the 2D plane. $\mathbf{T}$ is the Bellman operator.

$$\begin{aligned} \Rightarrow \boldsymbol{\theta} &= \Bigg[\underbrace{(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})^\top \boldsymbol{D}(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})}_{\boldsymbol{A}_{BRM}}\Bigg]^{-1} \underbrace{(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})^\top \boldsymbol{DR}}_{\boldsymbol{b}_{BRM}} \qquad (2.15) \\ &= \boldsymbol{A}_{BRM}^{-1}\boldsymbol{b}_{BRM} \end{aligned}$$

Note that inverse of $\boldsymbol{A}_{BRM}$ always exists (Scherrer, 2010). A simpler derivation can be obtained by setting the Bellman error to zero in the approximated case:

$$\begin{aligned} \mathbf{T}(\tilde{\boldsymbol{V}}) &= \tilde{\boldsymbol{V}} \\ (\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})\boldsymbol{\theta} &= \boldsymbol{R} \\ (\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})^\top \boldsymbol{D}(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})\boldsymbol{\theta} &= (\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})^\top \boldsymbol{DR} \\ \Rightarrow \boldsymbol{\theta} &= \left[(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})^\top \boldsymbol{D}(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})\right]^{-1} (\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})^\top \boldsymbol{DR} \\ &= \boldsymbol{A}_{BRM}^{-1}\boldsymbol{b}_{BRM} \end{aligned}$$

In the case of a tabular representation, where $\boldsymbol{\Phi} = \boldsymbol{I}$, Equation 2.8 is retrieved:

$$\begin{aligned} \boldsymbol{\theta} &= \left[(\boldsymbol{I} - \gamma \boldsymbol{P})^\top \boldsymbol{D}(\boldsymbol{I} - \gamma \boldsymbol{P})\right]^{-1} (\boldsymbol{I} - \gamma \boldsymbol{P})^\top \boldsymbol{DR} \\ &= (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\boldsymbol{D}^{-1}((\boldsymbol{I} - \gamma \boldsymbol{P})^\top)^{-1}(\boldsymbol{I} - \gamma \boldsymbol{P})^\top \boldsymbol{DR} \\ &= (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\boldsymbol{R}. \end{aligned}$$

The motivation for minimizing the Bellman error stems from the following bound (See

Scherrer, 2010):

$$\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\|^{\boldsymbol{D}}_{\ell_\infty} \leq \frac{1}{1-\gamma}\|\mathbf{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}\|^{\boldsymbol{D}}_{\ell_\infty}.$$

This bound has been also extended to the ℓ_2-norm (Scherrer, 2010):

$$\begin{aligned}\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| &\leq \frac{\sqrt{\kappa(\boldsymbol{D})}}{1-\gamma}\|\mathbf{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}\|,\\ \kappa(\boldsymbol{D}) &= \max_{i,j}\frac{\boldsymbol{P}_{ij}}{\boldsymbol{D}_{ii}},\end{aligned}$$

where $\kappa(\boldsymbol{D})$ represents the stochasticity of the transition model, capturing the deviation of the transition probability distribution from a uniform distribution (Scherrer, 2010).[4]

2.5.2 Projected Bellman Residual Minimization

The second major metric for finding the best approximation stems from minimizing the Projected Bellman Residual vector on the span of $\boldsymbol{\Phi}$, which is shown as the dotted blue vector in Fig. 2-4. The projection operator, $\boldsymbol{\Pi}$, maps each point $\boldsymbol{V} \in \mathbb{R}^{|S|}$ to $\tilde{\boldsymbol{V}} \in span(\boldsymbol{\Phi})$, minimizing the projection norm:

$$\begin{aligned}C(\boldsymbol{\theta}) &= \|\tilde{\boldsymbol{V}} - \boldsymbol{V}\|\\ &= (\tilde{\boldsymbol{V}} - \boldsymbol{V})^\top \boldsymbol{D}(\tilde{\boldsymbol{V}} - \boldsymbol{V})\\ &= (\boldsymbol{\Phi\theta} - \boldsymbol{V})^\top \boldsymbol{D}(\boldsymbol{\Phi\theta} - \boldsymbol{V})\\ &= \boldsymbol{E}^\top \boldsymbol{D}\boldsymbol{E},\end{aligned}$$

$$\begin{aligned}\Rightarrow \frac{\partial C}{\partial \boldsymbol{\theta}} &= \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{\theta}}\frac{\partial C}{\partial \boldsymbol{E}}\\ (\boldsymbol{D}^\top = \boldsymbol{D}) \qquad &= 2\boldsymbol{\Phi}^\top \boldsymbol{D}(\boldsymbol{\Phi\theta} - \boldsymbol{V}),\\ \frac{\partial C}{\partial \boldsymbol{\theta}} = 0 \Rightarrow \boldsymbol{\theta} &= (\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{V},\\ \Rightarrow \tilde{\boldsymbol{V}} &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{V},\\ \Rightarrow \boldsymbol{\Pi} &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}. \qquad (2.16)\end{aligned}$$

It is easy to show that $\boldsymbol{\Pi}$ is a projection matrix by verifying $\boldsymbol{\Pi}^2 = \boldsymbol{\Pi}$:

$$\begin{aligned}\boldsymbol{\Pi}^2 &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\\ &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\end{aligned}$$

[4] More information about $\kappa(\boldsymbol{D})$ can be found in Munos' work (2003)

$$= \Pi$$

Notice that,

$$\Pi\tilde{\boldsymbol{V}} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi}\boldsymbol{\theta} = \tilde{\boldsymbol{V}},$$

meaning that, as expected, the Π projects every point in the $\mathtt{span}(\boldsymbol{\Phi})$ to itself. The LSTD (Bradtke & Barto, 1996) solution minimizes the norm of the projected Bellman error:

$$\begin{aligned}
C(\boldsymbol{\theta}) &= \|\Pi \mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}\| \\
&= \|\Pi(\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}})\| \\
&= \left(\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}\right)^\top \Pi^\top \boldsymbol{D}\Pi \left(\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}\right), \\
\text{(Sutton et al., 2009)} \quad \Pi^\top \boldsymbol{D}\Pi &= (\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D})^\top \boldsymbol{D}\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D} \\
&= \boldsymbol{D}\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D} \\
&= \boldsymbol{D}\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D} \triangleq \mathrm{B},
\end{aligned}$$

$$\begin{aligned}
\Rightarrow C(\boldsymbol{\theta}) &= \left(\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}\right)^\top \mathrm{B} \left(\mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}\right), \\
(\mathrm{B}^\top = \mathrm{B}, \text{BRM derivation}) \quad \Rightarrow \boldsymbol{\theta} &= \left[(\boldsymbol{\Phi} - \gamma \boldsymbol{P}\boldsymbol{\Phi})^\top \mathrm{B}(\boldsymbol{\Phi} - \gamma \boldsymbol{P}\boldsymbol{\Phi})\right]^{-1} (\boldsymbol{\Phi} - \gamma \boldsymbol{P}\boldsymbol{\Phi})^\top \mathrm{B}\boldsymbol{R}, \\
\boldsymbol{L} &\triangleq (\boldsymbol{I} - \gamma \boldsymbol{P}), \\
\boldsymbol{M} &\triangleq \boldsymbol{D}\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi})^{-1} \Rightarrow \mathrm{B} = \boldsymbol{M}\boldsymbol{\Phi}^\top \boldsymbol{D}, \\
\mathrm{X} &\triangleq \boldsymbol{\Phi}^\top \boldsymbol{L}^\top \boldsymbol{M}, \\
\Rightarrow \boldsymbol{\theta} &= [\mathrm{X}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{L}_1\boldsymbol{\Phi}]^{-1}\mathrm{X}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{R} \\
&= [\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{L}_1\boldsymbol{\Phi}]^{-1}\mathrm{X}^{-1}\mathrm{X}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{R} \\
&= [\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{L}_1\boldsymbol{\Phi}]^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{R} \\
&= [\underbrace{\boldsymbol{\Phi}^\top \boldsymbol{D}(\boldsymbol{\Phi} - \gamma \boldsymbol{P}\boldsymbol{\Phi})}_{\boldsymbol{A}_{LSTD}}]^{-1} \underbrace{\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{R}}_{\boldsymbol{b}_{LSTD}} \qquad (2.17) \\
&= \boldsymbol{A}_{LSTD}^{-1}\boldsymbol{b}_{LSTD}.
\end{aligned}$$

$\boldsymbol{A}$ is full rank except for finitely many values of γ (Lagoudakis & Parr, 2003). Notice that again if $\boldsymbol{\Phi} = \boldsymbol{I}$, Equation. 2.8 is retrieved:

$$\begin{aligned}
\boldsymbol{\theta} &= (\boldsymbol{D}\boldsymbol{L})^{-1}\boldsymbol{D}\boldsymbol{R} \\
&= (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\boldsymbol{R}.
\end{aligned}$$

```
Algorithm 3: V-based Approximate Policy Iteration (V-API)              Complexity
Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
Output: $\pi$
$\pi(s) \leftarrow Random(\mathcal{A})$ for $s \in \mathcal{S}$
changed $\leftarrow$ True
while changed do
    Calculate $\boldsymbol{A}$ and $\boldsymbol{b}$ using Equation 2.15 or Equation 2.17    $\Theta(n|\mathcal{S}|^2)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{A}^{-1}\boldsymbol{b}$   (Use regularization if the inverse does not exist)    $\Theta(n^3)$
    $\boldsymbol{V}^\pi \leftarrow \boldsymbol{\Phi\theta}$
    for $s \in \mathcal{S}$ do
        $\pi^+(s) \leftarrow \text{argmax}_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$    $\Theta(|\mathcal{A}||\mathcal{S}|)$
    changed $\leftarrow (\pi^+ \neq \pi)$
    $\pi \leftarrow \pi^+$
return $\pi$
```

The LSTD solution can be derived in a simple form by forcing the Bellman equation in the span of $\boldsymbol{\Phi}$:

$$\begin{aligned}
\tilde{\boldsymbol{V}} &= \Pi \mathrm{T}(\tilde{\boldsymbol{V}}), \\
\boldsymbol{\Phi\theta} &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \boldsymbol{D\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}(\boldsymbol{R} + \gamma \boldsymbol{P\Phi\theta}), \\
\boldsymbol{\theta} &= (\boldsymbol{\Phi}^\top \boldsymbol{D\Phi})^{-1}\boldsymbol{\Phi}^\top \boldsymbol{D}(\boldsymbol{R} + \gamma \boldsymbol{P\Phi\theta}), \\
(\boldsymbol{\Phi}^\top \boldsymbol{D\Phi})\boldsymbol{\theta} &= \boldsymbol{\Phi}^\top \boldsymbol{D}(\boldsymbol{R} + \gamma \boldsymbol{P\Phi\theta}), \\
\left(\boldsymbol{\Phi}^\top \boldsymbol{D}(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})\right)\boldsymbol{\theta} &= \boldsymbol{\Phi}^\top \boldsymbol{DR}, \\
\boldsymbol{\theta} &= [\boldsymbol{\Phi}^\top \boldsymbol{D}(\boldsymbol{\Phi} - \gamma \boldsymbol{P\Phi})]^{-1}\boldsymbol{\Phi}^\top \boldsymbol{DR} \\
&= \boldsymbol{A}^{-1}_{LSTD}\boldsymbol{b}_{LSTD}.
\end{aligned}$$

While the LSTD solution can be far from $\boldsymbol{V}$, it has been shown that the approximation error is bounded (Parr et al., 2007):

$$||\boldsymbol{V} - \tilde{\boldsymbol{V}}|| \leq \frac{1}{\sqrt{1-\gamma^2}}||\boldsymbol{V} - \Pi\boldsymbol{V}||.$$

2.5.3 Using BRM and LSTD for Control

So far, BRM and LSTD were introduced as two approaches for policy evaluation. This section describes how these two methods can be integrated with the policy improvement step to solve MDPs. The first immediate algorithm shown in Algorithm 3 is the result of using Equation 2.9 as the policy improvement step. By comparing Equations 2.15 and

2.17 with Equation 2.8, one can see the benefit of using approximation methods over the tabular representation: the policy evaluation phase now requires $\Theta(n|\mathcal{S}|^2+n^3) = \Theta(n|\mathcal{S}|^2)$ computation.[5] Note that calculating the new policy still requires $\Theta(|\mathcal{A}||\mathcal{S}|^2)$ computations. Moreover calculating the steady state distribution ($\boldsymbol{D}$) is not always trivial (See Chapter 4 of Gallager, 2009).

Thus, compared to Algorithm 2, Algorithm 3 is not attractive, because it 1) requires additional computational complexity of $\Theta(n|\mathcal{S}|^2)$, 2) requires extra calculation of D, and 3) finds an approximate solution. In order to resolve this issue, the policy improvement must be calculated with complexity less than $\Theta(|\mathcal{A}||\mathcal{S}|^2)$. This can be done by using Equation 2.2. If the policy evaluation phase calculates the $\boldsymbol{Q}^\pi$ rather than the $\boldsymbol{V}^\pi$, then the policy improvement requires $\Theta(|\mathcal{A}||\mathcal{S}|)$.

Now consider the extra calculation involved in computing $\boldsymbol{Q}^\pi$ rather than $\boldsymbol{V}^\pi$ using linear function approximation. Similar to the derivation of Equation 2.6, Q^π can be written recursively as:

$$Q^\pi(s,a) = \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma Q^\pi(s', \pi(s'))\right]. \tag{2.18}$$

Given that $\mathcal{A} = \{a_1, \cdots, a_{|\mathcal{A}|}\}$, the above equation can be written in the matrix form. Symbols with the bar sign hold extra information required to calculate the Q values rather than the V values:

$$\boldsymbol{Q} = \bar{\boldsymbol{R}} + \gamma \bar{\boldsymbol{P}}\boldsymbol{Q},$$

where,

$$\boldsymbol{Q}_{|\mathcal{S}||\mathcal{A}|\times 1} = \begin{bmatrix} Q(s_1,a_1) \\ \vdots \\ Q(s_1,a_{|\mathcal{A}|}) \\ \vdots \\ Q(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \end{bmatrix} = \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\theta}},$$

$$\bar{\boldsymbol{\Phi}}_{|\mathcal{S}||\mathcal{A}|\times n|\mathcal{A}|} = \begin{bmatrix} \text{---}\ \phi^\top(s_1,a_1)\ \text{---} \\ \vdots \\ \text{---}\ \phi^\top(s_1,a_{|\mathcal{A}|})\ \text{---} \\ \vdots \\ \text{---}\ \phi^\top(s_{|\mathcal{S}|},a_{|\mathcal{A}|})\ \text{---} \end{bmatrix}, \bar{\boldsymbol{R}}_{|\mathcal{S}||\mathcal{A}|\times 1} = \begin{bmatrix} \mathcal{R}^{a_1}_{s_1} \\ \vdots \\ \mathcal{R}^{a_{|\mathcal{A}|}}_{s_1} \\ \vdots \\ \mathcal{R}^{a_{|\mathcal{A}|}}_{s_{|\mathcal{S}|}} \end{bmatrix},$$

[5] Calculating $\boldsymbol{A}$ requires $\Theta(n|\mathcal{S}|^2)$ due to $\boldsymbol{P\Phi}$.

$$\bar{\boldsymbol{P}}_{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}||\mathcal{A}|} = \begin{bmatrix} \mathcal{P}(s_1,a_1,s_1,a_1) & \cdots & \mathcal{P}(s_1,a_1,s_1,a_{|\mathcal{A}|}) & \cdots & \mathcal{P}(s_1,a_1,s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \\ \vdots & & \vdots & & \vdots \\ \mathcal{P}(s_1,a_{|\mathcal{A}|},s_1,a_1) & \cdots & \mathcal{P}(s_1,a_{|\mathcal{A}|},s_1,a_{|\mathcal{A}|}) & \cdots & \mathcal{P}(s_1,a_{|\mathcal{A}|},s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \\ \vdots & & \vdots & & \vdots \\ \mathcal{P}(s_{|\mathcal{S}|},a_{|\mathcal{A}|},s_1,a_1) & \cdots & \mathcal{P}(s_{|\mathcal{S}|},a_{|\mathcal{A}|},s_1,a_{|\mathcal{A}|}) & \cdots & \mathcal{P}(s_{|\mathcal{S}|},a_{|\mathcal{A}|},s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \end{bmatrix},$$

$$\begin{aligned} \mathcal{R}^a_s &= \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'}\mathcal{R}^a_{ss'}, \\ \mathcal{P}(s_i,a_j,s_k,a_l) &= \begin{cases} \mathcal{P}^{a_j}_{s_is_k} & \text{If } a_l = \pi(s_k) \\ 0 & \text{Otherwise} \end{cases}. \end{aligned}$$

In the tabular case:

$$\boldsymbol{Q} = (\boldsymbol{I} - \gamma\bar{\boldsymbol{P}})^{-1}\bar{\boldsymbol{R}}.$$

By extending the definition of steady state distribution in Equations 2.12 and 2.14 to include the $\bar{\boldsymbol{P}}$ matrix, $\bar{\boldsymbol{D}}$ is $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|$. Hence all derivations of LSTD and BRM remain intact, calculating $\boldsymbol{Q}^\pi$ instead of $\boldsymbol{V}^\pi$:

$$\begin{aligned} \bar{\boldsymbol{A}}_{BRM} &= (\bar{\boldsymbol{\Phi}} - \gamma\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}})^\top\bar{\boldsymbol{D}}(\bar{\boldsymbol{\Phi}} - \gamma\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}), \bar{\boldsymbol{b}}_{BRM} = (\bar{\boldsymbol{\Phi}} - \gamma\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}})^\top\bar{\boldsymbol{D}}\bar{\boldsymbol{R}} && (2.19) \\ \bar{\boldsymbol{A}}_{LSTD} &= \bar{\boldsymbol{\Phi}}^\top\bar{\boldsymbol{D}}(\bar{\boldsymbol{\Phi}} - \gamma\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}), \bar{\boldsymbol{b}}_{LSTD} = \bar{\boldsymbol{\Phi}}^\top\bar{\boldsymbol{D}}\bar{\boldsymbol{R}} && (2.20) \\ \bar{\boldsymbol{\theta}} &= \bar{\boldsymbol{A}}^{-1}\bar{\boldsymbol{b}} \\ \boldsymbol{Q}^\pi &= \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\theta}} \end{aligned}$$

Algorithm 4 shows the result of integrating these policy evaluation schemes with policy improvement according to Equation 2.2. While the new formulation reduces the cost of policy improvement from $\Theta(|\mathcal{S}||\mathcal{A}|^2)$ to $\Theta(n|\mathcal{S}|)$, it still increases the cost of policy evaluation from $\Theta(n||\mathcal{S}|^2)$ to $\Theta(n|\mathcal{S}||\mathcal{A}|^2)$ due to the calculation of $\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}$.[6] Moreover, using approximation techniques to represent Q instead of V, and performing the Bellman update only increases the computational complexity of the value iteration shown in Algorithm 2.[7]

In order to further reduce the complexity of each iteration, sampling methods are used

[6] In general this calculation requires $\Theta(n|\mathcal{S}|^2|\mathcal{A}|^3)$, but because of the deterministic policy assumption and the way $\phi(s,a)$ is constructed out of $\phi(s)$ (Equation 2.11), each row of $\bar{\boldsymbol{P}}$ and $\bar{\boldsymbol{\Phi}}$ has at most $|\mathcal{S}|$ and n non-zero elements correspondingly. Hence the complexity drop to $\Theta(n|\mathcal{S}|^2|\mathcal{A}|)$.

[7] Calculating the value of each state requires $\Theta(n)$ rather than $\Theta(1)$, hence the complexity of each iteration is raised to $\Theta(n|\mathcal{S}|^2|\mathcal{A}|)$.

Algorithm 4: Q-based Approximate Policy Iteration (Q-API) — Complexity

```
Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
Output: $\pi$
$\pi(s) \leftarrow Random(\mathcal{A})$ for $s \in \mathcal{S}$
changed $\leftarrow$ True
while changed do
    Calculate $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{b}}$ using Equation 2.19 or Equation 2.20        $\Theta(n|\mathcal{A}||\mathcal{S}|^2)$
    $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{A}}^{-1}\bar{\boldsymbol{b}}$   (Use regularization if the inverse does not exist)   $\Theta(n^3|\mathcal{A}|^3)$
    $\boldsymbol{Q}^{\pi} \leftarrow \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\theta}}$
    for $s \in \mathcal{S}$ do
        $\pi^+(s) \leftarrow \text{argmax}_{a\in\mathcal{A}} Q^{\pi}(s,a)$        $\Theta(|\mathcal{A}|)$
    changed $\leftarrow (\pi^+ \neq \pi)$
    $\pi \leftarrow \pi^+$
return $\pi$
```

to estimate values of $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{b}}$ for both BRM and LSTD approaches (*e.g.,* Lagoudakis & Parr, 2003). Given the fixed policy π, samples are gathered in the form of $\langle s_i, a_i\rangle, i \in \{1, \cdots, L_1\}$, where s_1 is the initial state, $a_i = \pi(s_i)$, and $s_{i+1} \sim \mathcal{P}^{a_i}_{s_i}$. This generative process assures that in the limit of infinite samples, averaged state visitations will converge to the steady state distribution under policy π (Lagoudakis & Parr, 2003). Let us estimate $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{b}}$ in Equations 2.19 and 2.20 using L_1 obtained samples:

$$
\begin{aligned}
\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}_{L_1\times n|\mathcal{A}|} &= \begin{bmatrix} \text{---}\ \varphi^{\top}(s_1,a_1)\ \text{---} \\ \text{---}\ \varphi^{\top}(s_2,a_2)\ \text{---} \\ \vdots \\ \text{---}\ \varphi^{\top}(s_{L_1},a_{L_1})\ \text{---} \end{bmatrix}, \\
\varphi(s_i,a_i) &= \sum_{s'_i\in\mathcal{S}} \mathcal{P}^{a_i}_{s_i s'_i}\phi(s'_i, \pi(s'_i)). \qquad (2.21) \\
\widetilde{\bar{\boldsymbol{R}}}_{L_1\times 1} &= \begin{bmatrix} \rho(s_1,a_1) \\ \rho(s_2,a_2) \\ \vdots \\ \rho(s_{L_1},a_{L_1}) \end{bmatrix}, \\
\rho(s_i,a_i) &= \sum_{s'_i\in\mathcal{S}} \mathcal{P}^{a_i}_{s_i s'}\mathcal{R}^{a_i}_{s_i s'_i}.
\end{aligned}
$$

Approximating the feature matrix by

$$\tilde{\bar{\boldsymbol{\Phi}}}_{L_1 \times n|\mathcal{A}|} = \begin{bmatrix} \text{---}\,\phi^{\mathsf{T}}(s_1, a_1)\,\text{---} \\ \text{---}\,\phi^{\mathsf{T}}(s_2, a_2)\,\text{---} \\ \vdots \\ \text{---}\,\phi^{\mathsf{T}}(s_{L_1}, a_{L_1})\,\text{---} \end{bmatrix}, \tag{2.22}$$

the approximated $\boldsymbol{A}$ and $\boldsymbol{b}$ are:

$$\begin{aligned} \tilde{\bar{\boldsymbol{A}}}_{LSTD} &= \frac{1}{L_1}\tilde{\bar{\boldsymbol{\Phi}}}^{\mathsf{T}}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\boldsymbol{P}\bar{\boldsymbol{\Phi}}}) && (2.23)\\ \tilde{\bar{\boldsymbol{b}}}_{LSTD} &= \frac{1}{L_1}\tilde{\bar{\boldsymbol{\Phi}}}^{\mathsf{T}}\tilde{\bar{\boldsymbol{R}}} && (2.24)\\ \tilde{\bar{\boldsymbol{A}}}_{BRM} &= \frac{1}{L_1}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\boldsymbol{P}\bar{\boldsymbol{\Phi}}})^{\mathsf{T}}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\boldsymbol{P}\bar{\boldsymbol{\Phi}}}) && (2.25)\\ \tilde{\bar{\boldsymbol{b}}}_{BRM} &= \frac{1}{L_1}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\boldsymbol{P}\bar{\boldsymbol{\Phi}}})^{\mathsf{T}}\tilde{\bar{\boldsymbol{R}}}. && (2.26) \end{aligned}$$

As the number of samples goes to infinity, the estimated values become exact if the policy is *ergodic*, meaning that in the limit all state-action pairs are visited infinitely often (Lagoudakis & Parr, 2003):

$$\begin{aligned} \lim_{L_1\to\infty} \tilde{\bar{\boldsymbol{A}}}_{LSTD} &= \bar{\boldsymbol{A}}_{LSTD}\\ \lim_{L_1\to\infty} \tilde{\bar{\boldsymbol{b}}}_{LSTD} &= \bar{\boldsymbol{b}}_{LSTD}\\ \lim_{L_1\to\infty} \tilde{\bar{\boldsymbol{A}}}_{BRM} &= \bar{\boldsymbol{A}}_{BRM}\\ \lim_{L_1\to\infty} \tilde{\bar{\boldsymbol{b}}}_{BRM} &= \bar{\boldsymbol{b}}_{BRM}. \end{aligned}$$

Figure 2-5 demonstrates the effect of non-ergodic policies through an example. The MDP has 3 states and 2 actions. Assume that $\pi(s_1) = a_1$ and $\pi(s_2) = a_2$. Rewards are highlighted as scalars on the right side of action labels located on arrows. Samples gathered by following π will not go through any of gray arrows, excluding the $+10$ reward. Hence using a tabular representation, matrix $\tilde{\bar{\boldsymbol{\Phi}}}$ will have only two distinct rows[8] corresponding to

[8] While more samples can have been obtained (*i.e.*, $L_1 > 2$), yet the resulting value function does not change due to the use of the tabular representation.

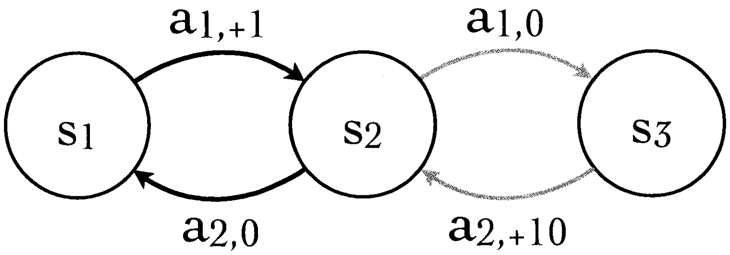


Figure 2-5: An MDP highlighting the importance of ergodicity for the policy. Gray actions are not chosen by the policy. The result of policy evaluation assigns highest values to $Q(s_1, a_1)$ and $Q(s_2, a_2)$. The greedy policy with respect to this value function is sub-optimal because it ignores the $+10$ reward.

$\phi(s_1, a_1)^\top$ and $\phi(s_2, a_2)^\top$:

$$\tilde{\tilde{\boldsymbol{\Phi}}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{similarly} \quad \widetilde{\boldsymbol{P}\tilde{\boldsymbol{\Phi}}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\Rightarrow \tilde{\tilde{\boldsymbol{A}}}_{LSTD} = \begin{bmatrix} 0.5000 & 0 & 0 & 0 & -0.4500 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.4500 & 0 & 0 & 0 & 0.5000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tilde{\tilde{\boldsymbol{b}}}_{LSTD} = \begin{bmatrix} 0.5000 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Given $\gamma = 0.9$, and using regularization,

$$\boldsymbol{Q} = \tilde{\tilde{\boldsymbol{A}}}_{LSTD}^{-1} \tilde{\tilde{\boldsymbol{b}}}_{LSTD} = \begin{bmatrix} 5.2532 \\ 0 \\ 0 \\ 0 \\ 4.7269 \\ 0 \end{bmatrix}.$$

While $\tilde{\tilde{\boldsymbol{A}}}_{LSTD}$ is not full-rank, any non-zero regularization value leads to zero value estimates for all unseen state-action pairs. Applying policy improvement on the new Q function leads to the same policy π, which is sub-optimal, as it ignores the $+10$ reward. A workaround for this problem is to make sure that the agent keeps *exploring* by using a pol-

Algorithm 5: Sampled Q-API-1 — Complexity

```
   Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
   Output: $\pi$
 1 $\bar{\boldsymbol{\theta}} \leftarrow Random()$
 2 $\pi(s) \leftarrow Random(\mathcal{A})$ for $s \in \mathcal{S}$
 3 changed $\leftarrow$ True
 4 while changed do
 5     Create $L_1$ samples $\langle s_i, a_i \rangle$ by following policy $\pi^\epsilon$              $\Theta(L_1)$
 6     Calculate $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$ using Equations 2.23-2.26                  $\Theta(nL_1|\mathcal{S}|)$
 7     $\bar{\boldsymbol{\theta}} \leftarrow \tilde{\bar{\boldsymbol{A}}}^{-1}\tilde{\bar{\boldsymbol{b}}}$   (Use regularization if the inverse does not exist)   $\Theta(n^3|\mathcal{A}|^3)$
 8     $\boldsymbol{Q}^\pi \leftarrow \tilde{\bar{\bar{\boldsymbol{\Phi}}}}\bar{\boldsymbol{\theta}}$
 9     for $s \in \mathcal{S}$ do
10         $\pi^+(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} Q^\pi(s, a)$                           $\Theta(|\mathcal{A}|)$
11     changed $\leftarrow (\pi^+ \neq \pi)$
12     $\pi \leftarrow \pi^+$
13 return $\pi$
```

Table 2.1: Computational Complexity of Calculating $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$

Calculation	Complexity	Comment
$\pi(s_i)$	$\Theta(1)$	Policy is saved explicitly.
$\varphi(s_i, a_i)$	$\Theta(n\|\mathcal{S}\|)$	$\phi(s, a)$ has at most n non-zero elements (Equation 2.11).
$\widetilde{\bar{\boldsymbol{P}}\bar{\bar{\boldsymbol{\Phi}}}}$	$\Theta(nL_1\|\mathcal{S}\|)$	
$\tilde{\bar{\bar{\boldsymbol{\Phi}}}}^\top(\tilde{\bar{\bar{\boldsymbol{\Phi}}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\bar{\boldsymbol{\Phi}}}})$	$\Theta(n^2L_1)$	Both $\tilde{\bar{\bar{\boldsymbol{\Phi}}}}$ and $(\tilde{\bar{\bar{\boldsymbol{\Phi}}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\bar{\boldsymbol{\Phi}}}})$ have $\Theta(n)$ non-zero columns.
$\tilde{\bar{\boldsymbol{A}}}_{LSTD}, \tilde{\bar{\boldsymbol{A}}}_{BRM}$	$\Theta(nL_1\|\mathcal{S}\|)$	Assuming $n \ll \|\mathcal{S}\|$.
$\tilde{\bar{\boldsymbol{b}}}_{LSTD}, \tilde{\bar{\boldsymbol{b}}}_{BRM}$	$\Theta(nL_1\|\mathcal{S}\|)$	

icy that is ergodic at all times. There are various ways to facilitate ergodicity; the easiest one is to introduce a random action with a small probability on every action selection. This mechanism is known as ϵ-greedy action selection, where ϵ is the probability of taking a random action. Hence an ϵ-greedy policy is defined as:

$$\pi^\epsilon(s) = \begin{cases} \text{argmax}_a Q^\pi(s, a) & \text{With probability } 1 - \epsilon \\ \texttt{Random}(\mathcal{A}) & \text{With probability } \epsilon \end{cases}. \tag{2.27}$$

Algorithm 5 reflects the use of the above policy evaluation scheme for solving an MDP. Table 2.1 shows the computational complexity involved in calculating $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$ (line 6) in detail. The total complexity for each iteration of Algorithm 5 is $\Theta(nL_1|\mathcal{S}| + |\mathcal{S}||\mathcal{A}| +$

```
Algorithm 6: Sampled Q-API-2                                                  Complexity
  Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
  Output: $\pi$
1 $\bar{\boldsymbol{\theta}} \leftarrow Random()$
2 while time left do
3     Create $L_1$ samples $\langle s_i, a_i \rangle$ following policy $\pi^\epsilon$          $\Theta(nL_1|\mathcal{A}|)$
4     Calculate $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$ using Equation 2.19 or Equation 2.20     $\Theta(nL_1L_2 + n^2L_1)$
5     $\bar{\boldsymbol{\theta}} \leftarrow \tilde{\bar{\boldsymbol{A}}}^{-1}\tilde{\bar{\boldsymbol{b}}}$   (Use regularization if the inverse does not exist)    $\Theta(n^3|\mathcal{A}|^3)$
6     $\boldsymbol{Q}^\pi \leftarrow \tilde{\bar{\boldsymbol{\Phi}}}\bar{\boldsymbol{\theta}}$
7 return $\pi$ greedy w.r.t. $Q$
```

$n^3|\mathcal{A}|^3)$. Note that the complexity is linear in $|\mathcal{S}|$ as opposed to all previous algorithms which were at least quadratic in $|\mathcal{S}|$. To eliminate the dependency on $|\mathcal{S}|$ all together, two steps are mandatory: 1) φ in Equation 2.21 should be calculated using sampling techniques, and 2) The policy must not be calculated explicitly for all states. To realize the first requirement, φ can be calculated using L_2 samples[9] instead of the exact expectation:

$$\varphi(s_i, a_i) \approx \frac{1}{L_2} \sum_{j \in \{1, \cdots, L_2\}} \phi(s_j, \pi(s_j)), \quad s_j \sim \mathcal{P}^{a_i}_{s_i}. \tag{2.28}$$

By calculating the policy on demand using Equation 2.2, the second requirement is satisfied, although the implicit policy improvement increases the complexity involved in calculating $\pi(s)$ from $\Theta(1)$ to $\Theta(n|\mathcal{A}|)$. Algorithm 6 shows the resulting algorithm. Notice that the condition on the loop on line 2 can no longer depend on maintaining a fixed policy for all states. An alternative way to break the loop is to check if the policy for a fixed set of states remains unchanged after one iteration. The computational complexity of each iteration of Algorithm 6 is $\Theta(nL_1|\mathcal{A}| + nL_1L_2 + n^2L_1 + n^3|\mathcal{A}|^3)$.

All improvements so far were made to the Policy Iteration method to reduce its computational complexity, so it is natural to think about improving the complexity of Value Iteration. The first step is to integrate function approximation with the algorithm. Consider Value Iteration as shown in Algorithm 2. While the new value function can be calculated at line 6 using $V(s) = \phi(s)^T\boldsymbol{\theta}$, there is no guarantee that the new value function lies in the span of $\boldsymbol{\Phi}$. There are two ways to mitigate this problem: 1) use a gradient descent technique to update $\boldsymbol{\theta}$, in the direction which reduces the error between the current value estimate and the resulting value estimate after applying the Bellman update or 2) project

[9] As a reminder, L_1 is the length of the trajectory simulated to approximate $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{b}}$. L_2 is the number of states sampled after taking action a_i at state s_i to estimate $\varphi(s_i, a_i)$.

the new calculated value estimate in the span of $\mathbf{\Phi}$ using orthogonal projection (*i.e.,* linear regression). Both these approaches have been used in the literature (Szepesvári & Munos, 2005; Wu & Givan, 2007).

Consider the gradient type methods through a mathematical example. Given a vector $\mathbf{X}$ and its approximation $\tilde{\mathbf{X}} = \mathbf{\Phi}\boldsymbol{\theta}$, how can $\boldsymbol{\theta}$ change slightly in order to get $\tilde{\mathbf{X}}$ closer to $\mathbf{X}$ in the ℓ_2-norm? The answer is to move it in the opposite direction of the gradient:

$$\begin{aligned} C &= ||\mathbf{X} - \tilde{\mathbf{X}}|| = ||\mathbf{X} - \mathbf{\Phi}\boldsymbol{\theta}|| = (\mathbf{X} - \mathbf{\Phi}\boldsymbol{\theta})^\top(\mathbf{X} - \mathbf{\Phi}\boldsymbol{\theta}), \\ \frac{\partial C}{\partial \boldsymbol{\theta}} &= -2(\mathbf{X} - \mathbf{\Phi}\boldsymbol{\theta})\mathbf{\Phi} = -2(\mathbf{X} - \tilde{\mathbf{X}})\mathbf{\Phi}, \\ \Rightarrow \boldsymbol{\theta} &= \boldsymbol{\theta} + \alpha(\mathbf{X} - \tilde{\mathbf{X}})\mathbf{\Phi}. \end{aligned} \tag{2.29}$$

The α parameter is the learning rate . If the $\mathbf{X}$ and $\tilde{\mathbf{X}}$ vectors only differ in their i^{th} element, the above gradient decent is simplified as follows:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha\Big(\mathbf{X}(i) - \tilde{\mathbf{X}}(i)\Big)\boldsymbol{e}_i^\top\mathbf{\Phi}, \tag{2.30}$$

where $\boldsymbol{e}_i$ is a $|\mathcal{S}| \times 1$ vector with all zero elements except for a 1 for its i^{th} element. Hence $\boldsymbol{e}_i^\top\mathbf{\Phi}$ is the i^{th} row of the $\mathbf{\Phi}$ matrix. Algorithm 7 shows the integration of the above gradient descent idea with value iteration. Note that the algorithm also shifts from calculating V values to Q values to reduce the computational complexity of maintaining the optimal policy on each iteration. Line 5 calculates the single element of the $\boldsymbol{Q}$ vector, Q^+, which is changed. Line 7 moves the $\bar{\boldsymbol{\theta}}$ following Equation 2.30. The row of $\mathbf{\Phi}$ corresponding to the changed value is $\phi(s,a)$. Unfortunately this algorithm increases the iteration complexity of Value Iteration, as it now requires $\Theta(n|\mathcal{A}|^2|\mathcal{S}|^2)$ computation.[10] Yet, this formulation allows for the use of sampling methods to further reduce the iteration complexity.

Similar to the Policy Iteration extensions (Algorithms 5 and 6), sampling can be used both for (1) applying the update on line 5 selectively by obtaining samples in the form $\langle s_i, a_i \rangle$ instead of looping through all state-action pairs (lines 3 and 4), and (2) calculating $Q^+(s,a)$ approximately on line 5. Algorithm 8 is the result of both these extensions (Wu & Givan, 2007) which eliminates the dependency on the size of the state space all together. Notice that, again, to make sure all state-action pairs are seen infinitely often the policy on line 3 is ergodic through the use of ϵ.

The above extensions of Value Iteration to incorporate linear function approximation used a gradient descent technique. Another approach is to find the minimum of the cost

[10] Calculating $Q(s,a) = \phi(s,a)^\top\boldsymbol{\theta}$, requires $\Theta(n)$ computation including the sparsity of $\phi(s,a)$.

```
Algorithm 7: Q-Based Approximate Value Iteration (Q-AVI)                          Complexity
Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
while time left do
    for $s \in \mathcal{S}$ do
        for $a \in \mathcal{A}$ do
            $Q^+(s,a) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q(s',a')]$      $\Theta(n|\mathcal{A}||\mathcal{S}|)$
            $\delta \leftarrow Q^+(s,a) - Q(s,a)$
            $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \alpha\delta\phi(s,a)$                              $\Theta(n)$
return $\pi$ greedy w.r.t. $Q$
```

```
Algorithm 8: Sampled Q-AVI                                                         Complexity
Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
while time left do
    Create $L_1$ samples $\langle s_i, a_i \rangle$ following policy $\pi^\epsilon$              $\Theta(nL_1|\mathcal{A}|)$
    for every sample $\langle s_i, a_i \rangle$ do
        Create $L_2$ samples $s'_i \sim \mathcal{P}^{a_i}_{s_i}$
        $Q^+(s_i,a_i) \leftarrow \frac{1}{L_2} \sum_{s'_i \in Samples} \mathcal{R}^{a_i}_{s_i s'_i} + \gamma \max_{a'_i} Q(s'_i, a'_i)$     $\Theta(nL_2|\mathcal{A}|)$
        $\delta \leftarrow Q^+(s,a) - Q(s,a)$
        $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \alpha\delta\phi(s,a)$                              $\Theta(n)$
return $\pi$ greedy w.r.t. $Q$
```

function mentioned in Equation 2.29 using linear regression:

$$\begin{aligned}
\frac{\partial C}{\partial \boldsymbol{\theta}} &= -2(\mathbf{X} - \boldsymbol{\Phi}\boldsymbol{\theta})\boldsymbol{\Phi} = 0 \\
\boldsymbol{\Phi}\boldsymbol{\theta} &= \mathbf{X} \\
\boldsymbol{\Phi}^\intercal\boldsymbol{\Phi}\boldsymbol{\theta} &= \boldsymbol{\Phi}^\intercal\mathbf{X} \\
\Rightarrow \boldsymbol{\theta} &= (\boldsymbol{\Phi}^\intercal\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\intercal\mathbf{X} \qquad (2.31)
\end{aligned}$$

The resulting $\tilde{\mathbf{X}} = \boldsymbol{\Phi}\boldsymbol{\theta}$ is the orthogonal projection of $\mathbf{X}$ in the column span of $\boldsymbol{\Phi}$. This will bring us to an algorithm known as Fitted Value Iteration (Justin Boyan, 1995) shown in Algorithm 9. The only change, compared to Algorithm 8, is the use of Equation 2.31 to find the best $\boldsymbol{\theta}$ rather than taking gradient steps for each sample. This new change increases the computational complexity of the method by $\Theta(n^2L_1 + n^3|\mathcal{A}|^3)$ due to $\widetilde{\bar{\boldsymbol{\Phi}}}^\intercal \widetilde{\bar{\boldsymbol{\Phi}}}$ calcula-

```
Algorithm 9:Fitted Value Iteration (FVI)                                                   Complexity
Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
while time left do
    Create $L_1$ samples $\langle s_i, a_i \rangle$ following policy $\pi^\epsilon$                  $\Theta(nL_1|\mathcal{A}|)$
    for every sample $\langle s_i, a_i \rangle$ do
        Create $L_2$ samples $s'_i \sim \mathcal{P}^{a_i}_{s_i}$
        $Q^+(s_i, a_i) \leftarrow \frac{1}{L_2}\sum_{s'_i \in Samples} \mathcal{R}^{a_i}_{s_i s'_i} + \gamma \max_{a'_i} Q(s'_i, a'_i)$      $\Theta(nL_2|\mathcal{A}|)$
    $\bar{\boldsymbol{\theta}} \leftarrow (\tilde{\tilde{\bar{\boldsymbol{\Phi}}}}^\top \tilde{\tilde{\bar{\boldsymbol{\Phi}}}})^{-1} \tilde{\tilde{\bar{\boldsymbol{\Phi}}}}^\top \boldsymbol{Q}^+$                       $\Theta(n^2 L_1 + n^3|\mathcal{A}|^3)$
return $\pi$ greedy w.r.t. $Q$
```

tion and matrix inversion respectively[11]. While this increase would seem to be a major drawback with respect to Algorithm 7, the extra computational cost provides a closed form solution to finding the minimum projection error that often translates into better policies after applying the policy improvement step. Moreover, several probabilistic bounds has been derived for FVI's performance with limited numbers of samples (Munos & Szepesvári, 2008; Szepesvári & Munos, 2005).

2.5.4 Discussion

Table 2.2 provides an overview of the model-based MDP solvers discussed in this chapter together with their iteration computational complexity. One might wonder which of these algorithms are better than the others? The answer is domain-dependent, as these methods trade off computational complexity with accuracy. Often the size of the state space, $|\mathcal{S}|$, for real-world problems is very large, which eliminates methods with computational complexity dependent on $|\mathcal{S}|$. However, more assumptions about a certain domain can help practitioners reducing the above complexities further. Also note that the main concern for using approximation was to eliminate the dependency on $|\mathcal{S}|$. While often $|\mathcal{A}| \ll |\mathcal{S}|$, for some domains (*e.g.*, controlling a helicopter with multi-dimensional continuous action space), having a dependency on $|\mathcal{A}|$ is not sustainable either. Hence further approximation is required to eliminate $|\mathcal{A}|$ from the above complexities. Finally it is critical to be reminded that this table merely states the iteration complexity of each method, yet it does not say anything about the number of iterations required to obtain a reasonable policy. While some methods might require more time to finish one iteration, they may require fewer iterations in total to find good policies. Recent methods combined the idea of BRM and LSTD

[11] As a reminder $\tilde{\tilde{\bar{\boldsymbol{\Phi}}}}$ is the features matrix defined in Equation 2.22.

Table 2.2: Model-based MDP solvers and their iteration computational complexity.

Algorithm	Iteration Complexity	Algorithm
Policy Iteration	$\Theta(\|\mathcal{S}\|^3 + \|\mathcal{A}\|\|\mathcal{S}\|^2)$	1
V-API	$\Theta(n\|\mathcal{S}\|^2 + \|\mathcal{A}\|\|\mathcal{S}\|^2)$	3
Q-API	$\Theta(n\|\mathcal{A}\|\|\mathcal{S}\|^2 + n^3\|\mathcal{A}\|^3)$	4
Sampled Q-API-1	$\Theta(nL_1\|\mathcal{S}\| + \|\mathcal{S}\|\|\mathcal{A}\| + n^3\|\mathcal{A}\|^3)$	5
Sampled Q-API-2	$\Theta(nL_1\|\mathcal{A}\| + nL_1L_2 + n^2L_1 + n^3\|\mathcal{A}\|^3)$	6
Value Iteration	$\Theta(\|\mathcal{A}\|\|\mathcal{S}\|^2)$	2
Q-AVI	$\Theta(n\|\mathcal{A}\|^2\|\mathcal{S}\|^2)$	7
Sampled Q-AVI	$\Theta(nL_1L_2\|\mathcal{A}\|)$	8
FVI	$\Theta(nL_1L_2\|\mathcal{A}\| + n^3\|\mathcal{A}\|^3)$	9

with kernelized representations to achieve good approximation of the value function with small amount of data (Bethke & How, 2009; Taylor & Parr, 2009).

2.6 Reinforcement Learning

In practical domains, often the MDP models (*i.e.,* $\mathcal{P}$ and $\mathcal{R}$) are not known and even for some domains the dynamics of the system are too complicated to be captured analytically. This problem setting can be addressed through the Reinforcement Learning (RL) framework shown in Figure 2-6, where an agent (left) interacts with the environment (right) on each time step, using a deterministic policy.[12] The goal of RL methods is to solve MDPs with unknown models solely by means of interacting with the environment. At first, RL might look very different from model-based MDP solvers discussed in the previous section. Yet, as we will see in this section, RL techniques can be seen as approximate dynamic programming methods where samples cannot be generated arbitrarily

RL methods are organized similar to model-based MDP solvers: 1) value-based methods (*e.g.,* Rummery & Niranjan, 1994) and 2) policy search techniques (*e.g.,* Bhatnagar et al., 2007). Since this thesis focuses on the first category, this section covers mostly value-based RL techniques. Again, the core idea of value-based techniques is to follow the policy evaluation-policy improvement loop, shown in Figure 2-1. Before continuing further, let us step back and see what extra restrictions an RL agent has to overcome compared to model-based methods:

- **Restriction I:** Since sampling from the environment is only available through inter-

[12] In general the policy can be stochastic.

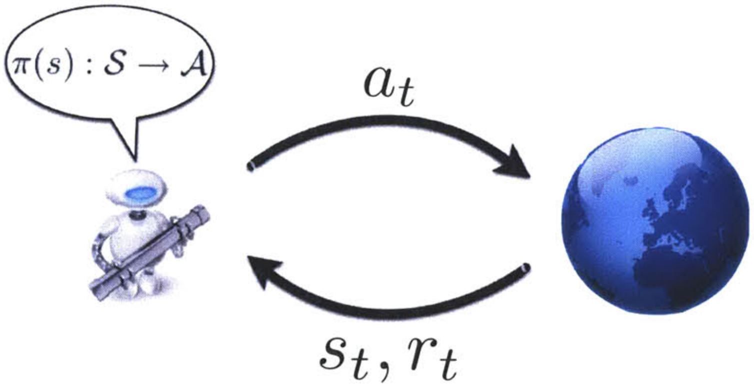


Figure 2-6: An agent interacting with an environment following policy π. The agent does not know the underlying reward or transition models of the MDP.

action, samples can only be gathered in the form of trajectories. For example in a navigation task, the agent cannot ask the environment about the consequence of turning left at arbitrary junctions; rather it can only obtain samples based on its current location.

- **Restriction II:** In the RL setting, one-step look-ahead (*i.e.,* Equation 2.9) cannot be used for policy improvement since it requires knowledge of both $\mathcal{P}$ and $\mathcal{R}$. Hence calculating V values does not allow policy improvement.

- **Restriction III:** In some settings, the time between interactions is limited. For example a flying robot can run out of power and crash while calculating the inverse of a large matrix. Hence the per-time-step complexity of the RL agent plays a critical role in its applicability to time sensitive domains.

The first restriction is addressed by learning schemes that do not require multiple samples from arbitrary states. In other words, learning is carried out by looking at sampled trajectories. RL methods hedge against the second restriction by estimating Q values instead of V and relying on Equation 2.2 to improve the policy. Finally, if interaction time is limited, RL methods either provide computationally cheap learning during interaction with the environment (*i.e.,* online learning) or carry out learning after obtaining samples from the environment (*i.e.,* batch learning). The next section describes popular online and batch RL methods and show that they can be viewed as ADP techniques.

```
Algorithm 10: Q-Learning                                          Complexity
  Input: $\boldsymbol{R}, \boldsymbol{P}, \gamma$
  Output: $\pi$
1 $\bar{\boldsymbol{\theta}} \leftarrow Random()$
2 $\langle s, a \rangle \leftarrow \langle s_0, \pi^{\epsilon}(s_0) \rangle$
3 while time left do
4     Send action $a$ and receive reward $r$ and next state $s'$
5     $Q^{+}(s,a) \leftarrow r + \gamma \max_{a'} Q(s', a')$          $\Theta(n|\mathcal{A}|)$
6     $\delta \leftarrow Q^{+}(s,a) - Q(s,a)$
7     $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \alpha\delta\phi(s,a)$          $\Theta(n)$
8     $\langle s, a \rangle \leftarrow \langle s', \pi^{\epsilon}(s') \rangle$          $\Theta(n|\mathcal{A}|)$
9 return $\pi$ greedy w.r.t. $Q$
```

2.6.1 Q-Learning

While for model-based MDP solvers, $\mathcal{P}$ and $\mathcal{R}$ are known, due to high computational complexity demands of using these models, they ended up being used as black boxes providing samples (*e.g.,* Algorithms 6, 8 and 9). Naturally, if these methods can accommodate the use of trajectories rather than arbitrary sampling techniques, they can realize RL techniques.

Let us focus on Algorithm 8. Considering **Restriction I**, both L_1 and L_2 must be 1 to allow for learning from a trajectory. The resulting online algorithm is known as Q-Learning (Watkins, 1992) and is shown in Algorithm 10. The complexities of online methods are described in terms of per-time-step computational complexity. As expected, Q-Learning has the same complexity as Algorithm 8 with $L_1 = L_2 = 1$ which is $\Theta(n|\mathcal{A}|)$. The δ calculated on line 6 highlights the difference between the better estimate of the Q function based on a single interaction, $Q^{+}(s,a)$, and the current estimate, $Q(s,a)$. Hence δ is called the *temporal difference* (TD) error in the literature (Sutton & Barto, 1998). Unfortunately this algorithm can diverge when function approximation is employed (Sutton & Barto, 1998).

2.6.2 SARSA

The divergence of Q-Learning stems from the fact that the policy that is used to gather samples (π^{ϵ}) is not the same as the policy used for learning (greedy w.r.t. Q values). Fortunately matching these two policies will lead to a convergent online method named SARSA (state action reward state action) (Rummery & Niranjan, 1994) shown in Algorithm 11. SARSA has the same per-time-step computational complexity as Q-Learning: $\Theta(n|\mathcal{A}|)$. Notice that compared to Algorithm 10, $Q^{+}(s,a)$ is now calculated based on $Q(s', \pi^{\epsilon}(s'))$ rather than

```
Algorithm 11:SARSA                                                  Complexity
Input: $\gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
$\langle s, a \rangle \leftarrow \langle s_0, \pi^{\epsilon}(s_0) \rangle$
while time left do
    Send action $a$ and receive reward $r$ and next state $s'$
    $a' \leftarrow \pi^{\epsilon}(s')$                                 $\Theta(n|\mathcal{A}|)$
    $Q^{+}(s, a) \leftarrow r + \gamma Q(s', a')$                       $\Theta(n)$
    $\delta \leftarrow Q^{+}(s, a) - Q(s, a)$
    $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \alpha\delta\phi(s, a)$            $\Theta(n)$
    $\langle s, a \rangle \leftarrow \langle s', a' \rangle$
return $\pi$ greedy w.r.t. $Q$
```

$\max_{a'} Q(s', a')$, making both the sampling and learning policies identical. This property is know as *on-policy* learning, meaning that the agent learns about the same policy used for generating trajectories. An alternative approach is *off-policy* learning (*e.g.,* Q-Learning), where the policy used for generating samples is different from the one the agent learns about. In other words, the temporal difference error is calculated off-policy.

2.6.3 Actor-Critic

So far, all algorithms guaranteed visiting all state-action pairs infinitely often (*i.e.,* ergodicity), by using an ϵ-greedy policy. The state-action space can be explored more efficiently by guiding the policy based on the past experiences. One such approach is to represent the policy as a separate entity, referred to as the "actor", implemented as the Gibbs softmax distribution (Sutton & Barto, 1998):

$$\pi^{actor}(s, a) = \frac{e^{\rho(s,a)/\tau}}{\sum_b e^{\rho(s,b)/\tau}}, \tag{2.32}$$

in which $\rho(s, a) \in \mathbb{R}$ is the preference of taking action a in state s, and the variable $\tau \in [0, \infty)$ can be used to shift between greedy with respect to Q values and random action selection. $\pi^{actor}(s)$ returns the action a with probability $\pi^{actor}(s, a)$. The Q values can be updated with an arbitrary RL update rule, and because the update rule essentially criticizes the agent for the observed TD error, it is named the critic. Again, in domains with large state-action pairs, preferences are represented using linear function approximation similar

```
Algorithm 12: Actor-Critic                                              Complexity
Input: $\gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
$\boldsymbol{\omega} \leftarrow Random()$
$\langle s, a \rangle \leftarrow \langle s_0, \pi^{actor}(s_0) \rangle$
while time left do
    Send action $a$ and receive reward $r$ and next state $s'$
    $a' \leftarrow \pi^{actor}(s')$                                      $\Theta(n|\mathcal{A}| + m|\mathcal{A}|)$
    $Q^+(s, a) \leftarrow r + \gamma Q(s', a')$                          $\Theta(n)$
    $\delta \leftarrow Q^+(s, a) - Q(s, a)$
    $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \alpha\delta\phi(s, a)$   (Critic Update)        $\Theta(n)$
    $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\delta\psi(s, a)$   (Actor Update)         $\Theta(m)$
    $\langle s, a \rangle \leftarrow \langle s', a' \rangle$                          $\Theta(n)$
return $\pi$ greedy w.r.t. $Q$
```

to the approximation of Q values:

$$\rho(s, a) = \boldsymbol{\omega}^T \psi(s, a), \tag{2.33}$$

where $\boldsymbol{\omega}_{m|\mathcal{A}| \times 1}$ is the weight vector and ψ is a function mapping each state-action pair to a feature vector with size $m|\mathcal{A}|$ akin to ϕ. Note that ψ and ϕ can be different functions. $\psi(s, a)$ is constructed based on $\psi(s)$ identical to how ϕ using Equation 2.11 was constructed.

There are various ways to update the actor (Bhatnagar et al., 2007; Peters & Schaal, 2008). Following the template of actor-critic methods (Bhatnagar et al., 2007), Algorithm 12 shows the realization of the actor-critic, with a gradient descent update rule for the actor update and on-policy TD error based learning (akin to SARSA) for the critic update. The per-time-step complexity of the algorithm is $\Theta(m|\mathcal{A}| + n|\mathcal{A}|)$ which is now dependent on both m and n due to the use of two function approximators. There are several convergence results for Actor-Critic learners (Bhatnagar et al., 2007). An interesting observation is that these methods are the intersection of value-based and policy search RL methods, because they both estimate the values, while representing the policy in a parametric form.

2.6.4 Least-Squares Policy Iteration

Policy Iteration methods can also be used in the context of RL by accounting for the 3 restrictions mentioned earlier. Let us revisit Algorithm 6. For now consider the use of LSTD formulation (*i.e.,* Equations 2.23 and 2.24). The next section discusses how BRM can be employed in the context of RL. To address **Restriction I**, in Equation 2.28 L_2 has to equal 1, since samples can be generated only by following a trajectory. Consequently, samples are gathered in from of $\langle s_i, a_i, r_i, s'_i, a'_i \rangle$, where $s_{i+1} = s'_i$ and $a_{i+1} = a'_i$. L_1 should be large enough to provide enough samples to form reasonable estimates of $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$:

$$\tilde{\bar{\boldsymbol{\Phi}}} = \begin{bmatrix} \text{---}\, \phi^\top(s_1, a_1) \,\text{---} \\ \text{---}\, \phi^\top(s_2, a_2) \,\text{---} \\ \vdots \\ \text{---}\, \phi^\top(s_{L_1}, a_{L_1}) \,\text{---} \end{bmatrix}, \widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}} = \begin{bmatrix} \text{---}\, \phi^\top(s'_1, a'_1) \,\text{---} \\ \text{---}\, \phi^\top(s'_2, a'_2) \,\text{---} \\ \vdots \\ \text{---}\, \phi^\top(s'_{L_1}, a'_{L_1}) \,\text{---} \end{bmatrix}, \tilde{\bar{\boldsymbol{R}}} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{L_1} \end{bmatrix} \tag{2.34}$$

$$\tilde{\bar{\boldsymbol{A}}}_{LSTD} = \frac{1}{L_1}\tilde{\bar{\boldsymbol{\Phi}}}^\top(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}), \tag{2.35}$$

$$\tilde{\bar{\boldsymbol{b}}}_{LSTD} = \frac{1}{L_1}\tilde{\bar{\boldsymbol{\Phi}}}^\top\tilde{\bar{\boldsymbol{R}}} \tag{2.36}$$

If the evaluation of a fixed policy is desired, the equations become:

$$\tilde{\boldsymbol{\Phi}} = \begin{bmatrix} \text{---}\, \phi^\top(s_1) \,\text{---} \\ \text{---}\, \phi^\top(s_2) \,\text{---} \\ \vdots \\ \text{---}\, \phi^\top(s_{L_1}) \,\text{---} \end{bmatrix}, \widetilde{\boldsymbol{P}\boldsymbol{\Phi}} = \begin{bmatrix} \text{---}\, \phi^\top(s'_1) \,\text{---} \\ \text{---}\, \phi^\top(s'_2) \,\text{---} \\ \vdots \\ \text{---}\, \phi^\top(s'_{L_1}) \,\text{---} \end{bmatrix}, \tilde{\boldsymbol{R}} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{L_1} \end{bmatrix} \tag{2.37}$$

$$\tilde{\boldsymbol{A}}_{LSTD} = \frac{1}{L_1}\tilde{\boldsymbol{\Phi}}^\top(\tilde{\boldsymbol{\Phi}} - \gamma\widetilde{\boldsymbol{P}\boldsymbol{\Phi}}), \tag{2.38}$$

$$\tilde{\boldsymbol{b}}_{LSTD} = \frac{1}{L_1}\tilde{\boldsymbol{\Phi}}^\top\tilde{\boldsymbol{R}} \tag{2.39}$$

Again, in the limit of infinite samples, approximations become exact provided the sampling policy is ergodic (Lagoudakis & Parr, 2003). The algorithm is already using Q values to represent the policy, addressing **Restriction II**. Finally **Restriction III** is resolved by the batch nature of LSPI; learning is done after the sample gathering phase. LSPI is shown in Algorithm 13. This algorithm has $\Theta(nL_1|\mathcal{A}|+n^2L_1+n^3|\mathcal{A}|^3)$ complexity per iteration. An important fact about this method is that samples generated to estimate $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$ are thrown away after each cycle, making this algorithm inefficient for domains with an expensive cost of sample acquisition (*e.g.*, rescue civilians after a hurricane).

Least-Squares Policy Iteration (LSPI) algorithm (Lagoudakis & Parr, 2003) mitigates

```
Algorithm 13: Trajectory Sampling Q-API                                              Complexity
Input: $\gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
while time left do
    Create $L_1$ samples $\langle s_i, a_i, r_i, s'_i, a'_i \rangle$ following policy $\pi^\epsilon$        $\Theta(nL_1|\mathcal{A}|)$
    Calculate $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$ using Equations 2.34-2.36                  $\Theta(n^2L_1)$
    $\bar{\boldsymbol{\theta}} \leftarrow \tilde{\bar{\boldsymbol{A}}}^{-1}\tilde{\bar{\boldsymbol{b}}}$   (Use regularization if the inverse does not exist)      $\Theta(n^3|\mathcal{A}|^3)$
    $\boldsymbol{Q} \leftarrow \tilde{\bar{\boldsymbol{\Phi}}}\bar{\boldsymbol{\theta}}$                                                   $\Theta(n^2|\mathcal{A}|)$
return $\pi$ greedy w.r.t. $Q$
```

```
Algorithm 14: Least-Squares Policy Iteration (LSPI)                                 Complexity
Input: $\gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
Create $L_1$ samples $\langle s_i, a_i, r_i, s'_i, a'_i \rangle$ following policy $\pi^\epsilon$
while time left do
    Calculate $\tilde{\bar{\boldsymbol{A}}}$ and $\tilde{\bar{\boldsymbol{b}}}$ using Equations 2.34-2.36                  $\Theta(n^2L_1)$
    $\bar{\boldsymbol{\theta}} \leftarrow \tilde{\bar{\boldsymbol{A}}}^{-1}\tilde{\bar{\boldsymbol{b}}}$   (Use regularization if the inverse does not exist)      $\Theta(n^3|\mathcal{A}|^3)$
    $\boldsymbol{Q} \leftarrow \tilde{\bar{\boldsymbol{\Phi}}}\bar{\boldsymbol{\theta}}$                                                   $\Theta(n^2|\mathcal{A}|)$
    $a'_i \leftarrow \text{argmax}_{a \in \mathcal{A}} Q(s'_i, a)$ For all $i$                        $\Theta(nL_1|\mathcal{A}|)$
return $\pi$ greedy w.r.t. $Q$
```

this problem of sample cost by reusing the same set of samples over and over in each iteration. On each iteration, LSPI biases the samples towards the new policy by switching a'_i to $\pi(s'_i)$, where π is greedy with respect to the most recent Q values. The result is shown in Algorithm 14. While the per-iteration complexity remains unchanged, the same set of data is reused through all iterations. Because the number of possible policies over a fixed set of samples is limited, LSPI in the worst case will switch between policies. With sufficient data, LSPI has been shown to work well in practice. The main drawback of LSPI is its strong bias introduced by the initial set of samples. If the initial sample distribution is not close enough to the sample distribution under the optimal policy, LSPI can perform poorly. This drawback is currently handled by manual filtering of samples gathered by a domain expert (Lagoudakis & Parr, 2003; Petrik et al., 2010), or by approximating a policy-independent model using the gathered samples (Bowling et al., 2008).

Unfortunately, applying the BRM approach to the RL setting is not as straightforward as the LSTD approach. To investigate this issue further, review **Restriction II**: samples can only be obtained in forms of trajectories. Now consider the calculation of $\tilde{\bar{\boldsymbol{A}}}$ in the BRM approach:

$$\tilde{\bar{\boldsymbol{A}}}_{BRM} = \frac{1}{L_1}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}})^{\top}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}). \tag{2.40}$$

Given that samples are in the form of $\langle s_i, a_i, s'_i, r_i\rangle$, $\tilde{\bar{\boldsymbol{\Phi}}}$, $\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}$ and $\tilde{\bar{\boldsymbol{R}}}$ have to be calculated through Equation 2.34, yet this causes both $\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}$ terms in Equation 2.40 to use the same set of samples, leading to a biased estimate of $\bar{\boldsymbol{A}}_{BRM}$ (Lagoudakis & Parr, 2003; Sutton & Barto, 1998). The solution to this problem is to calculate two $\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}$ matrices built upon independent samples. This independency constraint means samples should be gathered in the form of $\langle s_i, a_i, s'_i, a'_i, s''_i, a''_i, r_i\rangle$, where both s'_i and s''_i are sampled independently from state s_i after taking action a_i. Given samples in the above form an unbiased estimate of $\tilde{\bar{\boldsymbol{A}}}_{BRM}$ can then be formed:

$$\boldsymbol{\Phi} = \begin{bmatrix} \text{---}\ \phi^{\top}(s_1, a_1)\ \text{---} \\ \text{---}\ \phi^{\top}(s_2, a_2)\ \text{---} \\ \vdots \\ \text{---}\ \phi^{\top}(s_{L_1}, a_{L_1})\ \text{---} \end{bmatrix}, \tag{2.41}$$

$$\widetilde{\boldsymbol{P}\boldsymbol{\Phi}}_1 = \begin{bmatrix} \text{---}\ \phi^{\top}(s'_1, a'_1)\ \text{---} \\ \text{---}\ \phi^{\top}(s'_2, a'_2)\ \text{---} \\ \vdots \\ \text{---}\ \phi^{\top}(s'_{L_1}, a'_{L_1})\ \text{---} \end{bmatrix}, \widetilde{\boldsymbol{P}\boldsymbol{\Phi}}_2 = \begin{bmatrix} \text{---}\ \phi^{\top}(s''_1, a''_1)\ \text{---} \\ \text{---}\ \phi^{\top}(s''_2, a''_2)\ \text{---} \\ \vdots \\ \text{---}\ \phi^{\top}(s''_{L_1}, a''_{L_1})\ \text{---} \end{bmatrix}, \tag{2.42}$$

$$\tilde{\bar{\boldsymbol{A}}}_{BRM} = \frac{1}{L_1}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}_1)^{\top}(\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}_2). \tag{2.43}$$

The rest of the calculations for BRM remain unchanged:

$$\tilde{\bar{\boldsymbol{R}}} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{L_1} \end{bmatrix}, \tilde{\bar{\boldsymbol{b}}}_{BRM} = (\tilde{\bar{\boldsymbol{\Phi}}} - \gamma\widetilde{\bar{\boldsymbol{P}}\bar{\boldsymbol{\Phi}}}_1)^{\top}\boldsymbol{R} \tag{2.44}$$

The BRM requirement for calculating two independent $\widetilde{\boldsymbol{P}\boldsymbol{\Phi}}$ estimations is known as the double sampling requirement which makes the use of BRM challenging, because the ob-

```
Algorithm 15:LSPI-BRM                                                                Complexity
Input: $\gamma$
Output: $\pi$
$\bar{\boldsymbol{\theta}} \leftarrow Random()$
Create samples in form of $\langle s_i, a_i, r_i, s'_i, a'_i \rangle$ following policy $\pi^\epsilon$
Filter $L_1$ samples in form of $\langle s_i, a_i, r_i, s'_i, a'_i, s''_i, a''_i \rangle$
while time left do
    Calculate $\widetilde{\bar{\boldsymbol{A}}}_{BRM}$ and $\widetilde{\bar{\boldsymbol{b}}}_{BRM}$ using Equations 2.41-2.44                    $\Theta(n^2 L_1)$
    $\bar{\boldsymbol{\theta}} \leftarrow \widetilde{\bar{\boldsymbol{A}}}^{-1}_{BRM} \widetilde{\bar{\boldsymbol{b}}}_{BRM}$    (Use regularization if the inverse does not exist) $\Theta(n^3 |\mathcal{A}|^3)$
    $\boldsymbol{Q} \leftarrow \widetilde{\bar{\boldsymbol{\Phi}}} \bar{\boldsymbol{\theta}}$                                                                  $\Theta(n^2 |\mathcal{A}|)$
    $a'_i \leftarrow \text{argmax}_{a \in \mathcal{A}} Q(s'_i, a)$ For all $i$                                           $\Theta(n L_1 |\mathcal{A}|)$
    $a''_i \leftarrow \text{argmax}_{a \in \mathcal{A}} Q(s''_i, a)$ For all $i$                                         $\Theta(n L_1 |\mathcal{A}|)$
return $\pi$ greedy w.r.t. $Q$
```

tained samples have to be filtered. As a result, only those state-action pairs for which two consecutive samples exist are used for learning. Algorithm 15 shows the use of the BRM approach in the RL setting. Notice that creating L_1 samples to satisfy the double sampling requirement might be challenging in domains where the probability of visiting the same state is very low (*e.g.,* continuous state spaces). The iteration complexity of LSPI-BRM is identical to LSPI. There have been several arguments towards the use of BRM vs. LSTD in the literature (*e.g.,* Scherrer, 2010; Sutton et al., 2009). In summary, BRM has superior mathematical properties, while LSTD has been shown to work better in practice. For more discussion, refer to Sherrer's work (2010).

2.6.5 Discussion

Table 2.3 provides an overview of RL methods discussed in this chapter together with their computational complexity. For the upper part of the table, complexities correspond to the per-time-step computation, while for the bottom part, they highlight the iteration complexity. In general, the first three *online* methods provide cheap complexity per interaction, which is critical in dealing with domains where **Restriction III** allows for a short amount of interaction time. On the other hand, the last three *batch* algorithms often require fewer samples compared to online methods to produce good policies. Hence, if sample complexity is the only concern, batch methods are often preferred over online methods. There are still more RL algorithms in the literature not discussed in this chapter as they are not as popular in the RL community. Interested readers are referred to more detailed references (Bertsekas & Tsitsiklis, 1996; Buşoniu et al., 2010; Sutton & Barto, 1998; Szepesvári,

Table 2.3: RL methods and their per-time-step/iteration computational complexity.

Algorithm	Per-Time-Step Complexity	Algorithm Number
Q-Learning	$\Theta(n\|\mathcal{A}\|)$	10
SARSA	$\Theta(n\|\mathcal{A}\|)$	11
Actor-Critic	$\Theta(m\|\mathcal{A}\| + n\|\mathcal{A}\|)$	12
Algorithm	Iteration Complexity	Algorithm Number
Trajectory Sampling Q-API	$\Theta(nL_1\|\mathcal{A}\| + n^2L_1 + n^3\|\mathcal{A}\|^3)$	13
LSPI	$\Theta(nL_1\|\mathcal{A}\| + n^2L_1 + n^3\|\mathcal{A}\|^3)$	14
LSPI-BRM	$\Theta(nL_1\|\mathcal{A}\| + n^2L_1 + n^3\|\mathcal{A}\|^3)$	15

2010)

2.7 Summary

This chapter reviewed MDPs as a suitable framework for sequential decision making problems with uncertainty. MDP solvers together with their computational complexities were investigated in two major branches of model-based and model-free approaches. Using linear function approximation and sampling techniques, this chapter demonstrated how various DP methods are derived from Policy Iteration and Value Iteration by trading off accuracy with computational speed. Finally, this chapter showed that RL methods are essentially ADP methods, accommodating three extra restrictions.

While linear function approximation was introduced as one of the main tools to reduce the computational complexity of MDP solvers, the problem of finding a suitable set of basis functions (*i.e.,* an appropriate feature function) was not discussed so far. The next chapter focuses on this challenge.

Chapter 3

The *Right* Set of Features: A Theoretical View

The previous chapter explained how, with a fixed set of bases functions (*i.e.,* features), existing RL methods using linear function approximation can tackle large MDPs. Yet it did not answer a fundamental question: how to find the *right* set of features? This chapter investigates the answer to this question from a theoretical perspective. First the notion of the *right* set of features is defined, and then algorithms are derived to generate such features. While this chapter provides insight on why the proposed algorithms should perform well, readers seeking RL pseudo-code to tackle control problems should look at the next chapter, as this chapter solely focuses on policy evaluation.

The structure of this chapter is as follows. Section 3.1 discusses the desirable properties of features and representations that affect fast learning and cheap computational complexity. This chapter reviews Tile Coding (Sutton & Barto, 1998), showing its desirable properties and how its initialization can be viewed as defining linear independencies among features. Section 3.2 provides theoretical insights on *why* and *how* features should be added to linear representations. In particular, when feature values only take binary values (*i.e.,* a binary representation), this section provides theoretical results on how to select new features as the conjunction of the previous features in order to facilitate the best guaranteed convergence of the approximated value function. Based on these theoretical results, Section 3.3 introduces the incremental Feature Dependency Discovery (iFDD) family of representation expansion techniques and empirically demonstrates their advantage over random expansion methods. Section 3.4 highlights the computational demands of iFDD in its original form and introduces a sparsification technique that substantially reduces the computational complexity of iFDD. Section 3.5 wraps up this chapter by summarizing the contributions.

3.1 The *Right* Set of Features

The notion of *right* set of features can be defined arbitrarily based on some predefined criterion of interest. This thesis is interested in finding features that allow the linear function approximator to learn the underlying value function with a small number of samples (known as low sample complexity), and cheap computational complexity. This brings us to three characteristics of representations affecting their sample complexity and computational complexity:

1. feature coverage
2. sparse feature functions
3. binary feature functions

3.1.1 Feature Coverage

In order to reduce the sample complexity of learning techniques, learned values (*i.e.,* Q or V values depending on the algorithm) should propagate to several states. When the weight corresponding to a feature is changed, this change affects the value of all states for which the value of the feature is non-zero (*i.e.,* active). The larger the portion of the state space for which a feature is activated, the better the resulting coverage. This thesis formally defines *coverage* for arbitrary feature f, $G(f)$, as its proportional state space coverage. In particular, S is assumed to be defined over d-dimensional metric state spaces, where $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_d$ and $\forall s \in \mathcal{S}, s = \langle s_1, \cdots, s_d \rangle$, where $s_i \in \mathcal{S}_i$. For continuous dimensions, we assume the state space is a hyperrectangle: $\mathcal{S}_i \in [l_i, u_i]$, where l_i and u_i specify the lower and upper bounds on the i^{th} dimension respectively.

$$G(f) \triangleq \frac{|\mathcal{S}^f|}{|\mathcal{S}|}, \text{where} \quad \mathcal{S}^f = \{s | s \in \mathcal{S}, \phi_f(s) \neq 0\}. \tag{3.1}$$

The $|.|$ operator returns the number of elements for discrete spaces. For continuous spaces:

$$\begin{aligned} \mathcal{S}^f &= \int_{l_1}^{u_1} \cdots \int_{l^d}^{u_d} \mathcal{I}(\phi_f(s))ds, \\ |\mathcal{S}| &= \Pi_{i=1}^{d} u_i - l_i, \\ \text{where} \quad \mathcal{I}(x) &= \begin{cases} 1 & x \neq 0 \\ 0 & x = 0 \end{cases}. \end{aligned} \tag{3.2}$$

Extending the definition of coverage to hybrid spaces (*i.e.,* both discrete and continuous dimensions constitute the state space) is straightforward. Features with high coverage will

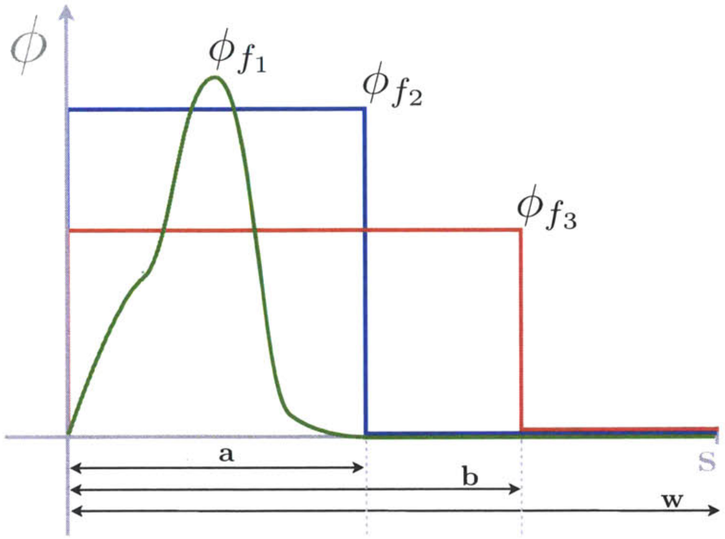


Figure 3-1: Three features defined over a one dimensional continuous state space. Features with more coverage and more uniformity have higher coverage values.

have G values close to 1. For example a constant feature that is active for all states has the G value of 1. On the other side of the spectrum are specific features with G values close to 0. For example in a tabular representation, for each feature $f, G(f) = \frac{1}{|S|}$. Figure 3-1 depicts an example where three feature functions: ϕ_{f_1}, ϕ_{f_2}, and ϕ_{f_3} are defined on a bounded one dimensional continuous state space. It can be seen that:

$$G(f_1) = G(f_2) = \frac{a}{w}, G(f_3) = \frac{b}{w}.$$

Features with high coverage facilitate fast learning in parts of the space where propagating learned values makes sense (*i.e.,* changing the weight corresponding to feature f makes the value function approximation more accurate for all states $s \in \mathcal{S}^f$). On the other hand, certain small subsets of the state space may require specific features with corresponding low coverage to capture the underlying shape of the value function. Section 3.2 theoretically shows how coverage plays a significant role in finding good features that reduces the approximation error of the value function quickly.

Dimensional Coverage: The coverage of each feature within each dimension is defined separately. In particular, the coverage of feature f at state s in dimension i is defined as follows:

$$G_{i,s}(f) \triangleq \frac{|\mathcal{S}^f_{i,s}|}{|\mathcal{S}_i|} \tag{3.3}$$

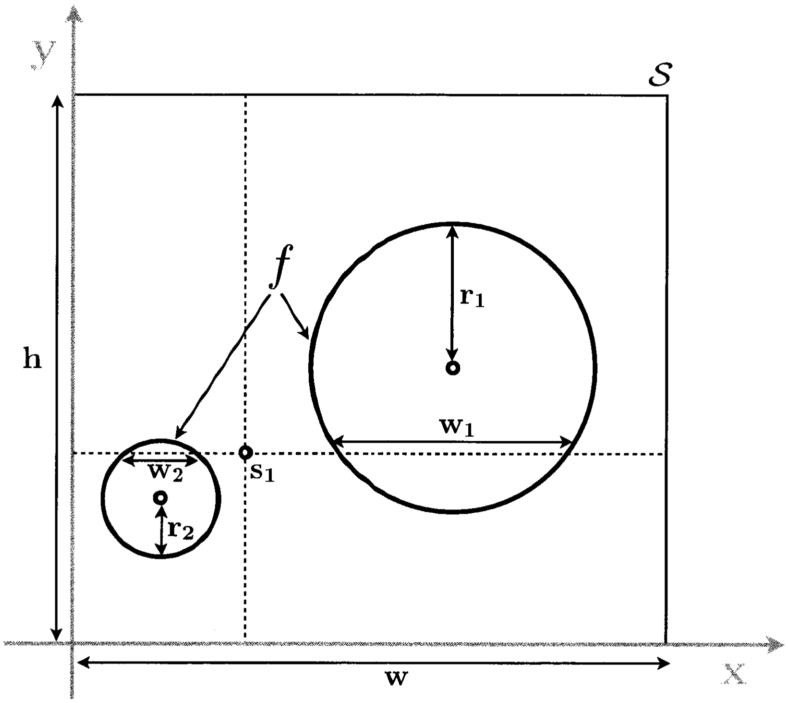


Figure 3-2: A two dimensional example illustrating the notion of coverage. Feature f is defined by two circles over the rectangular state space $\mathcal{S}$. Feature f has value 1 inside circles and 0 outside.

$$\text{where} \quad \mathcal{S}_{i,s}^f = \{x = \langle s_1, \cdots, s_{i-1}, x_i, s_{i+1}, \cdots, s_n \rangle | \phi_f(x) \neq 0, x_i \in \mathcal{S}_i\}.$$

Similar to feature coverage, for continuous domains:

$$\begin{aligned} |\mathcal{S}_{i,s}^f| &= \int_{l_i}^{u_i} \mathcal{I}(\phi_f(s))ds, \\ |\mathcal{S}_i| &= u_i - l_i, \end{aligned}$$

Figure 3-2 illustrates the concept of dimensional coverage through an example. The 2D state space, $\mathcal{S}$, is specified by the rectangle, while feature f is defined by two circles. Feature f has value 0 for states inside either of the circles and zero otherwise. Notice that in this example, the ϕ values are in the third dimension, as both x and y dimensions represents the state space $\mathcal{S}$. Hence:

$$\begin{aligned} G(f) &= \frac{\pi(r_1^2 + r_2^2)}{hw} \\ G_{x,s_1}(f) &= \frac{w_1 + w_2}{w} \\ G_{y,s_1}(f) &= 0 \end{aligned}$$

3.1.2 Sparse Features

A vector is *sparse* if the number of non-zero elements, k, is much less than the size of the vector, n (*i.e.,* $k \ll n$). In order to approximate the value of each state-action pair, the inner product of the existing weight vector, $\boldsymbol{\theta}$, and the corresponding feature function, $\phi(s, a)$, has to be calculated. While in general this calculation requires n multiplications and n summations, if either of these two vectors are sparse, this calculation drops to k multiplication and k summations.

The weight vector is often the free parameter adjusted through learning, yet the family of feature vectors is often selected by the user and thus can be forced to be sparse.[1] More formally, the sparsity for representation ϕ is:

$$\begin{aligned}\texttt{sparsity}(\phi) &\triangleq \frac{k}{n}\\ k &= \max_{s\in\mathcal{S}} \sum_{f\in\mathcal{X}} \mathcal{I}\Big(\phi_f(s)\Big),\end{aligned}$$

For sparse representations, $\texttt{sparsity}(\phi)$ is close to 0, while for dense representations, this value is close to 1. In general the per-time-step computational complexity of the approximation is $\Theta(\texttt{sparsity}(\phi)n) = \Theta(k)$ multiplications and summations.

3.1.3 Binary Features

Of special interest is the set of feature functions ϕ where the output is a binary vector ($\phi : \mathcal{S} \times \mathcal{A} \to \{0, 1\}^n$). In this case, not only can the estimated values be computed efficiently, since there is no need for multiplication (Buro, 1999), but each learned weight also indicates the importance of the corresponding binary feature. It is interesting to note that, if ϕ contains a unique feature for each state-action pair that is active only for that state-action pair, then the function approximation is reduced to a lookup table, assigning a separate value to each state-action pair (Equation 2.1). The next subsection describes Tile Coding as one of the most popular forms of generating sparse and binary features.

3.1.4 Tile Coding

Discretizing continuous state spaces into several layers (tilings) was first introduced by Albus (1971) as the Cerebellar Model Articulation Controller (CMAC) and was later named

[1] There are also methods that focus on calculating sparse weight vectors using ℓ_1 regularization. Chapter 4 will cover such methods.

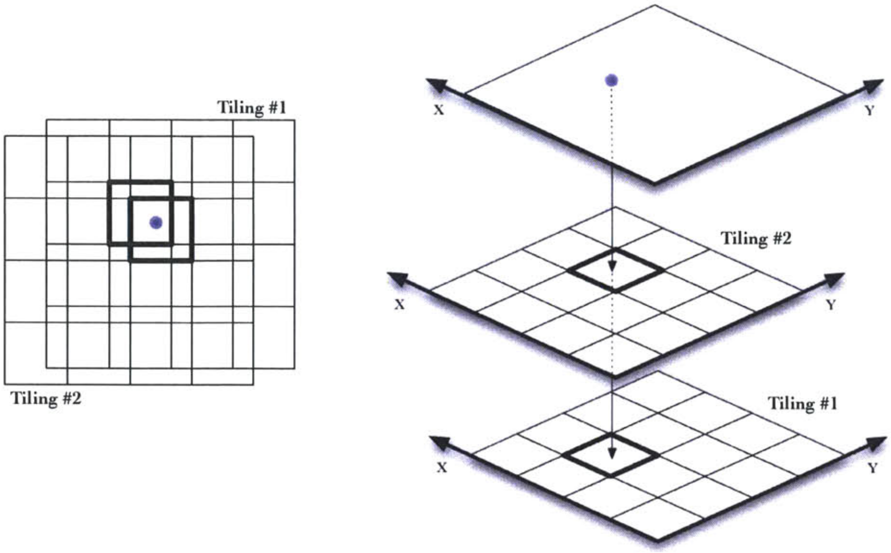


Figure 3-3: Tile Coding for a two dimensional space from top view (left) and side view (right). Note that tiling #2 is slightly shifted in the 2D space with respected to tiling #1.

Tile Coding (Sutton & Barto, 1998). Figure 3-3 illustrates how a two dimensional continuous space can be discretized into uniform grids along two layers. The left figure provides a view from the top, while the right figure illustrates the concept from the side view. Each tile represents a binary feature. Figure 3-3 shows how Tile Coding maps a point from the state space to a set of active features by locating the corresponding region (highlighted tile) within each layer. In this example, $\phi(s)$ has 32 elements corresponding to 32 tiles over the 2 tilings. $\phi(\bullet)$ has 30 zeros and two $1s$ corresponding to the two activated tiles. In general tiles can be arbitrary shaped, yet tiles defined as hyper-rectangles, the focus of this thesis, are mostly favored in practice due to the ease of implementation and cheap computational demands. Tile Coding has been also used with arbitrary boundaries (See Chapter 8 of Sutton & Barto, 1998).

For a binary feature f defined as a hyper-rectangular tile, the definition of dimensional coverage can be simplified because:

$$\forall x, y \in \mathcal{S}, G_{i,x}(f) = G_{i,y}(f), \iff \phi_f(x) = \phi_f(y).$$

Since the focus of this thesis is on such features, the term $G_i(f)$ will be used instead of $G_{i,s}(f)$, where $\phi_f(s) = 1$. Moreover, hyper-rectangular features have the following

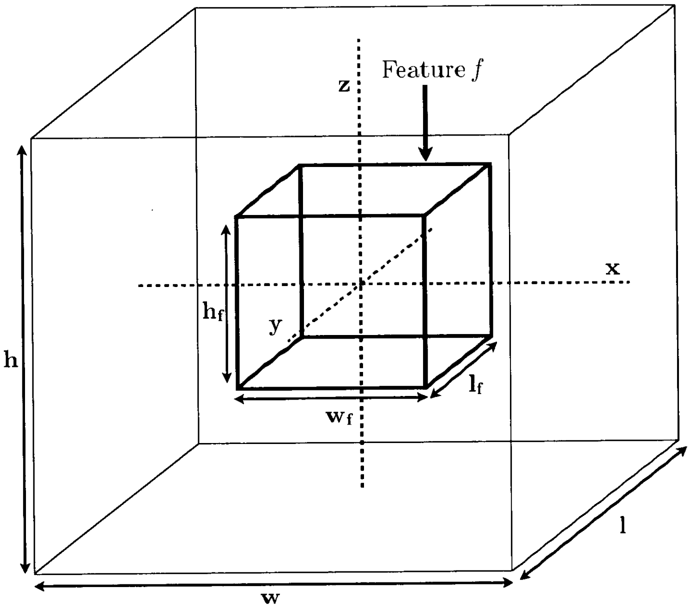


Figure 3-4: Feature f is defined by the inner cube. The state space is the larger cube. The coverage of feature f in dimensions x, y and z are $\frac{w_f}{w}$, $\frac{l_f}{l}$, and $\frac{h_f}{h}$ respectively.

property:

$$G(f) = \Pi_{i\in\{1,\cdots,d\}} G_i(f) \tag{3.4}$$

Figure 3-4 provides an example of a feature f defined by a cube in a bounded 3D space, where,

$$\phi_f(s) = \begin{cases} 1 & \text{if } s \text{ is inside the inner cube} \\ 0 & \text{otherwise} \end{cases}.$$

In this case:

$$\begin{aligned} G(f) &= \frac{l_f w_f h_f}{lwh}, \\ G_x(f) &= \frac{w_f}{w}, \\ G_y(f) &= \frac{l_f}{l}, \\ G_z(f) &= \frac{h_f}{h}. \end{aligned}$$

When a hyper-rectangular feature f fully covers dimension i, meaning that $G_i(f) = 1$, then feature f *ignores* dimension i. In other words, feature f is a *sub-dimensional* feature because the value of the state in dimension i does not affect $\phi_f(s)$. More formally:

$$\begin{aligned}
\texttt{dimensions}(f) &\triangleq \left\{i \middle| i \in \{1, \cdots, d\}, G_i(f) \neq 1\right\}, && (3.5)\\
\texttt{dimensionality}(f) &\triangleq \left|\texttt{dimensions}(f)\right|, && (3.6)\\
\text{feature } f \text{ is sub-dimensional} &\Leftrightarrow \texttt{dimensionality}(f) \neq d. && (3.7)
\end{aligned}$$

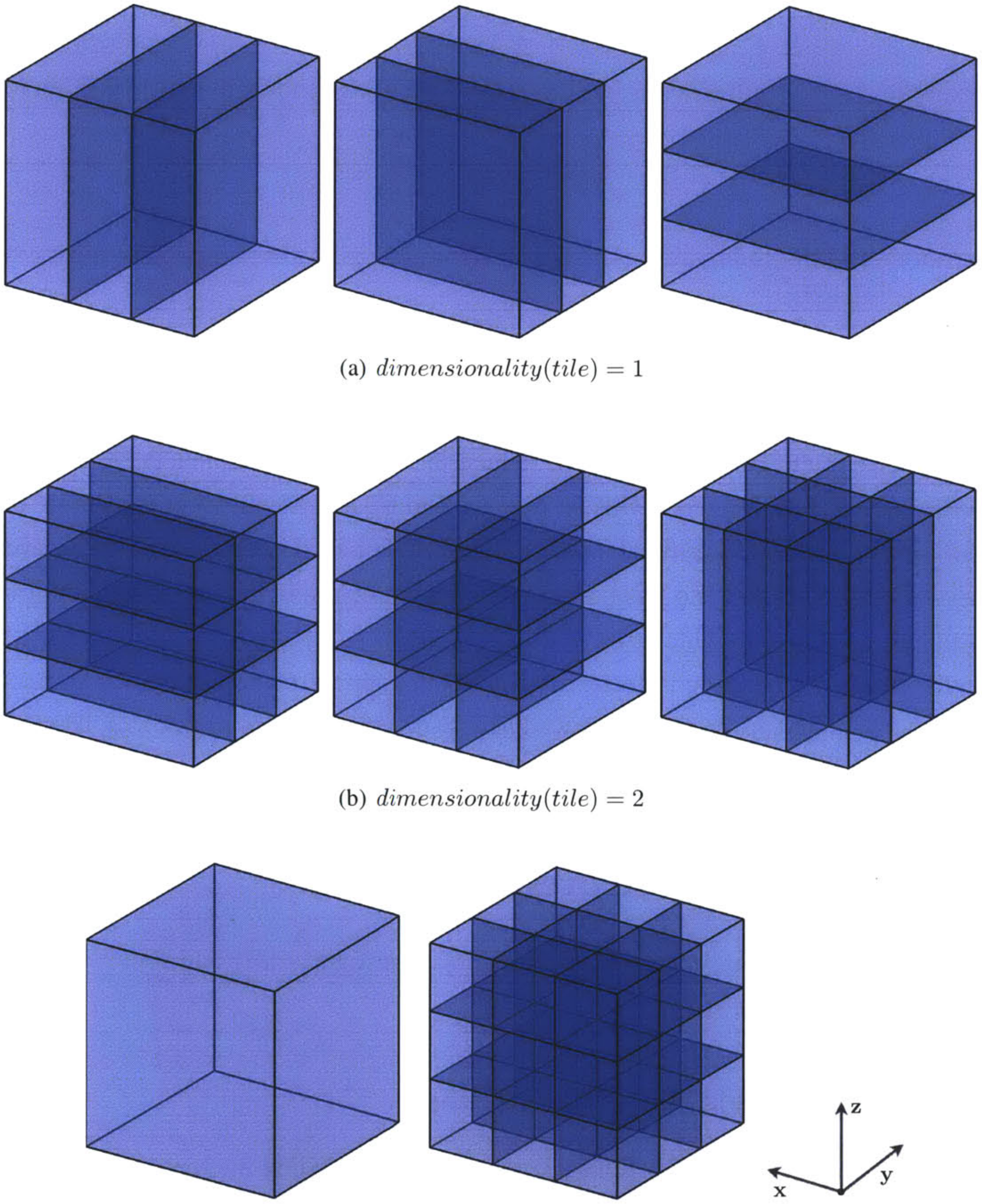
Practitioners have favored Tile Coding due to its successful results and ease of use (Leng et al., 2007; Stone & Sutton, 2001; Stone et al., 2005; Sutton, 1996; Torrey et al., 2005), yet there are two categories of manually tuned parameters: 1) number of tilings and 2) tile widths. The first parameter specifies the number of layers used for Tile Coding. Using more layers provides better resolution while increasing the memory requirement. The second set of parameters defines the width of each tile in each dimension of the state-space, controlling the coverage of tiles. The wider the tiles in each dimension, the higher their corresponding dimensional coverage will be.

Figure 3-5 depicts 8 types of uniform tilings covering a finite 3D state space specified by a cube. Figure 3-5(a) depicts three different tilings in which each tile corresponds to one of the dimensions, ignoring the other two dimensions. Figure 3-5(b) shows the same concept for tilings where tiles ignore one of the dimensions. Figure 3-5(c) corresponds to the unique tile/tiling which ignores all three dimensions. Finally, tiles in Figure 3-5(d) correspond to all 3 dimensions. A d-dimensional state space can be tiled using 2^d types of uniform hyper-rectangular tilings. Notice that similar to Figure 3-3, often more than one instance of a tiling type is used for better resolution. For example, in a 4-dimensional problem, Sutton used 48 tilings defined by 15 tiling types (1996).

Consider a d-dimensional continuous state space defined by a hyper-cube (*i.e.*, all dimensions have identical bounds), tiled with function ϕ using k tilings.[2] Each tiling $i \in \{1, \cdots, k\}$ consists of n_i uniform hyper-rectangular tiles, where for $j \in \{1, 2, \cdots, n_i\}$, tile t_{ij} has $\texttt{dimensionality}$ d_{ij}. Also assume that all tiles have equal widths in all $\texttt{dimensions}(t_{ij})$. Then:

$$\begin{aligned}
\text{(total number of features) } n &= \sum_{i=1}^{k} n_k\\
\texttt{sparsity}(\phi) &= \frac{k}{n},
\end{aligned}$$

[2] Notice that defining tiles for Tile Coding is equivalent to defining features for a linear function approximator, where each tile is a feature.



(a) $dimensionality(tile) = 1$

(b) $dimensionality(tile) = 2$

(c) $dimensionality(tile) = 0$ (d) $dimensionality(tile) = 3$

Figure 3-5: Eight Tiling Types for Tile Coding of a 3D state Space

$$
\begin{aligned}
G(t_{ij}) &= \frac{1}{n_i}, \\
G_u(t_{ij}) &= \begin{cases} 1 & \text{if } u \notin \texttt{dimensions}(t_{ij}) \\ \dfrac{1}{\sqrt[d_{ij}]{n_i}} & \text{otherwise} \end{cases} .
\end{aligned}
$$

For example if a 3D state space is tiled with all eight tilings shown in Figure 3-5, then:

$$n = 64, \quad \texttt{sparsity}(\phi) = \frac{8}{64} = \frac{1}{8}.$$

Furthermore, for tiles in Figure 3-5(b), tiles in the middle tiling have the following coverage values:

$$G(f) \;=\; \frac{1}{9}, \quad G_x(f) = \frac{1}{3}, \quad G_y(f) = 1, \quad G_z(f) = \frac{1}{3}.$$

There have been several attempts to reduce the number of parameters by automating the process, and these methods will be reviewed in Chapter 4. In summary, none of the current methods autonomously adapt both tiles and tilings simultaneously. Moreover, to the best of our knowledge, no one has addressed the problem of autonomously identifying important dimensions on which tilings should be defined. The next section describes why expanding the representation is necessary for a better approximation of the value function, and how such an expansion can be viewed as adding new tiles and tilings to a Tile Coding framework.

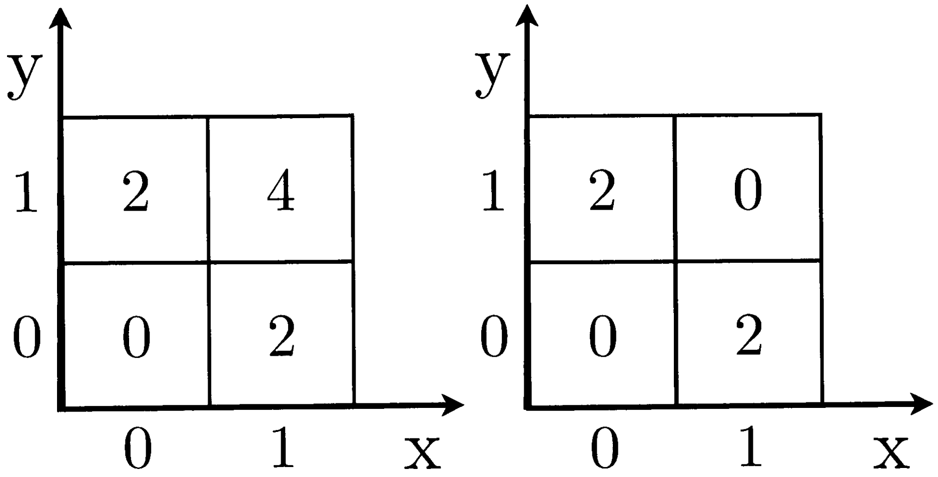
3.2 Expanding the Representation

Given a fixed set of binary features, this section investigates *why* and *how* new features should be added to the set of features.

3.2.1 Why Expand the Representation?

One may wonder, given a set of binary features, why expanding the representation is necessary. Figure 3-6 depicts two discrete functions V_1 and V_2 defined over a two dimensional space (x, y). The value of the function for each state is written in the grid cell corresponding to that state. For example, $V_1(0, 1) = 2$. Now, given the ordered set of features $\chi = \{f_1, f_2, f_3, f_4\}$ where

$$\phi_{f_1}(x, y) \;=\; \mathcal{I}(x = 0),$$



(a) $V_1(x,y)$

(b) $V_2(x,y)$

Figure 3-6: An example of two functions defined over two dimensional spaces. $V_1(x,y)$ and $V_2(x,y)$ are shown as the values inside the corresponding grid cell. The left function (a) is linear in the set of features ($x = 0, x = 1, y = 0, y = 1$) with the corresponding weight vector (0, 2, 0, 2). The right function (b) is not.

$$\begin{aligned}\phi_{f_2}(x,y) &= \mathcal{I}(x = 1),\\ \phi_{f_3}(x,y) &= \mathcal{I}(y = 0),\\ \phi_{f_4}(x,y) &= \mathcal{I}(y = 1),\end{aligned}$$

and ordered states $\langle s_1, s_2, s_3, s_4\rangle = \langle(0,0),(0,1),(1,0),(1,1)\rangle$, V_1 can be captured exactly using a linear combination of features:

$$V_1 = \mathbf{\Phi}\boldsymbol{\theta} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 2 \\ 4 \end{bmatrix}.$$

On the other hand, it is easy to verify that $V_2 \notin \text{span}(\mathbf{\Phi})$, hence it cannot be represented with such features. In other words, V_2 is not linear in the set of features χ, or V_2 has nonlinearities with respect to features in χ. Given $\wedge$ as the logical AND operator, if feature f_5 is added to χ with the following definition:

$$\phi_{f_5}(x,y) = \mathcal{I}(x = 1) \wedge \mathcal{I}(y = 1) = \mathcal{I}(x = 1 \wedge y = 1),$$

then V_2 becomes representable in the new feature set:

$$V_2 = \boldsymbol{\Phi}\boldsymbol{\theta} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 2 \\ -4 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 2 \\ 0 \end{bmatrix}.$$

Feature f_5 covers the nonlinearity of V_2 in the previous set of features, allowing it to be represented exactly using one additional weight. Note that the new -4 weight negates the $+4$ value accumulated for state $(1, 1)$, setting $V_2(1, 1) = 0$. Going back to the Tile Coding example (*i.e.,* Figure 3-5), assume a 3D state space is tiled using three tilings shown in Figure 3-5(a). Figure 3-7 depicts how features (tiles) defined in dimensions x and y shown in tiling (3-7-c), can be constructed by applying the conjunction operator to every pair of features, where one selected from tiling (3-7-a) and the other one is selected from tiling (3-7-b). If the conjunction operator is further applied to the resulting tiling (3-7-e) and the z-dimensional features (3-7-f), then tiling (3-7-g) is retrieved. If the granularity of splits for each dimension is fixed to three uniform buckets, tiling (3-7-g) represents a tabular representation, because each combination of dimensional values are captured through a unique feature.

So why not start with a powerful representation which makes the value function linear in those set of features? For a value function defined over $|S|$ states, why not pick a $\boldsymbol{\Phi}$ matrix with a column rank $|S|$? To answer this question, consider a simple multi-UAV mission planning scenario with 5 UAVs where each UAV has to track its 10 fuel levels and 100 locations. For this domain $|S| = 10^{15}$. To learn a value function in this scenario, 10^{15} parameters must be learned. Assuming each parameter is stored as 2 bytes, 2 petabyte of memory is required to store the value function. Furthermore, as the number of parameters (*i.e.,* the size of $\boldsymbol{\theta}$) grows, the amount of data required to find a reasonable approximation of the function increases correspondingly. Using a tabular representation challenges practitioners both in terms of memory consumption and sample complexity.

In general for an d dimensional state space where each dimension is split into b buckets, a lookup table representation including all linear independencies requires b^d features, known as the curse of dimensionality (Sutton & Barto, 1998). From an opposite perspective, the representation may ignore all possible linear independencies among the features. With the same definition of b and d, this representation requires bd features. Unfortunately, the resulting representation is extremely underpowered in terms of value functions it can represent. Hence, covering some linear independencies is often critical for representing the

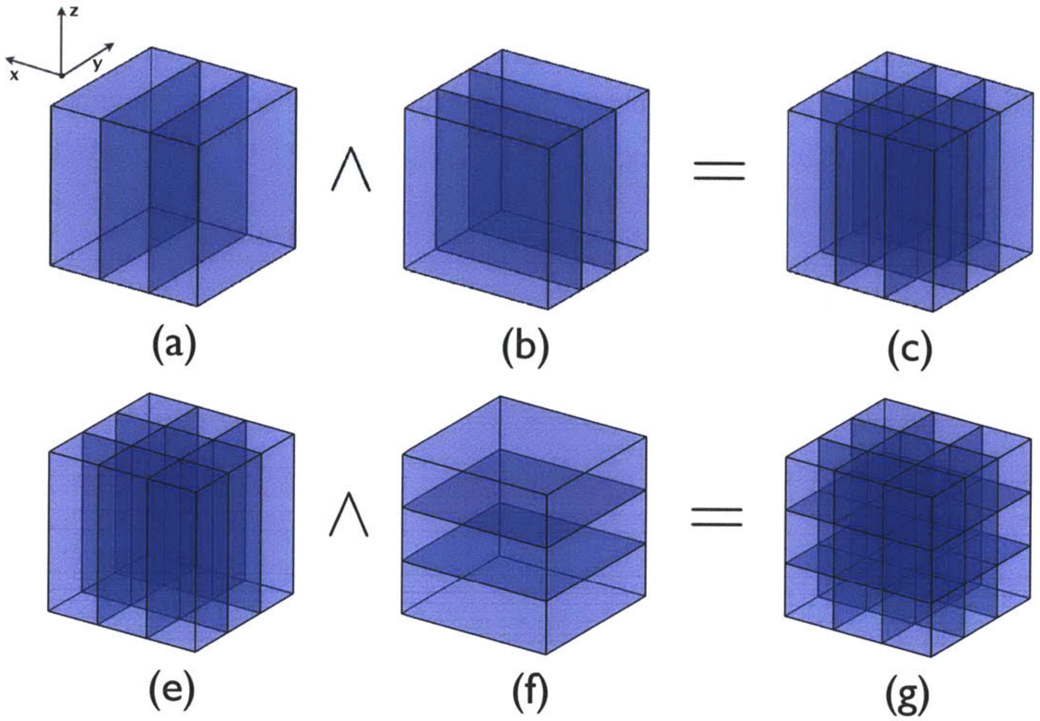


Figure 3-7: Two examples of creating features with higher dimensions using the conjunction operator.

value function of interest (*e.g.,* Figure 3-6). But how should such linear independencies be added? We observed including all of them is intractable. These two approaches are the extreme cases of feature selection. In practice, domain experts often move away from these extreme cases by manually adding important feature dependencies (*e.g.,* Silver et al., 2008; Sutton, 1996; Sutton & Barto, 1998). The next section answers the question of how to pick important feature dependencies from a theoretical perspective.

3.2.2 How to Expand the Representation?

Assuming an initial set of binary features (*e.g.,* $|\chi| = bd$) is given, we would like to expand the representation by taking into account linear dependencies using the conjunction operator.[3] This chapter focuses on the policy evaluation case, that is, given a fixed policy π, we will examine how to expand the representation in order to provide a good approximation of $\boldsymbol{V}^{\pi}$. Since the policy is assumed to be fixed, the π term is often dropped, and included implicitly.

[3] While dividing each dimension of the state space is a simple idea to create features, there are other tools for generating binary features for MDPs (See Sutton & Barto, 1998).

In order to be able to use the conjunction operator to define new features, we redefine the notion of a feature. More formally, given an initial feature function ϕ outputting vectors in $\mathbb{R}^n$, we address each of its n output elements as an *index* rather than a *feature* from this point on; $\phi_i(s) = c$ denotes index i of the initial feature function has value c in state s. [4]

$$\begin{aligned} \text{(set of indices)} \quad \mathcal{B}_n &= \{1, \cdots, n\}, \\ \text{(an index)} \quad i &\in \mathcal{B}_n. \end{aligned} \tag{3.8}$$

A feature is redefined as a set of indices:

$$\begin{aligned} \text{(a feature)} \quad f &\subseteq \mathcal{B}_n, \\ \phi_f(s) &\triangleq \bigwedge_{i \in f} \phi_i(s), \\ \phi_{\{\}}(s) &\triangleq \begin{cases} 1 & \text{if } \forall i \in \mathcal{B}_n, \phi_i(s) = 0 \\ 0 & \text{otherwise} \end{cases} . \end{aligned}$$

For example $\phi_{\{1,3\}}(.) = \phi_1(.) \wedge \phi_3(.)$. Notice that a single index can constitute a feature (*i.e.*, $f = \{1\}$). This thesis assumes that all sets are ordered based on the cardinality size of each element in ascending order, unless specified. Given a set of features $\chi = \{f_1, f_2, ...f_N\}$, the definition of $\mathbf{\Phi}$ is extended as follows:

$$\mathbf{\Phi}_\chi = \begin{bmatrix} \phi_{f_1}(s_1) & \phi_{f_2}(s_1) & \cdots & \phi_{f_N}(s_1) \\ \phi_{f_1}(s_2) & \phi_{f_2}(s_2) & \cdots & \phi_{f_N}(s_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{f_1}(s_{|\mathcal{S}|}) & \phi_{f_2}(s_{|\mathcal{S}|}) & \cdots & \phi_{f_N}(s_{|\mathcal{S}|}) \end{bmatrix}_{|\mathcal{S}| \times N} .$$

For brevity, we define ϕ_f as a column of feature matrix $\mathbf{\Phi}_\chi$ that corresponds to feature f.

$$\boldsymbol{\phi}_f = \mathbf{\Phi}_{\{f\}} = \begin{bmatrix} \phi_f(s_1) \\ \phi_f(s_2) \\ \vdots \\ \phi_f(s_m) \end{bmatrix} .$$

Also, define:

$$\text{(set of features with cardinality less than 2)} \qquad B_n \triangleq \{\varnothing, \{1\}, \{2\}, \cdots, \{n\}\},$$

[4] Notice the switch in the subscript from f (feature) to i (index).

$$\begin{aligned} \text{(set of all possible features)} \qquad \mathcal{F}_n &\triangleq \wp(\mathcal{B}_n), \\ \text{(arbitrary set of features)} \qquad \chi &\subseteq \mathcal{F}_n, \end{aligned} \tag{3.9}$$

where $\wp$ is the power set function (*i.e.,* the function that returns the set of all possible subsets). Also define operator pair : $\chi_1 \to \chi_2$, where $\chi_1, \chi_2 \subseteq \mathcal{F}_n$:

$$\begin{aligned} \texttt{pair}(\chi) &\triangleq \left\{ f \cup g \middle| f, g \in \chi, f \cup g \notin \chi \right\}, \\ \texttt{pair}^k(\chi) &\triangleq \underbrace{\texttt{pair}(\texttt{pair}(\cdots(\texttt{pair}(\chi))))}_{k \text{ times}}, \\ \texttt{pair}^0(\chi) &\triangleq \chi, \\ \texttt{full}(\chi) &\triangleq \bigcup_{i=0,\cdots,n} \texttt{pair}^i(B_n). \end{aligned}$$

Essentially, the pair operator provides the set of all possible new features built on the top of a given set of features using pairwise conjunction. The full operator generates all possible features given a set of features. It is easy to verify that:

$$\begin{aligned} \texttt{pair}(\mathcal{F}_n) = \texttt{full}(\mathcal{F}_n) &= \mathcal{F}_n \\ \forall \chi, B_n \subseteq \chi \Rightarrow \texttt{full}(\chi) &= \mathcal{F}_n \\ |\mathcal{F}_n| &= 2^n \end{aligned}$$

Now, given an MDP with a fixed policy, the feature expansion problem can be formulated mathematically. Given χ as a set of features and the corresponding approximation of the value function under the fixed policy, $\tilde{\boldsymbol{V}}_\chi = \boldsymbol{\Phi}_\chi \boldsymbol{\theta}$, find $f \in \texttt{pair}(\chi)$ that maximizes the following error reduction:

$$ER = \|\boldsymbol{V} - \tilde{\boldsymbol{V}}_\chi\| - \|\boldsymbol{V} - \tilde{\boldsymbol{V}}_{\chi \cup \{f\}}\|. \tag{3.10}$$

The following corollary provides an analytical answer to the above problem. The proof will follow.

Assumptions:

- **A1.** The MDP has a binary d-dimensional state space, $d \in \mathbb{N}^+$ (*i.e.,* $|\mathcal{S}| = 2^d$). Furthermore, each vertex in this binary space corresponds to one unique state; $s \in \{0,1\}^d$.

- **A2.** The agent's policy, π, is fixed.

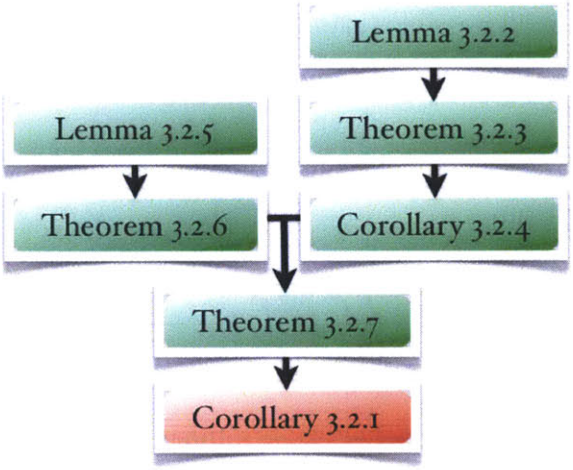


Figure 3-8: The structure of the analytical proof

- **A3.** Each feature corresponds to a coordinate of the state space (*i.e.*, $\phi(s) = s$). Hence the number of indices, n, is equal to the number of dimensions, d.

Assumption A1 is a more specific form of a general assumption where each dimension of the MDP can be represented as a finite vector and each dimension has finite number of possible values. It is simple to verify that such an MDP can be transformed into an MDP with binary dimensions. This can be done by transforming each dimension of the state space with M possible values into M binary dimensions. The MDP with binary dimensions was considered for brevity of the proofs.

Corollary 3.2.1 *Given 1) Assumptions A1-A3, 2) a set of features $\boldsymbol{\chi}$, where $B_n \subseteq \boldsymbol{\chi} \subseteq \mathcal{F}_n$, and 3) the definition of η_f in Theorem 3.2.7, if $\eta_f > \gamma$, then feature $f^* \in \mathtt{pair}(\boldsymbol{\chi})$ with the maximum guaranteed error reduction defined in Equation 3.10 can be calculated as:*

$$f^* = \operatorname{argmax}_{f \in \mathtt{pair}(\boldsymbol{\chi})} \frac{\sum_{\phi_f(s)=1} \mathbf{d}(s)\boldsymbol{\delta}(s)}{\sqrt{\left(\sum_{\phi_f(s)=1} \mathbf{d}(s)\right)}}, \tag{3.11}$$

where $\boldsymbol{\delta} = \mathbf{T}(\tilde{\boldsymbol{V}}_{\boldsymbol{\chi}}) - \tilde{\boldsymbol{V}}_{\boldsymbol{\chi}}$ is the Bellman error vector, and $\mathbf{d}$ is the steady state distribution defined in Equation 2.14.

The proof structure is shown in Figure 3-8. The rest of this section provides the building blocks of the proof, followed by a discussion of the theoretical result. Lemma 3.2.2 shows how matrices with linear independent columns can constitute larger matrices with linear independent columns. Theorem 3.2.3 states that the feature matrix corresponding to the full feature set is full column rank. Corollary 3.2.4 concludes that given an initial set of features, the feature matrix is always full column rank through the process of adding new features using the `pair` operator. Lemma 3.2.5 provides a geometrical property for vectors in d dimensional space under certain conditions. Theorem 3.2.6 provides a general guaranteed rate of error reduction for adding arbitrary features to the representation. Theorem 3.2.7 narrows down Theorem 3.2.6 to the case of binary features, where new features are built using the `pair` operator. Finally Corollary 3.2.1, as stated above, concludes that given the set of potential features obtained by the `pair` operator, the one with the maximum guaranteed error reduction is identified by Equation 3.11.

Lemma 3.2.2 *Given $m, n \in \mathbb{N}^+$ and $m \leq n$, if $\mathbf{X}_{m\times n}$ and $\mathbf{Z}_{m\times n}$ are full column rank matrices with real elements and $\mathbf{Y}_{m\times n}$ is arbitrary matrix, then matrix $\begin{bmatrix} \mathrm{X} & 0 \\ \mathrm{Y} & \mathrm{Z} \end{bmatrix}$ is a full column rank matrix.*

Proof *Matrix $\boldsymbol{U}$ is full column rank if $\boldsymbol{UD} = 0 \Rightarrow \boldsymbol{D} = 0$. Consider,*

$$\boldsymbol{U} = \begin{bmatrix} \mathrm{X} & 0 \\ \mathrm{Y} & \mathrm{Z} \end{bmatrix}, \boldsymbol{D} = \begin{bmatrix} \mathrm{A} \\ \mathrm{B} \end{bmatrix}.$$

$$\boldsymbol{UD} = 0 \Rightarrow \begin{bmatrix} \mathrm{X} & 0 \\ \mathrm{Y} & \mathrm{Z} \end{bmatrix} \begin{bmatrix} \mathrm{A} \\ \mathrm{B} \end{bmatrix} = 0 \Rightarrow \begin{cases} \mathrm{AX} = 0 \\ \mathrm{AY} + \mathrm{BZ} = 0 \end{cases}.$$

Since $\mathbf{X}$ is full column rank, $\mathrm{A} = 0 \Rightarrow \mathrm{BZ} = 0$. Now $\mathbf{Z}$ is also full column rank by assumption. Therefore $\mathrm{B} = 0$. Hence $\boldsymbol{D} = \begin{bmatrix} \mathrm{A} \\ \mathrm{B} \end{bmatrix} = 0$ ■

Theorem 3.2.3 *Given Assumptions A1-A3, $\forall n \in \mathbb{N}^+, \boldsymbol{\Phi}_{\mathcal{F}_n}$ is invertible.*

Proof *First note that $\boldsymbol{\Phi}_{\mathcal{F}_n}$ is a square matrix as $|\mathcal{F}_n| = |S| = 2^n$. Hence it would be sufficient to show that $\boldsymbol{\Phi}_{\mathcal{F}_n}$ has independent columns. The rest of the proof is through induction on n:*

$(n = 1)$: *The MDP has two states. Hence $\boldsymbol{\Phi}_{\mathcal{F}_1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $det(\boldsymbol{\Phi}_{\mathcal{F}_1}) = 1$. Notice that the first column corresponds to the null feature* (i.e., $\{\}$) *which returns* 1 *for the single state with no active feature.*

$(n = k)$: *Assume that $\boldsymbol{\Phi}_{\mathcal{F}_k}$ has independent columns.*

$(n = k + 1)$: *Based on the previous assumption, $\boldsymbol{\Phi}_{\mathcal{F}_k}$ has linear independent columns. Hence it is sufficient to show that $\boldsymbol{\Phi}_{\mathcal{F}_{k+1}}$ can be written as $\begin{bmatrix} \mathrm{X} & 0 \\ \mathrm{Y} & \mathrm{Z} \end{bmatrix}$, where $\mathbf{X} =$*

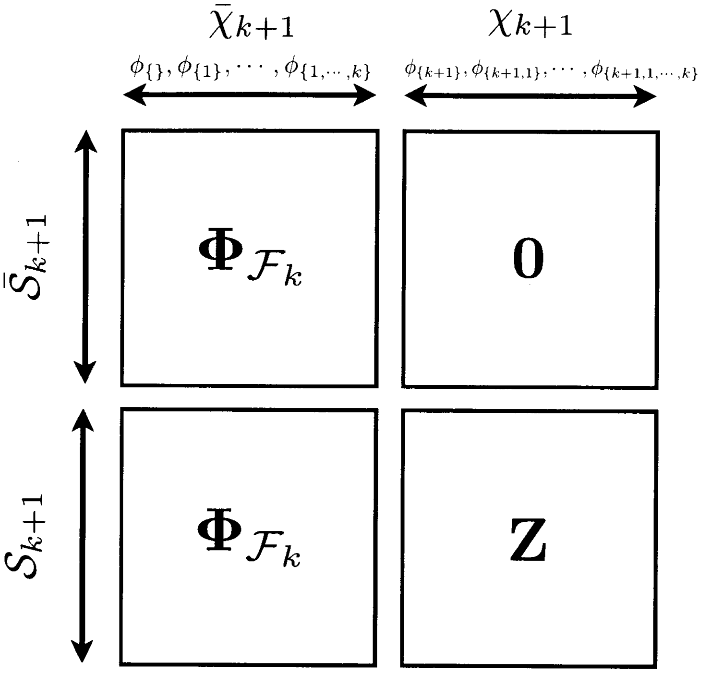

$\bar{\chi}_{k+1}$
χ_{k+1}
$\phi_{\{\}}, \phi_{\{1\}}, \cdots, \phi_{\{1,\cdots,k\}}$
$\phi_{\{k+1\}}, \phi_{\{k+1,1\}}, \cdots, \phi_{\{k+1,1,\cdots,k\}}$
$\bar{S}_{k+1}$
S_{k+1}
$\boldsymbol{\Phi}_{\mathcal{F}_k}$
$\mathbf{0}$
$\boldsymbol{\Phi}_{\mathcal{F}_k}$
$\mathbf{Z}$

Figure 3-9: Depiction of $\boldsymbol{\Phi}_{\mathcal{F}_{k+1}}$ using $\boldsymbol{\Phi}_{\mathcal{F}_k}$ as the building block. Note that features are not sorted based on their cardinality order, but it does not change the rank of the resulting matrix.

$\mathbf{Y} = \boldsymbol{\Phi}_{\mathcal{F}_k}$, *and* $\mathbf{Z}$ *has linear independent columns. Lemma 3.2.2 then completes the proof.*

The new added dimension, $k+1$, *doubles the number of states because* $|S| = 2^{k+1}$. *The new dimension also doubles the total number of possible features, as for any given set with size* k, *the total number of its subsets is* 2^k. *Divide states into the two following sets:*

$$\begin{aligned} S_{k+1} &= \{s|\phi_{\{k+1\}}(s) = 1\}, \\ \bar{S}_{k+1} &= \{s|\phi_{\{k+1\}}(s) = 0\}. \end{aligned}$$

Similarly, divide features into two sets:

$$\begin{aligned} \chi_{k+1} &= \{f|f \in \mathcal{F}_{k+1}, k+1 \in f\}, \\ \bar{\chi}_{k+1} &= \{f|f \in \mathcal{F}_{k+1}, k+1 \notin f\}. \end{aligned}$$

Construct rows and columns of $\boldsymbol{\Phi}_{\mathcal{F}_{k+1}}$*, following Figure 3-9. The values of the top left and bottom left of the matrix is* $\boldsymbol{\Phi}_{\mathcal{F}_k}$, *and the value of the top right of the matrix is* $\mathbf{0}$.

As for the bottom right ($\mathbf{Z}$), note that for all the corresponding states, $\phi_{\{k+1\}}(s) = 1$. Hence,

$$(\phi_{\{k+1\}}(s), \phi_{\{k+1,1\}}(s), \cdots, \phi_{\{k+1,1,\cdots,k\}}(s)) = (1, \phi_{\{1\}}(s), \cdots, \phi_{\{1,\cdots,k\}}(s)).$$

We know from the induction assumption that except for the first column, all other columns are linearly independent. Finally observe that the first column is the only column within $\mathbf{Z}$, with a 1 corresponding to the state with no active feature, hence independent of all other columns. ■

Corollary 3.2.4 *Given Assumptions A1-A3, $\forall \chi \subseteq \mathcal{F}_n$, $\mathbf{\Phi}_\chi$ has full rank.*

Corollary 3.2.4 highlights the fact that if feature expansion technique starts with the set of features with cardinality less than 2 (*i.e.*, B_n) and adds unique conjunctions incrementally, the resulting feature matrix always has full column rank. If after each feature expansion the weight corresponding to the new feature is set to zero, the approximate value function ($\tilde{\boldsymbol{V}}$) remains unchanged.

Insight: Theorem 3.2.3 shows that the conjunction operator creates a matrix $\mathbf{\Phi}_{\mathcal{F}_n}$ that forms a basis for $\mathbb{R}^{|S|}$ (*i.e.*, $\mathbf{\Phi}$ will have $|S|$ linearly independent columns). The $\boldsymbol{I}$ matrix is another basis for $\mathbb{R}^{|S|}$, yet no information flows between states (*i.e.*, the coverage of each feature is restricted to one state). When sorting columns of $\mathbf{\Phi}_{\mathcal{F}_n}$ based on the size of the features, it starts with features with high coverage (excluding the null feature). As more conjunctions are introduced, the coverage is reduced exponentially (*i.e.*, the number of active features are decreased exponentially by the size of the feature set). Next, we explain how adding binary features can lead to guaranteed approximation error reduction. First, we extend Theorem 3.6 of Parr et al. (2007) by deriving a lower bound on the improvement caused by adding a new feature.

Definition Define the angle between two vectors $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^d, d \in \mathbb{N}^+$ as

$$\angle(\boldsymbol{X}, \boldsymbol{Y}) = \arccos\left(\frac{\langle \boldsymbol{X} \cdot \boldsymbol{Y}\rangle}{||\boldsymbol{X}||.||\boldsymbol{Y}||}\right).$$

Note that $0 \leq \angle(\boldsymbol{X}, \boldsymbol{Y}) \leq \pi$.

Lemma 3.2.5 *Let L be the plane specified by three distinct points $P, Q, C \in \mathbb{R}^d$, with $\alpha = \angle(\boldsymbol{CQ}, \boldsymbol{CP}) > 0$. Assume that the additional point $X \in \mathbb{R}^d$ is not necessarily in L. Define the angles $\beta = \angle(\boldsymbol{CX}, \boldsymbol{CQ})$ and $\omega = \angle(\boldsymbol{CX}, \boldsymbol{CP})$. Now let $\boldsymbol{CP'}$ denote*

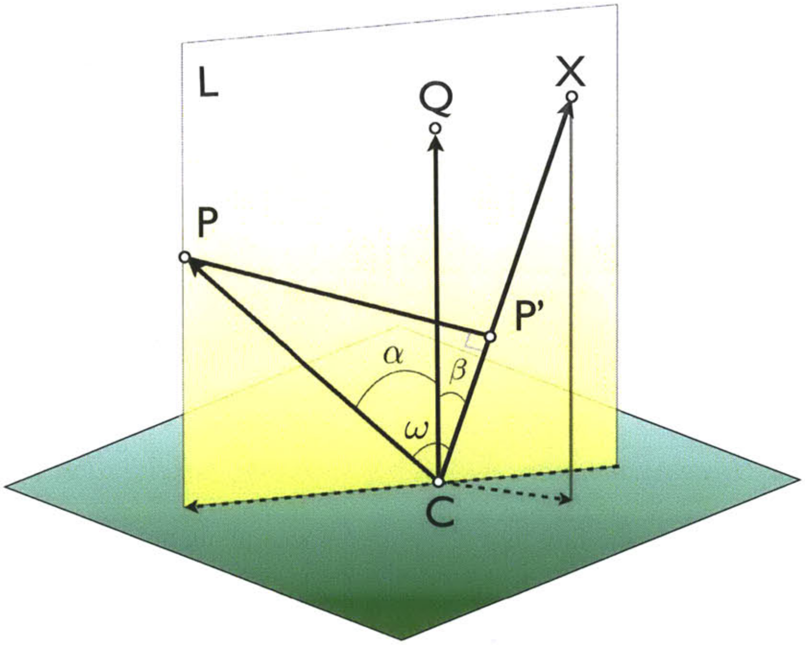


Figure 3-10: A $3D$ visualization of d dimensional points L_1 and Q and vector X with C as the center. $\|\boldsymbol{PP'}\|$ is maximized when $\omega = \alpha + \beta$.

the orthogonal projection of $\boldsymbol{CP}$ *on* $\boldsymbol{CX}$*. If* $\alpha + \beta < \frac{\pi}{2}$*, then* $\|\boldsymbol{PP'}\|$ *is maximized when* $\boldsymbol{CX} \in L$*.*

Proof Let us first assume that $\boldsymbol{CX} \notin L$, hence there exists a three dimensional subspace defined by $\boldsymbol{CX}$ and L. Figure 3-10 depicts such a space. Then,

$$\underset{\omega}{\operatorname{argmax}} \|\boldsymbol{PP'}\| = \underset{\omega}{\operatorname{argmax}} \|\boldsymbol{CP}\| \sin(\omega) = \underset{\omega}{\operatorname{argmax}} \sin(\omega).$$

Since $0 < |\alpha - \beta| \leq \omega \leq \alpha + \beta < \frac{\pi}{2}$, then $\underset{\omega}{\operatorname{argmax}} \|\boldsymbol{PP'}\| = \alpha + \beta$, which implies that $\boldsymbol{CX} \in L$ and thus is a contradiction. ■

Theorem 3.2.6 *Given an MDP with a fixed policy, where the value function is approximated as* $\tilde{\boldsymbol{V}}$*, define* $\boldsymbol{\delta} = \mathbf{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}$*, where* $\mathbf{T}$ *is the Bellman operator defined in Equation 2.7 and* $\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| = x > 0$*, where* $\boldsymbol{V}$ *is the optimal value for all states. Then* $\forall \boldsymbol{\phi}_f \in \mathbb{R}^{|\mathcal{S}|} : \beta = \angle(\boldsymbol{\phi}_f, \boldsymbol{\delta}) < \arccos(\gamma)$

$$\exists \xi \in \mathbb{R} : \|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| - \|\boldsymbol{V} - (\tilde{\boldsymbol{V}} + \xi \boldsymbol{\phi}_f)\| \geq \zeta x, \tag{3.12}$$

where γ is the discount factor and

$$\zeta = 1 - \gamma\cos(\beta) - \sqrt{1-\gamma^2}\sin(\beta) < 1.$$

Furthermore, if these conditions hold and $\tilde{\boldsymbol{V}} = \boldsymbol{\Phi\theta}$ with $\phi_f \notin \text{span}(\boldsymbol{\Phi})$ then:

$$\|\boldsymbol{V} - \boldsymbol{\Pi V}\| - \|\boldsymbol{V} - \boldsymbol{\Pi}'\boldsymbol{V}\| \geq \zeta x \tag{3.13}$$

where $\boldsymbol{\Pi}$ and $\boldsymbol{\Pi}'$ are defined by Equation 2.16 using $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}' = [\boldsymbol{\Phi}\ \phi_f]$ respectively.

Proof Consider both cases of the orientation of points $\boldsymbol{V}$ and $\mathbf{T}(\tilde{\boldsymbol{V}})$ with respect to each other:

- $(\mathbf{T}(\tilde{\boldsymbol{V}}) \neq \boldsymbol{V})$: Due to the contraction property of the Bellman operator, if $\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| = x$, then $\|\boldsymbol{V} - \mathbf{T}(\tilde{\boldsymbol{V}})\| \leq \gamma x$. Define α as the $\angle(\boldsymbol{V} - \tilde{\boldsymbol{V}}, \boldsymbol{\delta})$, then

  $$\sin(\alpha) = \frac{\|\boldsymbol{V} - \mathbf{T}(\tilde{\boldsymbol{V}})\|}{\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\|} \leq \gamma \quad \Rightarrow \quad \alpha \leq \arcsin(\gamma)$$

  Furthermore, by assumption $0 \leq \beta < \arccos(\gamma) = {}^{\pi}/_{2} - \arcsin(\gamma)$. Combined, these conditions indicate that $\alpha + \beta < {}^{\pi}/_{2}$.

  For the next step, given $\xi > 0$, mapping the points $\boldsymbol{V}, \tilde{\boldsymbol{V}}, \mathbf{T}(\tilde{\boldsymbol{V}}), \tilde{\boldsymbol{V}} + \xi\phi_f$ to P, C, Q, X in Lemma 3.2.5 shows that the orthogonal projection length of vector $\boldsymbol{V} - \tilde{\boldsymbol{V}}$ on $\tilde{\boldsymbol{V}} + \xi\phi_f - \tilde{\boldsymbol{V}}$ is maximized when all four points are coplanar and

  $$\omega = \angle(\boldsymbol{V} - \tilde{\boldsymbol{V}}, \xi\phi_f) = \alpha + \beta.$$

  Notice that the coplanar argument is implicit in the proof of Theorem 3.6 of Parr et al. (2007). Figure 3-11 depicts the geometrical view in such a plane, where $\xi^* = \text{argmin}_\xi \|\boldsymbol{V} - (\tilde{\boldsymbol{V}} + \xi\phi_f)\|$, $x' = x\sin(\omega)$. As shown above, $\alpha \leq \arcsin(\gamma)$ and $0 \leq \alpha + \beta < \frac{\pi}{2}$, thus

  $$\sin(\alpha + \beta) \leq \sin(\arcsin\gamma + \beta) = \gamma\cos(\beta) + \sin(\beta)\sqrt{1-\gamma^2}$$

  Hence

  $$\begin{aligned} x' &\leq x\left(\gamma\cos(\beta) + \sqrt{1-\gamma^2}\sin(\beta)\right) \\ \Rightarrow x - x' &\geq x\left(1 - \gamma\cos(\beta) - \sqrt{1-\gamma^2}\sin(\beta)\right) \equiv \zeta x \end{aligned}$$

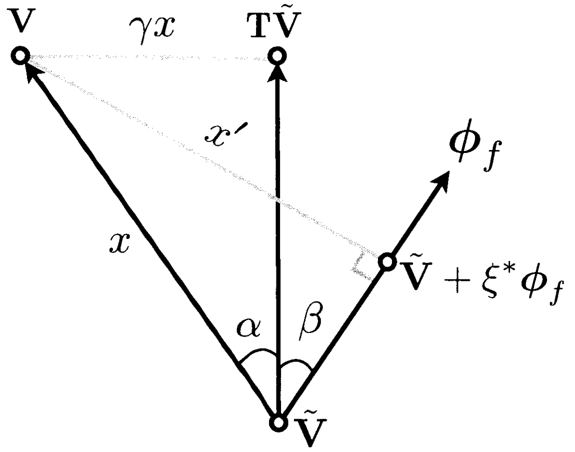

V
γx
$\mathbf{T}\tilde{\mathbf{V}}$
x'
ϕ_f
x
$\tilde{\mathbf{V}} + \xi^* \phi_f$
α
β
$\tilde{\mathbf{V}}$

Figure 3-11: Geometrical view of $\boldsymbol{V}, \tilde{\boldsymbol{V}}, \mathbf{T}(\tilde{\boldsymbol{V}})$, and ϕ_f. As β shrinks x' gets closer to γx.

Looking at Figure 3-11, it is easy to verify that $x - x' = \|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| - \|\boldsymbol{V} - (\tilde{\boldsymbol{V}} + \xi\phi_f)\|$, which completes the proof for the case $\mathbf{T}(\tilde{\boldsymbol{V}}) \neq \boldsymbol{V}$.

- $(\mathbf{T}(\tilde{\boldsymbol{V}}) = \boldsymbol{V})$: This means that $\alpha = 0$. If ϕ_f crosses $\boldsymbol{V}$, it means $\beta = 0$ and $\tilde{\boldsymbol{V}} + \xi^*\phi_f = \boldsymbol{V}$. Hence $\zeta = 1 - \gamma\cos(\beta) - \sqrt{1-\gamma^2}\sin(\beta) = 1 - \gamma$ and $\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| - \|\boldsymbol{V} - (\tilde{\boldsymbol{V}} + \xi^*\phi_f)\| = \|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| = x \geq \zeta x$. When ϕ_f does not cross $\boldsymbol{V}$, together they form a plane in which:

$$\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| - \|\boldsymbol{V} - (\tilde{\boldsymbol{V}} + \xi^*\phi_f)\| \;=\; x\Big(1 - \sin(\beta)\Big)$$

In order to complete the proof, a lower bound for the above error reduction is derived:

$$\begin{aligned} & 0 < \beta < \arccos(\gamma) = {}^{\pi}/_{2} - \arcsin(\gamma), 0 \leq \gamma < 1 \\ \Rightarrow\; & 0 < \beta + \arcsin(\gamma) < {}^{\pi}/_{2} \\ \Rightarrow\; & \sin(\beta) \leq \sin(\beta + \arcsin(\gamma)) = \gamma\cos(\beta) + \sqrt{1-\gamma^2}\sin(\beta) \\ \Rightarrow\; & (1 - \sin(\beta))x \geq (1 - \gamma\cos(\beta) - \sqrt{1-\gamma^2}\sin(\beta))x \equiv \zeta x \end{aligned}$$

In order to prove the second part of the theorem, the argument has to be specialized to the linear function approximation case, meaning $\tilde{\boldsymbol{V}} = \boldsymbol{\Phi\theta}$. Since the first part of the proof holds for any approximation, lets consider the case where $\tilde{\boldsymbol{V}} = \boldsymbol{\Pi V}$. Showing $\tilde{\boldsymbol{V}} + \xi^*\phi_f = \boldsymbol{\Pi' V}$ completes the proof as substituting $\tilde{\boldsymbol{V}}$ and $\tilde{\boldsymbol{V}} + \xi^*\phi_f$ to $\boldsymbol{\Pi V}$ and $\boldsymbol{\Pi' V}$ turns Inequality 3.12 to Inequality 3.13.

To proceed, decompose ϕ_f into two vectors $\phi_f^{\|}$ and $\phi_f^{\perp}$, where $\phi_f^{\|} \in \text{span}(\boldsymbol{\Phi})$ and $\phi_f^{\perp} \perp \text{span}(\boldsymbol{\Phi})$. Note that since $\phi_f \notin \text{span}(\boldsymbol{\Phi})$, then $\phi_f^{\perp}$ is not a null vector. The proof

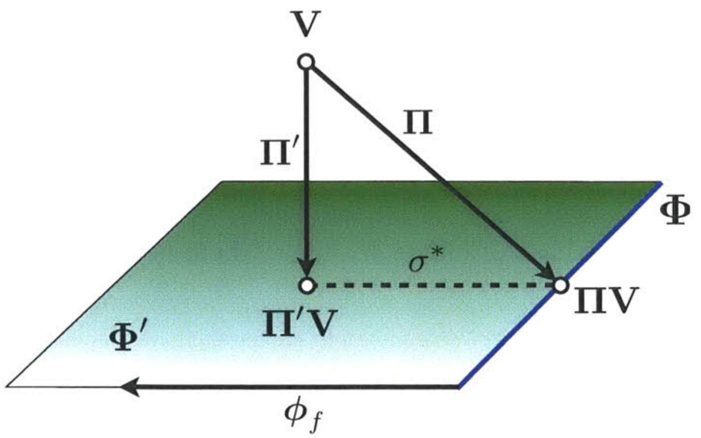


Figure 3-12: Increasing the dimensionality of the projection operator: $\phi_f \perp \texttt{span}(\Phi)$.

proceeds by showing $\tilde{V} + \xi^* \phi_f = \Pi' V$ with the assumptions of $\phi_f^{\|} = \mathbf{0}$ (*i.e.*, $\phi_f = \phi_f^{\perp}$), but this assumption will be relaxed shortly thereafter. Notice that $\Phi' = [\Phi \; \phi_f]$ has one extra column rank defined by ϕ_f, which is perpendicular to the $\texttt{span}(\Phi)$. Figure 3-12 provides a geometric view of the situation. The blue line and the green plane highlight $\texttt{span}(\Phi)$ and $\texttt{span}(\Phi')$ correspondingly. Both Π and Π' are orthogonal projections into these subspaces hence have the same span as Φ and Φ' respectively. Therefore:

$$\texttt{span}(\Pi') = \texttt{span}(\Pi) + \texttt{span}(\phi_f).$$

For any given value function V, because $\phi_f \perp \texttt{span}(\Pi)$:

$$\begin{aligned}
\Pi' V &= \Pi V + \sigma^* \phi_f, \\
\text{where, } \sigma^* &\triangleq \underset{\sigma}{\operatorname{argmin}} \|(V - \Pi V) - \sigma \phi_f\|, \\
(\Pi V = \tilde{V}) \quad &= \underset{\sigma}{\operatorname{argmin}} \|V - (\tilde{V} + \sigma \phi_f)\| \\
&= \xi^* \\
\Rightarrow \Pi' V &= \Pi V + \xi^* \phi_f, \\
&= \tilde{V} + \xi^* \phi_f
\end{aligned}$$

The extension to the case where $\phi_f^{\|} \neq \mathbf{0}$ is straightforward. Consider a subspace defined by two representation matrices Φ_1 and Φ_2 (*i.e.*, $\texttt{span}(\Phi_1) = \texttt{span}(\Phi_2)$), and corresponding orthogonal projection operators Π_1 and Π_2 as defined in Equation 2.16. Since both operators provide the solution to the same convex optimization (*i.e.*, $\min_{\theta} \|V - \tilde{V}\|$), where both domain and target space are identical, hence their outputs are equal (*i.e.*, $\tilde{V}_1 = \Pi_1 V =$

$\mathbf{\Pi}_2\boldsymbol{V} = \tilde{\boldsymbol{V}}_2$).[5] Consequently if $\phi_f^{\|}$ is added as the last column of $\boldsymbol{\Phi}'$, it does not change $\texttt{span}(\boldsymbol{\Phi}')$. Hence the result of the projection remains intact. ■

The previous theorem provided a general guaranteed rate of convergence for feature expansion techniques in the context of linear function approximators. The next theorem specializes the above result to the case of binary features, where new features are built using the conjunction operator.

Theorem 3.2.7 *Given Assumptions A1-A3,* $\chi \subseteq \mathcal{F}_n, \tilde{\boldsymbol{V}} = \boldsymbol{\Phi}_\chi\boldsymbol{\theta}, \boldsymbol{\delta} = \mathrm{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}$*, and* $\|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| = x > 0$*, then*

$$\forall f \in \texttt{pair}(\chi), \quad \textit{if} \quad \eta_f = \frac{\sum_{\phi_f(s)=1} \mathbf{d}(s)\boldsymbol{\delta}(s)}{\sqrt{\left(\sum_{\phi_f(s)=1}\mathbf{d}(s)\right)\left(\sum_{s\in\mathcal{S}}\mathbf{d}(s)\boldsymbol{\delta}^2(s)\right)}} > \gamma$$

then

$$\begin{aligned}
\exists\xi \in \mathbb{R} : \|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| - \|\boldsymbol{V} - (\tilde{\boldsymbol{V}} + \xi\boldsymbol{\phi}_f)\| &\geq \zeta x, && (3.14)\\
\|\boldsymbol{V} - \mathbf{\Pi}\boldsymbol{V}\| - \|\boldsymbol{V} - \mathbf{\Pi}'\boldsymbol{V}\| &\geq \zeta x, && (3.15)\\
\text{where } 1 - \gamma\eta_f - \sqrt{1-\gamma^2}\sqrt{1-\eta_f^2} &= \zeta && (3.16)
\end{aligned}$$

Proof Theorem 3.2.6 provides a general rate of convergence for approximation error, when arbitrary feature vectors are added to the feature matrix. Hence it is sufficient to show that the conditions of Theorem 3.2.6 holds in this new theorem, namely: 1) $\beta = \angle(\boldsymbol{\phi}_f, \boldsymbol{\delta}) < \arccos(\gamma)$ and 2) $\boldsymbol{\phi}_f \notin \texttt{span}(\boldsymbol{\Phi}_\chi)$. The latter is already shown through Corollary 3.2.4. As for the former:

$$\begin{aligned}
\cos(\beta) &= \frac{\langle\boldsymbol{\phi}_f \cdot \boldsymbol{\delta}\rangle}{\|\boldsymbol{\phi}_f\|.\|\boldsymbol{\delta}\|} = \frac{\sum_{\phi_f(s)=1} \mathbf{d}(s)\boldsymbol{\delta}(s)}{\sqrt{\left(\sum_{\phi_f(s)=1}\mathbf{d}(s)\right)\left(\sum_{s\in\mathcal{S}}\mathbf{d}(s)\boldsymbol{\delta}^2(s)\right)}} = \eta_f,\\
\beta &= \arccos(\eta_f).
\end{aligned}$$

By the assumption made earlier, $\eta_f > \gamma$. Hence $\beta < \arccos(\gamma)$. Satisfying preconditions of Theorem 3.2.6, both Equations 3.12 and 3.13 are obtained. Switching $\cos(\beta)$ with η_f completes the proof:

$$\begin{aligned}
\|\boldsymbol{V} - \mathbf{\Pi}\boldsymbol{V}\| - \|\boldsymbol{V} - \mathbf{\Pi}'\boldsymbol{V}\| &\geq \zeta x\\
\exists\xi \in \mathbb{R} : \|\boldsymbol{V} - \tilde{\boldsymbol{V}}\| - \|\boldsymbol{V} - (\tilde{\boldsymbol{V}} + \xi\boldsymbol{\phi}_f)\| &\geq \zeta x,
\end{aligned}$$

[5] Note that the corresponding coordinates, $\boldsymbol{\theta}$, in each case can be different, yet the resulting $\tilde{\boldsymbol{V}}$ are identical.

where,

$$\zeta = 1 - \gamma\eta_f - \sqrt{1-\gamma^2}\sqrt{1-\eta_f^2} < 1.$$

■

Theorem 3.2.7 provides sufficient conditions for guaranteed rate of convergence of the value function approximation by adding conjunctions of existing features. Hence the next corollary identifies the new feature, facilitating the best guaranteed rate of convergence of the value function.

Corollary 3.2.8 *Given Assumptions A1-A3,* $\chi \subseteq \mathcal{F}_n, \tilde{\boldsymbol{V}} = \boldsymbol{\Phi}_\chi \boldsymbol{\theta}, \boldsymbol{\delta} = \mathbf{T}(\tilde{\boldsymbol{V}}) - \tilde{\boldsymbol{V}}$*, and* $||\boldsymbol{V} - \tilde{\boldsymbol{V}}|| = x > 0$*, then the feature* $f \in \mathtt{pair}(\chi)$ *with the maximum guaranteed rate of convergence is:*

$$f^* = \underset{f \in \mathtt{pair}(\chi), \eta_f > \gamma}{\operatorname{argmax}} \frac{\sum_{\phi_f(s)=1} \mathbf{d}(s)\boldsymbol{\delta}(s)}{\sqrt{\sum_{\phi_f(s)=1} \mathbf{d}(s)}}. \tag{3.17}$$

Insight: This result shows how feature coverage affects the rate of convergence of the approximation. Equation 3.17 shows how the feature coverage is a double edge sword; while more coverage includes more weighted Bellman error (*i.e.,* the numerator) resulting in a higher convergence rate, it also contributes negatively to the rate of convergence (*i.e.,* the denominator). The ideal feature would be active in a single state with all of the Bellman error. Intuitively this is expected, because adding this feature makes the approximation exact. When the weighted sum of Bellman errors is equal for a set of features, the feature with the least coverage is preferable. On the other hand, when all features have the same coverage, the one with the highest weighted Bellman error coverage is ideal.

Another interesting observation is the difficulty of finding features that give the guaranteed convergence rate with respect to γ. In general, larger values of γ makes the task harder since the constraint $\eta_f > \gamma$ rejects more features in the set $\mathtt{pair}(\chi)$.

Finally, this theorem provides insight on why adding features with large coverage in early stages of learning and specific features later during the learning process would be beneficial. In the beginning of the learning process where feature weights have not been adjusted, the Bellman error is generally large everywhere. Therefore features with large coverage have higher chances of having good convergence rates due to the numerator of Equation 3.17. As weights are updated, the Bellman error is reduced correspondingly. This will make the denominator of Equation 3.17 the deciding factor, rendering features with large coverage ineffective. This transition in the effect of feature coverage may explain the empirical results of Whiteson *et al.* (2007), where the agent starts with one big tile and adaptively increases the number of tiles, learning faster compared to it starting with a fixed learned representation.

Can Assumption A3 be relaxed? One may wonder about the final result of this feature expansion scheme started with an arbitrary feature function which does not comply with Assumption A3. While Assumption A3 is a sufficient condition for realizing the best representation (*i.e.,* the representation that assigns one unique feature to each state) in the limit, we conjecture that this assumption is not necessary. More specifically, we conjecture that given an arbitrary initial features $\chi \subseteq \mathcal{F}_n$, the following assumption, which is more general than assumption A3, is sufficient to reach the best representation:

$$\texttt{full}(\chi) = \mathcal{F}_n.$$

Assumption A3 is equivalent to $B_n \subseteq \chi$, which immediately satisfies the above condition.

3.3 Incremental Feature Dependency Discovery

Corollary 3.2.8 forms the theoretical foundation of the family of incremental Feature Dependency Discovery (iFDD) algorithms. The general process of iFDD methods is to start with an initial set of binary features and expand the representation by adding new features that approximate f^* in Equation 3.17. The $\boldsymbol{\theta}$ can be calculated using the BRM or LSTD methods explained in Chapter 2. There are two problems in calculating f^* exactly: 1) calculating $\mathrm{d}(s)$ and the Bellman error $\boldsymbol{\delta}(s)$ is not possible without knowing the MDP completely, and 2) calculating η_f requires a sweep through the whole state space, resulting in computational complexity dependent on $|S|$. Sampling techniques mitigate the first problem. In particular, L_1 samples in the form of $\langle s_i, r_i, s'_i \rangle$ are obtained through executing the fixed policy π. Consequently the Bellman error is replaced with the temporal difference error. To address the second problem, the $\eta_f > \gamma$ condition is dropped from the maximization. This exclusion brings no harm, as long as one feature $f \in \texttt{pair}(\chi)$ exists for which $\eta_f > \gamma$. Taking both these modifications into account, Equation 3.17 can be approximated as follows:

$$\tilde{f}^* = \operatorname*{argmax}_{f \in \texttt{pair}(\chi)} \frac{\sum_{i\in\{1,\cdots,L_1\},\phi_f(s_i)=1} r_i + [\gamma\phi(s'_i) - \phi(s_i)]^\top \boldsymbol{\theta}}{\sqrt{\sum_{i\in\{1,\cdots,L_1\},\phi_f(s_i)=1} 1}}. \tag{3.18}$$

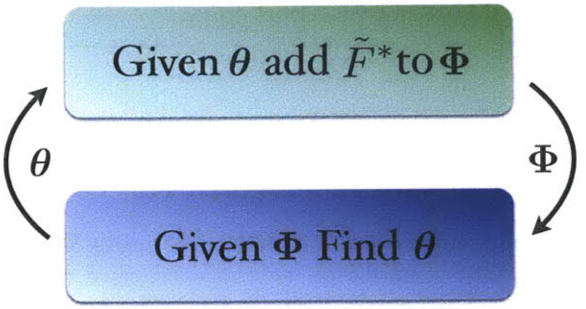

Given θ add $\tilde{F}^*$ to Φ
θ
Φ
Given Φ Find θ

Figure 3-13: The feature discovery/policy evaluation loop. Note there is no control involved, so the output of the loop is an approximate value function for a fixed policy π.

Empirically, the following approximation of the $\tilde{f}^*$ also has been shown to yield good results in the online control case (Geramifard et al., 2011c):

$$\tilde{f}^* = \operatorname*{argmax}_{f \in \texttt{pair}(\chi)} \sum_{i \in \{1,\dots,L_1\}, \phi_f(s_i)=1} |r_i + [\gamma\phi(s'_i) - \phi(s_i)]^\top \boldsymbol{\theta}| \tag{3.19}$$

Both these approximations will be empirically probed in the policy evaluation case in Section 3.3.1. Given $\boldsymbol{\theta}$, the above equations add a new feature to the representation. To close the loop, a policy evaluation technique should be employed to find the new $\boldsymbol{\theta}$, as shown in Figure 3-13. Algorithm 16 shows the corresponding pseudo-code for combining iFDD and LSTD, named FDLSTD.[6] Each iteration of the loop realizes the feature discovery (lines 5-6) / policy evaluation (lines 7-8) loop. The feature discovery step requires inspecting all pairs of features, incurring $\Theta(n^2)$ complexity, where each inspection requires $\Theta(nL_1)$ computation, leading to $\Theta(n^3L_1)$ total complexity. Note that n is increased on every iteration by one.

3.3.1 Empirical Results

In order to empirically evaluate the performance of FDLSTD, this algorithm was used to evaluate the value function corresponding to a fixed policy for the classical pendulum domain with a fixed number of samples.

Inverted Pendulum Following Lagoudakis and Parr's work (2003), Figure 3-14 depicts the problem. The system's state is the pendulum's angle and angular velocity, $(\theta, \dot{\theta})$, actions are three different torques, episodes last up to 3,000 steps, and $\gamma = 0.95$. The initial

[6] The BRM approach can also be used, yet the double sampling requirement should be handled carefully, akin to Algorithm 15.

```
Algorithm 16:Feature Discovery LSTD (FDLSTD)                                              Complexity
    Input: $\pi$,$\chi \subseteq \mathcal{F}$
    Output: $\tilde{\boldsymbol{V}}^{\pi}$
1   Create $L_1$ samples in the form of $\langle s_i, r_i, s_i' \rangle$ following policy $\pi$
2   Calculate $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{b}}$ using Equations 2.37-2.39 and $\boldsymbol{\Phi}_\chi$                $\Theta(n^2 L_1)$
3   $\boldsymbol{\theta} \leftarrow \tilde{\boldsymbol{A}}^{-1}\tilde{\boldsymbol{b}}$    (Use regularization if the inverse does not exist)                  $\Theta(n^3)$
4   repeat
5       Calculate $\tilde{f}^*$ using Equation 3.18 or Equation 3.19                             $\Theta(n^3 L_1)$
6       $\chi \leftarrow \chi \cup \{\tilde{f}^*\}$
7       Calculate $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{b}}$ using Equations 2.37-2.39 and $\boldsymbol{\Phi}_\chi$            $\Theta(n^2 L_1)$
8       $\boldsymbol{\theta} \leftarrow \tilde{\boldsymbol{A}}^{-1}\tilde{\boldsymbol{b}}$    (Use regularization if the inverse does not exist)              $\Theta(n^3)$
9   until maximum iterations is reached
10  return $\tilde{\boldsymbol{V}}^{\pi} = \boldsymbol{\Phi}_\chi \boldsymbol{\theta}$
```

features consisted of discretizing each dimension into 20 buckets (for 120 features total). The policy π was fixed to apply the torque in the opposite direction of $\dot{\theta}$. Algorithm 16 was implemented with four types of feature expansion techniques (*i.e.,* line 5):

- **Random**: $\tilde{f}^* = \texttt{random}\{f | f \in \texttt{pair}(\chi), \exists s \in \texttt{Samples}, \phi_f(s) \neq 0\}$. This random expansion method ignores features that are inactive for all samples, because they cannot improve the approximation.

- **FDLST**: Equation 3.19.

- **FDLST$^+$**: Equation 3.18.

- **Best**: $\tilde{f}^* = \text{argmin}_{f \in \texttt{pair}(\chi)} \sqrt{\sum_{s \in \texttt{Samples}} (\boldsymbol{V} - \tilde{\boldsymbol{V}}_{\chi \cup \{f\}})^2}$, where $\boldsymbol{V}$ was calculated by running LSTD on 100,000 samples using a tabular representation with $1,200$ features.

The "Best" method is not feasible in practice as it requires access to the true value function, yet it provides the best feature conjunction that minimizes the optimization criteria. The number of samples was fixed to 1,000, 2,000, and 5,000 for 3 set of batch experiments. The performance of each method was calculated based on the RMSE of $\tilde{\boldsymbol{V}}$ and $\boldsymbol{V}$. Results were averaged over 20 runs. The random seed was fixed among all methods, resulting in identical set of data across all expansion techniques.

Figure 3-15 depicts the results of running these four algorithms. Shaded areas around each plot show 95% confidence interval regions. The X-axis corresponds to the number of iterations (*i.e.,* number of features added to the representation), while the Y-axis depicts

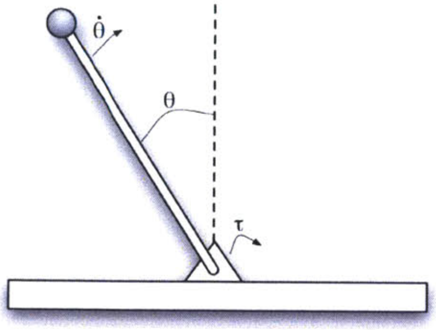
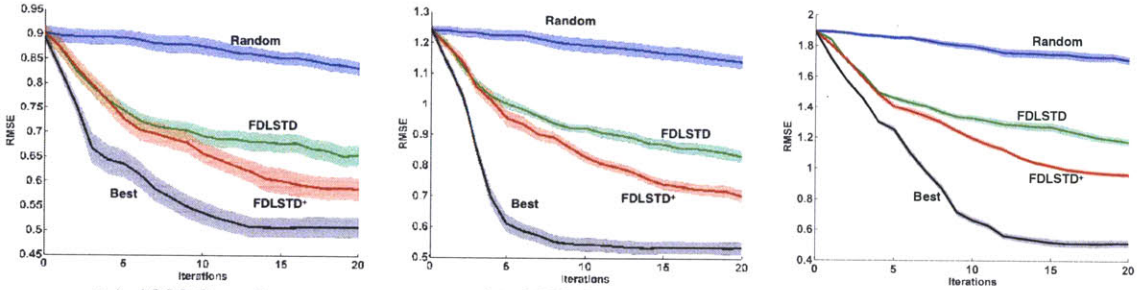


Figure 3-14: The inverted pendulum domain



(a) 1000 Samples

(b) 2000 Samples

(c) 5000 Samples

Figure 3-15: The accuracy of policy evaluation results of Random, iFDD and iFDD$^+$ approaches compared to the Best possible feature expansion. All methods add a conjunction of two existing features to the pool of features on each iteration.

the RMSE of the resulting value function approximation with respect to $\boldsymbol{V}$. As expected, both FDLSTD and FDLSTD$^+$ methods provided much faster convergence rates than the Random approach. Another interesting observation is that FDLSTD$^+$, which provided a better approximation of Equation 3.17 , yielded better convergence compared to FDLSTD. As the number of samples is increased, the sub-optimality of FDLSTD and FDLSTD$^+$ became more evident. It is important to understand that Equation 3.17 identifies the feature with the best guaranteed convergence rate in the worst case scenario. Hence in practice, for a specific MDP. a different feature may yield better convergence result. In other words, if a new feature was generated by solving Equation 3.17 exactly for the Pendulum domain, the resulting learning speed could be still slower than the best plot shown here as black.

3.4 Reducing the Computational Complexity of iFDD

As mentioned in the previous section, executing both versions of iFDD (*i.e.,* Equation 3.18 and 3.19) requires $\Theta(n^3 L_1)$ complexity, where n is the total number of features, increased on each iteration by one. Since, the goal is to use iFDD in large state spaces, this section focuses on reducing this complexity.

With the use of initial set of features with sparsity of $^k/_n$, the complexity of iFDD is reduced to $\Theta(n^2 k L_1)$ due to the fast calculation of $\phi(s_i)$. To accommodate the time dependency of features, define k_t and n_t as the maximum number of active features and total number of features at iteration t. Hence the complexity of feature discovery can be written as $\Theta(n_t^2 k_t L_1)$. It is easy to verify that if iFDD starts with B_n as the initial representation, then

$$\begin{aligned} \lim_{t\to\infty} k_t &= 2^{k_0}, \\ \lim_{t\to\infty} n_t &= 2^{n_0}. \end{aligned} \tag{3.20}$$

Hence,

$$\lim_{t\to\infty} \Theta(n_t^2 k_t L_1 |\mathcal{A}|) = \Theta(2^{2n_0+k_0} L_1).$$

This complexity poses a challenge for online methods because even for small number of k_0 and n_0, $2^{2n_0+k_0}$ can be computationally intensive.

The next chapter explains how, by incrementally estimating the optimization criteria in Equations 3.18 and 3.19, the worse case complexity of iFDD can be dropped to $\Theta(k_t^3) = \Theta(2^{3k_0})$.[7] It would be ideal if feature values are changed in a way so that k_t is not increased on each iteration, resulting in the complexity of iFDD to be $\Theta(k_0^3)$. But what is the cause for k_t parameter to grow? The answer is: *new features added to the representation do not change the existing feature values.* For example, if feature $f = g \wedge h$ is active, then features g and h must also be active. Hence in the limit where all combinations are covered, the number of active features becomes exponential in the number of initial active features.

In order to break this trend, a greedy activation mechanism is employed to create a sparse set of features that provides a summary of all active features. In general finding the minimum covering set is NP-complete but greedy selection gives the best polynomial time approximation because the set coverage problem is submodular. For example, if initial features g and h are active in state s and feature $f = g \wedge h$ has been discovered, then for state s feature f is activated, while both features g and h are deactivated since g and h are

[7] Note that k_t is upper bounded by 2^{k_0} due to Equation 3.20.

```
Algorithm 17:Generate Sparse Feature Vector ($\mathring{\phi}$)                 Complexity
  Input: $\phi^0(s), \chi \subseteq \mathcal{F}_n$
  Output: $\phi(s)$
1 $\mathring{\phi}(s) \leftarrow \bar{0}$
2 activeInitialFeatures $\leftarrow \{i|\phi_i^0(s) = 1\}$
3 Candidates $\leftarrow$ SortedPowerSet(activeInitialFeatures)            $\Theta(2^{k_0})$
4 while activeInitialFeatures $\neq \varnothing$ do
5     $f \leftarrow$ Candidates.next()
6     if $f \in \chi$ then
7         activeInitialFeatures $\leftarrow$ activeInitialFeatures $\setminus f$     $\Theta(k_0)$
8         $\mathring{\phi}_f(s) \leftarrow 1$
9 return $\mathring{\phi}(s)$
```

covered by f. Algorithm 17 describes the above process more formally: given the initial feature vector, $\phi^0(s)$ and the current pool of features χ, candidate features are found by identifying the active initial features and calculating the power set ($\wp$) sorted by set sizes (lines 2,3). The loop (line 4) keeps activating candidate features that exist in the feature set χ until all active initial features are covered (lines 5-8). The outer loop of Algorithm 17 requires $\Theta(2^{k_0})$ operations in the worst case and each iteration through the loop involves $\Theta(1)$ lookup (using a hashing mechanism) and $\Theta(k_0)$ set difference. While the result of this algorithm is a sparse set of features, running the algorithm requires $\Theta(k_0 2^{k_0})$ complexity. The next chapter shows how this sparsification plays a critical role in creating an online iFDD process with the worst per-time-step complexity of $\Theta(k_0^3 2^{k_0})$. The next section investigate the theoretical consequences of applying the sparsification to the representation.

3.4.1 Sparsifying $\mathbf{\Phi}$: Theoretical Implications

In order to investigate the effects of sparsification of $\mathbf{\Phi}$ theoretically, Algorithm 17 is defined in a matrix form. Define:

$$\begin{aligned}\bar{\mathbf{X}} &\triangleq \neg\mathbf{X},\\ \mathbf{XY} &\triangleq \mathbf{X} \wedge \mathbf{Y},\\ \mathbf{X}+\mathbf{Y} &\triangleq \mathbf{X} \vee \mathbf{Y}.\end{aligned}$$

The abjunction operator on two binary scalar variables x and y is defined as: $x \nrightarrow y = x\bar{y}$. For binary vectors the operation is performed bitwise: $\mathbf{X} \nrightarrow \mathbf{Y} = \mathbf{X}\bar{\mathbf{Y}}$. Algorithm 17 in the matrix form is defined through the $\mho$ operator shown in Algorithm 18. For any given

Algorithm 18:$\mho$

```
Input: $\chi \underset{\circ}{=} \{f_1, f_2, \cdots, f_N\}$
Output: $\mathring{\Phi}_{M \times n}$
$\Phi \leftarrow \Phi_\chi$
for $i \leftarrow n$ downto $2$ do
    for $j \leftarrow i - 1$ downto $1$ do
        if $f_i \cap f_j \neq \varnothing$ then
            $\Phi e_j \leftarrow \Phi e_j \nrightarrow \Phi e_i$
return $\Phi$
```

set of features $\mathcal{B}_n \subseteq \chi \subseteq \mathcal{F}_n$, the $\mho$ operator creates a sparse feature matrix $\mathring{\Phi}_\chi$. Note that the original matrix Φ_χ is not necessarily sparse. Hence the $\circ$ notation highlights sparsity of the feature matrix $\mathring{\Phi}_\chi$. The Φe_i in the algorithm selects the i^{th} column of Φ. On each iteration of the outer loop, $\mho$ picks a column starting from the right (Φe_i) and applies it to every column on its right (Φe_j) using the abjunction operator, only if both f_i and f_j share a common index.

Remark For any $\mathbf{X}, \mathbf{Y} \in \{0, 1\}^n, \mathbf{X}, \mathbf{Y} \neq \mathbf{0}$, then if $\mathbf{Z} = \mathbf{XY} \neq \mathbf{0}$, then $\mathbf{Z}$ is not orthogonal to both $\mathbf{X}$ and $\mathbf{Y}$.

Proof $\langle \mathbf{Z}, \mathbf{X} \rangle = \langle \mathbf{Z}, \mathbf{Y} \rangle = ||\mathbf{XY}||_{\ell_1} \neq 0$ ■

This highlights the fact that the new feature built using the conjunction of two feature vectors is not orthogonal to the span of the existing set of features. On the contrary, as it will be proved in Theorem 3.4.3, when sparsification is applied to the matrix Φ, it forms a full-rank matrix with orthogonal basis.

Lemma 3.4.1 *For any* $\mathbf{X}, \mathbf{Z} \in \{0, 1\}^n, \mathbf{X}, \mathbf{Z} \neq \mathbf{0}, \mathbf{X} \neq \mathbf{Z}$, *then* $\mathbf{Z} \perp (\mathbf{X} \nrightarrow \mathbf{Z})$.

Proof $\underbrace{\langle \mathbf{Z}, \mathbf{X} \nrightarrow \mathbf{Z} \rangle}_{\text{Inner product}} = ||\mathbf{Z}(\mathbf{X} \nrightarrow \mathbf{Z})|| = ||\mathbf{Z}\mathbf{X}\bar{\mathbf{Z}}|| = 0.$ ■

Fig. 3-16 provides a geometrical interpretation of the abjunction operator in 2D (a) and 3D (b) spaces. In Fig. 3-16(a),

$$\boldsymbol{Q} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{V} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \boldsymbol{Q}^{\nrightarrow} = \boldsymbol{Q} \nrightarrow \boldsymbol{V} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

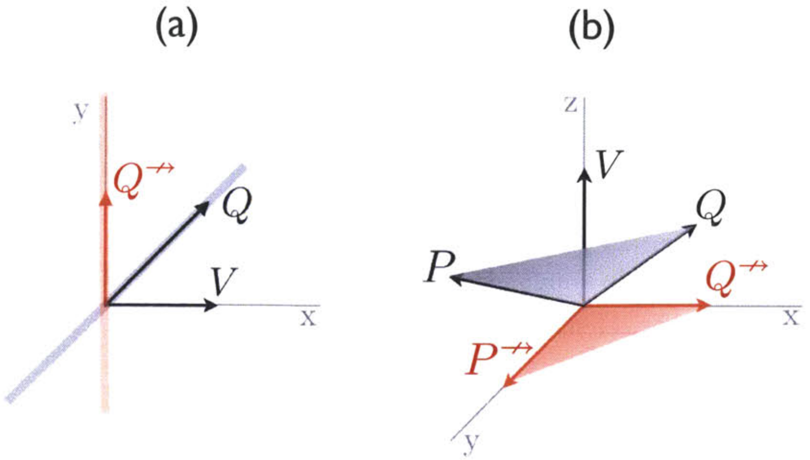

(a)
(b)
y
$Q^{\not\twoheadrightarrow}$
Q
V
x
z
V
Q
P
$Q^{\not\twoheadrightarrow}$
x
$P^{\not\twoheadrightarrow}$
y

Figure 3-16: A geometric interpretation of the abjunction operator applied to vectors in 2D (a) and 3D (b) spaces.

Notice how the one dimensional subspace defined by $\boldsymbol{Q}$, shown as the dark transparent line, changed to the red transparent line, turning it orthogonal to $\boldsymbol{V}$. Fig. 3-16(b) depicts another example in a 3D space where,

$$\boldsymbol{Q} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \boldsymbol{P} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \boldsymbol{V} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Rightarrow \begin{cases} \boldsymbol{Q}^{\not\twoheadrightarrow} = \boldsymbol{Q} \not\twoheadrightarrow \boldsymbol{V} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ \boldsymbol{P}^{\not\twoheadrightarrow} = \boldsymbol{P} \not\twoheadrightarrow \boldsymbol{V} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{cases}.$$

The black transparent plane shows the subspace spanned by $\boldsymbol{P}$ and $\boldsymbol{Q}$. The red plane shows the transformation of the space after applying the abjunction operator to $\boldsymbol{P}$ and $\boldsymbol{Q}$ (*i.e.,* the black subspace).

Lemma 3.4.2 *Given Assumptions A1-A3, $E_n \subseteq \mathcal{F}_n - B_n, \boldsymbol{\chi} = B_n \cup E_n$ then, matrix $\boldsymbol{\Phi}'_{\boldsymbol{\chi}} = \mho(f)$ has no zero column vector.*

Proof Given any feature $f \in \boldsymbol{\chi}$, there exists at least one state for which $\phi_f(s) \neq 0$. Find state s for which only indices covered by f are active (*i.e.,* $\forall f \in f, \phi_f(s) = 1, \forall f \notin f, \phi_f(s) = 0$). It is sufficient to show that $\phi_f(s)$ remains unchanged after applying the $\mho$ operator. Since $\boldsymbol{\chi}$ is sorted in ascending order based on feature sizes, $\forall g \in \boldsymbol{\chi}, |g| \geq |f| \Rightarrow g \nsubseteq f \Rightarrow \phi_g(s) = 0$. Hence $\phi_f(s) \not\twoheadrightarrow \phi_g(s) = 1 \not\twoheadrightarrow 0 = 1$. ■

Theorem 3.4.3 *Given Assumptions A1-A3, $E_n \subseteq \mathcal{F}_n - B_n, \chi = B_n \cup E_n$ then, matrix $\mathring{\mathbf{\Phi}}_\chi = \mho(\chi)$ has full rank.*

Proof We apply induction on the dimensions of the state space (n) to prove this. Note that we can write $\mathbf{\Phi}_\chi = [\mathbf{\Phi}_{B_n} \mathbf{\Phi}_{E_n}]$ and $\mathring{\mathbf{\Phi}}_\chi = [\mathring{\mathbf{\Phi}}_{B_n} \mathring{\mathbf{\Phi}}_{E_n}]$.

$(n = 2)$: The MDP has two states. Hence $B_2 = \{\varnothing, \{1\}, \{2\}\}$. The non-trivial case is when $E_2 = \{\{1, 2\}\}$, Hence:

$$\mathbf{\Phi}_\chi = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \mathring{\mathbf{\Phi}}_\chi = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},$$

which has linear independent columns.

$(n = k)$: Assume that for any $E_k \subseteq \mathcal{F}_k - B_k, \mathring{\mathbf{\Phi}}_\chi = [\mathring{\mathbf{\Phi}}_{B_k} \mathring{\mathbf{\Phi}}_{E_k}]$ has linear independent columns.

$(n = k + 1)$: Divide the set of states into two sets: $\mathcal{S}^-$ which includes states that does not activate feature $k+1$ and $\mathcal{S}^+ = \mathcal{S} - \mathcal{S}^-$. The new features matrix with the additional dimension can be decomposed into $\mathbf{\Phi}_\chi = [\mathbf{\Phi}_{B_{k+1}} \mathbf{\Phi}_{E_{k+1}}]$. Following Theorem 3.2.3,

$$\mathbf{\Phi}_{B_{k+1}} = \begin{bmatrix} \mathbf{\Phi}_{B_k} & \mathbf{0} \\ \mathbf{\Phi}_{B_k} & \mathbf{1} \end{bmatrix}$$

where the top and bottom rows correspond to $\mathcal{S}^-$ and $\mathcal{S}^+$ and the last column corresponds to the feature $k + 1$. Similarly, define the following three extended feature sets:

$$\begin{aligned} E_k^- &= \{f | f \in E_{k+1}, k + 1 \notin f\}, \\ E_{k+1}^+ &= \{f | f \in E_{k+1}, k + 1 \in f\}, \\ E_k^+ &= \{f \smallsetminus \{k + 1\} | f \in E_{k+1}^+\}. \end{aligned}$$

Notice that, the subscript k is used to highlight the fact that feature $k+1$ in not present in either of E_k^+ and E_k^- sets. Following the same row ordering used for $\mathbf{\Phi}_{B_{k+1}}$, matrix $\mathbf{\Phi}_\chi$ with the column reordering of $\{B_k, E_k^-, k + 1, E_{k+1}^+ \smallsetminus \{k + 1\}\}$ can be written as:

$$\begin{bmatrix} \mathbf{\Phi}_{B_k} & \mathbf{\Phi}_{E_k^-} & 0 & 0 \\ \mathbf{\Phi}_{B_k} & \mathbf{\Phi}_{E_k^-} & \mathbf{1} & \mathbf{\Phi}_{E_k^+} \end{bmatrix}.$$

Similarly matrix $\mathring{\mathbf{\Phi}}_{\chi} = \mho(\chi)$ can be written as:

$$\begin{bmatrix} \mathring{\mathbf{\Phi}}_{B_k} & \mathring{\mathbf{\Phi}}_{E_k^-} & 0 & 0 \\ \mathring{\mathbf{\Phi}}'_{B_k} & \mathring{\mathbf{\Phi}}'_{E_k^-} & q & \mathring{\mathbf{\Phi}}_{E_k^+} \end{bmatrix}.$$

Define $\mathbf{X} = [\mathring{\mathbf{\Phi}}_{B_k} \mathring{\mathbf{\Phi}}_{E_k^-}], \mathbf{Y} = [\mathring{\mathbf{\Phi}}'_{B_k} \mathring{\mathbf{\Phi}}'_{E_k^-}], \mathbf{Z} = [q\ \mathring{\mathbf{\Phi}}_{E_k^+}]$. If both $\mathbf{X}$ and $\mathbf{Z}$ have linear independent columns then Lemma 3.2.2 completes the proof.[8] $\mathbf{X}$ and $\mathring{\mathbf{\Phi}}_{E_k^+}$ have full rank due to the induction assumption. So it is sufficient to show that q does not lie in the column space of $\mathring{\mathbf{\Phi}}_{E_k^+}$. Similar to Lemma 3.4.2, consider state s for which only feature $k+1$ is active. Since $\forall g \in \chi, |g| > |\{k+1\}| \Rightarrow \phi_g(s) = 0$. Hence the row corresponding to state s in vector q has one unique 1, while the same row in matrix $\mathring{\mathbf{\Phi}}_{E_k^+}$ is 0. Therefore Q is not in the span of $\mathring{\mathbf{\Phi}}_{E_k^+}$. ■

Remark Given Assumptions A1-A3, $E_n \subsetneq \mathcal{F}_n - B_n, \chi = B_n \cup E_n$, then for any $f \in \texttt{pair}(\chi)$, the column spaces of $\mathring{\mathbf{\Phi}}_{\chi \cup \{f\}} = \mho(\chi \cup \{f\})$ does not necessarily cover the column space of $\mathring{\mathbf{\Phi}}_{\chi} = \mho(\chi)$.

Proof We provide one example where a column of $\mathring{\mathbf{\Phi}}_{\chi \cup \{f\}}$ does not lie in the column space of $\mathring{\mathbf{\Phi}}_{\chi}$. Consider an MDP with 3 binary dimensions, where $f = \{1,3\}, B_3 = \{\varnothing, \{1\}, \{2\}, \{3\}\}, E_3 = \{\{1,2\}\}$. Hence:

$$\mathbf{\Phi}_{\chi} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{\Phi}_{\chi \cup \{f\}} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

[8] The values of $\mathring{\mathbf{\Phi}}'_{B_k}$ and $\mathring{\mathbf{\Phi}}'_{E_k^-}$ does not affect the proof.

$$\mathring{\boldsymbol{\Phi}}_{\chi} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathring{\boldsymbol{\Phi}}_{\chi\cup\{f\}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Let $A = [\mathring{\boldsymbol{\Phi}}_{\chi\cup\{f\}}\ \mathring{\boldsymbol{\Phi}}_{\chi}\boldsymbol{e}_2] \Rightarrow det(A^\top A) = 3 \Rightarrow \mathtt{span}(\mathring{\boldsymbol{\Phi}}_{\chi}) \not\subseteq \mathtt{span}(\mathring{\boldsymbol{\Phi}}_{\chi\cup\{f\}})$ ■

The $\mho$ operator retains the full rank property of $\boldsymbol{\Phi}_{\chi}$. The columns of $\boldsymbol{\Phi}_{\chi}$ form an orthogonal basis. Unfortunately as shown by the above counter example, the resulting subspaces after adding the new feature do not cover the previous subspace. As a result the rate of convergence results derived in Corollary 3.2.8 does not hold when sparsification is applied. Theorem 3.4.4 provides an upper bound on the maximum number of states for which their values are not necessarily retained after expanding the representation and applying the sparsification method.

Theorem 3.4.4 *Given Assumptions A1-A3, $E_n \subseteq \mathcal{F}_n - B_n, \chi = B_n \cup E_n, |\chi| = n$, and $f, g \in \chi, h \in \mathtt{pair}(\chi) = f \cup g$, where f, g and h have order numbers i, j and k in the ordered set $\chi \cup \{h\}$ correspondingly, considering the following two representations:*

- *R1: $\boldsymbol{\Phi} = \mho(\chi)$ where $\phi_p(s)$ return the value of feature p at state s.*
- *R2: $\mathring{\boldsymbol{\Phi}} = \mho(\chi \cup \{h\})$ where $\mathring{\phi}_p(s)$ return the value of feature p at state s.*

$\forall \boldsymbol{\theta} \in \mathbb{R}^n, \forall s \in \mathcal{S}$, if $\phi_f(s) = \phi_g(s) = 1$ or $\mathring{\phi}_h(s) = 0$, then $\phi(s)^\top\boldsymbol{\theta} = \mathring{\phi}(s)^\top\boldsymbol{\theta}'$, if

$$\boldsymbol{\theta}' = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_{k-1} \\ \theta_i + \theta_j \\ \theta_{k+1} \\ \vdots \\ \theta_n \end{bmatrix}$$

Proof Let us consider the two cases mentioned in the theorem separately:

- $\{s | s \in \mathcal{S}, \mathring{\phi}_h(s) = 0\}$: it is easy to see that $\forall u \in h, \phi_u(s) = \mathring{\phi}_u(s)$ because $x \nrightarrow 0 = x$. Furthermore, based on our assumption, the value of the new feature at state s is zero. Hence the new added weight is ignored, resulting in $\phi(s)^\top \boldsymbol{\theta} = \mathring{\phi}(s)^\top \boldsymbol{\theta}'$.

- $\{s | s \in \mathcal{S}, \phi_f(s) = \phi_g(s) = 1\}$: In this case, $\mathring{\phi}_h(s) = 1$, and due to sparsification $\mathring{\phi}_f(s) = \mathring{\phi}_g(s) = 0$. Because the weight corresponding to $\mathring{\phi}_h(s)$ includes the sum of weights corresponding to $\mathring{\phi}_f(s)$ and $\mathring{\phi}_g(s)$, it is sufficient to show that $\forall u \in \boldsymbol{\chi} \setminus (f \cup g), \phi_u(s) = \mathring{\phi}_u(s)$. This is proved by contradiction. Assume $\exists u \in \boldsymbol{\chi} \setminus (f \cup g)$, where $\phi_u(s) \neq \mathring{\phi}_u(s)$. Because the $\mho$ operator cannot turn a zero value to one for the previous feature columns[9], then $\phi_u(s) = 1$ and $\mathring{\phi}_u(s) = 0$. Hence $h \cap u \neq \varnothing$. On the other hand, by the definition of $\mho$, $\forall x, y \in \boldsymbol{\chi}, \phi_x(s) = \phi_y(s) = 1 \Rightarrow x \cap y = \varnothing$, otherwise one would have eliminated the other. This case assumed $\phi_f(s) = \phi_g(s) = 1$. Therefore:

$$\begin{aligned} u \cap f &= \varnothing, u \cap g = \varnothing, \\ \Rightarrow u \cap (f \cup g) &= \varnothing, \\ u \cap h &= \varnothing, \end{aligned}$$

which contradicts the earlier statement. ■

Corollary 3.4.5 *Given Assumptions A1-A3 and* $\boldsymbol{\chi} \subseteq \mathcal{F}_n$, $\boldsymbol{\theta} \in \mathbb{R}^n$, $\mathring{\boldsymbol{\Phi}}_{\boldsymbol{\chi}} = \mho(\boldsymbol{\chi})$, $\mathring{\boldsymbol{\Phi}}_{\boldsymbol{\chi} \cup \{f\}} = \mho(\boldsymbol{\chi} \cup \{f\})$, *and* $\boldsymbol{\theta}'$ *built according to Theorem 3.4.4, if the portion of elements of vectors* $\tilde{\boldsymbol{V}} = \mathring{\boldsymbol{\Phi}}_{\boldsymbol{\chi}} \boldsymbol{\theta}$ *and* $\tilde{\boldsymbol{V}}' = \mathring{\boldsymbol{\Phi}}_{\boldsymbol{\chi} \cup \{f\}} \boldsymbol{\theta}'$ *that are not identical is defined as* β*, then* $\beta \leq \frac{1}{2^{|f|}}$.

Proof Theorem 3.4.4 shows that for all states that $\mathring{\phi}_f(s) = 0$, the value function remains the same. For an MDP with n binary state variables, the minimum number of states for which $\mathring{\phi}_f(s) = 0$ is $1 - G(f) = 2^{n-|f|}$.[10] Hence β is bounded by $2^{-|f|}$. ■

Note that in our setting $|f| \geq 2$, which means in the worst case β is $\frac{1}{4}$ and it drops exponentially fast because the coverage of new features exponentially decays with the number of conjunctions. Figure 3-17 depicts this concept in a graph. The next chapter empirically investigates online control methods built on top of iFDD using the sparsification technique combined with the temporal difference learning.

[9] $0 \nrightarrow x = 0$

[10] The $\mho$ operator can also remove some of the activations, hence the term minimum is used.

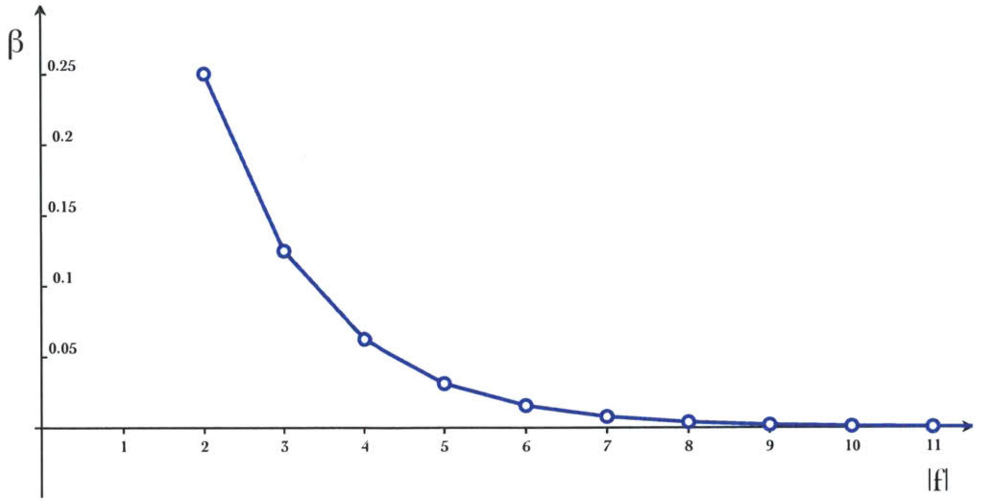


Figure 3-17: The upper bound on the portion of the state space for which the resulting value functions can disagree if sparsification ($\mho$) is used and the weight vector is changed according to Theorem 3.4.4.

3.5 Contributions

This chapter provided a theoretical analysis on the notion of *right* set of features used for approximating the value function of a fixed policy in a linear form. We introduced the notion of feature coverage and showed that coverage plays a fundamental role in identifying features with a guaranteed rate of convergence in reducing the approximation error. Theorem 3.2.6 provided a generic convergence rate for feature expansion techniques. Corollary 3.2.8 yielded a closed form solution to identify new binary features using conjunctions of previous features with best guaranteed convergence rates. This theoretical result formed the core idea of the incremental Feature Dependency Discovery (iFDD) family of feature expansion techniques. Empirical results in a batch setting demonstrated faster convergence rate of the value function using iFDD techniques over a random approach in the task of inverted pendulum. Finally in order to reduce the complexity of iFDD methods, a sparsification technique was introduced and its theoretical consequences were probed.

Chapter 4

The *Right* Set of Features: An Empirical View

The previous chapter introduced the family of iFDD feature expansion techniques for solving the policy evaluation problem and derived several theoretical results. This chapter empirically investigates the use of iFDD for online control. The structure of this chapter is as follows. Section 4.1 introduces the online iFDD family of expansion techniques. Combined with temporal difference learning, new methods are examined in control problems including large scale UAV mission planning scenarios. Section 4.2 extends iFDD by relaxing the dependency on the initial feature set. Section 4.3 reviews existing adaptive function approximators and relates iFDD with discussed methods within the big picture. Section 4.4 provides a summary of the contributions. Portions of this chapter have been published as a conference paper (Geramifard et al., 2011c).

4.1 Online iFDD

This section shows how iFDD can be used online. Since the focus of this chapter is the practicality of the algorithms in large domains, cheap computational complexity is a major concern. Reviewing from the previous chapter, given a set of L_1 samples, the current weight vector $\boldsymbol{\theta}$, and the current set of features χ, a new feature can be constructed using the following optimization:

$$\tilde{f}^* = \operatorname*{argmax}_{f\in\texttt{pair}(\chi)} \psi_f(\texttt{Samples}, \boldsymbol{\theta}, \chi), \tag{4.1}$$

where ψ, which is defined as *relevance* from this point on, was implemented in two forms

$$\begin{aligned}
\delta_i &= r_i + [\gamma\boldsymbol{\phi}(s_i') - \boldsymbol{\phi}(s_i)]^\top \boldsymbol{\theta}, \\
\psi_f(\texttt{Samples}, \boldsymbol{\theta}, \boldsymbol{\chi}) &= \sum_{i\in\{1,\cdots,L_1\}, \phi_f(s_i)=1} |\delta_i|, && (4.2) \\
\psi_f^+(\texttt{Samples}, \boldsymbol{\theta}, \boldsymbol{\chi}) &= \frac{\sum_{i\in\{1,\cdots,L_1\}, \phi_f(s_i)=1} \delta_i}{\sqrt{\sum_{i\in\{1,\cdots,L_1\}, \phi_f(s_i)=1} 1}}. && (4.3)
\end{aligned}$$

This chapter focuses on the control problem. As seen in Chapter 2, value-based RL techniques require access to Q values to solve control problems. Hence, from this point the temporal difference is defined based on Q values rather than V values. Hence:

$$\delta_i = r_i + [\gamma\boldsymbol{\phi}(s_i', a_i') - \boldsymbol{\phi}(s_i, a_i)]^\top \boldsymbol{\theta}. \tag{4.4}$$

Notice that if the underlying policy changes during feature expansion (which is often the case in the control case), theoretical results do not immediately apply, yet they suggest the resulting control algorithms will perform well in practice. To proceed, define each $f \in \texttt{pair}(\boldsymbol{\chi})$ as a *potential* feature with ψ_f as its corresponding relevance. The iFDD algorithm seeks to add the potential feature with the highest relevance to the representation. The main insight in iFDD to reduce the complexity is the creation of an online process that calculates the relevances defined in Equations 4.2 and 4.3 **incrementally**. Given that the representation has sparsity ${}^{k_t}/_{n_t}$ at time step t, then the total number of pair combination of active feature is bounded by $\Theta(k_t^2)$, hence the feature relevance of at most k_t^2 potential features can be updated on every time step. This observation is one of the key concepts in major complexity reduction of iFDD. Furthermore, from this point, all feature vectors will be sparsified using Algorithm 19 discussed in the previous chapter, hence we use the $\circ$ notation (*i.e.,* $\overset{\circ}{\boldsymbol{\phi}}$ is used for function approximation instead of $\boldsymbol{\phi}$). Notice that this change will again move the algorithm further away from the main theoretical results (*i.e.,* Theorem 3.2.7) because the span of the feature matrix after feature expansion does not necessarily include the previous span (*i.e.,* Remark 6). Yet from the practical viewpoint, it reduces the computational complexity of iFDD substantially.

The process begins by building a linear approximation to the value function online using the initial set of binary features. It tracks the relevance measure for all simultaneously activated feature pairs (*i.e.,* potential features) for every visited state. On every time step, Equation 4.1 can be used to discover the potential feature with the maximum relevance value, but this approach will generate one feature on every time step, exhausting the mem-

Algorithm 19: Generate Feature Vector ($\mathring{\phi}$) — Complexity

```
Input: $\phi^0(s), \chi$
Output: $\mathring{\phi}(s)$
$\mathring{\phi}(s) \leftarrow \bar{0}$
activeInitialFeatures $\leftarrow \{i | \phi^0_i(s) = 1\}$
Candidates $\leftarrow$ SortedPowerSet(activeInitialFeatures)                    $\Theta(2^{k_0})$
while activeInitialFeatures $\neq \varnothing$ do
    $f \leftarrow$ Candidates.next()
    if $f \in \chi$ then
        activeInitialFeatures $\leftarrow$ activeInitialFeatures $\setminus f$    $\Theta(k_0)$
        $\mathring{\phi}_f(s) \leftarrow 1$
return $\mathring{\phi}(s)$
```

ory space quickly. To circumvent this problem, a thresholding mechanism replaces the argmax operator in Equation 4.1. In particular, whenever the relevance for a potential feature exceeds a user-defined threshold, it will be discovered and added to the representation, capturing the nonlinearity between the corresponding feature pair. Note that stochastic transitions may introduce some complications that will be discussed later in Chapter 6. The algorithm proceeds in three steps:

(i) Identify *potential* features that can reduce the approximation error,

(ii) Track the relevance of each potential feature, and

(iii) Add potential features with relevance above a discovery threshold to the pool of features used for approximation.

Figure 4-1 shows online iFDD in progress. The circles represent initial features, while rectangles depict conjunctive features. The relevance of each potential feature f, ψ_f, is the filled part of the rectangle. The discovery threshold ξ, shown as the length of rectangles, controls the rate of expansion and is the only parameter of online iFDD. This parameter is domain-dependent and requires expert knowledge to set appropriately. However, intuitively, lower values encourage faster expansion and improve the convergence to the best possible representation, while higher values slow down the expansion and allow for a better exploitation of feature coverage. While the ideal value for ξ will depend on the stochasticity of the environment, empirical results were robust to the choice of the discovery threshold value. Section 4.1.4 covers the details.

Online iFDD can be integrated with any value-based RL method. This thesis investigates the integration of online iFDD with TD learning due to its simplicity, speed, and

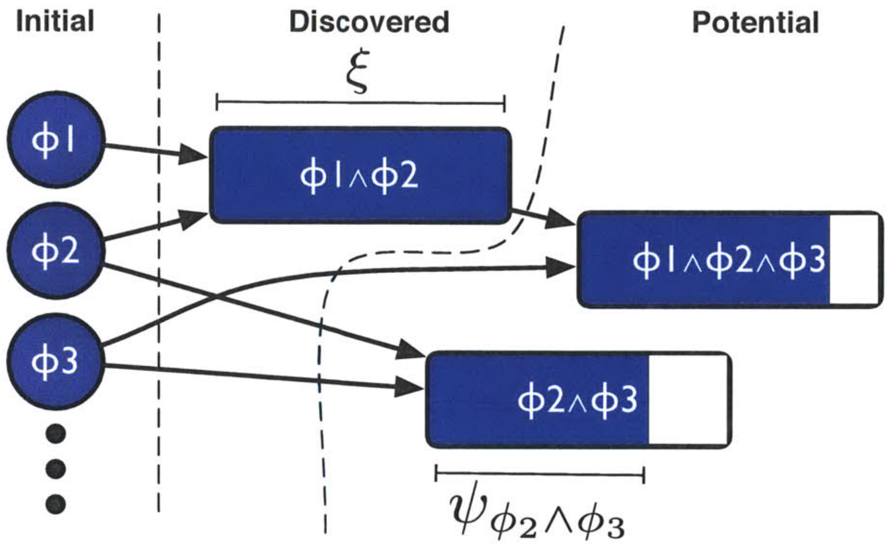


Figure 4-1: A snapshot of online iFDD: Initial features are circles, conjunctive features are rectangles. The relevance ψ_f of a potential feature f is the filled part of the rectangle. Potential features are discovered if their relevance ψ reaches the discovery threshold ξ.

its previous empirical results in the literature (Geramifard et al., 2006, 2007; Sutton, 1996; Sutton & Barto, 1998). It will be shown that for the policy evaluation case, the online iFDD algorithm with TD learning will converge to the best possible function approximation given an initial set of binary features. We note that, if the initial features are such that no function approximation – linear or nonlinear – can satisfactorily approximate the underlying value function, then applying online iFDD will not help. For example, if a key feature such as a UAV's fuel is not included in the initial set of features, then the value function approximation will be poor even after applying online iFDD. For more discussion, refer to the discussion of relaxing Assumption A3 in Section 3.2.2.

4.1.1 Algorithm Details

For simplicity, first assume that the relevance of a potential feature is defined by Equation 4.2. The use of Equation 4.3 for calculating relevances will be discussed at the end of this section. The process begins with an initial set of binary features; let χ be the current set of features used for the linear function approximation at any point in time. After every interaction, first the RL method updates the weight vector $\boldsymbol{\theta}$. In the case of TD, $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \delta_t \mathring{\boldsymbol{\phi}}(s_t, a_t)$, where α_t is the learning rate and δ_t is defined in Equation 4.4. Next, Algorithm 20 is applied to discover new features.

The first step in the discovery process (lines 1,2) identifies all conjunctions of active fea-

Algorithm 20: Discover using Equation 4.2

```
  Input: $\mathring{\phi}(s), \delta_t, \xi, \chi, \psi$
  Output: $\chi, \psi$
1 foreach $(g, h) \in \{(i, j) | \mathring{\phi}_i(s)\mathring{\phi}_j(s) = 1\}$ do
2     $f \leftarrow g \cup h$
3     if $f \notin \chi$ then
4         $\psi_f \leftarrow \psi_f + |\delta_t|$
5         if $\psi_f > \xi$ then
6             $\chi \leftarrow \chi \cup f$
```

tures as potential features.[1] Considering only conjunctive features is sufficient for online iFDD to converge to the best approximation possible given the initial feature set; conjunctive features also remain sparse and thus keep the per-time-step computation low (refer to the complexity analysis of Algorithm 17). The relevance ψ_f of each potential feature $f = g \cup h$ is then incremented by the absolute approximation error $|\delta_t|$ (line 4). If the relevance ψ_f of a feature f exceeds the discovery threshold ξ, then feature f is added to the set χ and used for future approximation (lines 5,6).

In order to bound the number of states for which the value function might change, Theorem 3.4.4 is used to determine the weight of the new feature. In particular, if feature $f = g \cup h$ is added to the representation, the online iFDD algorithm initializes the coefficient for the new feature f as $\theta_f = \theta_g + \theta_h$.

4.1.2 Theory

The main virtue of online iFDD is the way it expands the feature representation. Unlike other representation-expanding techniques (*e.g.*, Ratitch & Precup, 2004; Whiteson et al., 2007), iFDD increases the dimensionality of the space over which features are defined gradually as opposed to utilizing the full-dimensional state space. This section shows that online iFDD with TD learning asymptotically leads to the best performance possible given the initial features: first it is shown that online iFDD does not stop expanding the representation unless the representation is perfect or it is fully expanded. Next, the asymptotic approximation error with respect to the true value function is bounded.

Let s^i denote the i^{th} state. The feature function $\mathring{\phi}(s)$ outputs which features are active in state s and $\mathring{\Phi}_{|S| \times n}$ captures all such feature vectors, where the ith row corresponds to

[1] Conjunctions are stored in a "flat" representation, so there is only one conjunctive feature $a \cup b \cup c$ for the conjunction of features $a \cup (b \cup c)$ and $(a \cup b) \cup c$.

$\mathring{\phi}(s^i)^\top$. As feature conjunctions are added as new features, the output dimensionality of $\mathring{\phi}(s)$ grows and adds columns to $\mathring{\boldsymbol{\Phi}}$.

Completeness of Feature Expansion To show that online iFDD-TD will find the best representation possible given the initial set of features, the following shows that the online iFDD-TD will either find a perfect representation or will add all possible conjunctive features:

Theorem 4.1.1 *Given initial features and a fixed policy π that turns the underlying MDP into an ergodic Markov chain, online iFDD-TD is guaranteed to discover all possible feature conjunctions or converge to a point where the TD error is identically zero with probability one.*

Proof Suppose online iFDD-TD has found $\mathring{\phi}$ as its final representation which neither sets the TD error zero everywhere nor includes all possible feature conjunctions. These properties imply that there is at least one state s that has at least two active features, g and h, where $g \cup h$ has not been explored (if no state has more than one active feature, then the feature discovery algorithm would have no combined features to propose and the representation would have been fully expanded). The absolute sum of TD errors for this state s and this feature pair g and h after some time T_0 is $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$, where $\mathcal{I}$ indicates whether the agent was in state s at time t, and δ_t is the corresponding temporal difference error. By the ergodicity of the underlying Markov chain, state s will be visited infinitely many times.

Since the value function approximation is not perfect, there exists some state s' for which the TD error $\delta(s')$ is nonzero. The Markov chain induced by the policy was assumed to be ergodic; ergodicity implies that there exists a path of finite length from state s' to state s. Thus, over time, the TD error at state s' will propagate to state s. The only way for feature $f = g \cup h$ to not be added is if the sum $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$ converges to some non-zero value that is less than the discovery threshold ξ.

It can be argued that the sum $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$ diverges. Since the policy is fixed, consider the value function $V(s)$ instead of the action-values $Q(s, a)$. Let $V_\infty(s)$ be the converged fixed-point value function that would result from this learning process and $V_t(s)$ be the value function at time t. Let $\epsilon(s,t) = V_t(s) - V_\infty(s)$. Then we can rewrite the absolute TD error at time t as

$$\begin{aligned}
|\delta_t| &= |V_t(s) - r(s) - \gamma V_t(s')| \\
&= |V_\infty(s) + \epsilon(s,t) - r(s) - \gamma V_\infty(s') - \gamma \epsilon(s',t)| \\
&= |(V_\infty(s) - r(s) - \gamma V_\infty(s')) + (\epsilon(s,t) - \gamma \epsilon(s',t))|
\end{aligned} \tag{4.5}$$

where, if the value function is not perfect, the first term $(V_\infty(s) - r(s) - \gamma V_\infty(s'))$ is some constant c (if the MDP is stochastic, the absolute TD error $|\delta_t| = |V_\infty(s) - r(s,a) - \gamma V_\infty(s')|$ will be nonzero simply because the successor state s' will vary). The second term $(\epsilon(s,t) - \gamma\epsilon(s',t))$ decreases as $V_t(s)$ converges toward $V_\infty(s)$; find a time $t = T_0$ such that $|\epsilon(s,t) - \gamma\epsilon(s',t)| < \frac{c}{2}$. Therefore, the sum of absolute TD errors $\sum_{t=T_0}^{\infty} |\delta_t| \mathcal{I}(s_t = s)$ is lower bounded by $\sum_{t=T_0}^{\infty} \frac{c}{2}\mathcal{I}(s_t = s)$ and diverges.[2] ■

Corollary 4.1.2 *If at each step of online iFDD-TD the policy changes but still induces an ergodic Markov chain (*e.g., *via ϵ-greedy or Boltzmann exploration), then online iFDD-TD will explore all reachable features or converge to a point where the TD error is identically zero with probability one.*

Asymptotic Quality of Approximation Theorem 4.1.1 implies that online iFDD-TD will converge to a fixed point where the TD error is zero everywhere or the feature space is fully explored. Using the bound derived in Tsitsiklis and Van Roy (1997), the asymptotic approximation error of iFDD with respect to this final representation is bounded. Let $\overset{\circ}{\boldsymbol{\Phi}}_\infty$ be the feature matrix that includes all conjunctive features (including initial features), and $\boldsymbol{V}_{|\mathcal{S}|\times 1}$ be the vector representing the optimal value of all states.

Corollary 4.1.3 *With probability one, online iFDD-TD converges to a weight vector $\boldsymbol{\theta}$ and feature matrix $\overset{\circ}{\boldsymbol{\Phi}}$, where the approximated value function error, as originally shown by Tsitsiklis and Van Roy (1997) for a fixed set of linear bases, is bounded by:*

$$||\overset{\circ}{\boldsymbol{\Phi}}\boldsymbol{\theta} - \boldsymbol{V}||_{\boldsymbol{D}} \leq \frac{1}{1-\gamma}||\boldsymbol{\Pi}\boldsymbol{V} - \boldsymbol{V}||_{\boldsymbol{D}},$$

where $\boldsymbol{D}_{|\mathcal{S}|\times|\mathcal{S}|}$ is a diagonal matrix with the stationary distribution along its diagonal, $\boldsymbol{\Pi} = \overset{\circ}{\boldsymbol{\Phi}}_\infty(\overset{\circ}{\boldsymbol{\Phi}}{}_\infty^\top \boldsymbol{D}\overset{\circ}{\boldsymbol{\Phi}}_\infty)^{-1}\overset{\circ}{\boldsymbol{\Phi}}{}_\infty^\top \boldsymbol{D}$, and $||.||$ stands for the weighted Euclidean norm.

Proof Theorem. 4.1.1 states that online iFDD-TD either finds a perfect representation with TD error zero everywhere (the approximation is exact) or exhaustively expands the whole representation. In the fully-expanded case, each state will have exactly one active feature, and thus the final feature matrix $\overset{\circ}{\boldsymbol{\Phi}}$ (excluding zero columns) will have full column rank. Apply Theorem 1 of Tsitsiklis and Van Roy's work (1997) to bound the error in the value function approximation. ■

Corollary 4.1.3 guarantees that online iFDD-TD will achieve the best approximation for the value of policy π given the initial feature set. Later online iFDD-TD will be examined empirically to show this approach learns more quickly than a full tabular approach (equivalent

[2] This proof was jointly derived with Finale Doshi (Geramifard et al., 2011c).

to starting out with a full set of "converged" conjunctions of basic features). Whether that value function is optimal depends on the initial choice of features; Corollary 4.1.3 states that online iFDD-TD makes the best possible use of given features asymptotically. Determining a good set of initial features is also an important (but orthogonal) task. However, some basic guidelines can be given for selecting initial features. For finite state MDPs, the feature representation is guaranteed to be sufficient if each state has a unique set of associated features. For continuous state spaces, discretizing dimensions separately into buckets is one way of providing initial binary features; more popular approaches such as tile coding (Albus, 1971) can be also applied as long as the granularity is fine enough for the task to be solvable. Section 4.2 provides a heuristic method for expanding the set of initial features online.

Maximum Features Explored In general, if online iFDD-TD begins with an initial feature set χ with $|\chi| = n$ elements, then the total number of possible features is the size of the power set of χ, $|\wp(\chi)| = 2^n$. In the specific case where initial features correspond to d independent variables that can each take q values – such as a continuous MDP discretized into q buckets for d independent dimensions – a tighter bound on the number of features to be explored can be obtained because only one initial feature will be active for each dimension.

Remark Assume the initial set of features is defined for an MDP over d variables, each with domain size q. The maximum number of features explored (initial + discovered) using iFDD for such an MDP is $(q+1)^d - 1$.

Proof The number of feature conjunctions of size k is q^k. Hence the maximum number of features using Pascal's triangle amounts to:

$$\sum_{k=1}^{d} \binom{d}{k} q^d = \sum_{k=0}^{d} \binom{d}{k} q^d - 1 = (q+1)^d - 1. \blacksquare$$

A tabular representation for d variables uses q^d features, less than the $(q+1)^d - 1$ bound on features explored by iFDD. The difference occurs because the lookup table is the fringe of the feature tree expanded by iFDD process. While iFDD might explore more features than the tabular representation, it is shown empirically that iFDD often retains many fewer features due to its nature of gradual expansion. Also, because a minimal set of highest order clauses are active in any state, the asymptotic effective number of features used by iFDD is bounded by $|S|$ (equal to the number of features in a tabular representation) unless a perfect representation is reached before discovering all possible features.

4.1.3 Computational Complexity

As a reminder k_t and n_t are the maximum number of active features for any state and the total number of features in use at time t. The online iFDD algorithm does not forget discovered features and $\mathring{\phi}$ uses a greedy set covering approach to form new feature vectors. Therefore, for all times $i < j \Rightarrow n_i \leq n_j, k_i \geq k_j$. Hence, $\forall t > 0, k_t \leq k_0$. The main loop of the *Discover* function (Algorithm 20) requires k_t^2 operations. Using advanced hashing functions such as Fibonacci heaps, both χ and ψ updates require $\Theta(1)$ operations. Hence the per-time-step complexity of Algorithm 20 is $\Theta(k_t^2) \leq \Theta(k_0^2)$. It is interesting to see that the discovery process runs faster as more features are added to the representation. The outer loop of Algorithm 19 requires $\Theta(2^{k_0})$ operations in the worst case and each iteration through the loop involves $\Theta(1)$ lookup and $\Theta(k_0)$ set difference. Hence the total per-time-step complexity of evaluating the feature function $\mathring{\phi}$ is $\Theta(k_0 2^{k_0})$. The computational complexity of both algorithms depends on k_0, the number of active features, and not n_t, the total number of features. Thus, even if a method like Tile Coding, which may introduce large numbers of features, is used to create initial features, online iFDD will still execute quickly.

4.1.4 Empirical Results

This section compares the effectiveness of online iFDD with SARSA (see Sutton & Barto, 1998) against representations that (i) use only the initial features, (ii) use the full tabular representation, and (iii) use two state-of-the-art representation-expansion methods: the adaptive Tile Coding (ATC) method, which cuts the space into finer regions through time (Whiteson et al., 2007), and the sparse distributed memories approach (SDM), which creates overlapping sets of regions (Ratitch & Precup, 2004). Following earlier successful RL papers (Boyan, 2002; Geramifard et al., 2007), all cases used learning rates

$$\alpha_t = \frac{\alpha_0}{k_t}\frac{N_0 + 1}{N_0 + \text{Episode}\#^{1.1}},$$

where k_t was the number of active features at time t. For each algorithm and domain, the best α_0 and N_0 were selected from $\{0.01, 0.1, 1\}$ and $\{100, 1000, 10^6\}$ correspondingly. For exploration the ϵ-greedy policy with $\epsilon = 0.1$ was employed. Each algorithm was tested on each domain for 30 runs (60 for the rescue mission). iFDD was fairly robust with respect to the threshold, ξ, outperforming the initial and tabular representations in 7 out of 8 cases.

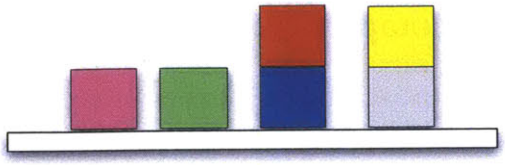
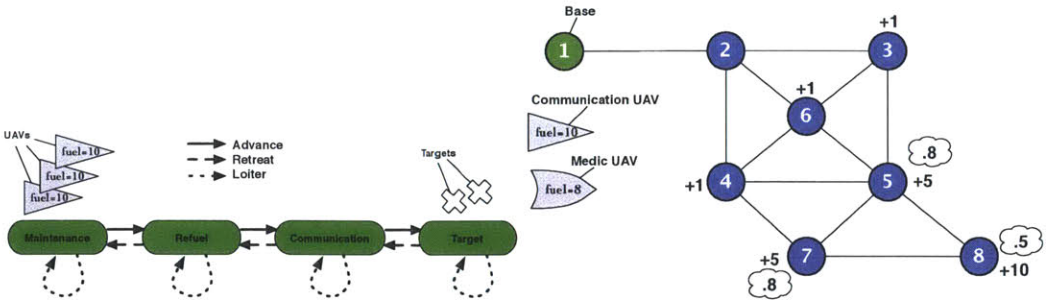


Figure 4-2: The BlocksWorld domain: the goal is to build a tower of blocks with a pre-ordered set of colors. For every movement of a block, there is 30% chance of dropping the block. All blocks are on the table initially.

Inverted Pendulum The description of the pendulum domain is identical to the previous chapter. Note that the policy is not fixed anymore. The goal is to balance the pendulum as long as possible capped at 3000 steps. The initial features consisted of discretizing each dimension into 20 levels (for 120 features total). Figure 4-5(a) plots the number of steps the pendulum remained balanced versus the total steps experienced (SARSA using the original Gaussian representation is also plotted). In this relatively low-dimensional space, SDM and iFDD found good policies quickly, although SDM outperformed iFDD for the first test case after $10,000$ steps. SARSA with the initial features never learned to balance the pendulum for more than $2,500$ steps, suggesting that the optimal value function was not linear in the initial features. The tabular representation reached near-optimal performance after about $60,000$ steps, while the Gaussian representation approach performed well after $40,000$ steps. ATC's initial advantage disappeared after many unhelpful splits that slowed generalization.

BlocksWorld The BlocksWorld task, shown in Figure 4-2, is to build a color-ordered 6-block tower. The trajectory starts with all blocks on the table. The action set is defined as $move(b, d)$ where $b \in B$ and $d \in B \cup \{T\}$, amounting to 36 distinct actions. An object is called clear if there exist no other objects on it or if it is table. The action is possible if $b \neq d$ and both b and d are clear. Each move has 30% chance of failure, resulting in dropping the moving block on the table. Initial features were directly derived from the relations: $On(o_1, o_2)$ which led to $6 \times 6 \times 36 = 1296$ features. The state is defined as a 36 dimensional Boolean vector, specifying the $On(.,.)$ relation for all pairs. A tabular representation potentially demands $b^2 \sum_{i=1}^{b} \binom{b}{i} i^{(b-i)} \times (b-i)! = 368,316$ states. Reward is 10^{-3} for all moves except the terminating move that completes the desired tower with reward of $+1$. Episodes were capped by 1000 moves and $\gamma = 1$. Figure 4-5(b) shows the return per episode. As expected, the initial representation does poorly because it cannot capture correlations between blocks. Our iFDD approach on the other hand discovered the necessary feature dependencies. The tabular representation could express the optimal



(a) Persistent Surveillance

(b) Rescue Mission

Figure 4-3: Two UAV Mission Planning Scenarios

policy, but its expressiveness hindered generalization: it needed three times the data as iFDD to achieve the same performance. Despite our optimization attempts, ATC and SDM both learned poorly in this larger, 36-dimensional domain.

Persistent Surveillance Figure 4-3(a) shows an unmanned aerial vehicle (UAV) mission-planning task where three fuel-limited UAVs must provide continuous surveillance of two targets. At any time, the UAVs may be in maintenance, refuel, communication, or target states; deterministic actions allow them to stay in place or move to adjacent states. All actions except maintenance or refuel cost a unit of fuel; UAVs refuel completely by staying in the refuel state. UAVs have perfect sensors to monitor for motor and camera failures; failed parts can be repaired by going to maintenance. Parts have a 5% chance of failing at each time step: broken motors require immediate fixing while broken cameras prevent the UAV from monitoring a target. All together, the state is a 12-dimensional vector of remaining fuel, location, motor sensor status and camera sensor status for each of the three UAVs for a total of approximately 150 million state-action pairs. Initial features for each UAV are the fuel indicator, location, and the state of each of the two sensors. The action space is the combination of actions for all UAVs. γ was set to 0.9 and episodes were capped at $1,000$ steps.

The team received a $+20$ reward for every time step a target was reported; to report a target a UAV had to see it from the target state and a UAV had to relay the message from the communication state. UAVs were penalized for each unit of fuel used; running out fuel outside the refuel area cost -50 for each UAV and ended the episode. The results in Figure 4-5(c) show that the lack of generalization slowed learning for the tabular case: even after 10^5 steps, it held all agents in the maintenance area for the entire mission. ATC and SDM had similarly inefficient generalization in this high-dimensional space. As before, the initial feature set could not capture the required correlations: it incorrectly generalized the

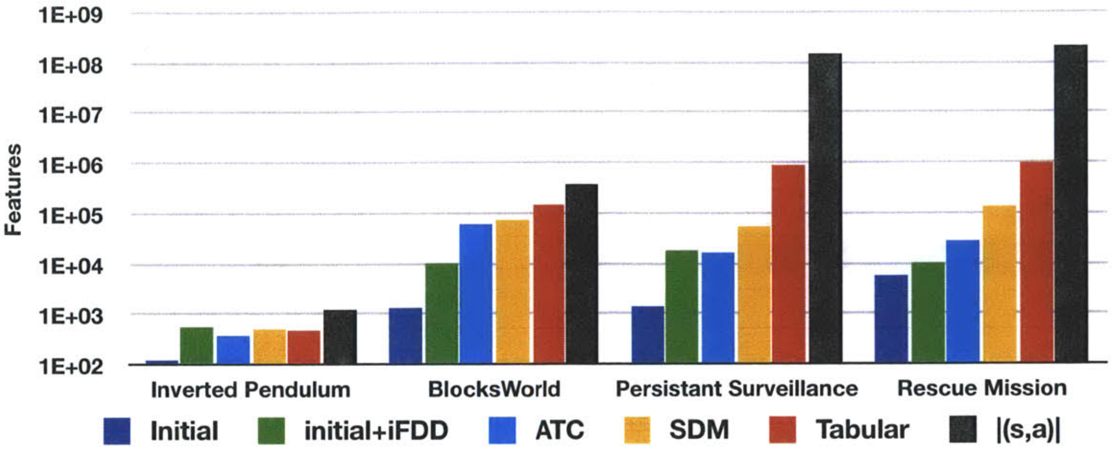


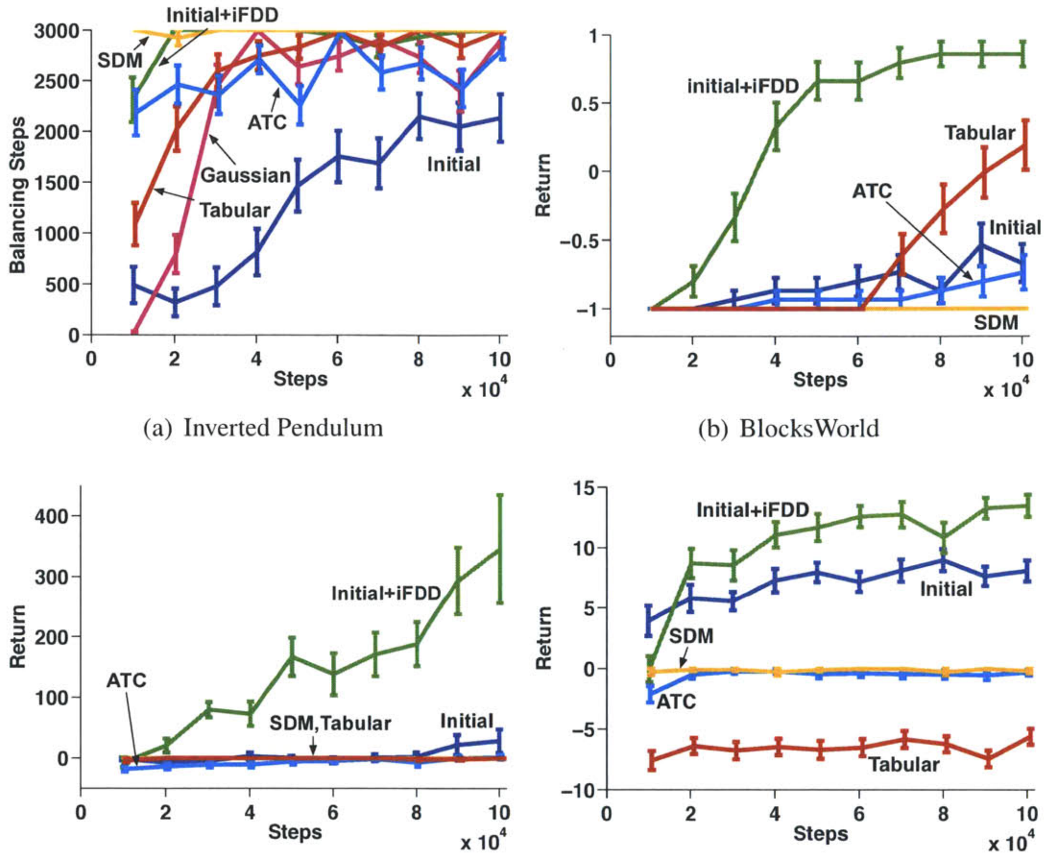
Figure 4-4: Average final feature counts. ATC and SDM, even using more features, performed poorly on high-dimensional examples. The black bar depicts the total number of state-action pairs.

consequences of running out of fuel. In contrast, the iFDD method corrected the improper generalization by incrementally adding feature conjunctions combining fuel and location. The resulting representation was able to switch to the better policy of sending UAVs out after $20,000$ steps. After $100,000$ steps, the performance of iFDD was more than 12 better on average compared to the second best competitor (Initial).

Rescue Mission Figure 4-3(b) shows a mission-planning task where a medic UAV and a communication UAV must complete a rescue mission. The green circle shows UAVs' base location; numbers above the remaining nodes indicate the number of injured people at that node; and the cloud numbers are the probability of successful rescue. Victims are saved when the medic UAV is at their node and the communication UAV is no farther than one edge away to relay back information. The medic UAV consumes a unit of fuel per movement or hover; the communication UAV may move (costs one fuel cell) or perch (costs nothing).[3] Initial features were the fuel and position of each UAV, the communication UAV mode, and the rescue status at each node. The total state-action pairs exceeded 200 million. γ was set to 1.

The team received $+1$ for each person rescued, -0.1 for each unit of fuel spent, and -23 if not at base after 10 time steps or depleting all fuel. Figure 4-5(d) shows the tabular representation was crippled by the scale of the problem. ATC and SDM fared somewhat better by capturing the notion of crashing early on but could not capture the complex reward structure. Learning with only the initial features proceeded quickly for the first $10,000$ steps,

[3] The perched UAV must hover before moving.



(a) Inverted Pendulum

(b) BlocksWorld

(c) Persistent Surveillance

(d) Rescue Mission

Figure 4-5: Empirical results of SARSA algorithm using various representational schemes in in four RL domains: Inverted Pendulum, BlocksWorld, Persistent Surveillance, and Rescue Mission.

showing that the initial features are largely independent for this domain. However, after $20,000$ steps, iFDD's richer representation allowed it to encode a policy that outperformed all other methods.

Figure 4-4 shows the average final feature counts for each domain. Previously, it was demonstrated that online iFDD can lead to a representation with more features than the tabular approach, but in the UAV domains, iFDD discovered approximately two orders of magnitude fewer features than tabular approach. Even when ATC and SDM had more features than online iFDD, they still did not match online iFDD's performance (except for SDM on pendulum). This observation suggests that the good performance of online iFDD is not due to the quantity, but the quality, of discovered features.

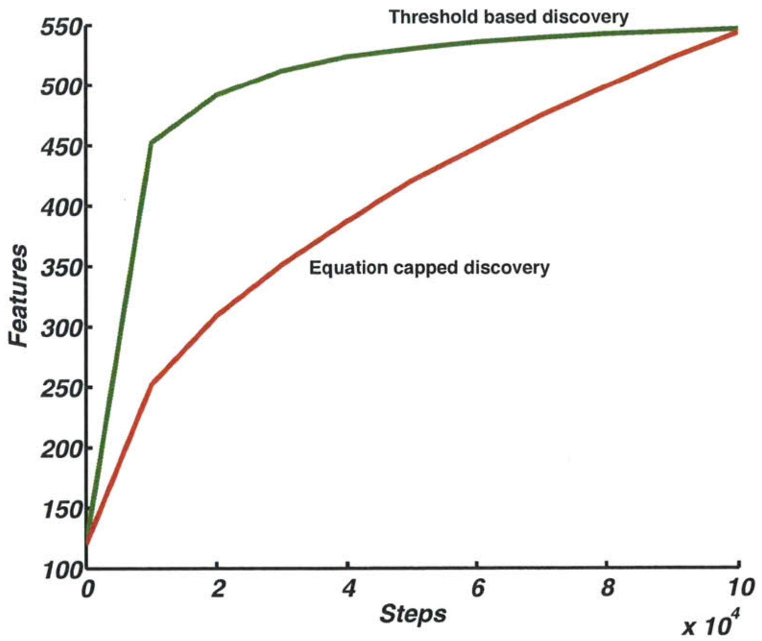


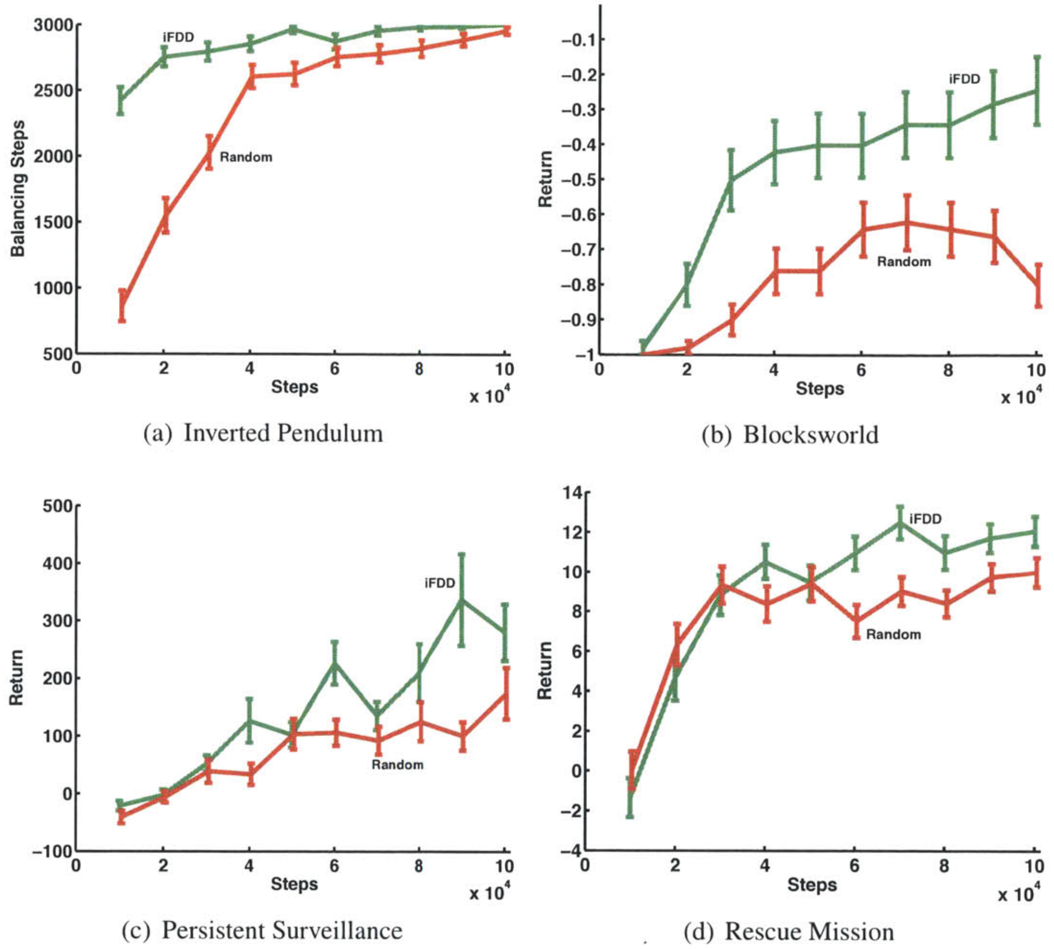
Figure 4-6: The resulting feature discovery rate based on Equation 4.6 (red) and the number of features discovered in the previous set of experiments (green) for the inverted pendulum problem. C was picked manually, so that the pool of potential features would not run out during the simulation.

Online iFDD vs. Random Expansion

This section investigates the effectiveness of the online iFDD process by comparing it carefully against the random approach selection where the relevance of the encountered potential features is ignored. In order to provide a careful comparison, the rate of feature discovery for all methods was dictated by:

$$n_t = n_0 + \sqrt{Ct}, \tag{4.6}$$

where t is the time step, n_t is the total number of features, n_0 is the number of initial features and C is a constant controlling the rate of discovery. Notice that this mechanism is different from the previous online iFDD algorithm where potential features with relevance more than a certain threshold were discovered. Instead, on each step, a certain number of features are discovered in order to maintain n_t features at time t. This will assure that both Random and online iFDD have the same number of features at every time step. The Random approach adds potential features uniformly randomly out of the pool of potential features, while iFDD adds new features by looking at the sorted list of potential features based on their relevance magnitude. The resulting algorithms were probed in the same



(a) Inverted Pendulum

(b) Blocksworld

(c) Persistent Surveillance

(d) Rescue Mission

Figure 4-7: Performance of online iFDD and Random feature discovery schemes with a fixed rate of discovery in four RL domains: Inverted Pendulum, BlocksWorld, Persistent Surveillance, and Rescue Mission. Notice how iFDD outperforms the random approach in all domains.

battery of four domains with the corresponding C values set to $\{0.2, 0.4, 2, 0.004\}$. These values assured that the pool of potential features was never empty, maximizing the number of times online iFDD and Random could pick different features. For example, Figure 4-6 shows the rate of feature discovery for the online iFDD based on the threshold discovery (green) and based on Equation 4.6 (red) for the inverted pendulum problem. The green plot essentially shows the rate of discovery for the online iFDD in plot 4-5(a). Note how the red plot remained below the green plot statistically significantly for the most parts of the regions. For all experimental domains, the number of features was capped akin to the Figure 4-6. In all domains, simulation results were averaged over 100 runs except the Persistent Surveillance with 30 runs. Error bars represent 95% confidence intervals.

Algorithm 21: Discover using Equation 4.3

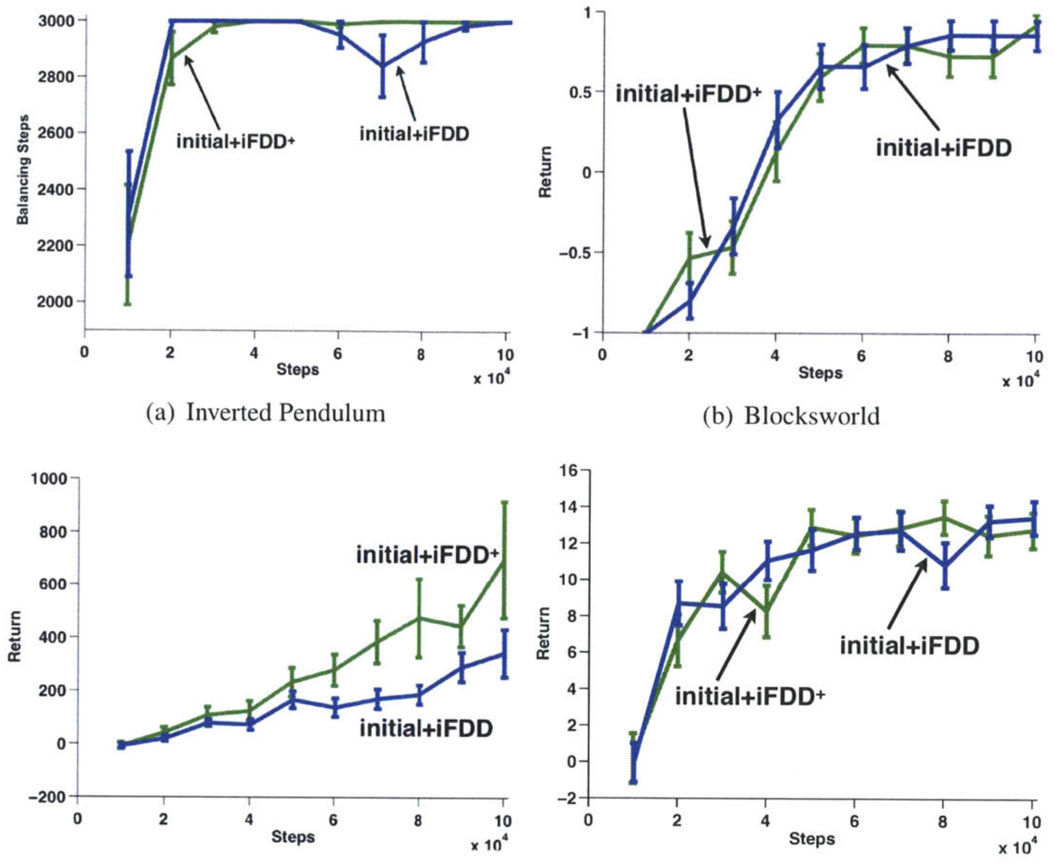
```
Input: $\mathring{\phi}(s), \delta_t, \xi, \chi, \Delta, \nu$
Output: $\chi, \psi$
foreach $(g, h) \in \{(i, j) | \mathring{\phi}_i(s)\mathring{\phi}_j(s) = 1\}$ do
    $f \leftarrow g \cup h$
    if $f \notin \chi$ then
        $\Delta_f \leftarrow \Delta_f + \delta_t$
        $\nu_f \leftarrow \nu_f + 1$
        $\psi_f \leftarrow {}^{\Delta_f}/_{\sqrt{\nu_f}}$
        if $\psi_f > \xi$ then
            $\chi \leftarrow \chi \cup f$
```

Figure 4-7 depicts the performance of online iFDD and Random approaches combined with SARSA akin to Figures 4-5(a)-4-5(d). In all domains, the error driven feature discovery used in iFDD provided a statistically significant boost to the learning rate of the algorithm compared to the random discovery approach. Also notice that in the BlocksWorld domain, on average, adding more features using the random approach led to worse performance after $70,000$ steps. The performance drop from $70,000$ steps to $100,000$ was statistically significant.

Online iFDD vs. Online iFDD$^+$

So far the relevance of potential features was calculated using Equation 4.2. This section introduces the online iFDD$^+$ algorithm, in which relevances are calculated according to Equation 4.3. While Algorithm 19 still provides the sparsified $\mathring{\phi}$, Algorithm 21 is used to track relevances and discover features. Notice that calculating the new relevance formulation (*i.e.,* Equation 4.3), requires two vectors: (i) Δ, storing the sum of TD errors, and (ii) ν, representing the visitation frequency of each potential feature. At any given time the relevance of feature f is calculated as ${}^{\Delta_f}/_{\sqrt{\nu_f}}$. While Theorem 4.1.1 showed the convergence of online iFDD-TD, it does not immediately result in the asymptotic convergence of online iFDD$^+$-TD. In particular there are two pieces required to prove the convergence of online iFDD$^+$-TD: 1) the sum of TD errors (as opposed to the sum of absolute TD errors) for a particular feature f is still unbounded, and 2) asymptotically the ratio of this sum (Δ_f) over $\sqrt{\nu_f}$ is also unbounded. The extension of the asymptotic convergence of online iFDD$^+$ is left for the future work.

The setting for running online iFDD$^+$ was identical to online iFDD except for the set of candidate discovery thresholds. The best threshold value for online iFDD$^+$ was



(a) Inverted Pendulum

(b) Blocksworld

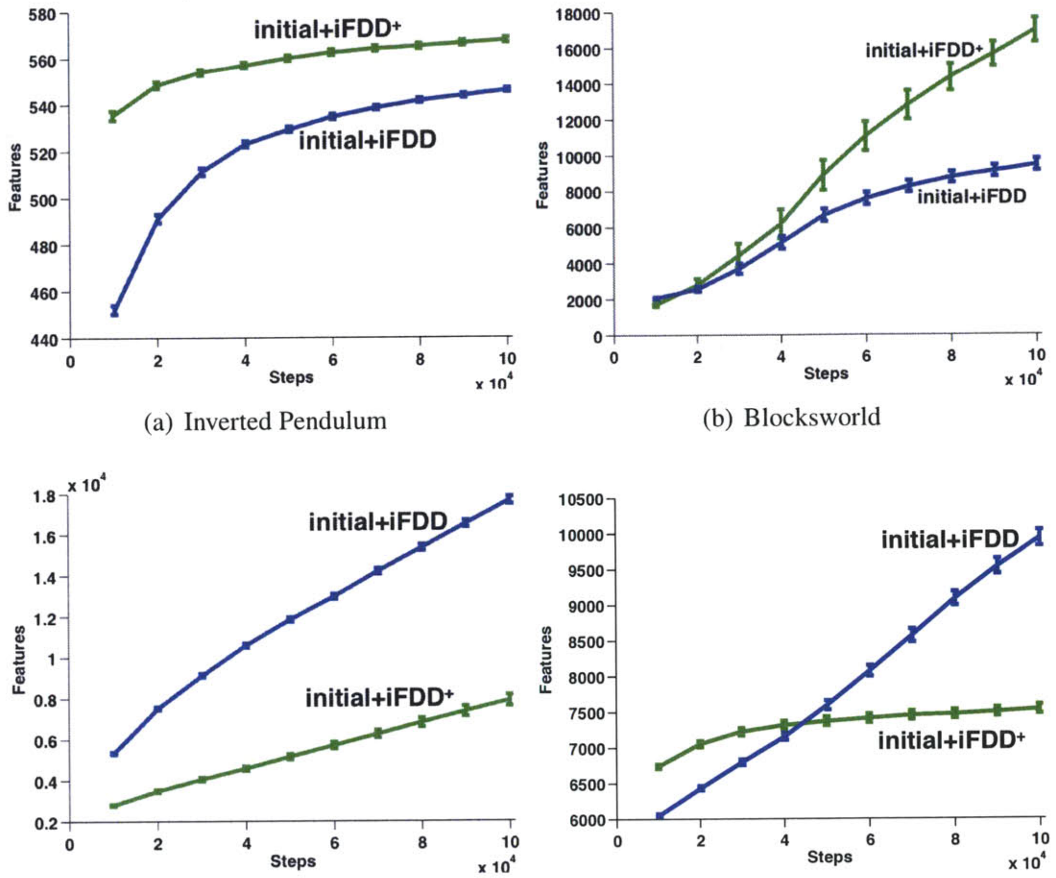
(c) Persistent Surveillance

(d) Rescue Mission

Figure 4-8: Empirical results of SARSA algorithm combined with online iFDD and online iFDD$^+$ representational schemes in in four RL domains: Inverted Pendulum, BlocksWorld, Persistent Surveillance, and Rescue Mission. The Y-axis represents the performance of each method.

picked empirically out of $\{0.02, 0.05, 0.1\}$ for both Inverted Pendulum and BlocksWorld, $\{50, 100, 200\}$ for Persistent Surveillance, and $\{100, 200, 500\}$ for the Rescue domain. Compared to online iFDD, candidate threshold values were smaller, because online iFDD$^+$ retained smaller relevances caused by the absence of the $|.|$ operator and the use of feature frequencies in the denominator. Figure 4-8 shows the performance of online iFDD against online iFDD$^+$ in all the domains, while Figure 4-9 shows the corresponding feature sizes used by each method.

- **Inverted Pendulum**: The performance of online iFDD$^+$ had less variance compared to online iFDD. In particular, after $40,000$ steps the resulting policy using online iFDD$^+$ always balanced the pendulum for $3,000$ steps, while as for online iFDD the policy sometimes experienced some difficulties around $70,000$ steps. This may be



(a) Inverted Pendulum

(b) Blocksworld

(c) Persistent Surveillance

(d) Rescue Mission

Figure 4-9: Empirical results of SARSA algorithm combined with online iFDD and online iFDD$^+$ representational schemes in in four RL domains: Inverted Pendulum, BlocksWorld, Persistent Surveillance, and Rescue Mission. The Y-axis represents the total number of features used by each method.

due to the fact that some of the discovered features by online iFDD were not useful. Hence the agent required some time to have their weights adjusted. Overall, online iFDD$^+$ discovered more features compared to online iFDD, which can explain the reduction in the variance of the performance due to better approximation of the value function.

- **BlocksWorld**: In this domain both methods performed similarly, yet online iFDD$^+$ ended up with roughly two times the number of features compared to online iFDD. To investigate the effect of the threshold, new values of $\{0.2, 0.5, 1\}$ were added to the pool. While larger threshold values led to fewer number of features discoveries, the corresponding performances were not on par with online iFDD. Because online iFDD$^+$ is a closer approximation of the original rate of convergence stated in

Corollary 3.2.8, this observation seems counter intuitive. Although applying iFDD methods to the control case violates the assumptions of the corresponding Corollary, hence results are not totally unexpected.

- **Persistent Surveillance**: The performance advantage of online iFDD$^+$ over online iFDD was statistically significant in the Persistent Surveillance domain after $60,000$ steps. At the same time, the final number of features used by online iFDD$^+$ was less than one-half that used by online iFDD.

- **Rescue Domain**: In terms of the performance both methods had similar results, but online iFDD$^+$ again finished with fewer features.

Overall, the empirical results suggest that using online iFDD$^+$ over online iFDD is beneficial, because in terms of performance, online iFDD$^+$ was always better or on par with online iFDD. Note that the computational complexities of both Algorithm 20 and Algorithm 21 are equal.

4.2 Adaptive Resolution iFDD (ARiFDD)

The main drawback of iFDD is its reliance on the knowledge of the user to provide a reasonable initial representation. This is due to the fact that iFDD cannot further split initial features. Hence, if the representation including all feature dependencies is not powerful enough to capture the objective function, applying iFDD will not be helpful. For discrete functions, finding a comprehensive set of tiles is simple: provide binary features corresponding to values of each dimension, independent of the other dimensions. While this set of features allows iFDD to represent any objective function defined over the discrete set of states asymptotically, the generalization occurring within each dimension is limited. For example, consider an objective function for a UAV mission planning defined over a set of discrete variables including the fuel indicator with 30 possible values. Assume states with less than $1/3$ of the fuel level translates into very low state values. With 30 features corresponding to each fuel level, if the UAV reduces the weight corresponding to feature (fuel = 1), it will not affect the weights corresponding to features (fuel = $2, \cdots 10$). Hence, the concept of low fuel must be learned for each of the fuel values separately. Applying the same method to continuous domains is even more challenging. The conservative approach is to discretize each dimension with the finest possible granularity. This will reduce the generalization within each dimension, requiring a plethora of training samples.

Adaptive Resolution iFDD (ARiFDD), elevates the capability of iFDD by augmenting an autonomous feature splitting mechanism. Figures 4-10 illustrates the process through a

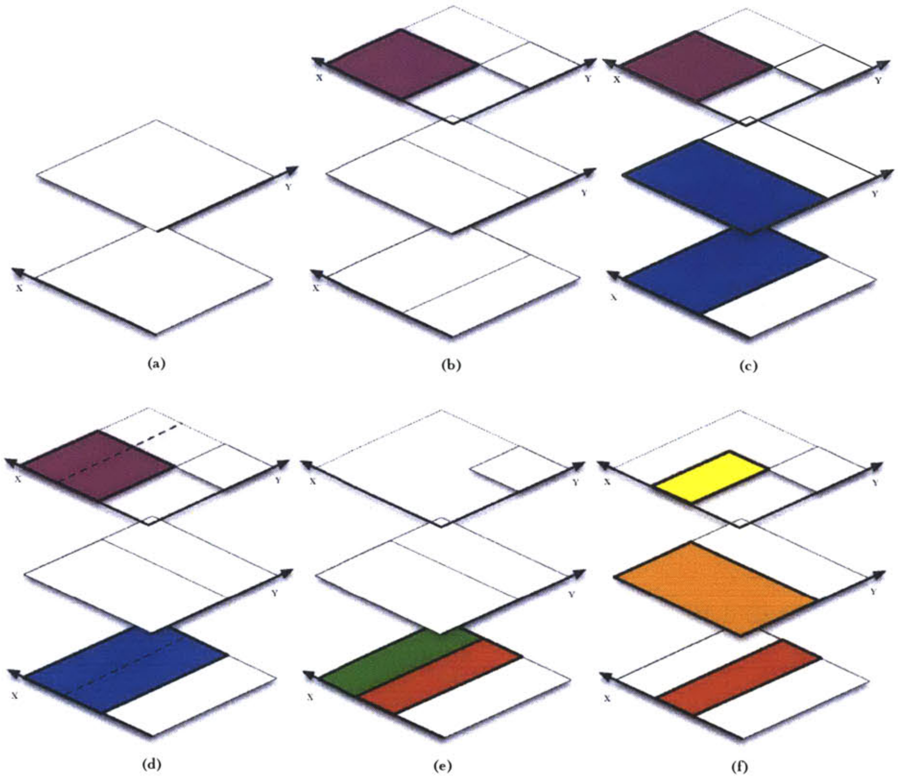


Figure 4-10: ARiFDD representation expansion process: a) Initialization. b) A possible representation found by ARiFDD. The representation for points falling in the purple tile cannot be further expanded since there are no more conjunctions in that region. c) ARiFDD considers splitting all basic tiles corresponding to the purple tile. d) The basic (blue) tile which is going to be split and the split line are identified. e) The red and green tiles are introduced as new basic tiles while the purple tile, built on the top of the split tile, is eliminated. f) After some time iFDD discovers the yellow tile as a new expanded tile capturing the dependency between the red tile and the orange tile.

Tile Coding example. For a d dimensional space, ARiFDD sets up the representation by creating d *basic*[4] tiles, each defined on a single tiling covering a unique dimension of the space. Figure 4-10-a illustrates this on a 2D space where two basic features (tiles) cover the state space in x and y dimensions correspondingly.[5] Figure 4-10-b shows a representation after running ARiFDD for some time. For points mapped to a single expanded tile, iFDD cannot expand the representation further. ARiFDD facilitates further expansion by splitting

[4] The term "basic" replaces the term "initial" to highlight the fact that these features can be split further, as opposed to initial features that are fixed.

[5] While ARiFDD can also be initialized with a user specified representation, unlike iFDD, the choice of initialization cannot be arbitrary. In particular, tiles must be rectangular as statistics stored for each dimension are stored separately. Additionally if features are not one dimensional, a further process is required to extract the corresponding one dimensional tiles for each feature.

basic tiles corresponding to such points. Figure 4-10-c shows the corresponding basic tiles for the purple tile as two blue tiles that are considered for further split. Figure 4-10-d shows the selected basic (blue) tile and its split line. Notice that splitting the blue tile renders the existence of the purple tile obsolete because the blue tile constitutes the purple tile. Figure 4-10-e depicts how the basic tile is split into two new basic tiles, shown as red and green tiles. Accordingly, all affected expanded tiles (which in this example includes only the purple tile) are removed from the representation. In Figure 4-10-f, iFDD discovers a new expanded tile (shown in yellow) capturing the dependency between the red tile and the orange tile.

Algorithm 22 illustrates this process in more detail. The input arguments to the algorithm are the current state, (s), current active features calculated by Algorithm 19 $\big(\mathring{\phi}(s)\big)$, the corresponding basic tiles $\big(\phi^0(s)\big)$, the current error measure (δ_t), split threshold (ξ), and current list of tiles (χ). Additionally the algorithm keeps track of three statistical measures for each basic tile. The first parameter is akin to the relevance measure discussed in online iFDD. The Ξ vector contains the sum of absolute errors corresponding to each basic tile. The next two parameters are used to decide the priority and the split point for each basic tile. Note that each basic tile, by definition, corresponds to a single dimension of the state space. Hence parameters μ, and σ^2 track the weighted mean and variance of observed states within each basic tile, where the weights are the absolute TD error.

The algorithm simply executes the iFDD discover, if more features can be expanded at state s (lines 1 and 20). Otherwise, the statistics for each basic tile corresponding to state s are updated incrementally (lines 4-8) using Finch's derivation (2009). If the threshold for any basic features exceeds a user-defined threshold (ς), the basic feature is added to the list of `potentialSplits`. If at the end of the loop `potentialSplits` is not empty, the basic tile with the least weighted variance is split along its weighted mean (lines 12-15). Consequently all expanded features built on top of the split tile are removed from the pool of features (lines 17 and 18). Munos *et al.* applied a similar concept for discretizing kd-trees, yet they focused expansion on areas with the largest variance. This scheme turned out to be not very helpful compared to the other competitors they introduced (Munos & Moore, 2002).

The intuition behind splitting a feature on its weighted mean stems from the fact that the resolution should increase close to areas of the state spaces where the error persists and no feature can be added through online iFDD. Picking the tile with the least variance as opposed to largest variance was found empirically to yield better results. Intuitively, the dimension with the least variance has the highest density of error, which is suggested by Corollary 3.2.8 to be a positive indicator of expanding good features. As expected, by

Algorithm 22: ARiFDD - Discover

```
   Input: $s, \phi^0(s), \mathring{\phi}(s), \delta_t, \chi, \Xi, \mu, \sigma^2, \varsigma$
   Output: $\chi, \Xi, \mu, \sigma^2$
1  if $\mathring{\phi}(s)$ has one active feature then
2      potentialSplits $\leftarrow \varnothing$
3      foreach basic feature $f \in \phi^0(s)$ do
4          $\Xi_f^+ \leftarrow \Xi_f + |\delta_t|$
5          $\mu_f^+ \leftarrow \mu_f + \frac{\delta_t}{\Xi_f^+}(s_f - \mu_f)$
6          $\sigma_f^2 \leftarrow \frac{1}{\Xi_f^+}\left[\sigma_f^2\Xi_f + \delta_t(s_f - \mu_f)(s_f - \mu_f^+)\right]$
7          $\Xi_f \leftarrow \Xi_f^+$
8          $\mu_f \leftarrow \mu_f^+$
9          if $\Xi_f > \xi$ then
10             potentialSplits $\leftarrow$ potentialSplits $\cup \{f\}$
11     if potentialSplits $\neq \varnothing$ then
12         $f \leftarrow \mathrm{argmin}_{f \in \mathtt{potentialSplits}} \sigma_f^2$
13         $f_1, f_2 \leftarrow$ Split basic feature $f$ along $\mu_f$
14         $\chi \leftarrow \chi \setminus \{\{f\}\}$
15         $\chi \leftarrow \chi \cup \{\{f_1\}, \{f_2\}\}$
16         forall the $g \in \chi$ do
17             if $f \in g$ then
18                 $\chi \leftarrow \chi \setminus g$
19 else
20     Run iFDD Discover (Algorithm 20 or 21)
```

setting $\varsigma = \infty$, the iFDD algorithm is retrieved. Also the extra computation required for ARiFDD compared to iFDD does not change the per-time-step computational complexity. Executing lines 16-18 is the most expensive part of Algorithm 22. By storing maps from basic tiles to the set of features built on top of them, running lines 16-18 requires $\Theta(2^{k_0})$, because the number of features built on top of a specific tile is bounded by 2^{k_0} for sparse representations.

4.2.1 Empirical Results

The SARSA algorithm was combined with ARiFDD (replacing line 20 with Algorithm 20), online iFDD , initial, and tabular representations, and tested in the Inverted Pendulum domain. All settings were identical to Section 4.1.4 and $\varsigma = 1$. Also in order to avoid memory overrun, the maximum number of splits were bounded. In particular, any split that resulted

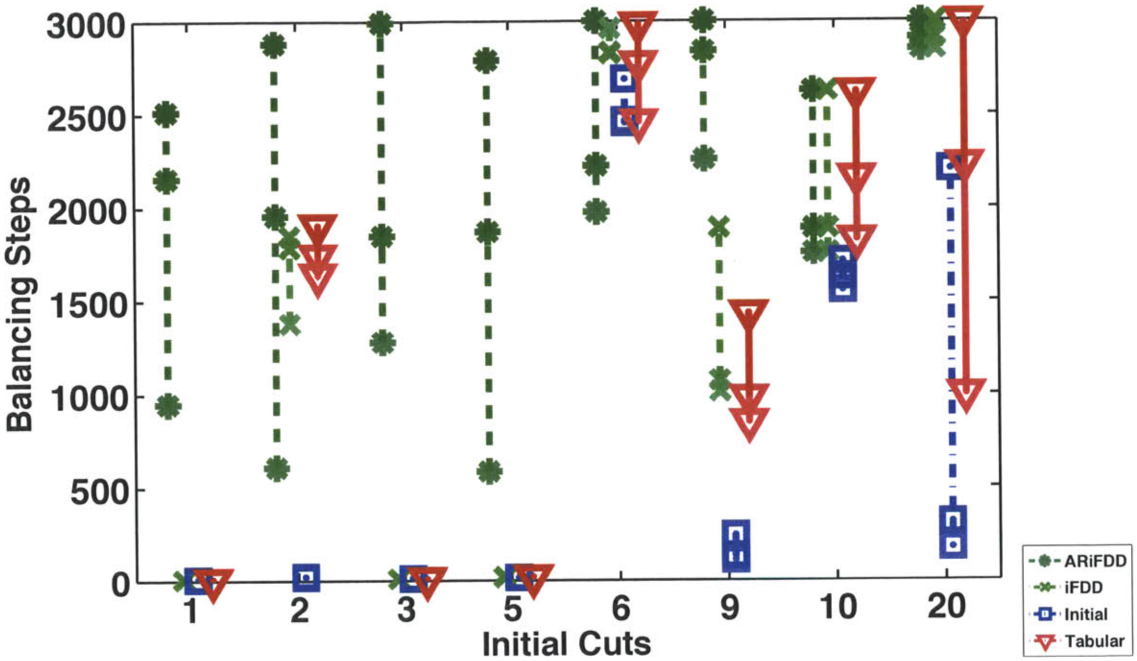


Figure 4-11: Comparison of SARSA using ARiFDD, iFDD, initial, and tabular representations. Each vertical line depicts the performance of the corresponding algorithm algorithm after $10,000$, $20,000$ and $100,000$ interactions from bottom to top. The X-axis represents the number of uniform cuts put through the two dimensions of the state space to generate the initial set of features.

in features with dimensional coverage less than $1/20$ of the domain size in that dimension was rejected.[6]

Figure 4-11 depicts the empirical results in the Inverted Pendulum domain. The X-axis depicts the number of uniform cuts put in each dimension initially to create basic/initial features. The Y-axis represents the number of steps each method balanced the pendulum averaged over 30 runs. Each vertical line depicts the performance of the corresponding algorithm algorithm after $10,000$, $20,000$ and $100,000$ interactions from bottom to top.

There are two stark observations: 1) For all initial cut values, ARiFDD achieved the best performance compared to all other methods asymptotically. The reason that iFDD and ARiFDD performed identically with 10 initial cuts was that all suggested splits were rejected due to coverage restriction mentioned above.[7] 2) Starting with an odd or even number of initial cuts played a major role in the performance of all methods except ARiFDD. For example, with 2 tiles per dimension, both iFDD and tabular methods could balance the

[6] $[\theta, \dot{\theta}] \in [\frac{-\pi}{2}, \frac{\pi}{2}] \times [-2, 2]$

[7] Theoretically, the split in the exactly mid point of each feature was feasible but it was never suggested.

pendulum close to $2,000$ steps at the end of the training. This is due to the fact that directions of θ and $\dot{\theta}$ provide important information for capturing good policies. Yet picking 1 or 3 uniform cuts led to the poor performance of iFDD, initial, and tabular representations. Similarly moving from 5 initial cuts to 6 made a dramatic change in the performance of the three methods.

In summary, picking the granularity of initial features can determine the success or failure of iFDD, initial, and tabular representations. Allowing feature expansion within the basic tiles specified by the user, ARiFDD relaxes the dependency of iFDD on the initial set of features. In the case where the user has no insight on how to specify basic features, ARiFDD can start with one feature per dimension and expands the granularity of the representation adaptively. Of course, it is expected that this favorable property will also increase the sample complexity.

4.3 Related Work

Figure 4-12 illustrates the taxonomy of Adaptive Function Approximators (AFAs) as suggested by Buşoniu *et al.* (2010). On the highest level, AFAs are divided into non-parametric and parametric methods. Non-parametric methods mostly utilize kernel-based estimation in which the value of each state is approximated as a function of its similarity to a set of previously known samples (Bethke & How, 2009; Ormoneit & Sen, 2002; Reisinger et al., 2008; Taylor & Parr, 2009). The per-time-step computational complexity of non-parametric methods is expensive as these methods often require matrix inversion where the size of the matrix is dependent on the number of samples. This property prevents the use of non-parametric methods in online settings where samples are generated with high frequency. Additionally the user requires prior skill in order to pick the right kernel and tune its specifications.

The parametric branch of AFAs assumes a functional form for the value function which can be non-linear or linear. Artificial Neural Networks (ANNs) are one of the most famous family of non-linear approximators. This mathematical model has been utilized in the context of adaptive representation within the RL framework through the idea of cascade correlation networks (Girgin & Preux, 2007; Girgin & Preux, 2008; Rivest & Precup, 2003) and evolutionary networks (Whiteson & Stone, 2006). Unfortunately, ANNs, and in general non-linear function approximators, lack the convergence property when combined with even most basic learning techniques (Tsitsiklis & Van Roy, 1997). Moreover, they require several sweeps through labeled data in order to provide a reasonable approximation, which is often not possible in online settings. In contrast, online temporal difference

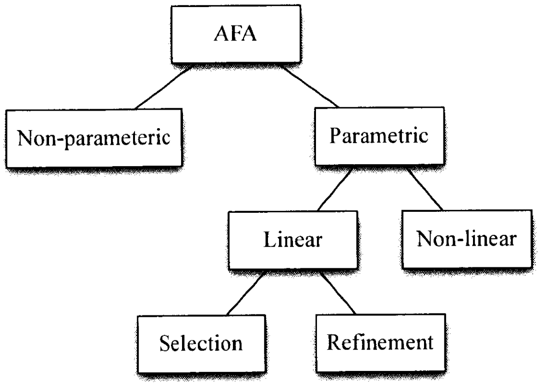


Figure 4-12: Taxonomy of Adaptive Function Approximators adopted from the work of Buşoniu *et al.* (Buşoniu et al., 2010)

learning has been shown to converge when combined with linear function approximation (Tsitsiklis & Van Roy, 1997).

Creating linear AFAs has been an active area of research for the past decade, and has been mainly pursued from two perspectives: selection and refinement. While L. Buşoniu *et al.* (2010) considered Bellman error-based AFAs as a separate category, we include them in the refinement branch, as they merely use a different measure in order to refine the representation. Starting with a large set of basis functions, selection (bottom-up) methods condense the representation by eliminating unpromising basis functions. This process has been realized through regularization (Farahmand et al., 2008; Kolter & Ng, 2009), temporal difference learning (Li et al., 2009), dynamic Bayesian networks (Kroon & Whiteson, 2009), and homotopy methods (Petrik et al., 2010). Finding a comprehensive set of basis functions for high dimensional problems often requires extensive domain knowledge. Moreover, as the per-time-step computational complexity of these methods depends on the number of features, the large size of the initial set of basis functions reduces the speed of selection methods substantially.

Refinement (top-down) methods take an opposite approach compared to selection algorithms by starting off with a small representation and adding new features to the representation as more data is observed. Early refinement techniques added random feature conjunction as new features to the representation (Sutton & Whitehead, 1993). Tree based partitioning of the state space has been a popular research trend for more than a decade (Lampton & Valasek, 2009; Lin & Wright, 2010; Moore & Atkeson, 1995; Reynolds, 2000; Uther & Veloso, 1998; Whiteson et al., 2007). This approach is also known as adaptive Tile Coding in one layer by which the state space is partitioned into non-overlapping regions (*i.e.,* state aggregation). Regions are further split into smaller areas based on some criteria specified by the algorithm. While early work (Moore & Atkeson, 1995) scaled RL methods up to 9 dimensional state spaces, the deterministic dynamics and known single goal region assumptions limited the applicability of this approach. Relaxing these assumptions, later work exhibited compelling results in low dimensional spaces, yet they could not scale up to large domains as no generalization is allowed between tiles (*i.e.,* each state is mapped into one tile). The only work that focused on adaptive tiling adjustment was by Sherstov and Stone (2005) in which tilings were defined in the full dimensional state space and the effective resolution of tiles were both uniform and fixed. In general, adaptive Tile Coding techniques have never considered automatic tile creation in sub-dimensional spaces which is one of the main ideas introduced in this dissertation. Researchers also used linear instead of binary features[8] for representations (Munos & Moore, 2002; Waldock & Carse, 2008). While the initial idea of splitting regions still remains the same, new points are evaluated by interpolating the values of neighboring points. Among interpolation techniques, the idea of Sparse Distributed Memory (*a.k.a* Kanerva coding) is unique as it operates with a fixed amount of memory (Ratitch & Precup, 2004). The main insight of SDM is to maintain at least N neighbors for a given sampled point. Once this rule is violated, random points are shifted to fulfill this constraint for the new point. While SDM exhibited superior results in our low dimensional Inverted Pendulum problem, the performance was degraded drastically as it was applied to problems with larger number of dimensions.

Perceiving the correlation between the Bellman error vector and space spanned by the current features, researchers investigated the idea of adding features that reduce the Bellman error vector. The direct approach adds the Bellman vector as a new basis function on each iteration of policy iteration (Parr et al., 2007; Valenti, 2007). While the new feature vectors are guaranteed to be helpful in reducing the current Bellman error, the computational demands of this approach make it unappealing for online settings. On the same track, Wu *et al.* applied machine learning techniques to introduce new binary features identifying the sign of the Bellman error (2005). They extended their work by discovering real-valued features exhibiting high correlation with the magnitude of the Bellman error (Wu & Givan, 2007). Both efforts led to batch techniques using approximate value iteration not amenable to online settings. Keller *et al.* aggregated states with similar Bellman error using neighborhood component analysis and added the aggregation function as the new basis function on each iteration (Mannor & Precup, 2006). Their method applies only to the policy evaluation case. Furthermore, applying this technique to online problems is challenging as it requires matrix inversion on every iteration. Representational policy iteration (Mahade-

[8] Notice that both linear and binary features are used within the context of linear function approximation

van, 2005) is another computationally intensive approach for adding features based on harmonic analysis. Sanner *et al.* introduced a batch feature discovery technique for first-order MDPs (Sanner, 2006a), although the method does not generalize to universal reward functions, demands the world model, and requires high computational complexity. Sanner also introduced a fast method for online feature discovery (Sanner, 2006b), although the approach is restricted to relational MDPs with a single goal state. Manual feature expansion techniques has been also practiced in the literature. A brute-force approach for including all feature conjunctions up to a certain depth was applied to the TD learning method in the games of hearts (Sturtevant & White, 2006) and Go (Silver et al., 2008). Outside the scope of reinforcement learning, from a biological prospective, Valiant introduced adding feature conjunctions as a provably evolvable structure for supervised learning. He verified that this mechanism under certain assumptions can learn binary functions with arbitrary precision in polynomial time (Valiant, 2009). Kalai *et al.* provided PAC bounds for the greedy feature construction algorithm used to extend the representation of a binary classifier in which the conjunction correlated with the highest amount of error is added to the pool of features (Kalai et al., 2009).

The iFDD family of expansions is a new member of the linear AFAs within the refinement category. It is simple to implement, requires low per-time-step computational complexity, and scales better to high dimensional state spaces. The main difference between the iFDD family of methods and other refinement techniques used for control is their ability to assign multiple features to a state, each of which corresponds to a selected set of dimensions. In the Tile Coding framework, this can be thought of defining tilings in sub-dimensional state spaces as opposed to other refinement techniques which define tilings in the full dimensional state space.

A Note on Batch Refinement Techniques

While batch refinement techniques often have lower sample complexity compared to online techniques, the associated computational demands often limit their application in online settings. Furthermore, batch methods face the inconsistency between the distribution of obtained samples and the distribution of samples under the current policy. Mahadevan *et al.* suggested the use of a fixed policy for the sample gathering phase (2006). This fixed policy can still cause problems for some domains, as the representation expansion method can exert a lot of effort representing complex value functions corresponding to poor initial policies. This observation motivated researchers to manually include samples that are highly likely to be visited during the execution of the optimal policy (*e.g.*, the bicycle domain in Petrik et al., 2010) which cannot be generalized to arbitrary problems.

Finally methods that do not provide a functional form for the new features (*e.g.*, Parr et al., 2007) cannot provide feature values for unseen samples. Hence for new visited states, another approximation technique such as linear interpolation is required to provide the value estimate.

4.4 Contributions

This chapter introduced online iFDD and online iFDD$^+$and showed how the iFDD family of methods can be combined with any online value-based RL algorithm. The combination of temporal difference learning and online iFDD was investigated across four domains with the planning spaces varying from 1.2×10^3 to $\sim 2 \times 10^8$. Empirical results showed the advantage of online iFDD over using two fixed representations (initial and tabular) and two adaptive methods (ATC and SDM) for tackling large domains. Moreover the convergence of online iFDD combined with temporal difference learning was proved in the case of policy evaluation. It was also shown that when using sparse features, the per-time-step computational complexity of online iFDD is independent of the total number of features, making it an attractive choice for practical domains with constrained interaction time. This chapter also analyzed the performance of online iFDD with respect to online iFDD$^+$ and Random expansion mechanisms. In order to relax the dependency of online iFDD on the initial set of features, the ARiFDD algorithm was introduced as a promising solution capable of expanding the initial representation. Finally this chapter discussed related work on adaptive function approximation techniques, placing iFDD in the context of that work.

Chapter 5

Learning Within Planning

So far, this thesis has focused on model-free reinforcement learning approaches for solving large scale control problems. While such techniques are computationally cheap and can deal with unknown models, they do not have a mechanism to avoid risky behaviors. For example, in the context of a UAV mission planning scenario, the learner might send UAVs with low fuel to remote locations solely for the purpose of learning about the consequences of such behaviors. In the eyes of a human operator who has domain knowledge, such actions may not be acceptable, because losing a UAV is costly and the risk of losing a UAV is high under this plan. Consequently, several cooperative planners exist in the literature for solving control problems based on prior knowledge (Alighanbari, 2004; Alighanbari et al., 2003; Beard et al., 2002; Casal, 2002; Choi et al., 2009; Ryan et al., 2004; Saligrama & Castañón, 2006; Wang et al., 2007; Xu & Ozguner, 9-12 Dec. 2003). This chapter explains how cooperative planners and domain knowledge can help mitigate the risk of learning, reducing the overall sample complexity while boosting the performance of cooperative planners. Parts of this chapter were published as separate papers (Geramifard et al., 2011a,b; Redding et al., 2010a,b).

The structure of this chapter is as follows. Section 5.1 illustrates the goal of this chapter through a pedagogical example. Section 5.2 explains the intelligent cooperative control architecture (iCCA) as a template for integrating cooperative planners with learning algorithms.[1] Section 5.3 focuses on the integration of RL methods that have explicit policy formulations (*e.g.,* Actor-Critic) with cooperative planners where the risk model is deterministic. Section 5.4 extends the previous approach to support RL methods with implicit policy forms. Section 5.5 elevates the risk assessments process by taking a probabilistic approach to risk calculation. It also introduces a method that encapsulates the prior

[1] While all contributions of this chapter are built on top of the iCCA template, the template itself was introduced by Josh Redding (2010b) and is not part of the contributions of this thesis.

knowledge as a separate entity, which can be adaptively changed through time, improving the quality of cooperative planner and risk assessment accordingly. Section 5.6 provides the literature review on cooperative planners and safety in RL. Section 5.7 concludes the chapter by highlighting the contributions.

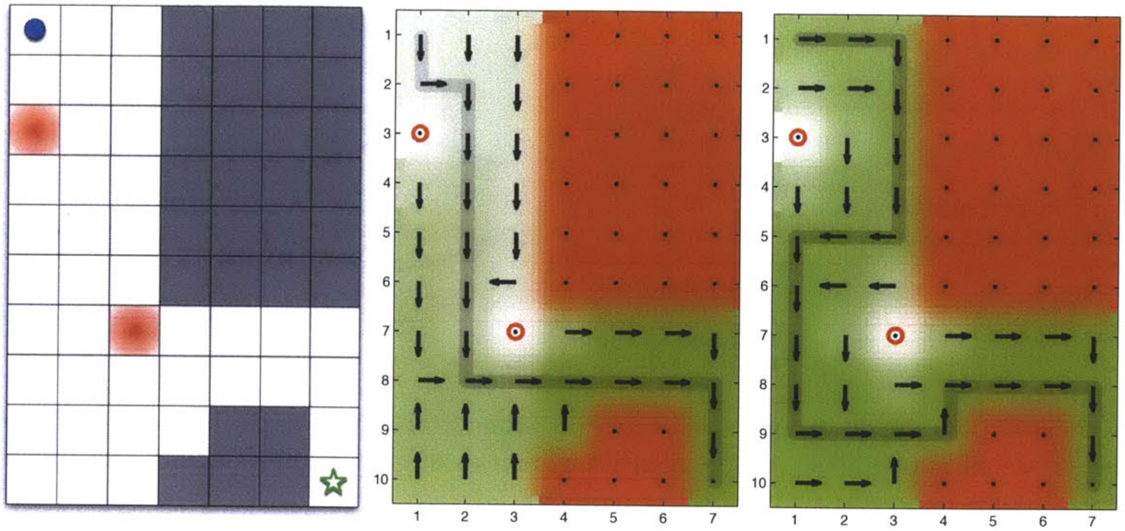
5.1 A Pedagogical Example: GridWorld-1

Consider a grid world scenario shown in Figure 5-1(a), in which the task is to navigate a UAV from the top-left corner ($\bullet$) to the bottom-right corner ($\star$). Red areas highlight the danger zones where the UAV will be eliminated upon entrance. At each step the UAV can take any action from the set $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. However, due to wind disturbances, there is 30% chance that the UAV is pushed into any unoccupied neighboring cell while executing the selected action. The reward for reaching the goal region and off-limit regions are $+1$ and -1 respectively, while every other move results in -0.001 reward.

Figure 5-1(b) illustrates the policy (shown as arrows) calculated by a planner using dynamic programming that is unaware of the wind, together with the nominal path highlighted as a gray tube. As expected, the path suggested by the planner follows the shortest path that avoids directly passing through off-limit areas. The color of each cell represents the true value of each state (*i.e.,* including the wind) under the planner's policy. Green indicates positive, white indicates zero, and red indicates negative values[2]. The optimal policy and its corresponding value function and nominal path are shown in Figure 5-1(c). Notice how the optimal policy avoids the risk of getting close to off-limit areas by making wider turns. While the new nominal path is longer, it mitigates the risk better. In fact, the new policy raises the mission success rate from 29% to 80%, while boosting the value of the initial state by a factor of $\approx$3. Model-free learning techniques such as SARSA can find the optimal policy through mere interaction, although they require a plethora training examples. More importantly, they might deliberately move the UAV towards off-limit regions just to gain information about those areas. However, when integrated with the planner, the learner can rule out intentionally poor decisions. Furthermore, the planner's policy can be used as a starting point for the learner to bootstrap on, reducing the amount of data the learner requires to master the task.

The chapter explains how planner solutions based on approximated models (*i.e.,* Figure 5-1-b) can be improved using learning techniques, while at the same time, the risk in the learning process is reduced reduced. The next section explains the template framework used to combine learning and planning methods together.

[2] We set the value for blocked areas to $-\infty$, hence the intense red color



(a) GridWorld-1 (b) Plan with the noiseless model (c) Plan with the true model

Figure 5-1: GridWorld-1 (a), the corresponding policy calculated with a planner assuming deterministic movement model and its true value function (b) and the optimal policy with the perfect model and its value function (c). The task is to navigate from the top left corner highlighted as • to the right bottom corner identified as ⋆. Red regions are off-limit areas which the UAV should avoid. The dynamics model has 30% noise of moving the UAV to a random free neighboring grid cell. Gray cells are not traversable.

5.2 Intelligent Cooperative Control Architecture (iCCA)

Figure 5-2 depicts the general template of intelligent cooperative control architecture (iCCA) (Redding et al., 2010b). The left rectangle with the gray boundary is the control box and consists of three elements:

- **Cooperative Planner**: Given a problem model, this module provides *safe* solutions with cheap computational complexity, often gained by simplifying the model. Cooperative planners are usually domain-dependent.

- **Learning Algorithm**: This component implements learning by looking at the past experiences of interactions. While in general any machine learning algorithm can sit in this box, in this thesis, RL methods instantiate this component.

- **Performance Analysis**: In the big picture, this module regulates the interaction between the learner and the cooperative planner. The duties of this module can vary based on its instantiation. In this thesis, its purpose is to evaluate the risk involved in executing the actions suggested by the learner.

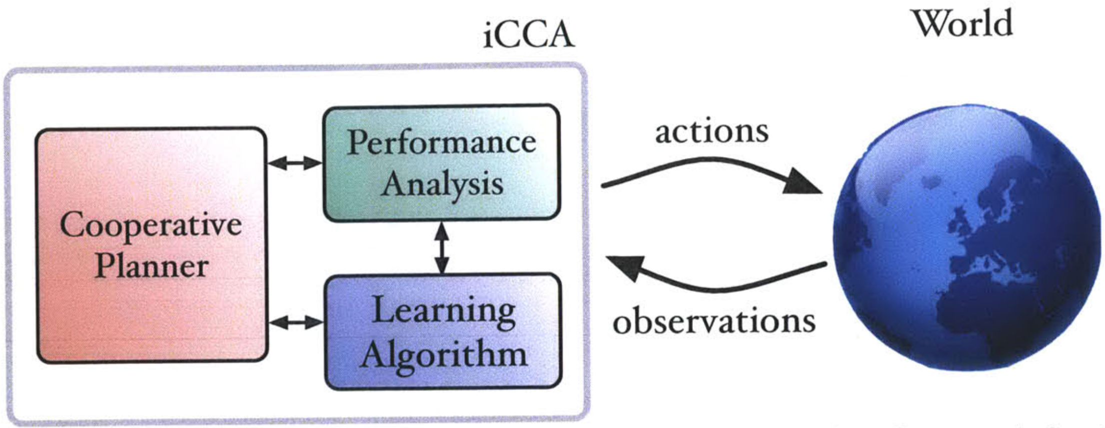


Figure 5-2: **i**ntelligent **C**ooperative **C**ontrol **A**rchitecture, a template framework for the integration of cooperative control algorithms and machine learning techniques (Redding et al., 2010b).

The rest of the figure resembles Figure 2-6 closely, in which in every step the decision made by the control box is sent to the world and executed (*e.g.*, move a UAV to a certain location). During the execution, the command might get distorted (*e.g.*, the UAV moves forward but the wind gust pushes it away). The outcome of each command execution affects the world. Consequently the world sends back the observations to the the control box. The observation may also get distorted through noise, yet this thesis will not consider partial observability. Note that Figure 5-2 depicts a general framework, and depending on the instantiation of the template, several algorithms can be derived.

5.3 Learning Methods with Explicit Policy Forms

This sections explains how RL methods can be combined with cooperative planners in order to 1) mitigate the risk involved in the learning process, 2) improve the sample complexity of the learning methods, and 3) improve the performance of the cooperative planners. The high level idea is to use the solution of the cooperative planner fed with an approximate model to bias the policy of the RL agent in order to explore solutions close to the behavior of the cooperative planner. Furthermore, actions that are deemed not *safe* (*i.e.*, risky) are switched with the cooperative planner solution.

Figure 5-3 depicts the instantiation of the iCCA framework for merging RL methods with cooperative planners. An RL agent realizes the learning algorithm module, while the risk analyzer instantiates the performance analysis box. Also note that observations are replaced with s, r, due to the full observability assumption. The underlying problem is formulated as an MDP with the true model $T = (\mathcal{P}, \mathcal{R})$. An approximate model of the MDP, $\hat{T} = (\hat{\mathcal{P}}, \hat{\mathcal{R}})$ is assumed to be available.

5.3.1 Cooperative Planner

For small MDPs, given the approximated model, value iteration (*i.e.,* Algorithm 2) is used to generate the planner's policy, (*i.e.,* π^p). For large UAV mission planning scenarios, however, running value iteration is not feasible. Therefore the consensus based bundle algorithm (CBBA) (Choi et al., 2009) provides the solution. CBBA is a fast algorithm for task assignment among UAVs where the model of the system has to be deterministic. All stochasticities in $\hat{T}$ are replaced with the events with maximum expected values before being fed into CBBA. For example if there is an 80% chance to move from one state to another state, achieving a reward of 100, the transition is assumed to be successful all the time with the corresponding reward of 80. More information about CBBA can be found in (Choi et al., 2009; Redding et al., 2010a). This thesis uses CBBA as a black box, which takes an approximate model of the system and provides a safe policy quickly that obtains good cumulative rewards. Other cooperative planners can easily replace CBBA box as long as they return safe policies quickly.

5.3.2 RL Agent

For the learning module, the Actor-Critic algorithm with a tabular representation is employed (*i.e.,* Algorithm 12). Since the representation is tabular, instead of using $\boldsymbol{\omega}$ as the weight vector, preferences for each state-action pair are explicitly stored in $\rho(s, a)$. In order to bias the policy of the actor initially, the preferences of state-action pairs sampled from π^p are increased by the user-defined value λ. This will encourage the agent to select actions similar to the cooperative planner initially.

5.3.3 Risk Analyzer

The framework assume the presence of a function named *safe*: $\mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\}$ that returns $True$ if the execution of action a at state s will result in a catastrophic outcome and $False$ otherwise. Also it is assumed that the risk model is deterministic, meaning that actions may not stochastically result in catastrophic outcomes. The deterministic risk model assumption will be relaxed in Section 5.5.

5.3.4 Cooperative Learning: The High Level Control Loop

All sections so far described the mechanics inside each individual piece. Consequently, a high level process is required to tailor the functionality of all these modules together. Algorithm 23 shows the pseudo-code for the main loop of the iCCA control box, named

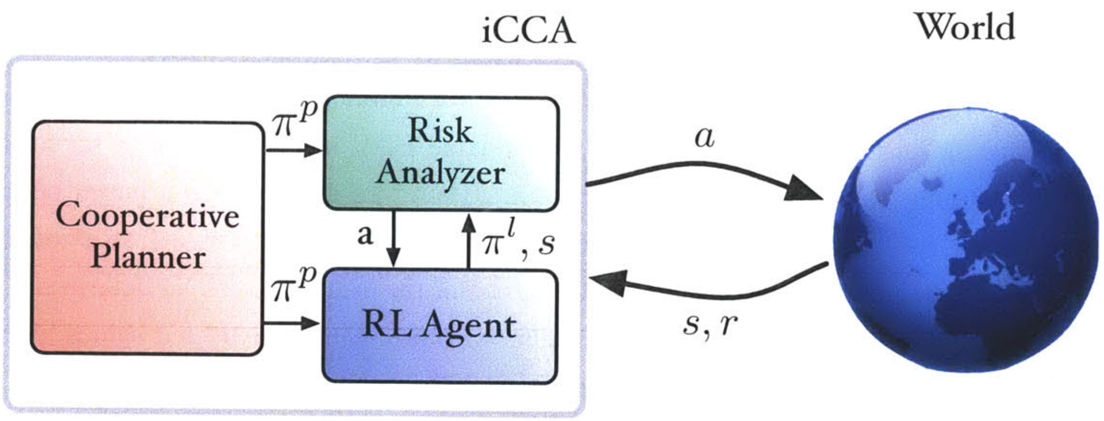
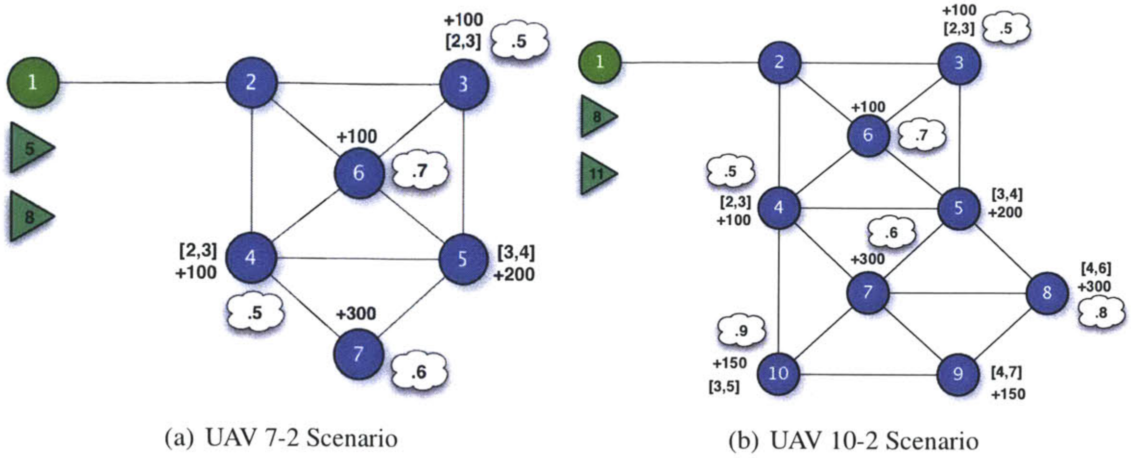


Figure 5-3: iCCA framework instantiation for RL

```
Algorithm 23: Cooperative Learning-1
   Input: $s, r$
   Output: $a$
 1 $a^p \sim \pi^p(s)$                          /* CooperativePlanner */
 2 $a^l \sim \pi^l(s)$                                    /* Learner */
 3 $a \leftarrow a^l$
 4 if not safe$(s, a)$ then
 5     $a \leftarrow \pi^p$
 6     $\rho(s, a) \leftarrow \rho(s, a) - \lambda$
 7 ActorCritic.learn$(s, r, a)$        /* lines 7-10 of Algorithm 12 */
 8 return $a$
```

Cooperative Learning. First the safe action of the planner, a^p, and the learner action a^l are generated using the corresponding policies π^p and π^l (lines1,2). The safety of the learning agent is then tested using the *safe* function (line 4). If the action is safe, it will be executed on the next step, otherwise the action is replaced with the planner's action, a^p, which is assumed to be safe (line 5). What this process dictates, however, is that state-action pairs explicitly forbidden by the *safe* function will not be intentionally visited. Therefore, if the *safe* function is built on a poor model, it can hinder the learning process in parts of the state space for which the safety is miscalculated. To reduce the probability of the learner suggesting the same action, the preference corresponding to the unsafe action is reducing by λ (line 6). This parameter is picked by the domain expert to discourage suggesting unsafe actions by the learner .



(a) UAV 7-2 Scenario

(b) UAV 10-2 Scenario

Figure 5-4: The mission scenarios of interest: a team of two UAVs plan to maximize their cumulative reward along the mission by cooperating to visit targets. Target nodes are shown as circles with rewards noted as positive values and the probability of receiving the reward shown in the accompanying cloud. Note that some target nodes have no value. Constraints on the allowable visit time of a target are shown in square brackets.

5.3.5 Empirical Evaluation

This section compares the performance of iCCA with respect to pure learning and pure cooperative planning approaches in both GridWorld-1 and more complicated UAV mission planning scenarios. Figures 5-4 depicts the mission scenarios of interest where a team of two fuel-limited UAVs cooperate to maximize their total reward by visiting valuable target nodes in the network. The base is highlighted as node 1 (green circle), targets are shown as blue circles and agents as triangles. The total amount of fuel for each agent is highlighted by the number inside each triangle. For those targets with an associated reward, it is given as a positive number nearby. The constraints on the allowable times when the target can be visited are given in square brackets and the probability of receiving the known reward when the target is visited is given in the white cloud nearest the node.[3] Each reward can be obtained only once and traversing each edge takes one fuel cell and one time step. UAVs may loiter at any of the nodes indefinitely if, for some reason, they believe loitering to be the "optimal" action. The fuel burn for a loitering action is also one unit, except for any UAV at the base, where it is assumed to be stationary and its fuel level is therefore not depleted. The mission horizon was set to 8 time steps for UAV 7-2 scenario and 11 for the UAV 10-2 scenario.

[3] If two agents visit a node at the same time, the probability of visiting the node would increase accordingly.

UAV Mission Planning: MDP formulation

The state space was formulated as $[l_1, f_1, \dots, l_n, f_n, v_1, \dots, v_m, t]^T$, where l_i and f_i were integer values highlighting the location and the remaining fuel respectively for UAV i $(i \in 1 \dots n)$. v_j was a single bit signaling if node j had been visited before, where $(j \in 1 \dots m)$, and t was the current time step. There were n UAVs and m nodes participating in the scenario. The action space was $[l_1^+, \dots, l_n^+]$ where l_i^+ was the node to which the agent was traveling. The transition function $(\mathcal{P}_{ss'}^a)$ was deterministic for the UAV position, fuel consumption, and time variables of the state space, while it was stochastic for the visited list of targets. The detailed derivation of the complete transition function should be trivial following the corresponding graph in Figure 5-4. That is, transitions were allowed between nodes for which there was an edge on the graph. The reward on each time step was stochastic and calculated as the sum of rewards from visiting new desired targets minus the total burnt fuel cells on the last move. Notice that a UAV received the target reward only if it landed on an unvisited node and lucky enough to obtain the reward. In that case, the corresponding visibility bit turned on, and the agent received the reward. The crash penalty or mission failure was equal to the negative sum of rewards at all nodes for both scenarios in order to prioritize safety over visiting targets. The mission failed if any UAV ran out of fuel or was not at the base by the end of the mission horizon.

Experimental Results

Both for GridWorld-1 and UAV 7-2 scenario, the optimal solutions were obtained using dynamic programming and used as the baseline for the optimality. Unfortunately, calculating an optimal solution was not feasible for the UAV 10-2 case, with about 9 billion state action pairs.[4] For GridWorld-1, based on the noise-free model, an action was assumed safe if it does not deliberately move the UAV to one of the off-limit grid cells. As for the UAV mission planning scenarios, first all-pairs shortest paths were calculated using Floyd-Warshall algorithm (Cormen et al., 2001) and stored. On each step, an action was assumed unsafe if after executing the action, the UAV does not have enough fuel to return to the base using the shortest path values. For baseline planners, the Value Iteration and CBBA methods were used for GridWorld-1 and the UAV mission planning scenarios correspondingly. Note that Value Iteration did not have access to the true model, while CBBA could not use the exact stochastic model due to its deterministic assumption of the dynamics. The quality of CBBA was probed on each domain by executing its policy online for 10,000 episodes.

[4] This computation for UAV 7-2 scenario took about a day to calculate all expected values over more than 100 million state action pairs. Thus, this approach cannot be easily scaled for larger sizes of the problem.

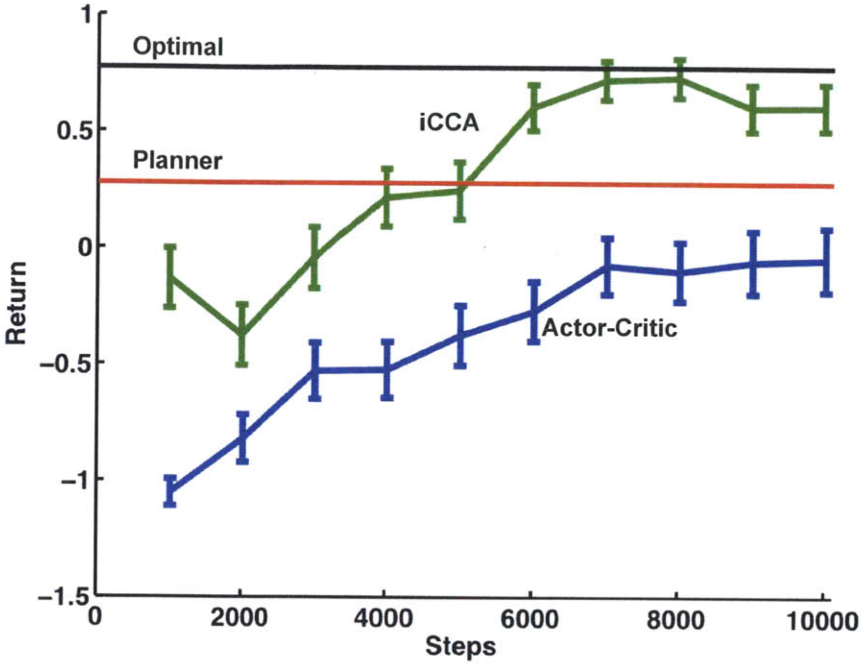


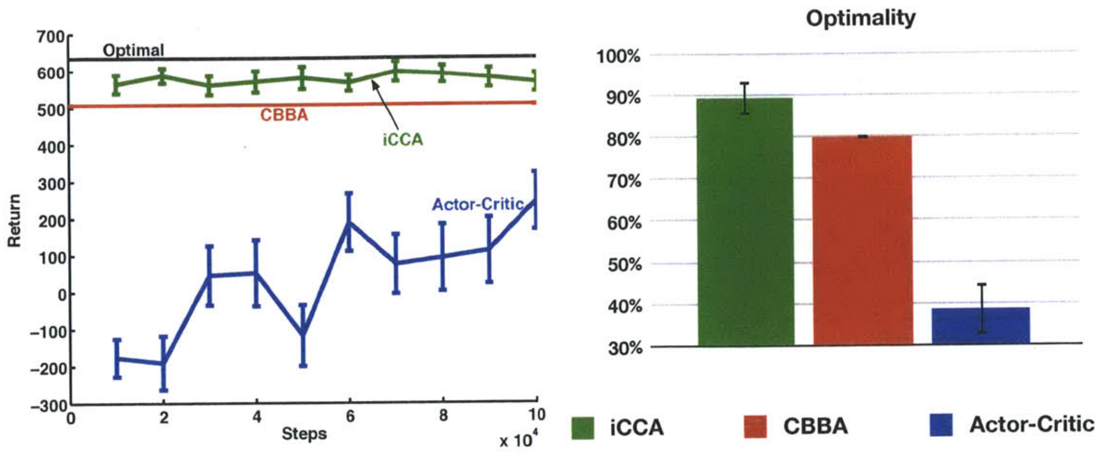
Figure 5-5: A comparison of the collective rewards received in GridWorld-1 using cooperative planning alone (red), pure learning (blue), and when both are coupled via the iCCA framework (green). The optimal performance (black) was calculated via dynamic programming.

For Value Iteration the expected value of the initial state was simply fetched from the table. For each learning algorithm (*i.e.,* Actor-Critic and iCCA) the best learning rate was found empirically where the learning rate was calculated by:

$$\alpha_t = \alpha_0 \frac{N_0 + 1}{N_0 + \text{Episode \#}^{1.1}}.$$

The best α_0 and N_0 were selected through experimental search of the sets of $\alpha_0 \in \{0.01, 0.1, 1\}$ and $N_0 \in \{100, 1000, 10^6\}$ for each algorithm and scenario. λ was set 100 and τ was set to 1 for the actor. The number of interactions for each simulation was limited to 10^4 and 10^5 steps for GridWorld-1 and UAV mission planning scenarios respectively. This led to a cap of 40 minutes of computation for each simulation on an Intel Xeon 2.40 Ghz with 4 GB of RAM and Linux Debian 5.0. The performance of learning algorithms was extracted by running the greedy policy with respect to the existing preferences of the actor. For iCCA, unsafe moves again were replaced by the cooperative planner's solution. All learning method results were averaged over 60 runs except for the UAV 10-2 scenario for which it was averaged over 30 runs. Error bars represent 95% confidence intervals.

Figure 5-5 compares the performance of Actor-Critic, iCCA, the baseline planner (Fig 5-



(a) Step based performance

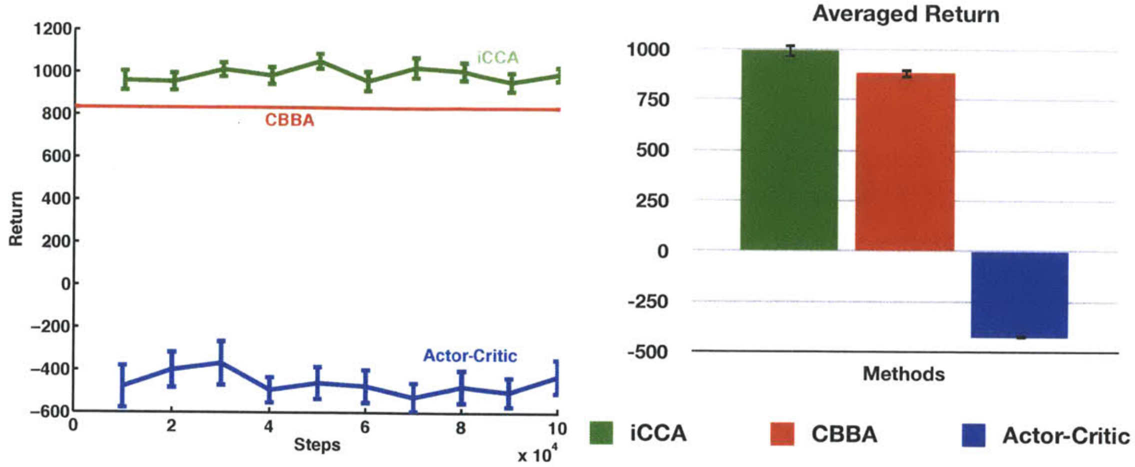
(b) Optimality after training

Figure 5-6: (a) A comparison of the collective rewards received in UAV 7-2 using cooperative planning alone (red), pure learning (blue), and when both are coupled via the iCCA framework (green). The optimal performance (black) was calculated via dynamic programming. (b) The final performance of all methods scaled based on the expected performance of the worst and best policies.

1-b), and the expected optimal solution (Fig 5-1-c) in GridWorld-1. The X-axis shows the number of steps the agent executed an action, while the Y-axis highlights the cumulative rewards of each method after each $1{,}000$ steps. Notice how iCCA outperformed pure learning Actor-Critic. In particular iCCA outperformed the planner (red) after $6{,}000$ steps by navigating farther from the danger zones. Actor-Critic, on the other hand, could not outperform the planner by the end of $10{,}000$ steps.

Similarly, Figure 5-6(a) depicts the performance of Actor-Critic, iCCA, CBBA, and optimal policies in the UAV 7-2 scenario. The Y-axis shows the cumulative reward, while the X-axis represents the number of interactions. It is clear that the Actor-Critic performed much better inside the iCCA framework and performed better than CBBA alone. The reason is that CBBA provided a good starting point for the Actor-Critic to explore the state space, while the risk analyzer filtered risky actions of the actor leading to catastrophic situations. Figure 5-6(b) shows the performance of iCCA and Actor-Critic relative to the optimal policy after 10^5 steps of interaction with the domain and the averaged optimality of CBBA through $10{,}000$ trials. Notice how the integrated algorithm could on average boost the best individual optimality performance (*i.e.,* CBBA's result) by $\%10$.

UAV 10-2 Scenario Figure 5-7(a) depicts the same set of results for the UAV 10-2 scenario. Since the size of the state-action pairs for this domain is about 9 billion, running



(a) Step based performance

(b) Averaged return after training

Figure 5-7: A comparison of the collective rewards received in UAV 10-2 scenario when strictly following plans generated by CBBA alone, Actor-Critic reinforcement learning outside of the iCCA environment, *i.e.,* without initialization and guidance from CBBA, and the result when these are coupled via the iCCA framework.

dynamic programming to obtain the optimal policy was not feasible. For that reason the performance of the optimal policy is excluded. Since the state space was much larger than the UAV 7-2 scenario, Actor-Critic method had a hard time to find a sensible policy even after 10^5 steps. Online CBBA still could find a good policy to the approximated problem. When both CBBA and Actor-Critic were put together through the iCCA framework, the agent could achieve better performance even early on, after only 10^4 steps. Figure 5-7(b) shows the averaged performance of each method at the end of the learning phase. Notice that iCCA again could boost the performance of CBBA solution statistically significantly.

5.4 Learning Methods with Implicit Policy Function

The initial policy of Actor-Critic type learners can be biased simply as they parameterize the policy explicitly. For learning schemes that do not represent the policy as a separate entity, such as SARSA, integration within the iCCA framework is not immediately obvious. This section presents a new approach for integrating learning approaches without an explicit actor component. The idea is motivated by the concept of the R_{max} algorithm (Brafman & Tennenholtz, 2001). The approach can be explained through the mentor-protégé analogy, where the planner takes the role of the mentor and the learner takes the role of the protégé. In the beginning, the protégé does not know much about the world,

Algorithm 24: Cooperative Learning-2

```
Input: $s, r$
Output: $a$
1  $a \sim \pi^{p}(s)$                                   /* CooperativePlanner */
2  knownness $\leftarrow \min\{1, \frac{\texttt{count}(s,a)}{\mathcal{K}}\}$
3  if $rand() <$ knownness then
4      $a' \sim \pi^{l}(s)$                              /* Learner */
5      if $safe(s, a')$ then
6          $a \leftarrow a'$
7  else
8      count$(s, a) \leftarrow$ count$(s, a) + 1$
9  learner.update$(s, r, a)$
10 return $a$
```

hence, for the most part s/he takes actions advised by the mentor. While learning from such actions, after a while, the protégé feels comfortable about taking a self-motivated actions as s/he has been through the same situation many times. Seeking permission from the mentor, the protégé could take the action if the mentor thinks the action is safe. Otherwise the protégé should follow the action suggested by the mentor.

Algorithm 24 details the new cooperative learning process. On every step, the learner inspects the suggested action by the planner and estimates the "knownness" of the state-action pair by considering the number of times that state-action pair has been experienced following the planner's suggestion. The $\mathcal{K}$ parameter controls the transition speed from following the planner's policy to following the learner's policy. Given the knownness of the state-action pair, the learner probabilistically decides to select an action from its own policy. If the action is deemed to be safe, it is executed. Otherwise, the planner's policy overrides the learner's choice (lines 4-6). If the planner's action is selected, the knownness count of the corresponding state-action pair is incremented. Finally the learner is executed depending on the choice of the learning algorithm. Note that any control RL algorithm, even the Actor-Critic family of methods, can be integrated with cooperative planners using Algorithm 24 as line 9 is the only learner-dependent line, defined in the general form.

5.4.1 Experimental Results

This section compares the empirical performance of SARSA combined with cooperative planners (CSARSA), with pure learning and pure planning methods in both the GridWorld-1 domain and the UAV 7-2 mission planning scenario. All cooperative planners and settings

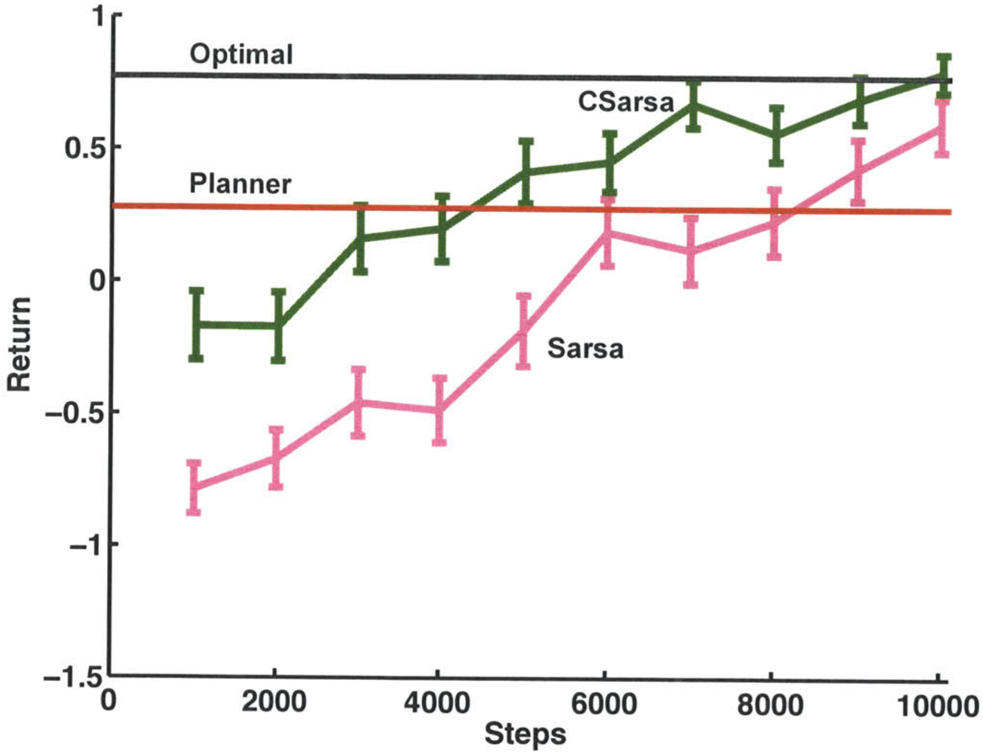


Figure 5-8: In the pedagogical GridWorld domain, the performance of the optimal solution is given in black. The solution generated by the deterministic planner is shown in red. In addition, the performance of SARSA and CSARSA are shown. It is clear that CSARSA outperform SARSA and eventually outperform the baseline planner when given a sufficient number of interactions.

remained the same from the previous set of experiments in Section 5.3.5. The knownness parameter, $\mathcal{K}$, for CSARSA was selected out of $\{10, 20, 50\}$. The exploration rate (ϵ) for SARSA and CSARSA was set to 0.1. All learning method results were averaged over 60 runs.

GridWorld-1 Figure 5-8 compares the performance of CSARSA, SARSA, the baseline planner (Fig 5-1-b), and the expected optimal solution (Fig 5-1-right) in the pedagogical GridWorld domain. The X-axis shows the number of steps the agent executed an action, while the Y-axis highlights the cumulative rewards of each method after each $1{,}000$ steps. Notice how CSARSA outperformed pure learning approaches. Compared to Figure 5-5, SARSA based methods learned faster. This observation can be explained by two facts: 1) SARSA's policy is embedded in the $Q-$value function, whereas the actor requires another level of learning for the policy on the top of learning the $Q-$value function and 2) Algorithm 24 provides a better exploration mechanism (*i.e.,* R_{max} like) compared to Algo-

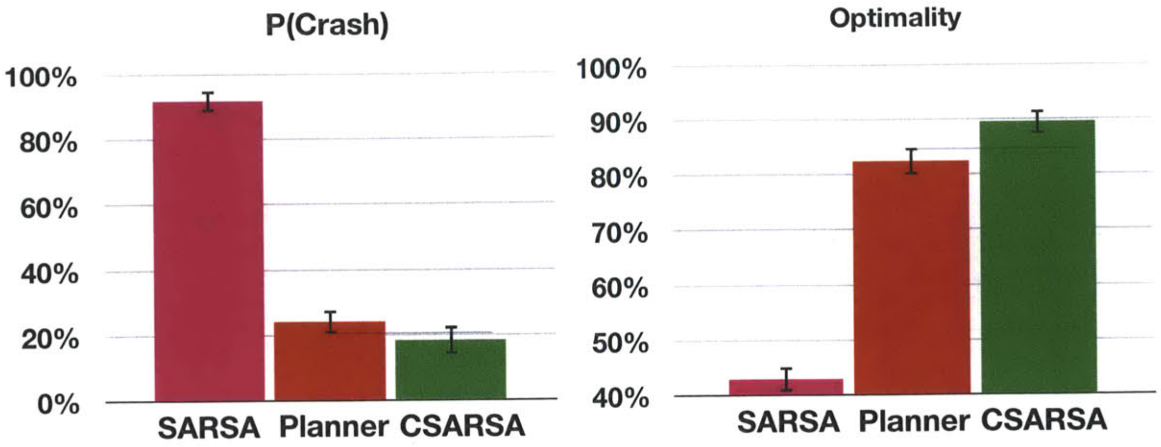


Figure 5-9: Probability of crash (left) and optimality (right) of SARSA, CBBA (planner), and CSARSA algorithms at the end of the training session in the UAV 7-2 mission planning scenario. CSARSA improved the performance of both CBBA and SARSA. The optimality improvement of CSARSA was statistically significant.

rithm 23, where exploration is realized internally by the actor.

UAV 7-2 Scenario In order to test our risk mitigation approach under harder circumstances, 5% chance of edge traverse failure was added for each UAV, resulting in a fuel burn without no movement. This noise value was not included in the approximated model, hence the safety checking mechanism could not consider it. Figure 5-9 shows the results of the same battery of learning algorithms used in GridWorld-1 applied to the UAV mission planning scenario at the end of learning (*i.e.,*10^5 steps of interaction). Akin to the previous section, CBBA was used as the base line planner. The left plot exhibits the risk of executing the corresponding policy while the right plot depicts the optimality of each solution. At the end of learning, SARSA could barely avoid crashing scenarios (about 90%), thus yielding low performance with less than 50% optimality. This observation coincides with the previous experiments with this domain where the movement model was noise free (Figure 5-6), highlighting the importance of biasing the policy of learners in large domains and avoiding risky behaviors. On average, CSARSA reduced the probability of failure of CBBA by 6%, yet this improvement was not statistically significant. At the same time, CSARSA raised the optimality of CBBA by 7%. This improvement was statistically significant.

5.5 Adaptive Models

So far the approximated model of the MDP, $\hat{T}$, was assumed to be static during the course of interaction with the world. Moreover, the safety of actions was filtered based on the deterministic *safe* function (*e.g.,* movement and fuel burn did not involve uncertainty in the approximated model). In this section, the iCCA framework is extended so that the approximate model can be adapted through the course of online learning. In particular the parametric form of the model is assumed to be able to capture the true model, where parameters of the model are adjusted using adaptive parameter estimation. Moreover, a probabilistic approach towards safety is introduced which allows the use of stochastic approximated models to estimate risk. Empirical results demonstrate that the performance of the resulting system increases. Finally the drawbacks of having an underpowered model representation are covered and an alternative solution is suggested.

5.5.1 Pedagogical GridWorld-2

Consider the GridWorld-2 domain shown in Figure 5-10(a), in which the task is to navigate from the bottom-middle (•) to one of the top corner grid cells ($\star$), while avoiding the danger zone (∘), where the agent will be eliminated upon entrance. At each step the agent can take any action from the set $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. However, due to wind disturbances unbeknownst to the agent, there is a 20% chance the agent will be transferred into a neighboring unoccupied grid cell upon executing each action. The reward for reaching either of the goal regions and the danger zone are $+1$ and -1, respectively, while every other action results in -0.01 reward.

First consider the conservative policy shown in Figure 5-10(b) designed for high values of wind noise. As expected, the nominal path, highlighted as a gray watermark, follows the long but safe path to the top left goal. The color of each grid represents the true value of each state under the policy. Green indicates positive, and white indicates zero. The value of blocked grid cells are shown as red. Figure 5-10(c) depicts a policy designed to reach the right goal corner from every location. This policy ignores the existence of the noise, hence the nominal path in this case gets close to the danger zone. Finally Figure 5-10(d) shows the optimal solution. Notice how the nominal path under this policy avoids getting close to the danger zone. Model-free learning techniques such as SARSA can find the optimal policy of the noisy environment through interaction, but require a great deal of training examples. More critically, they may deliberately move the agent towards dangerous regions in order to gain information about those areas. Section 5.4 demonstrated that when an approximate model (*e.g.,* the model used to generate policies in Figure 5-10-b,c) is integrated with a

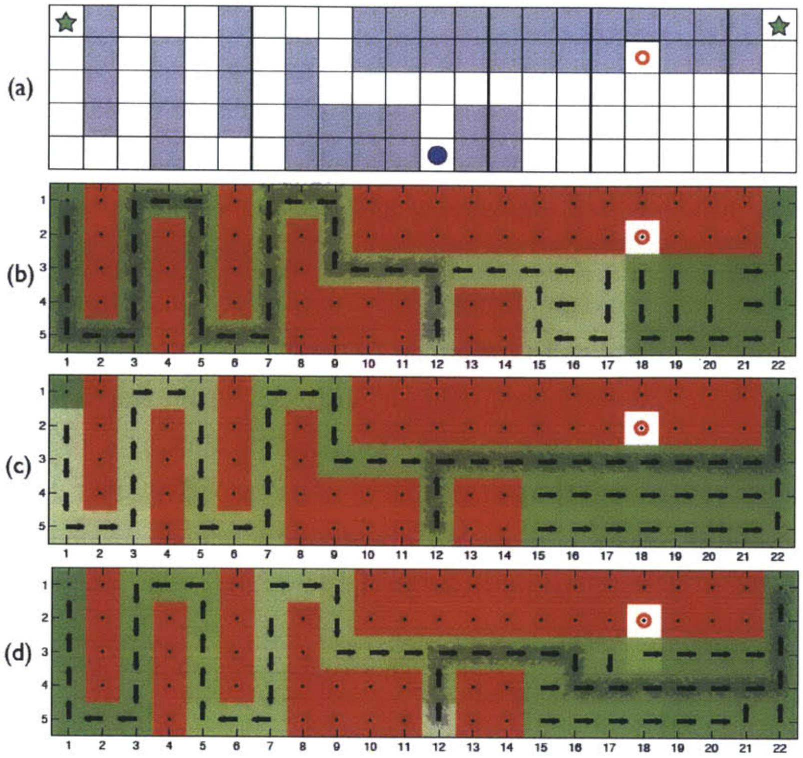


Figure 5-10: The gridworld domain is shown in (a), where the task is to navigate from the bottom middle (•) to one of the top corners (⋆). The the danger region (○) is an off-limit area where the agent should avoid. The corresponding policy and value function, are depicted with respect to (b) a conservative policy to reach the left corner in most states, (c) an aggressive policy which aims for the top right corner, and (d) the optimal policy.

learner, it can rule out suggested actions by the learner that are poor in the eyes of the planner, resulting in safer exploration. Furthermore, the planner's policy can be used as a starting point for the learner to bootstrap on, potentially reducing the amount of data required by the learner to master the task. However, the model used for planning and risk analysis were static. This section expands the iCCA framework by representing the model as a separate entity which can be adapted through the learning process. The focus here is on the case where the parametric form of the approximated model ($\hat{T}$) includes the true underlying model (T) (*e.g.,* assuming an unknown uniform noise parameter for the gridworld domain). Later the drawbacks of this approach when $\hat{T}$ is unable to exactly represent T are discussed and a potential alternative is introduced. Adding a parametric model to the planning and learning scheme is easily motivated by the case when the initial bootstrapped policy is wrong, or built from incorrect assumptions. In such a case, it is more effective to simply switch the underlying policy with a better one, rather than requiring a plethora of interactions to learn from and refine a poor initial policy. The remainder of this

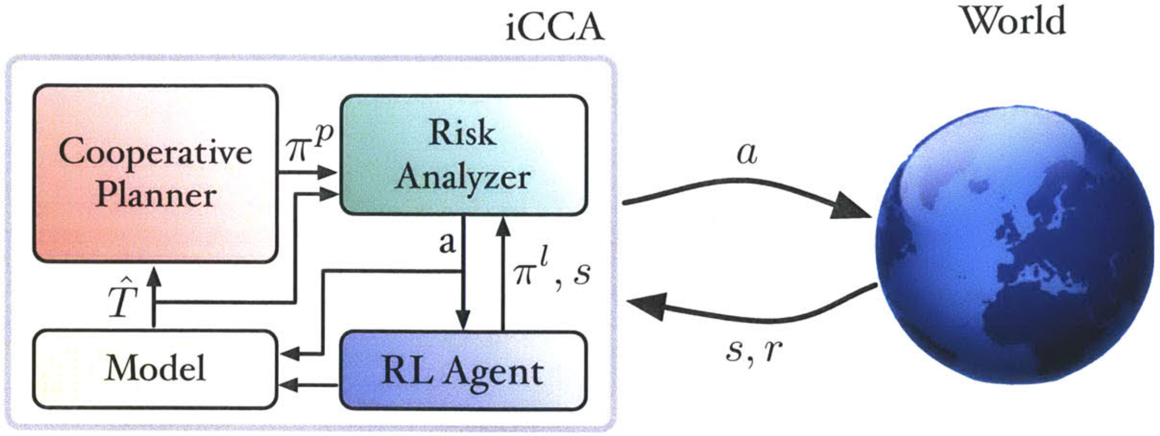


Figure 5-11: iCCA framework instantiation for RL with separate Model module

section introduces a new stochastic risk analyzer and a cooperative learning process that is able to intelligently switch-out the underlying policy, refined by the learning process.

Figure 5-11 depicts the new iCCA instantiation. Note that the model, (*i.e.,* $\hat{T}$) now has its own encapsulation, and is shared both by the cooperative planner and the risk analyzer. Moreover, the model module uses perceived state-action pairs to adjust its own parameters. In order to take advantage of a dynamic model, the risk analyzer should support stochastic risk models. Algorithm 25 explains the new risk analysis process in which the notation of safety is defined probabilistically. In particular, it is assumed that there exists a function $constrained : \mathcal{S} \to \{0, 1\}$, which indicates if being in a particular state is allowed or not. Risk is defined as the probability of visiting any of the constrained states. The core idea is to use Monte-Carlo simulation to estimate the risk level associated with the given state-action pair if planner's policy is applied thereafter by simulating $\mathcal{M}$ trajectories from the current state s. The first action is the suggested action a, and the rest of actions come from the planner's policy, π^p. The approximated model, $\hat{T}$, is utilized to sample successive states. Each trajectory is bounded to a fixed horizon $\mathcal{H}$. The risk of taking action a from state s is estimated by the probability of a simulated trajectory reaching a risky state within horizon $\mathcal{H}$. If this risk is below a user-defined threshold, ε, the action is deemed to be safe.

Algorithm 26 depicts the new cooperative algorithm. Lines 1-7 are identical to Algorithm 24, while lines 8-13 highlights the new part of the algorithm which includes model adaptation. The risk mitigation process is the same as Algorithm 24, yet the *safe* function is now defined by Algorithm 25 which uses the most recent version of the estimated model, $\hat{T}$. Line 9 updates the current estimate of the model, based on the observations, providing more accurate safety estimations compared to a static risk analyzer. Furthermore, if the

Algorithm 25: safe (check the safety of the action suggested by the learner)

```
Input: $s, a$
Output: isSafe
risk $\leftarrow 0$
for $i \leftarrow 1$ to $\mathcal{M}$ do
    $t \leftarrow 1$
    $s_t \sim \hat{T}(s, a)$
    while not constrained($s_t$) and not isTerminal($s_t$) and $t < \mathcal{H}$ do
        $s_{t+1} \sim \hat{T}(s_t, \pi^p(s_t))$
        $t \leftarrow t + 1$
    risk $\leftarrow$ risk $+ \frac{1}{i}($constrained$(s_t) -$ risk$)$
isSafe $\leftarrow$ (risk $< \varepsilon$)
```

change to the model used for planning crosses a user-defined threshold (ΔT), the planner revisit its policy and keeps record of the new model (lines 10-12). If the policy changes, the *counts* of all state-action pairs are set to zero so that the learner start watching the new policy (mentor) from the scratch (line 13,14). An important observation is that the planner's policy should be seen safe through the eyes of the risk analyzer at all times. Otherwise, most actions suggested by the learner will be deemed too risky by mistake, as they are followed by the planner's policy. Hence the output of the iCCA is reduced to the baseline planner's policy.

5.5.2 Experimental Results

This part probes the effectiveness of the adaptive modeling approach (*i.e.,* Algorithm 26), called AM-iCCA, compared to (i) the static model iCCA (*i.e.,* Algorithm 24) and (ii) the pure learning approach. Both iCCA and AM-iCCA used Algorithm 25 to estimate the safety, since the approximated model ($\hat{T}$) is stochastic. The empirical settings omitted here were identical to those of Section 5.3.5. 5 Monte-Carlo simulations used to evaluate risk (*i.e.,* $\mathcal{M} = 5$). Each algorithm was tested for 100 trials. The risk tolerance (ε) was set to 20%. For the AM-iCCA, the noise parameter was estimated as:

$$noise = \frac{\#\text{unintended agents moves} + \text{initial weight}}{\#\text{total number of moves} + \text{initial weight}}.$$

Both iCCA methods started with the noise estimate of 40% with the count weight of 100.

Algorithm 26: Cooperative Learning-3

```
Input: $s, r$
Output: $a$
$a \sim \pi^p(s)$
knownness $\leftarrow \min\{1, \frac{\texttt{count}(s,a)}{\mathcal{K}}\}$
if $rand() <$ knownness then
    $a' \sim \pi^l(s)$
    if $safe(s, a')$ then
        $a \leftarrow a'$
else
    count$(s, a) \leftarrow$ count$(s, a) + 1$
learner.update$(s, r, a)$
model.update$(s, r, a)$                                /* Update $\hat{T}$ */
if $||\hat{T}^p - \hat{T}|| > \Delta T$ then
    $\hat{T}^p \leftarrow \hat{T}$
    planner.update()                                   /* Update $\pi^p$ */
    if $\pi^p$ is changed then
        reset all counts to zero
return $a$
```

Pedagogical GridWorld-2 For the iCCA algorithm, the planner followed the conservative policy (Figure 5-10-b). As for AM-iCCA, the planner switched from the conservative to the aggressive policy (Figure 5-10-c), whenever the noise estimate dropped below 25%. The knownness parameter ($\mathcal{K}$) was set to 10. Figure 5-12 compares the cumulative return obtained in the GridWorld-2 domain for SARSA, iCCA, and AM-iCCA based on the number of interactions. The expected performance of both static policies are shown as horizontal lines, estimated by 10,000 simulated trajectories. The improvement of iCCA with a static model over the pure learning approach is statistically significant in the beginning, while the improvement is less significant as more interactions were obtained. Although initialized with the conservative policy, the adaptive model approach within iCCA (shown as green in Figure 5-12) quickly learned that the actual noise in the system was much less than the initial 40% estimate and switched to using (and refining) the aggressive policy. As a result of this early discovery and switching planner's policy, AM-iCCA outperformed both iCCA and SARSA, requiring two times less data compared to other learning methods to reach the asymptotic performance.[5] Over time, however, all methods reached the same level of performance. On that note, it is important to see that all learning methods

[5] Compare AM-iCCA's performance after 4,000 steps to other learning methods' performance after 8,000 steps.

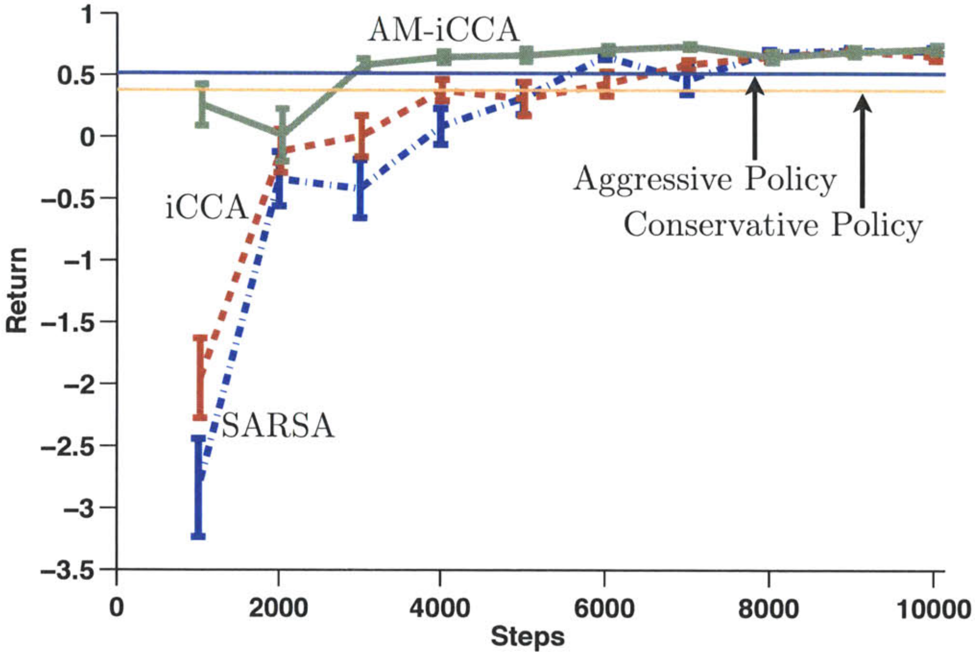


Figure 5-12: Empirical results of AM-iCCA, iCCA, and SARSA algorithms in the GridWorld-2.

(SARSA, iCCA, AM-iCCA) improved on the baseline static policies, highlighting their sub-optimality.

UAV 7-2 Scenario The UAV 7-2 Scenario was implemented with 5% movement noise identical to Section 5.4.1. As for the baseline cooperative planner, CBBA (Choi et al., 2009) was implemented in two versions: aggressive and conservative. The aggressive version used all remaining fuel cells in one iteration to plan the best set of target assignments ignoring the possible noise in the movement. Algorithm 27 illustrates the conservative CBBA algorithm. The input to the algorithm is the collection of UAVs (U). First the current fuel of UAVs are saved and decremented by 3 (lines 1-2). Then on each iteration, CBBA is called with the reduced amount of fuel cells. Consequently, the plan will be more conservative compared to the case where all fuel cells are considered. If the resulting plan allows all UAVs to get back to the base safely, it will be returned as the solution. Otherwise, UAVs with no feasible plan (*i.e.,* $Plan[u] = \emptyset$) will have their fuels incremented, as long as the fuel does not exceed the original fuel value (line 8). Notice that aggressive CBBA is equivalent to calling CBBA method on line 5 with the original fuel levels. Akin to the GridWorld-2 domain, the iCCA algorithm only took advantage of the conservative CBBA because the noise assumed to be fixed at 40%. As for AM-iCCA, the planner switched from the conservative to the aggressive CBBA, whenever the noise estimate dropped below

Algorithm 27: Conservative CBBA

```
Input: UAVs
Output: Plan
MaxFuel ← U.fuel
UAVs.fuel ← UAVs.fuel − 3
ok ← False
while not $ok$ or MaxFuel = UAVs.fuel do
    Plan ←CBBA(UAVs)
    ok ← True
    for $u$ ∈ UAVs, Plan[$u$] = ∅ do
        UAVs.fuel[$u$] ← min(MaxFuel[$u$], UAVs.fuel[$u$] + 1)
        ok ← False
return Plan
```

25%. The best knowness parameter ($\mathcal{K}$) was selected from $\{10, 20, 50\}$ for both iCCA and AM-iCCA.

Figures 5-13 shows the results of learning methods (SARSA, iCCA, and AM-iCCA) together with two variations of CBBA (conservative and aggressive) applied to the UAV mission planning scenario. Figure 5-13(a) represents the solution quality of each learning method after 10^5 steps of interactions. The quality of fixed CBBA methods were obtained through averaging over 10,000 simulated trajectories, where on each step of the simulation a new plan was derived to cope with the stochasticity of the environment. Figure 5-13(b) depicts the optimality of each solution, while Figure 5-13(c) exhibits the risk of executing the corresponding policy. First note that SARSA at the end of training yielded 50% optimal performance, together with more than 80% chance of crashing a UAV. Both CBBA variations outperformed SARSA. The aggressive CBBA achieved more than 80% optimality in cost of 25% crash probability, while conservative CBBA had 5% less performance, as expected, it realized a safe policy with rare chances of crashing. The iCCA algorithm improved the performance of the conservative CBBA planner again by introducing risk of crash around 20%. While on average it performed better than that aggressive policy, the difference was not statistically significant. Finally AM-iCCA outperformed all other methods statistically significantly, obtaining close to 90% optimality. AM-iCCA boosted the best performance of all other methods by 22% on average (Figure 5-13-a). The risk involved in running AM-iCCA was also close to 20%, matching the selected ε value.

These result highlights the importance of an adaptive model within the iCCA framework: 1) model adaptation provides a better simulator for evaluating the risk involved in taking learning actions, and 2) planners can adjust their behaviors according to the model,

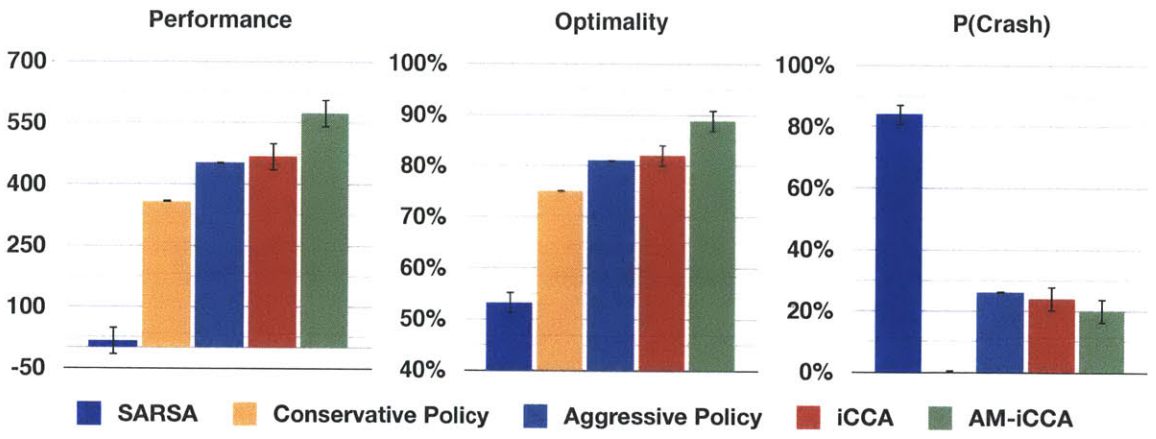


Figure 5-13: Results of SARSA, CBBA-conservative, CBBA-Aggressive, iCCA and AM-iCCA algorithms at the end of the training session in the UAV mission planning scenario. AM-iCCA improved the best performance by 22% with respect to the allowed risk level of 20%.

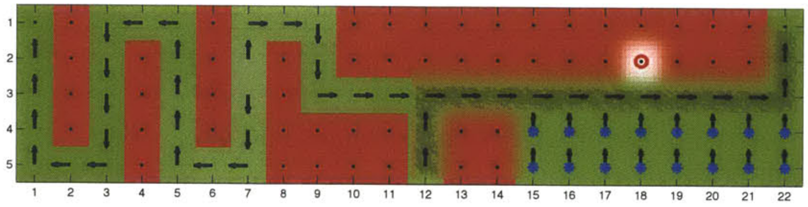


Figure 5-14: The GridWorld-3 scenario: identical to GridWorld-2, yet the noise is only applied in windy grid cells ($*$).

resulting in better policies serving as the stepping stones for the learning algorithms to build upon.

5.5.3 Extensions

So far, the true model was assumed to be representable within the functional form of the approximated model. But what are the consequences of using cooperative learning if this assumption does not hold? Returning to the GridWorld-2 domain, consider the case where the 20% noise is not applied to all states. Figure 5-14 depicts such a scenario through GridWorld-3 where the noise is only applied to the grid cells marked with a $*$. While passing close to the danger zone is safe, when the agent assumes the uniform noise model by mistake, it generalizes the noisy movements to all states including the area close to the danger zone. This can cause the AM-iCCA to converge to a suboptimal policy, as the risk analyzer filters optimal actions suggested by the learner due to the inaccurate model assumption.

The root of this problem is that iCCA always filters the risky actions of the learner, even though the underlying assumed model might be incorrect. In order to allow the learning agent the freedom of trying potentially risky actions asymptotically, the safety check should be turned off for all states at some point. Back to the mentor/protégé analogy, the protégé may simply stop checking if the mentor thinks an action is safe once s/he feels comfortable taking a self-motivated action. Thus, the learner will eventually circumvent the need for a planner altogether. More specifically, line 4 of Algorithm 26 is changed, so that if the knownness of a particular state reaches a certain threshold, probing the safety of the action is not mandatory anymore. While this approach introduces another parameter to the framework, we conjecture that the resulting process converges to the optimal policy under certain conditions. This is due to the fact that under an ergodic policy realized by the ϵ-greedy policy, all state-action pairs will be visited infinitely often. Thus at some point the knownness of all states exceeds any predefined threshold, which leads to 1) having SARSA suggest an action for every state, and 2) turning the risk filtering mechanism off for all states. As a result, the whole iCCA framework is reduced to pure SARSA with an initial set of weights. Under certain conditions, it can be shown that the resulting method is convergent to the optimal policy with probability one (Melo et al., 2008).

5.6 Related Work

In the controls community, many researchers have developed algorithms for task assignment among teams of vehicles under the name of cooperative planning (Alighanbari, 2004; Alighanbari et al., 2003; Beard et al., 2002; Berman et al., 2009; Cassandras & Li, 2002; Castanon & Wohletz, 2009; Choi et al., 2009; Ryan et al., 2004; Saligrama & Castañón, 2006; Wang et al., 2007, 2005). As the name suggests, these methods use an existing world model for planning vehicles task assignment. While, in theory, such planning problems can be solved using dynamic programming (DP) (Bellman, 2003), the computation time required for solving realistic problems using DP is not practically feasible. Consequently, cooperative planners are often focused on problems with specific properties such as convexity, submodularity, *etc.* that render the problem more tractable. Furthermore, they investigate approximation techniques for solving planning problems.

Cassandras *et al.* adopted a receding horizon approach for planning in obstacle free 2D spaces where vehicles with constant speeds move according to deterministic dynamics (Cassandras & Li, 2002). Singh *et al.* proposed a non-myopic approach and derived theoretical guarantees on the performance of their algorithm (Singh et al., 2009). Their main results are built upon the submodularity (a diminishing returns property) assumption which is a limiting factor. Alighanbari *et al.* formulated the task allocation problem as a mixed-integer linear program (Alighanbari et al., 2003). They scaled their solution to large domains including 6 vehicles by approximating the decomposition of task assignment among vehicles. The main drawback of the work is the assumption of deterministic dynamics for vehicle movements. Wang *et al.* introduced a new method for maintaining the formation among vehicles in domains with dynamic obstacles (Wang et al., 2005). Their approach assumes a quadratic cost function and a Gaussian noise model. In our UAV mission planning scenarios both these assumptions are violated: the penalty function is step-shaped and the noise involved in the reward function has a multinomial distribution. Castanon *et al.* focused on stochastic resource allocation problem (Castanon & Wohletz, 2009). While their method scaled well to domains with 20 resources and 20 tasks, it cannot be applied to cases where multiple assignments of simultaneous resources are required for the task completion. Berman *et al.* tackled the problem of task allocation for swarms of robots in a distributed fashion using stochastic policies (Berman et al., 2009). Their approach addresses the problem of maintaining a predefined distribution of robots on a set of locations, yet does not support time varying distributions. Choi *et al.* used a distributed auction-based approach for task assignment among agents (2009). They bounded the sub-optimality of their solution, yet extending their work to stochastic models is still an open problem. In summary, cooperative planners often result in sub-optimal solutions due to the model inaccuracy or unsatisfied assumptions. Furthermore, these methods do not incorporate learning to use perceived trajectories in order to improve future performance.

Safe exploration has been of special interest to practitioners applying RL techniques to real world domains involving expensive resources. Pessimistic approaches find optimal policies with respect to the worse possible outcome of selected actions (Heger, 1994). Consequently, resulting policies are often overly constrained, yielding undesirable behavior. Bounding the probability of failure, risk-sensitive RL methods (Geibel & Wysotzki, 2005; Mihatsch & Neuneier, 2002) provide a less conservative approach by maximizing the performance subject to an acceptable risk level. Because these methods do not guarantee the performance before the learning phase, they are not suitable for online settings. Abbeel *et al.* investigated exploration within the context of batch apprenticeship learning (2005). They conjectured that running least-squares policy iteration (Lagoudakis & Parr, 2003) over teacher generated trajectories yields safe policies for practical domains. While in practice they reported safe resulting policies, their approach does not have mathematical guarantees. For deterministic MDPs, Hans *et al.* extended the risk-sensitive RL approach by identifying states as super critical, critical, and safe (Hans et al., 2008). While they demonstrated promising empirical results, their method cannot be applied to stochastic domains. Knox *et al.* discussed various ways that an initial policy can bias the learning process (2010), yet their approach requires access to the value function of the given policy which may be hard to calculate. Within the controls community, safe exploration is pursued under robust RL in which the stability of the system is of main interest (Anderson et al., 2007). Current state of the art methods do not extend to general MDPs as they consider systems with linear transition models (Anderson et al., 2007).

5.7 Contributions

This chapter introduced cooperative learners formed by combining cooperative planners with RL algorithms. The first set of cooperative learners were built on top of RL methods with explicit policy parameterization using deterministic risk models. The second set of cooperative learners relaxed the assumption of an explicit parametric form for the RL methods, allowing the fusion of any online RL technique with cooperative planners. The third set of cooperative learners accommodated the notion of adaptive modeling within the framework and also supported stochastic risk models. Resulting methods were empirically investigated in pedagogical grid-world domains and UAV mission planning scenarios with state-action spaces up to 9 billions. Empirical results demonstrated the ability of cooperative learners in boosting the performance of fixed planners, while reducing the risk and sample complexity involved in the learning process. This chapter also explained the drawbacks of adaptive modeling when the parametric form of the model cannot be captured in the exact model. Finally this chapter provided a literature review of cooperative planners and safety within learning.

Chapter 6

Conclusion

This chapter provides a summary of the thesis and presents a list of the contributions. It also sheds light on potential future work that has been motivated by the results of this thesis.

6.1 Contributions

The main goal of this thesis has been to develop algorithms that find good policies in large-scale sequential decision making problems under uncertainty by interacting with the system. The motivating example that was carried through the thesis was the mission planning for a set of fuel-limited unmanned air vehicles (UAVs) with more than 100 million state-action pairs. The three main challenges to scale existing methods up to such scenarios were: (I) sample complexity, (II) limited computation, and (III) safe exploration. This thesis tackled these challenges within the following three thrusts.

6.1.1 A Theoretical View of Finding the Right Set of Features

Chapter 3 focused on reinforcement learning algorithms using linear function approximation to represent the value function. Addressing mainly challenge (I) and partially challenge (II), this chapter:

- Defined the notion of feature coverage in the context of linear function approximation and provided a theoretical analysis on how feature coverage plays a critical role in providing a guaranteed rate of convergence in the policy evaluation case (*i.e.*, Theorem 3.2.6 and Corollary 3.2.8).

- Introduced the incremental feature dependency discovery (iFDD) family of algorithms based on the theoretical results as a novel feature expansion process. It also showed how iFDD can be viewed geometrically as a dynamic Tile Coding scheme.

- Empirically verified the advantage of discovering features using batch iFDD as opposed to batch random expansion in the task of inverted pendulum.
- Presented a sparsification method to reduce the computational complexity of iFDD, and analyzed the theoretical consequences of such sparsification.

6.1.2 An Empirical View of Finding the Right Set of Features

Chapter 4 concentrated on practical implementations of iFDD. The aim was to approximate the optimization criteria of iFDD with the goal of applying the resulting algorithm to the online setting. Addressing both challenges (I) and (II) this chapter:

- Introduced online iFDD and online iFDD$^+$ as two members of iFDD family which can be combined with any online value based RL method. Given a set of initial features, in the case of policy evaluation, this chapter also proved the asymptotic convergence of online iFDD combined with temporal difference learning to the best possible approximation (*i.e.,* Corollary 4.1.3).
- Empirically verified the advantage of online methods over two fixed representations (initial and tabular) and two state-of-the-art online expansion techniques (ATC and SDM) in a battery of domains including UAV mission planning scenarios, where the size of the state-action pairs ranged from 10^3 to more than 10^8.
- Presented adaptive resolution iFDD (ARiFDD) as a promising extension which can adaptively reshape the initial set of features, relaxing the dependency of iFDD on the initialization process. Empirical results exhibited the advantage of using online ARiFDD over online iFDD in the task of inverted pendulum with various set of initializations.
- Provided an overview of the related work for adaptive function approximators that covered the current existing gaps and placed iFDD in the context of that work.

6.1.3 Learning with Safe Exploration

Chapter 5 mainly addressed challenges (I) and partly (III) by integrating online RL methods with cooperative planners. In particular this chapter:

- Introduced cooperative learners as the fusion of online RL methods and cooperative planners.

- Empirically verified the advantage of cooperative learners over pure learning and pure planning methods in the pedagogical GridWorld domains and UAV Mission planning scenarios with about 9×10^9 state-action pairs. In particular cooperative planners boosted the performance of pure planners, while reduced the sample complexity and risk involved in pure learning methods.

- Extended the cooperative learners to support adaptive model approximation and stochastic risk models. These extensions facilitated better risk estimation and the capability to switch the underlying policy suggested by the planner. The resulting methods were shown to improve the sample complexity in a GridWorld domain and a UAV mission planning scenario.

- Provided a literature review both on existing cooperative planners for solving UAV mission planning problems, and RL methods aimed at realizing safety.

6.2 Future Work

This section describes potential extensions for which this thesis can be used as a stepping stone.

6.2.1 Solving Partially Observable MDPs (POMDPs)

A POMDP is defined as tuple $(\mathcal{S}, \mathcal{A}, \Omega, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \mathcal{O}^a_{s'o}, \gamma)$ where the definitions of $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$, and γ are identical to the discussion of MDPs covered in Chapter 2. Ω is the set of possible observations, and $\mathcal{O}^a_{s'o} : \mathcal{S} \times \mathcal{A} \times \Omega \to [0,1]$ provides the probability of observing o after taking action a and reaching state s. A POMDP can be cast as a equivalent to a belief MDP defined as tuple $(\mathbf{B}, \mathcal{A}, \Upsilon^a_{bb'}, \Psi^a_{bb'}, \gamma)$, where $\mathbf{B} \in [0,1]^{|\mathcal{S}|}$ is the set of all possible probability distributions (beliefs) over the state space, $\Upsilon^a_{bb'}$ is the probability of going from belief b to belief b' taking action a, and Ψ^a_b is the expected reward along the way. The derivation of $\Upsilon^a_{bb'}$ and Ψ^a_b from $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$, and $\mathcal{O}^a_{s'o}$, as presented in the work of Kaelbling *et al.* (1998), is as follows:

$$
\begin{aligned}
\Upsilon^a_{bb'} &= P(b'|b,a) = \sum_{o \in \Omega} P(b'|b,a,o)P(o|b,a), \\
P(b'|b,a,o) &= \begin{cases} 1 & SE(b,a,o) = b' \\ 0 & otherwise \end{cases}, \\
b'(s') &= P(s'|o,a,b) = \mathcal{O}^a_{s'o} \sum_{s \in \mathcal{S}} \mathcal{P}^a_{ss'} b(s),
\end{aligned}
$$

$$
\begin{aligned}
P(o|b,a) &= \sum_{s'\in\mathcal{S}} P(o|b,a,s')P(s'|b,a) = \sum_{s'\in\mathcal{S}} \mathcal{O}^a_{s'o} P(s'|b,a) \\
&= \sum_{s'\in\mathcal{S}} \mathcal{O}^a_{s'o} \sum_{s\in\mathcal{S}} \mathcal{P}^a_{ss'} b(s), \\
\Psi^a_b &= \sum_{s\in\mathcal{S}} b(s) \sum_{s'\in\mathcal{S}} \mathcal{R}^a_{ss'},
\end{aligned}
$$

where SE is the state-estimation function that has the new belief state b' as its output. Using this mapping, each POMDP can be translated into a continuous $|\mathcal{S}|$ dimensional MDP. Hence, the result of this thesis provides an attractive tool for dealing with POMDPs where all elements of the POMDP are known. It should be noted that the variation in the value function defined over the belief space is often more dramatic compared to the value function for the state space, due to the fact that in real-world domains the state of the agent often locally changes, while the change in the belief space can be arbitrary large. Furthermore, tackling POMDPs with unknown transition or observation model remains a challenge, as the agent cannot calculate its state in the new MDP formulation.

6.2.2 Model Learning

One area for future refinement is the noise sensitivity of online iFDD methods. Highly stochastic domains often lead to more frequent TD errors encouraging the feature discovery process to quickly add many features or add unnecessary features after the convergence of weights. While these factors do not affect the asymptotic guarantees, unnecessary features can increase computational load and slow down learning by refining the representation too quickly. Utilizing techniques that learn a model online, such as linear Dyna (Sutton et al., 2008; Yao et al., 2009) can hedge against this problem. Moreover, because these methods assume that the transition and reward models are linear in the feature space, in theory, iFDD can be used to approximate such models. In other words, iFDD can be used to provide a good fit to both the model and the value function. However, the source of the model error is no longer the Bellman error. It would be interesting to see if theoretical results can be extended to the case where the underlying function of interest can be sampled directly.[1]

6.2.3 Moving Beyond Binary Features

While the original Tile Coding framework used a binary (*i.e.,* piecewise constant) function to generate features, researchers have considered other functions, such as hat functions (Waldock & Carse, 2008) and Gaussians (Eldracher et al., 1997; Nguyen et al., 2005).

[1] Note that in the case of RL, the true value function can not be sampled directly.

Exploring how such functions can be used in the context of iFDD while maintaining the low per-time-step computational complexity would be an interesting extension to this thesis. For example, given continuous features on $[0, 1]$, a threshold can convert continuous features into binary values.

6.2.4 Exploration and Knownness

Another developing line of research in the RL community is efficient exploration with the aim of minimizing the regret (*i.e.,* the difference between the performance under optimal policy, and the actual performance obtained) during learning. While the R$_{max}$ type of algorithms have been shown both empirically and theoretically to yield promising results (Brafman & Tennenholtz, 2001), applying such methods to high-dimensional problems or continuous state spaces poses a challenge for RL practitioners. Recently the Multi-Resolution Exploration (MRE) technique (Nouri & Littman, 2008) has been shown to work well in low-dimensional continuous problems, yet we conjecture MRE does not scale to high-dimensional problems, because the core idea of MRE is very similar to the adaptive Tile Coding method (ATC) used to approximate the value function. Both techniques add full-dimensional features to capture the function of interest (knownness in case of MRE, and value function in case of ATC), which often have low coverage. Another interesting research direction is to use iFDD to approximate knownness rather than the value function. Empirical results suggest that the resulting method should demonstrate better sample complexity compared to MRE, because features can be defined in lower dimensional spaces.

6.2.5 Decentralized Cooperative Learning

Another alternative for dealing with large state-spaces in multi-agent mission planning domains is to decentralize the decision making process, so that each agent partially contributes to the overall plan formation (Choi et al., 2009). While cooperative learners in this thesis employed decentralized planners (*i.e.,* CBBA), the learning methods required a centralized process. It would be interesting to see how the learning methods can also be decentralized in order to have the whole framework running separately on each agent.

6.2.6 Cooperative Learning with Adaptive Function Approximators

This thesis introduced cooperative learning and iFDD as two different tools for dealing with large control problems, yet it did not consider the fusion of both approaches. While combining iFDD within cooperative learners with explicit policy forms should be simple, extending the fusion to RL methods with implicit policies is not immediately obvious. The main gap lies in formulating the knownness for continuous state spaces. Hence, the solution to Section 6.2.4 provides the stepping stone for this extension as well.

6.2.7 Relation of iFDD to Brain and Cognitive Science

While the importance of starting with a small representation for learning has been noted in the cognitive science literature (Elman, 1993), similar results have been obtained in the RL domain. Sherstov *et al.* demonstrated that reducing the generalization among states (*i.e.,* decaying the number of tilings while increasing the resolution of tiles) boosts the learning speed (2005). Ahmadi *et al.* showed that exposing features by a domain expert incrementally reduces the sample complexity (2007). Finally, Whiteson *et al.* empirically verified that agents starting with the learned representation required more samples compared to those starting with a simple representation which was refined through the learning process (2007). All such evidence motivates the use of an adaptive representation, starting with a broad coverage, while subsequently narrowing the coverage as time evolves. We think the theoretical analysis on the convergence rate of feature expansion techniques (*i.e.,* Corollary 3.2.8) provides an interesting lead on why, in general, adaptive representations and in particular a move from simple to complicated features is superior to static representations. In a broader sense, we hypothesize that children have limited representational power on early stages of life using features with broad coverage. Hence, finding a set of weights for a compact representation consumes less time, while broad features cover a lot of discrepancies early in the learning process. This hypothesis can explain why children are good at picking up second languages compared to adults (Bjorklund, 1997). As children grow older, the representation in their brain expands. This phenomenon encourages older humans to learn a task using a more complicated representation, therefore requiring more parameter tuning.

We would like to close this thesis by introducing the following experiment template which can highlight more of the connection between iFDD and human learning: select two sets of children A and B. The goal for all children is to solve a specific task using a set of binary features (*e.g.,* identify an object type). To excel in the task children have to consider a list of feature conjunctions in their decision making. Group A will be taught features and all necessary feature conjunctions (without being exposed to any samples), while group B will only be taught features but not any feature conjunctions. My hypothesis is that group A will require more training to master the task compared to group B.

Bibliography

Abbeel, Pieter and Ng, Andrew Y. Exploration and apprenticeship learning in reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 1–8, 2005.

Ahmadi, Mazda, Taylor, Matthew E., and Stone, Peter. IFSA: incremental feature-set augmentation for reinforcement learning tasks. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1–8, New York, NY, USA, 2007. ACM. ISBN 978-81-904262-7-5. doi: http://doi.acm.org/10.1145/1329125.1329351.

Albus, James S. A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61, 1971.

Alighanbari, M. Task assignment algorithms for teams of UAVs in dynamic environments. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 2004.

Alighanbari, M., Kuwata, Y., and How, J. P. Coordination and control of multiple UAVs with timing constraints and loitering. In *American Control Conference (ACC)*, volume 6, pp. 5311–5316 vol.6, 4-6 June 2003. ISBN 0743-1619. URL `http://acl.mit.edu/papers/FP13_41.pdf`.

Anderson, CW, Young, PM, Knight, MR Buehnerand JN, Bush, KA, and Hittle, DC. Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks. *IEEE Transactions on Neural Networks*, 18(4):993–1002, 2007.

Barto, Andrew and Duff, Michael. Monte carlo matrix inversion and reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 687–694. Morgan Kaufmann, 1994.

Beard, R., McLain, T., Goodrich, M., and Anderson, E. Coordinated target assignment and intercept for unmanned air vehicles. *IEEE Transactions on Robotics and Automation*, 18(6):911–922, 2002.

Bellman, Richard. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958.

Bellman, Richard. *Dynamic Programming*. Dover Publications, March 2003. ISBN 0486428095.

Berman, Spring, Halász, Ádám, Hsieh, M. Ani, and Kumar, Vijay. Optimized stochastic policies for task allocationin swarms of robots. *Transactions Rob.*, 25(4):927–937, 2009. ISSN 1552-3098. doi: http://dx.doi.org/10.1109/TRO.2009.2024997.

Bertsekas, D. and Tsitsiklis, J. Neuro-dynamic programming: an overview, 1995.

Bertsekas, D. and Tsitsiklis, J. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

Bethke, B. and How, J. Approximate Dynamic Programming Using Bellman Residual Elimination and Gaussian Process Regression. In *American Control Conference (ACC)*, St. Louis, MO, 2009.

Bhatnagar, Shalabh, Sutton, Richard S., Ghavamzadeh, Mohammad, and Lee, Mark. Incremental natural actor-critic algorithms. In Platt, John C., Koller, Daphne, Singer, Yoram, and Roweis, Sam T. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 105–112. MIT Press, 2007.

Bjorklund, D. The role of immaturity in human development. *Psychological bulletin*, 122 (2):153–169, 1997.

Bowling, M. and Veloso, M. Simultaneous adversarial multirobot learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, August 2003.

Bowling, Michael, Geramifard, Alborz, and Wingate, David. Sigma Point Policy Iteration. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pp. 379–386, Richland, SC, 2008. ISBN 978-0-9817381-0-9. URL `http://dl.acm.org/citation.cfm?id=1402439`.

Boyan, Justin A. Technical update: Least-squares temporal difference learning. *Journal of Machine Learning Research (JMLR)*, 49:233–246, 2002.

Bradtke, S. and Barto, A. Linear least-squares algorithms for temporal difference learning. *Journal of Machine Learning Research (JMLR)*, 22:33–57, 1996.

Brafman, Ronen I. and Tennenholtz, Moshe. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 3:213–231, 2001.

Buro, Michael. From simple features to sophisticated evaluation functions. In *Computers and Games, Proceedings of CG98, LNCS 1558*, pp. 126–145. Springer-Verlag, 1999.

Buşoniu, L., Babuška, R., De Schutter, B., and Ernst, D. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida, 2010.

Casal, A. *Reconfiguration Planning for Modular Self-Reconfigurable Robots*. PhD thesis, Stanford University, Stanford, CA, 2002.

Cassandras, C. and Li, W. A receding horizon approach for solving some cooperative control problems. In *IEEE Conference on Decision and Control (CDC)*, pp. 3760–3765, 2002.

Castanon, D. A. and Wohletz, J. M. Model predictive control for stochastic resource allocation. *IEEE Transactions on Automatic Control*, 54(8):1739 – 1750, 2009.

Choi, H.-L., Brunet, L., and How, J. P. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, August 2009. ISSN 1552-3098. doi: 10.1109/TRO.2009.2022423. URL `http://acl.mit.edu/papers/email.html`.

Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., and Stein, Clifford. *Introduction to Algorithms*. The MIT Press, 2nd revised edition edition, September 2001. ISBN 0262531968.

Eldracher, Martin, Staller, Alexander, and Pompl, René. Adaptive encoding strongly improves function approximation with CMAC. *Neural Comput.*, 9(2):403–417, 1997. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/neco.1997.9.2.403.

Elman, J.L. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993. doi: 10.1016/0010-0277(93)90058-4.

Farahmand, Amir Massoud, Ghavamzadeh, Mohammad, Szepesvári, Csaba, and Mannor, Shie. Regularized policy iteration. In Koller, Daphne, Schuurmans, Dale, Bengio, Yoshua, and Bottou, Léon (eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 441–448. MIT Press, 2008.

Finch, Tony. Incremental calculation of weighted mean and variance, 2009.

Finlayson, Mark A. and Winston, Patrick H. Analogical Retrieval via Intermediate Features: The Goldilocks Hypothesis. Research Report 2006-071, CSAIL, 2006.

Gallager, R.G. *Discrete stochastic processes.* Kluwer international series in engineering and computer science. Kluwer Academic Publishers, 2nd edition, 2009.

Geibel, Peter and Wysotzki, Fritz. Risk-sensitive reinforcement learning applied to chance constrained control. *Journal of Artificial Intelligence and Research (JAIR)*, 24, 2005.

Geramifard, A., Redding, J., Joseph, J., and How, J. P. Model Estimation Within Planning and Learning. In *Workshop on Planning and Acting with Uncertain Models, ICML, Bellevue, WA, USA*, June 2011a.

Geramifard, A., Redding, J., Roy, N., and How, J. P. UAV Cooperative Control with Stochastic Risk Models. In *American Control Conference (ACC)*, pp. 3393 – 3398, June 2011b. URL `http://people.csail.mit.edu/agf/Files/11ACC-iCCARisk.pdf`.

Geramifard, Alborz, Bowling, Michael, and Sutton, Richard S. Incremental least-square temporal difference learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 356–361, 2006.

Geramifard, Alborz, Bowling, Michael, Zinkevich, Martin, and Sutton, Richard. iLSTD: Eligibility traces and convergence analysis. In Schölkopf, B., Platt, J., and Hoffman, T. (eds.), *Advances in Neural Information Processing Systems (NIPS).* MIT Press, Cambridge, MA, 2007.

Geramifard, Alborz, Doshi, Finale, Redding, Joshua, Roy, Nicholas, and How, Jonathan. Online discovery of feature dependencies. In Getoor, Lise and Scheffer, Tobias (eds.), *International Conference on Machine Learning (ICML)*, pp. 881–888, New York, NY, USA, June 2011c. ACM. ISBN 978-1-4503-0619-5. URL `http://people.csail.mit.edu/agf/Files/11ICML-iFDD.pdf`.

Girgin, Sertan and Preux, Philippe. Feature Discovery in Reinforcement Learning using Genetic Programming. Research Report RR-6358, INRIA, 2007.

Girgin, Sertan and Preux, Philippe. Basis function construction in reinforcement learning using cascade-correlation learning architecture. In *International Conference on Machine Learning and Applications (ICMLA)*, pp. 75–82, Washington, DC, USA, 2008. IEEE

Computer Society. ISBN 978-0-7695-3495-4. doi: http://dx.doi.org/10.1109/ICMLA.2008.24.

Goodman, N. D., Tenenbaum, J. B., Griffiths, T. L., and Feldman, J. Compositionality in rational analysis: Grammar-based induction for concept learning. In Oaksford, M. and Chater, N. (eds.), *The probabilistic mind: Prospects for Bayesian cognitive science*, 2008.

Hans, Alexander, Schneegaß, Daniel, Schäfer, Anton Maximilian, and Udluft, Steffen. Safe exploration for reinforcement learning. In *Proceedings of the 16th European Symposium on Artificial Neural Networks*, 2008.

Heger, Matthias. Consideration of risk and reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 105–111, 1994.

Justin Boyan, Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G., Touretzky, D.S., and Lee, T.K. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 369–376, Cambridge, MA, 1995. The MIT Press.

Kaelbling, Leslie Pack, Littman, Michael L., and Cassandra, Anthony R. Planning and acting in partially observable stochastic domains. *Journal of Artificial Intelligence and Research (JAIR)*, 101:99–134, 1998.

Kalai, Adam Tauman, Samorodnitsky, Alex, and Teng, Shang-Hua. Learning and smoothed analysis. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Atlanta, Georgia, USA*, pp. 395–404, 2009.

Knox, W. Bradley and Stone, Peter. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2010.

Kohl, Nate and Stone, Peter. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2619–2624, 2004.

Kolter, J. Zico and Ng, Andrew Y. Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning (ICML)*, pp. 521–528, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: http://doi.acm.org/10.1145/1553374.1553442.

Kroon, Mark and Whiteson, Shimon. Automatic feature selection for model-based reinforcement learning in factored MDPs. In *International Conference on Machine Learning and Applications (ICMLA)*, pp. 324–330, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3926-3. doi: http://dx.doi.org/10.1109/ICMLA.2009.71.

Lagoudakis, Michail G. and Parr, Ronald. Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149, 2003.

Lampton, Amanda and Valasek, John. Multiresolution state-space discretization method for q-learning. In *American Control Conference (ACC)*, pp. 1646–1651, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-4523-3.

Leng, Jinsong, Fyfe, Colin, and Jain, Lakhmi. Reinforcement learning of competitive skills with soccer agents. In *KES'07/WIRN'07: Proceedings of the 11th international conference, KES 2007 and XVII Italian workshop on neural networks conference on Knowledge-based intelligent information and engineering systems*, pp. 572–579, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74817-5.

Li, Lihong, Williams, Jason D., and Balakrishnan, Suhrid. Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection. In *New York Academy of Sciences Symposium on Machine Learning*, 2009.

Lin, Stephen and Wright, Robert. Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning. In *AAAI Workshops*, 2010.

Mahadevan, Sridhar. Representation policy iteration. *International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

Mahadevan, Sridhar, Maggioni, Mauro, and Guestrin, Carlos. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research (JMLR)*, 8:2007, 2006.

Mannor, Shie and Precup, Doina. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 449–456. ACM Press, 2006.

Melo, Francisco S., Meyn, Sean P., and Ribeiro, M. Isabel. An analysis of reinforcement learning with function approximation. In *International Conference on Machine Learning (ICML)*, pp. 664–671, 2008.

Mihatsch, Oliver and Neuneier, Ralph. Risk-sensitive reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 49(2-3):267–290, 2002. ISSN 0885-6125. doi: 10.1023/A:1017940631555.

Moore, Andrew W. and Atkeson, Christopher G. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Journal of Machine Learning Research (JMLR)*, 21:199–233, December 1995. ISSN 0885-6125. doi: 10.1023/A:1022656217772. URL `http://dl.acm.org/citation.cfm?id=223124.223125`.

Munos, R. and Szepesvári, C. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research (JMLR)*, 1:815–857, 2008.

Munos, Rémi. Error bounds for approximate policy iteration. In Fawcett, Tom and Mishra, Nina (eds.), *International Conference on Machine Learning (ICML)*, pp. 560–567. AAAI Press, 2003. ISBN 1-57735-189-4.

Munos, Rémi and Moore, Andrew. Variable resolution discretization in optimal control. *Journal of Machine Learning Research (JMLR)*, 49(2-3):291–323, 2002. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1017992615625.

Nguyen, M. N., Shi, D., and Quek, C. Self-organizing gaussian fuzzy CMAC with truth value restriction. In *ICITA '05: Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume 2*, pp. 185–190, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2316-1. doi: http://dx.doi.org/10.1109/ICITA.2005.250.

Nouri, Ali and Littman, Michael L. Multi-resolution exploration in continuous spaces. In Koller, Daphne, Schuurmans, Dale, Bengio, Yoshua, and Bottou, Léon (eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 1209–1216. MIT Press, 2008.

Ormoneit, Dirk and Sen, Śaunak. Kernel-based reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 49(2-3):161–178, 2002. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1017928328829.

Parr, Ronald, Painter-Wakefield, Christopher, Li, Lihong, and Littman, Michael. Analyzing feature generation for value-function approximation. In *International Conference on Machine Learning (ICML)*, pp. 737–744, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: http://doi.acm.org/10.1145/1273496.1273589.

Peters, Jan and Schaal, Stefan. Natural actor-critic. *Neurocomput.*, 71:1180–1190, March 2008. ISSN 0925-2312. doi: 10.1016/j.neucom.2007.11.026.

Petrik, Marek, Taylor, Gavin, Parr, Ronald, and Zilberstein, Shlomo. Feature selection using regularization in approximate linear programs for Markov decision processes. In *International Conference on Machine Learning (ICML)*, 2010.

Ratitch, Bohdana and Precup, Doina. Sparse distributed memories for on-line value-based reinforcement learning. In *European Conference on Machine Learning (ECML)*, pp. 347–358, 2004.

Redding, J., Geramifard, A., Choi, H.-L., and How, J. P. Actor-Critic Policy Learning in Cooperative Planning. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2010a. (AIAA-2010-7586).

Redding, J., Geramifard, A., Undurti, A., Choi, H., and How, J. An intelligent cooperative control architecture. In *American Control Conference (ACC)*, pp. 57–62, Baltimore, MD, July 2010b. URL `http://people.csail.mit.edu/agf/Files/10ACC-iCCA.pdf`.

Reisinger, Joseph, Stone, Peter, and Miikkulainen, Risto. Online kernel selection for Bayesian reinforcement learning. In *International Conference on Machine Learning (ICML)*, July 2008.

Reynolds, Stuart I. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *International Conference on Machine Learning (ICML)*, pp. 783–790. Morgan Kaufmann, 2000.

Rivest, François and Precup, Doina. Combining TD-learning with cascade-correlation networks. In *International Conference on Machine Learning (ICML)*, pp. 632–639. AAAI Press, 2003.

Rummery, G. A. and Niranjan, M. Online Q-learning using connectionist systems (tech. rep. no. cued/f-infeng/tr 166). *Cambridge University Engineering Department*, 1994.

Ryan, Allison, Zennaro, Marco, Howell, Adam, Sengupta, Raja, and Hedrick, J. Karl. An overview of emerging results in cooperative UAV control. In *IEEE Conference on Decision and Control (CDC)*, pp. 602–607, 2004.

Saligrama, V. and Castañón, D.A. Reliable distributed estimation with intermittent communications. In *IEEE Conference on Decision and Control (CDC)*, pp. 6763–6768, Dec. 2006. doi: 10.1109/CDC.2006.377646.

Sanner, Scott. Practical linear value approximation techniques for first-order MDPs. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2006a.

Sanner, Scott. Online feature discovery in relational reinforcement learning. In *Proceedings of the Open Problems in Statistical Relational Learning Workshop*, 2006b.

Scherrer, Bruno. Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. In *International Conference on Machine Learning (ICML)*, 2010.

Sherstov, Alexander A. and Stone, Peter. Function approximation via tile coding: Automating parameter choice. In Zucker, J.-D. and Saitta, I. (eds.), *Symposium on Abstraction, Reformulation, and Approximation (SARA)*, volume 3607 of *Lecture Notes in Artificial Intelligence*, pp. 194–205. Springer Verlag, Berlin, 2005.

Silver, David, Sutton, Richard S., and Müller, Martin. Sample-based learning and search with permanent and transient memories. In *International Conference on Machine Learning (ICML)*, pp. 968–975, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: http://doi.acm.org/10.1145/1390156.1390278.

Singh, Amarjeet, Krause, Andreas, and Kaiser, William. Nonmyopic adaptive informative path planning for multiple robots. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

Stone, Peter and Sutton, Richard S. Scaling reinforcement learning toward RoboCup soccer. In *International Conference on Machine Learning (ICML)*, pp. 537–544. Morgan Kaufmann, San Francisco, CA, 2001.

Stone, Peter, Sutton, Richard S., and Kuhlmann, Gregory. Reinforcement learning for RoboCup-soccer keepaway. *International Society for Adaptive Behavior*, 13(3):165–188, 2005.

Sturtevant, Nathan R. and White, Adam M. Feature construction for reinforcement learning in hearts. In *5th International Conference on Computers and Games*, 2006.

Sutton, R. S., Szepesvari, Cs, Geramifard, A., and Bowling, M. Dyna-style planning with linear function approximation and prioritized sweeping. In *International Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 528–536, 2008.

Sutton, Richard S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1038–1044. The MIT Press, 1996.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

Sutton, Richard S. and Whitehead, Steven D. Online learning with random representations. In *International Conference on Machine Learning (ICML)*, pp. 314–321. Morgan Kaufmann, 1993.

Sutton, Richard S., McAllester, David, Singh, Satinder, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1057–1063. MIT Press, 1999.

Sutton, Richard S., Maei, Hamid Reza, Precup, Doina, Bhatnagar, Shalabh, Silver, David, Szepesvári, Csaba, and Wiewiora, Eric. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning (ICML)*, pp. 993–1000, New York, NY, USA, 2009. ACM.

Szepesvári, Csaba. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

Szepesvári, Csaba and Munos, Rémi. Finite time bounds for sampling based fitted value iteration. In *International Conference on Machine Learning (ICML)*, pp. 881–886, 2005.

Taylor, Gavin and Parr, Ronald. Kernelized value function approximation for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 1017–1024, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: http://doi.acm.org/10.1145/1553374.1553504. URL `http://doi.acm.org/10.1145/1553374.1553504`.

Tedrake, Russ, Zhang, Teresa Weirui, and Seung, H. Sebastian. Stochastic policy gradient reinforcement learning on a simple 3D Biped. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pp. 2849–2854, 2004.

Tesauro, Gerald. Programming backgammon using self-teaching neural nets. *Journal of Artificial Intelligence and Research (JAIR)*, 134(1-2):181–199, 2002. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(01)00110-2.

Torrey, Lisa, Walker, Trevor, Shavlik, Jude, and Maclin, Richard. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *International Conference on Machine Learning (ICML)*, pp. 412–424, 2005.

Tsitsiklis, John N. and Van Roy, Benjamin. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

Ullman, Shimon, Vidal-Naquet, Michel, and Sali, Erez. Visual features of intermediate complexity and their use in classification. *Nature neuroscience*, 5(7):682–687, July 2002. ISSN 1097-6256. doi: 10.1038/nn870.

Uther, William T. B. and Veloso, Manuela M. Tree based discretization for continuous state space reinforcement learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 769–774, 1998.

Valenti, Mario J. *Approximate Dynamic Programming with Applications in Multi-Agent Systems*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge MA, May 2007. URL `http://acl.mit.edu/papers/Valenti_PhD_2007.pdf`.

Valiant, Leslie G. Evolvability. *J. ACM*, 56:3:1–3:21, February 2009. ISSN 0004-5411.

Waldock, A. and Carse, B. Fuzzy Q-learning with an adaptive representation. In *Fuzzy Systems, IEEE World Congress on Computational Intelligence*, pp. 720–725, 2008.

Wang, X., Yadav, V., and Balakrishnan, S. N. Cooperative UAV formation flying with obstacle/collision avoidance. *Control Systems Technology, IEEE Transactions on*, 15(4): 672–679, 2007. doi: 10.1109/TCST.2007.899191.

Wang, Xiaohua, Yadav, Vivek, and Balakrishnan, S.N. Cooperative UAV formation flying with stochastic obstacle avoidance. *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2005.

Watkins, C. J. Q-learning. *Journal of Machine Learning Research (JMLR)*, 8(3):279–292, 1992.

Whiteson, Shimon and Stone, Peter. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 7:877–917, May 2006.

Whiteson, Shimon, Taylor, Matthew E., and Stone, Peter. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007.

Wu, Jia-hong and Givan, Robert. Feature-discovering approximate value iteration methods. In *Symposium on Abstraction, Reformulation, and Approximation (SARA)*, 2005.

Wu, Jia-Hong and Givan, Robert. Discovering relational domain features for probabilistic planning. In Boddy, Mark S., Fox, Maria, and Thiébaux, Sylvie (eds.), *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 344–351. AAAI, 2007. ISBN 978-1-57735-344-7.

Xu, L. and Ozguner, U. Battle management for unmanned aerial vehicles. In *IEEE Conference on Decision and Control (CDC)*, volume 4, pp. 3585–3590. 9-12 Dec. 2003. ISBN 0191-2216.

Yao, Hengshuai, Sutton, Rich, Bhatnagar, Shalabh, Diao, Dongcui, and Szepesvari, Csaba. Multi-Step Dyna Planning for Policy Evaluation and Control. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 2187–2195. 2009.

Zhang, Wei and Dietterich, Thomas G. High-Performance Job-Shop Scheduling With A Time-Delay TD(λ) Network. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1024–1030. MIT Press, 1995.