Alborz Geramifard
#532 - 8801 - 111St.
Edmonton, AB
Canada, T6G2X5

Date: ____________________

Think? Why think! We have computers to do that for us.

— Jean Rostand

University of Alberta

INCREMENTAL LEAST-SQUARES TEMPORAL DIFFERENCE LEARNING

by

Alborz Geramifard

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2007

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Incremental Least-Squares Temporal Difference Learning** submitted by Alborz Geramifard in partial fulfillment of the requirements for the degree of **Master of Science**.

____

Richard S. Sutton

____

Michael Bowling

____

Dale Schuurmans

____

Petr Musilek

Date: ___________________

To my beloved parents

Abstract

Sequential decision making is a challenging problem for the artificial intelligence community. It can be modeled as an *agent* interacting with an *environment* according to its policy. *Policy iteration* methods are a popular approach involving the interleaving of two stages: *policy evaluation* to compute the desirability of each state with respect to the policy, and *policy improvement* to improve the current policy with respect to the state values. The effectiveness of this approach is highly dependent on the effectiveness of policy evaluation, which is the focus of this dissertation. The per time step complexity of traditional methods like temporal difference learning (TD) are sublinear in the number of features. Thus, they can be scaled to large environments, however they use training data relatively inefficiently and so require a large number of sample interactions. The least-squares TD (LSTD) method addresses the data inefficiency of TD by using the sum of the TD updates on all past experiences. This makes LSTD a formidable algorithm for tackling problems where data is limited or expensive to gather. However, the computational cost of LSTD cripples its applicability in most large environments.We introduce an incremental version of the LSTD method, called iLSTD, for online policy evaluation in large problems.

On each time step, iLSTD uses the sum TD update vector in a gradient fashion by selecting and descending in a limited set of dimensions. We show that if a sparse feature representation is being used, the iLSTD algorithm's per time step complexity is linear in the number of features whereas for LSTD, it is quadratic. This allows iLSTD to scale up to large environments with many features where LSTD cannot be applied. On the other hand, because iLSTD takes advantage of all data on each time step, it requires far less data than the TD method. Empirical results in the Boyan chain and mountain car environments shows the superiority of iLSTD with respect to TD and the speed advantage of iLSTD with respect to LSTD. We also extend iLSTD with eligibility traces, resulting in iLSTD(λ), and show that the additional computation does not change the linear per time step complexity. Additionally, we investigate the performance and convergence properties of iLSTD with different dimension selection mechanisms. Finally, we discuss the limitations of this study.

Preface

A long time ago, there was a king ruling over the vast land of Agencia. He was old but generous, wise, and kind, and all of his citizens were happy to have such a king. The country was wealthy because of its business relationship with the neighbor country: Envirocia. Desiring to extend the relationship between the two countries, the king of Envirocia sent an ambassador to Agencia in order to open the discussion about new domains of cooperation. After a few months, on a lovely morning, the Vazir of Agencia saw the ambassador of Envirocia coming down the stairs of the palace with a gloomy face.

Being asked by the Vazir, the ambassador replied sadly:

"*Your king is generous, wise, and kind, but he is forgetful. Sometimes he makes decisions which are not wise in light of our earlier conversations.*"

The Vazir started to think about this problem. It was true that sometimes the king would forget some details of the previous conversations, and make unwise decisions accidentally. However, he was a bit old and no one should expect him to remember all previous discussions. After consulting on this issue with the king, it was decided to assign a consultant for each subject and before any decision, the king would negotiate with all of the consultants and make an informed decision.

After a while, consultants of agriculture, politics, etc. were elected, and the course of meetings was adopted accordingly. Everything was going well for a few months. One day the Vazir saw the same ambassador. Being puzzled by his unhappy face again, he asked the ambassador about the recent flow of meetings.

“*I should admit that recently the king makes decent judgments, but as the number of our meetings increases, it takes a while for me to hear back from the majesty, since he reviews all previous discussions with his consultants and states his final decision only after much consultation ...*”, the ambassador replied with a tired face.

This story highlights the main drawbacks of two alternative approaches of online policy evaluation in the reinforcement learning framework. The per time step complexity of TD is low, but it does not use all past experiences efficiently. On the other hand, the LSTD method considers all of the previous interactions. Doing so, it requires far less data, but its expensive per time step complexity makes it inapplicable for large problems. The iLSTD method, introduced in this thesis, achieves the best of both worlds: it makes use of all data while maintaining a linear per time step complexity.

Acknowledgements

I would like to thank both of my supervisors, Richard Sutton and Michael Bowling, who kindly guided me through every step of my research and donated a great deal of their precious time. Rich always had a good grasp of the big picture of this work, which was a blessing when you want to travel a few miles in the research field. On the other hand, whenever I felt that I was confused about the next step, Mike was always there for me with a bag full of new interesting ideas. He devoted lots of his time in order to polish different aspects of this research. Without both of them, I could not have made it this far.

I want to recognize the insightful comments of Dale Schuurmans and his permission for using the Salient cluster for my experiments. I would also like to thank Mohammad Ghavamzadeh, Amir massoud Farahmand, Martin Zinkevich, Mark Ring, Dan Lizotte and all other members of the RLAI lab at the University of Alberta for proofreading my thesis and helping me through my research.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

x	Scaler (any lowercase variable)	N/A
$\mathbf{x}$	Vector (any bolded lowercase variable)	N/A
$\mathbf{X}$	Matrix (any bolded uppercase variable)	N/A
a	Action	7
$\mathcal{A}$	Action space	7
s	State	7
$\mathcal{S}$	State space	7
r	Reward	7
R	Return	7
π	Policy	7
γ	Discount factor	7
V	Value function	8
$\boldsymbol{\theta}$	Weight vector	9
$\boldsymbol{\phi}$	Feature vector	9
$\boldsymbol{\phi}_t$	Feature vector corresponding to s_t	9
n	Number of features	10
k	Number of active features	10
α	Learning rate	10
δ	TD error	10
$\mathbf{u}$	Temporal difference update	11
R^n	n-step return	12
R^λ	λ-return	12
λ	Eligibility trace factor	12
$\mathbf{z}$	Eligibility trace vector	13
l	Maximum number of active eligibility traces	13
$\boldsymbol{\mu}$	Sum of temporal difference updates	14
$\mathbf{A}$	Transition model	15
$\mathbf{b}$	Reward model	15
$\tilde{\mathbf{A}}$	Approximation of $\mathbf{A}^{-1}$	17
ω	Initial regularization factor for $\tilde{\mathbf{A}}$	17
$\mathbf{I}$	Identity matrix	17
m	Number of descents per time step in iLSTD	21
$\boldsymbol{e}_i$	Identity vector	21
ϵ	Probability for random selection in ϵ-greedy mechanism	49
τ	Temperature parameter used for Boltzmann distribution	50
ψ	Lower bound probability for Boltzmann distribution	50

Chapter 1

Introduction

Humans are smart, demanding, and lazy. They have always sough to invent tools. Long ago, these tools were only meant to help them to perform their tasks easier and faster. Demanding even more free time, they started to create computer programs to perform their jobs related to military, medicine, education, business, *etc.* These programs or agents may be a part of a vision system recognizing vehicles along a road, or software installed on a physical robot in a rehabilitation center helping a patient recover after an accident. This new research field is called *artificial intelligence* (AI), and has raised many new challenges. These programs should be able to accomplish complicated tasks, be flexible to resolve new situations reasonably, and make fast decisions to be applicable in real-time domains. One significant characteristic that all problems in the AI area share, is experience. A popular subfield of AI, called machine learning, is based on the idea of using experience through "learning." ML has been categorized into three main problems:

- *Supervised learning*: In this problem, the agent is provided with labeled examples, with which it finds a solution to map a new unlabeled example to its label (*classification*). For example, a vision system may be trained with a large set of images. Each image contains one specific object and tagged with the name of the object. After training the system, it is able to recognize the same objects in the new images. Support vector machines (SVM) and decision trees [*e.g.*, see Alpaydin, 2004] are instances of methods that are used to solve supervised learning problems.

- *Unsupervised learning*: There are cases where the examples are not labeled before being given to the agent. The agent must assign the labels based on some notion of

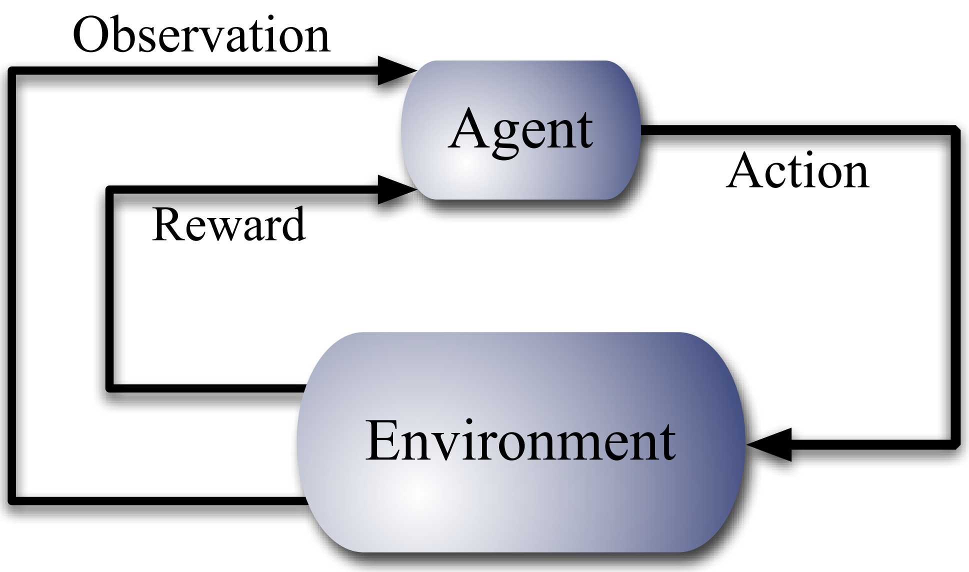

Figure 1.1: Reinforcement Learning Framework: At each time step, the agent select an action and send it to the environment and receives the resulting observation and reward.

similarity, which is called *clustering*. For example, given a bunch of points along a line and the number of classes, a program may cluster each point into a class. Kohonen self-organizing maps (KSOM) [Kohonen, 2001] and K-means clustering [*e.g.*, see Jain *et al.*, 1999] are examples of unsupervised learning methods.

- *Reinforcement learning (RL)*: As shown in Figure 1.1, in this problem the agent interacts with an *environment*, sends an *action*, and receives a resulting *observation* and a *reward signal*. Each action can change the state of the environment, while the reward signal is a score stating how well the agent behaved in the last interaction. The goal of the agent is to maximize some measure of long-term reward which we call the *goal function*. For example, the game of chess can be mapped into the RL framework: the position of all pieces on the board forms the current state of the environment, possible moves for all pieces of the board define the action set, and the reward signal can be +1 for winning, -1 for losing, and 0 for a tie. For all other non-finishing moves during the game the reward is also 0. Notice that being greedy with respect to the reward function on each time step will not necessarily maximize the goal function, which makes RL problems interesting and distinct from the supervised learning problems. The effect of some early actions might not be visible only till the very end. Q-Learning and Sarsa [*e.g.*, see Sutton and Barto, 1998] are examples of popular RL algorithms.

This thesis focuses on the reinforcement learning problem. In general, the state of the environment and the reward signal are dependent on all previous observations and actions, but in the special case of Markov decision processes (MDPs),[1] they only depend on the last observed state and executed action (*Markov property*). On each time-step, the agent chooses an action according to its *policy*, which is a mapping from state-action pairs to probabilities. Searching for a policy that maximizes the goal function is the main objective. This is achieved commonly by RL methods through two phases: the agent evaluates the values of state-action pairs with respect to its current policy (*policy evaluation*), and then improves the policy on the next iteration (*policy improvement*). This loop continues until no further improvement can be made to the policy. Policy evaluation determines the desirability of each state given the current policy (*value function*) by evaluating each state. However in real applications the number of states can be large, and the value function is usually approximated with a function approximator which can be either non-linear (*e.g.*, neural networks [Kohonen, 1988]) or linear (*e.g.*, coarse coding [Sutton and Barto, 1998]). Policy evaluation is similar to the way humans cogitate about their actions. For example, people do not get too close to the edge of a cliff. This is not because standing close to the edge is painful by itself (*i.e.*, immediate reward), but it is due to the higher probability of falling down from the cliff (*i.e.*, state value). This dissertation addresses the policy evaluation problem with linear function approximators, where each state is represented by a feature vector and the value function is considered to be a linear combination of the features.

1.1 Motivation

Temporal difference (TD) learning [Sutton, 1988] is a traditional method of solving MDPs. This method has a sublinear running time in the number of features and can be scaled to large problems because of its low computational complexity, although TD is forgetful

[1]For the reminder of the thesis, we will refer to the *observation* as *state* in MDPs, since it uniquely defines the environment's state.

and consequently it uses the data inefficiently. Bradtke and Barto [1996] extended the TD method by introducing least-squares TD algorithm (LSTD), which finds the solution to the zero sum TD updates for all past experiences. However, when this method is applied to large problems, its running time becomes a critical obstacle because LSTD is quadratic in the number of features. This fact motivated us to search for an algorithm which is less computationally expensive, thus scalable to larger problems, but at the same time uses the data more efficiently than TD.

1.2 The New Approach

We introduce iLSTD[2] as a new TD based method, which achieves the best of the TD and LSTD worlds: its per time step complexity is linear in the number of features, and it makes use of all data on each time step. These two properties lead to an algorithm which is dramatically faster than LSTD in large environments and performs better than TD. The main idea behind the iLSTD method is that it evades the computation for solving the zero sum TD update ($n \times n$ matrix inversion) on each time step, and instead it uses the sum TD update vector as a gradient to descend in a limited number of dimensions.[3] After each descent, it updates the sum TD update vector. By taking advantage of a sparse feature representation, this update costs only linear computation in the number of features.

1.3 Contributions

- Introducing the incremental version of the LSTD algorithm, iLSTD, as a new policy evaluation method.
- Extending the iLSTD method with eligibility traces: iLSTD(λ)
- Proving the linear running time of iLSTD(λ) with a sparse feature representation in the general case.
- Comparing the experimental results of TD(λ), iLSTD(λ), and LSTD(λ) methods in Boyan chain and mountain car environments.

[2]Some of the material in this thesis has been published previously [Geramifard *et al.*, 2006, Geramifard *et al.*, 2007].

[3]In this thesis, we use *dimension* and *feature* terms interchangeably.

- Studying the performance and convergence properties of the iLSTD algorithm with different dimension selection rules.

1.4 Overview

We first begin by reviewing the RL framework and the MDP class of problems in more detail in Chapter 2. TD methods [Sutton, 1988] will be introduced as traditional algorithms of solving online policy evaluation. After explaining how they were extended to large problems using function approximators, the merits and limitations of TD methods will be covered. Then, eligibility traces are introduced as a technique for improving TD methods. We explain how these ideas which led to more advanced methods such as experience replay [Lin, 1993], least-squares TD (LSTD) [Bradtke and Barto, 1996] and LSTD(λ) [Boyan, 1999], and analyze the LSTD methods in more detail. In Chapter 3, we present incremental LSTD (iLSTD) and study its convergence. We also prove that with any sparse feature representation, the running time of this method is linear in the number of features. This property together with the usage of all past experience at any moment, make iLSTD a capable algorithm which can be scaled to large problems with running time restrictions, and at the same time, its performance exceeds TD's significantly. In Chapter 4, we investigate the use of eligibility traces with the iLSTD algorithm and show that this extension does not hurt the per time step complexity of the algorithm. Various feature selection mechanisms that can be combined with iLSTD(λ) method are introduced in Chapter 5 and their performance and convergence properties are examined. We finally conclude the thesis in Chapter 6 by highlighting the main contributions of this study. We also discuss the extensions and limitations of this work.

Chapter 2

Background

In this chapter, we present background work in reinforcement learning (RL), although for a more comprehensive introduction, we encourage the reader to look at more detailed text books [Sutton and Barto, 1998, Bertsekas and Tsitsiklis, 1995]. First we present the RL framework and the Markov Decision Processes (MDP) formalism. We also review temporal difference learning (TD) as a traditional method for solving MDPs, and describe how it can be applied to problems with large state spaces using linear function approximation. After a brief review of tile coding and its use as a common linear function approximator, we move on and present eligibility traces and least-squares methods that are the building blocks of more data efficient MDP solutions such as TD(λ), LSTD(λ), and experience replay. Finally, the TD and LSTD methods are presented in more detail and their positive and negative points are discussed serving as a basis for the new algorithms presented in the following chapters.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a class of learning problems in which an agent interacts with an unfamiliar, dynamic and stochastic environment with the goal of minimizing some measure of its long-term performance [Sutton and Barto, 1998]. This interaction is conventionally modeled as an MDP, or if the environment state is not always completely observable, as a partially observable MDP (POMDP) [Jaakkola *et al.*, 1995]. In this thesis we restrict our attention to the discrete-time MDP setting. An MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma \rangle$, in which $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}^a_{ss'}$ is the

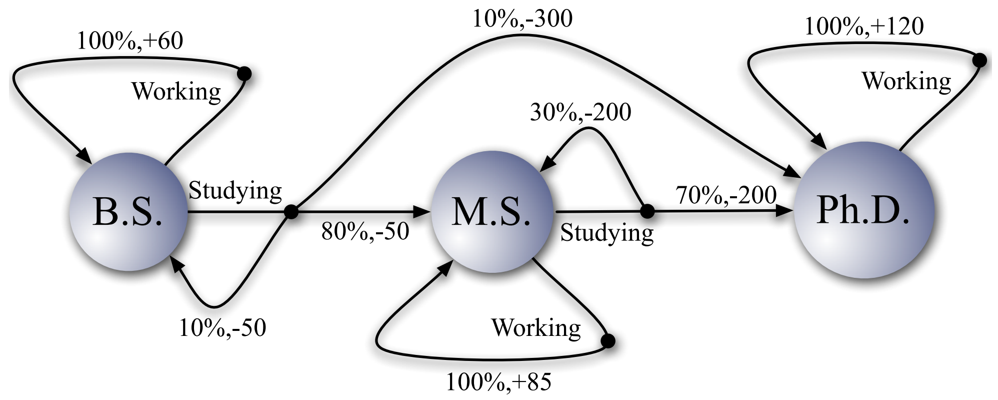

Figure 2.1: A simple MDP: The student starts with a bachelor degree with two actions of studying and working. Numbers on arrows represents the probability of transition and consequent reward respectively.

probability of ending in state s' by taking action a from state s, $\mathcal{R}^a_{ss'}$ is the reward value corresponding to that transition, and $\gamma \in [0, 1]$ is a discount factor. On each time step, the agent selects an action, which changes the state of the environment and receives the consequent state and reward. If this loop continues forever the task is called *continuing*, if it stops after a finite number of time steps, it is called an *episodic* task.

Figure 2.1 depicts a simple MDP modeling a graduate student. Each large circle stands for the state node which is an educational degree and each small solid circle represents an action. Arrows indicate the probability of transitions ($\mathcal{P}^a_{ss'}$) together with the consequent rewards ($\mathcal{R}^a_{ss'}$). The outcome of each action is dependent on the state it was executed from. For example, the higher educational level you achieve, the more benefit you earn for working. An agent's trajectory, beginning in state s_0, is a sequence $s_0, a_1, r_1, s_1, a_2, r_2, s_2, ...$, where the agent takes action a_1, receives reward r_1, and ends up in state s_1, and so on. The *return* at time t, R_t, is defined as the sum of future discounted rewards from time t:

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}.$$

The goal of the agent is to take actions in order to maximize the expected return. The agent selects actions according to its policy, π. In general, the policy is defined as a function $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, which gives the probability of selecting each action given the current state.

The value of a state is defined as the expected sum of discounted rewards when following the policy π:

$$V^{\pi}(s) \;=\; E\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s, \pi\right].$$

The value function can be written recursively using the well-known *Bellman equation*:

$$\begin{aligned} V^{\pi}(s) &= E\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, \pi\right] \qquad (2.1) \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right]. \end{aligned}$$

The goal of researchers in reinforcement learning is to build agents that learn to maximize the expected return, also known as the *optimal policy*. A wide range of MDP solvers accomplish this task through *policy iteration* which consist of two phases: *policy evaluation* in which the state-action pairs are evaluated according to the current policy, and *policy improvement* which improves the current policy according to the state-action values. This dissertation addresses the policy evaluation problem, and unless specified a fixed policy is assumed to be given. Note that, with a discrete state and action space, if prior knowledge about the $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ is provided (*i.e.* the agent knows the model of the environment), V^{π} has a closed form solution. The *value iteration* method (also known as *dynamic programming*) solves the MDP problem using its model to perform policy evaluation [Sutton and Barto, 1998]. In real applications, usually the model is not known, which is the focus of this research.

There are many ways to categorize RL methods. One distinction is between online and offline algorithms. In online learning, the agent interacts with the environment and learns at the same time. Thus, it holds the solution on every time step. If the environment puts any constraints on the interaction speed, the agent has a limited amount of time per interaction for learning and decision-making. In offline algorithms, the learning phase happens after a certain number of interactions. Doing so, the agent does not have any time restrictions on learning, but, at the same time, it does not have the solution during interaction. For example, in batch TD method, all of the TD updates are saved during the interaction, and afterwards, state values are updated based on the sum of TD updates.

2.2 Function Approximation

In many realistic reinforcement learning problems, the number of states is large or infinite.[1] In such cases, the memory required to hold large tables becomes a critical issue. In addition, learning the value of each individual state requires many visits to that state. With a large number of states, a great deal of experience is necessary before all states are visited even a small number of times. The solution to this problem is *generalization*, where the values of "similar" states are kept similar. In particular, for value prediction, we are interested in estimating the whole value function based on some sample values. This kind of generalization is called *function approximation*. There are non-linear function approximators like artificial neural networks, yet they lack convergence guarantees. In this work, we focus on linear function approximation, a well studied area, in which the value function is approximated by a weighted combination of a set of state features:

$$V(s) = \boldsymbol{\theta} \cdot \boldsymbol{\phi}(s) = \sum_{i=1}^{n} \theta_i \phi_i(s), \tag{2.2}$$

where the weight vector, $\boldsymbol{\theta}$, is the parameter set the agent learns while $\boldsymbol{\phi}$ maps each state to a feature vector ($\boldsymbol{\phi} : \mathcal{S} \to \Re^n$). Through the rest of this thesis, $\boldsymbol{\phi}_t$ means $\boldsymbol{\phi}(s_t)$. Radial basis functions, tile coding, and Kanerva coding are examples of linear function approximators. We discuss tile coding in more detail in the next section. Readers should refer to [Sutton and Barto, 1998] for explanations of the other methods.

2.2.1 Tile Coding

Tile coding is one of the popular methods for applying linear function approximation to reinforcement learning (also known as CMACS [Albus, 1971]). This method discretizes the state space with many tiles such that each point in the state space is within a small number of tiles. $\boldsymbol{\phi}$(s) is the function that maps each point to a feature vector which has ones in the rows corresponding to "active" tiles for that state and zeros for the rest. Figure

[1]Even a reinforcement learning task with one continuous state variable has an infinite number of states.

2.2 illustrates an example of applying tile coding to a 2D state space. Two sets of tiles, *tilings*, are used and each point corresponds to two tiles, so the feature vector $\boldsymbol{\phi}(s)$ will have two ones and the rest zeros. The number of tilings and the way they cover the space can be selected arbitrarily [*i.e.* see Sutton and Barto, 1998], although simple grids are most common. As the accuracy of tile coding can be controlled through the number of tilings and the tile size, this method is capable of approximating any underlying non-linear function. These properties make tile coding an easy to implement yet powerful method and it has been used extensively [Tham, 1995, Sutton, 1996, Bowling and Veloso, 2002, Bowling and Veloso, 2003, Stone *et al.*, 2005, Sherstov and Stone, 2005]. An interesting property of tile coding, which is common among most linear function approximators, is that it represents the state space with sparse feature vectors: the maximum number of active features at any given moment (the number of tilings, k), is much smaller than the total number of features (n) or simply $k \ll n$. Throughout the rest of this thesis, we assume this sparsity property on all feature vectors. Sparse feature vectors are commonly used to deal with large state spaces. Stone *et al.* used ten thousand features for the keep away soccer domain [2005], but only 416 of these features were active at any given moment. Bowling *et al.* used tile coding to represent the state of a card game using only three tilings for one million features [Bowling and Veloso, 2002].

2.3 Temporal Difference Learning

Temporal difference methods are a class of methods for learning a value function from experience [*e.g.*, Sutton and Barto, 1998]. At each time step, given a new experience tuple $(s_t, a_t, r_{t+1}, s_{t+1)}$, TD reduces the error of estimated values by shifting the value $V(s_t)$ towards $r_t + \gamma V(s_{t+1})$:

$$V(s_t) \quad \leftarrow \quad V(s_t) + \alpha_t \delta_t,$$

where

$$\delta_t \quad = \quad r_t + \gamma V(s_{t+1}) - V(s_t).$$

The α_t parameter is the learning rate, and δ_t is the TD error at time t. If states are

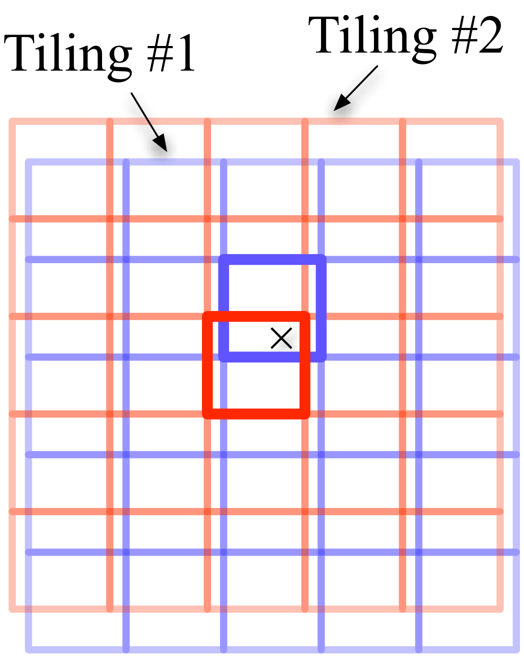

Figure 2.2: Tile Coding of a 2D state space with two tilings

represented by feature vectors then the above equations turn to:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \mathbf{u}_t(\boldsymbol{\theta}_t), \tag{2.3}$$

where

$$\mathbf{u}_t(\boldsymbol{\theta}_t) = \boldsymbol{\phi}(s_t)\delta_t(V_{\boldsymbol{\theta}_t}), \tag{2.4}$$

$$\delta_t(V_{\boldsymbol{\theta}_t}) = r_t + \gamma V_{\boldsymbol{\theta}_t}(s_{t+1}) - V_{\boldsymbol{\theta}_t}(s_t),$$

Using Equation 2.2, we get:

$$= r + \big(\gamma\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)\big)^T \boldsymbol{\theta}_t. \tag{2.5}$$

We call $\mathbf{u}_t(\boldsymbol{\theta}_t)$ the TD update at time t . Notice that δ_t is now a function of $V_{\boldsymbol{\theta}}$ and computes the TD error of the estimated values with respect to $\boldsymbol{\theta}_t$.

Algorithm 1 shows the complete TD algorithm with the computational complexity of its main line. The simplicity of the TD method has made this algorithm a common method for solving reinforcement learning problems. Line 4 computes the TD update according to Equations 2.3–2.5 and updates the weight vector $(\boldsymbol{\theta})$ at each time step. Line 4 first com-

Algorithm 1: TD Complexity

```
0  $s \leftarrow s_0$
1  Initialize $\boldsymbol{\theta}$ arbitrarily
2  **repeat**
3    Take an action according to $\pi$ and observe $r$, $s'$
4    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{\phi}(s)\Big(r + \big[\gamma\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)\big]^T\boldsymbol{\theta}\Big)$        $O(k)$
5    $s \leftarrow s'$
6  **end repeat**
```

putes the inner product of two vectors, of which one is sparse, and then multiplies another sparse vector by the result. Thus, as Algorithm 1 states, with the sparsity assumption, TD's complexity per time step is $O(k)$. However, the main drawback of the TD method is its forgetfulness. At each time step, TD takes only the current tuple, $(s_t, a_t, r_{t+1}, s_{t+1})$, into account. This inefficient use of data has encouraged researchers to look for methods that take advantage of more information at each time step. Experience replay [Lin, 1993] was one success of this line of research. It saves trajectories and repeatedly takes TD steps from the end of the trajectory back to the start state over the past experience. This helps the algorithm to propagate the TD error faster through the state space. Although the idea is useful and improves the performance of TD, it can not be applied online since it demands considerable time to take TD steps over the past experience.

2.3.1 TD with Eligibility Traces

Traditional TD methods adjust $V(s_t)$ toward $r + \gamma V(s_{t+1})$ (one step backups), but it is possible to look further ahead. In general, the n-step return, R_t^n, can be used to estimate the learning target:

$$R_t^{(n)} = \sum_{i=1}^{n} \left(\gamma^{i-1} r_{t+i}\right) + \gamma^n V_t(s_{t+n})$$

Now the question is: how should one pick n? One answer to this question is to weight each of these return values and then sum them all together. Following this idea, the λ-return is defined:

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}, \qquad 0 \leq \lambda \leq 1.$$

By this definition, $R_t^{(n)}$ will have a weight of $(1-\lambda)\lambda^n$. By changing λ we can get a variety of TD methods, which put different weight on immediate or distant backups as λ

changes. If $\lambda = 0$ then we obtain the traditional TD(0) method which only uses the one-step backup, while setting $\lambda = 1$ in an episodic task, we get TD(1), also called the *Monte Carlo* method, which does not rely on any intermediate value and only backs up the results from the end of the episode.

This explanation of λ is called the forward view, which is not suitable for implementation because it requires the agent to wait until the end of the episode to compute all the returns and back up the values. Another way to consider λ is the backward view, which is more suitable for implementation. For each state encountered along a trajectory, an eligibility trace named $e_t(s)$ is maintained and, at each time step, we decay all of the traces by multiplying them by $\gamma\lambda$. Whenever the algorithm experiences a TD error, instead of only updating the value of the previous state, it will update the values of all states according to their eligibility traces:

$$\begin{aligned} e_t(s) &= \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t; \\ \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \end{cases} \\ V_t(s) &= V_{t-1}(s) + \alpha\delta_t e_t(s) \quad \text{for all } s \in S. \end{aligned}$$

Although the forward and backward views may seem to perform different updates, they are equivalent in the offline mode [Sutton and Barto, 1998]. With eligibility traces, the TD error is backed up more quickly through the state space. Eligibility traces can also be applied to the function approximation case. Instead of each state, each feature has a trace parameter which we call $\mathbf{z}(i)$. At each time step, all traces are decayed and traces corresponding to active features are increased:

$$\mathbf{z}_t(i) = \begin{cases} \gamma\lambda\mathbf{z}_{t-1}(i) + 1 & \text{if } \mathbf{z}(i) \in \text{present features of } \boldsymbol{\phi}(s_t); \\ \gamma\lambda\mathbf{z}_{t-1}(i) & \text{otherwise;} \end{cases} \tag{2.6}$$

$$\mathbf{u}_t(\boldsymbol{\theta}_t) = \mathbf{z}_t\delta_t(V_{\boldsymbol{\theta}_t}) \tag{2.7}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \alpha\mathbf{u}_t(\boldsymbol{\theta}_{t-1}). \tag{2.8}$$

In practice, a threshold (ϵ) can be used to cut small eligibility traces to zero. Let l be the number of time steps that an eligibility trace is decayed to reach the threshold (*i.e.*, $l = log_{\lambda}^{\epsilon}$), then the maximum number of non zero elements of **z** is lk.

```
Algorithm 2: TD(λ)                                              Complexity
0 $s \leftarrow s_0, \mathbf{z} \leftarrow \mathbf{0}$
1 Initialize $\boldsymbol{\theta}$ arbitrarily
2 repeat
3   Take an action according to $\pi$ and observe $r$, $s'$
4   $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \boldsymbol{\phi}(s)$                                   $O(lk)$
5   $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\mathbf{z}\Big(r + \big[\gamma\boldsymbol{\phi}(s') - \boldsymbol{\phi}(s)\big]^T\boldsymbol{\theta}\Big)$         $O(lk)$
6   $s \leftarrow s'$
7 end repeat
```

Algorithm 2 shows the pseudo-code of TD(λ) with the computational complexity of important lines.[2] Line 4 computes the eligibility trace vector (**z**) based on Equation 2.6 and Line 5 updates the weight vector ($\boldsymbol{\theta}$) following Equations 2.7 and 2.8. We can see that incorporating eligibility traces into the TD method increases the per-time complexity from $O(k)$ to $O(lk)$,[3] but as shown by Sutton [1988] eligibility traces can considerably improve the performance of the TD method. It helps the algorithm to propagate the TD error faster, and as the value of λ is increased, the value function will have less bias and more variance. Tsitsiklis and Van Roy [1997] proved the convergence of online TD(λ) with linear function approximation.

2.4 Least-Squares TD

First introduced by Bradtke and Barto [1996], the least-squares TD (LSTD) algorithm was an attempt to use past experience more efficiently than in traditional TD. The basic idea behind the LSTD method is to compute the $\boldsymbol{\theta}$ vector that minimizes the sum of TD errors over all past experiences.

2.4.1 Derivation

In this section, we present the derivation of the LSTD method [Bradtke and Barto, 1996]. Let $\boldsymbol{\mu}_t(\boldsymbol{\theta})$ be the sum of TD updates $(\mathbf{u}_t(\boldsymbol{\theta}))$ up to time t,

$$\boldsymbol{\mu}_t(\boldsymbol{\theta}_t) = \sum_{i=1}^{t} \mathbf{u}_i(\boldsymbol{\theta}_t).$$

[2]As described in Section 2.2.1, we assume that feature vectors are sparse.

[3]With a very small threshold, **z** is not going to be sparse which translates into $O(n)$ per time step complexity for TD(λ).

Applying equations 2.4 and 2.5,

$$\begin{aligned}
\boldsymbol{\mu}_t(\boldsymbol{\theta}) &= \sum_{i=1}^{t} \boldsymbol{\phi}_i \delta_i(V_{\boldsymbol{\theta}}) \\
&= \sum_{i=1}^{t} \boldsymbol{\phi}_i \left(r_{i+1} + \gamma \boldsymbol{\phi}_{i+1}^T \boldsymbol{\theta} - \boldsymbol{\phi}_i^T \boldsymbol{\theta} \right) \\
&= \sum_{i=1}^{t} \left(\boldsymbol{\phi}_i r_{i+1} - \boldsymbol{\phi}_i (\boldsymbol{\phi}_i - \gamma \boldsymbol{\phi}_{i+1})^T \boldsymbol{\theta} \right) \\
&= \underbrace{\sum_{i=1}^{t} \boldsymbol{\phi}_i r_{i+1}}_{\mathbf{b}_t} - \underbrace{\sum_{i=1}^{t} \boldsymbol{\phi}_i (\boldsymbol{\phi}_i - \gamma \boldsymbol{\phi}_{i+1})^T}_{\mathbf{A}_t} \boldsymbol{\theta} \qquad (2.9) \\
&= \mathbf{b}_t - \mathbf{A}_t \boldsymbol{\theta}.
\end{aligned}$$

Setting $\boldsymbol{\mu}$ to zero, we get:

$$\boldsymbol{\theta} = \mathbf{A}_t^{-1} \mathbf{b}_t.$$

As described by Boyan [1999], the **A** and **b** matrices model the environment based on the observed trajectories using maximum likelihood. The **A** matrix holds the information about the transition function: given $\boldsymbol{\phi}_t$, what is the expected $\boldsymbol{\phi}_{t+1}$. Thus, we call **A** the transition model. The **b** vector represents a reward model based on all experiences: given $\boldsymbol{\phi}_t$, what is the expected reward on the next time step. Hence we call **b** the reward model..

2.4.2 Alternative Derivation

An alternative view of the LSTD algorithm is a method for minimizing the mean square error (MSE) of all state values with respect to $\boldsymbol{\theta}$ weighted by their probability distribution following policy $\pi, P_\pi(s)$. Since $V^\pi(s)$ might not be in the space spanned by the feature vectors, we want to find a set of parameters, $\boldsymbol{\theta}$, that makes $V(\cdot)$ as close as possible to $V^\pi(\cdot)$. Let $C(\boldsymbol{\theta})$ be the objective function that we seek to minimize.

$$C(\boldsymbol{\theta}) = \sum_{s \in S} P_\pi(s) (V^\pi(s) - V(s))^2$$

Taking the gradient of the function with respect to $\boldsymbol{\theta}$, we get:

$$\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = \sum_{s \in S} P_\pi(s) \nabla_{\boldsymbol{\theta}} (V^\pi(s) - V(s))^2$$

Assuming that the visited states are drawn from the same distribution as $P_\pi(s)$:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) &= \frac{1}{t}\sum_{i=1}^{t} \nabla_{\boldsymbol{\theta}} (V^\pi(s_i) - V(s_i))^2 \\
&= -\frac{2}{t}\sum_{i=1}^{t} (V^\pi(s_i) - V(s_i)) \nabla_{\boldsymbol{\theta}} V(s_i)
\end{aligned}$$

Since the real value of $V^\pi(s)$ is not available, it is estimated with $r_t + \gamma V(s_{t+1})$, which leads to:

$$\begin{aligned}
&\text{Let } a_t = -\frac{2}{t}, \\
\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) &\approx a_t \sum_{i=1}^{t} \left(r_i + \gamma V(s_{t+1}) - V(s_i)\right) \boldsymbol{\phi}_i \\
&= a_t \sum_{i=1}^{t} \left(r_i + \gamma \boldsymbol{\phi}_{i+1}^T \boldsymbol{\theta} - \boldsymbol{\phi}_i^T \boldsymbol{\theta}\right) \boldsymbol{\phi}_i \\
&= a_t \sum_{i=1}^{t} \left(r_i \boldsymbol{\phi}_i - \left(\boldsymbol{\phi}_i - \gamma \boldsymbol{\phi}_{i+1}\right)^T \boldsymbol{\theta} \boldsymbol{\phi}_i\right) \\
&= a_t \left(\underbrace{\sum_{t=1}^{t} r_i \boldsymbol{\phi}_i}_{\mathbf{b}_t} - \underbrace{\sum_{t=1}^{t} \boldsymbol{\phi}_i \left(\boldsymbol{\phi}_i - \gamma \boldsymbol{\phi}_{i+1}\right)^T}_{\mathbf{A}_t} \boldsymbol{\theta} \right) \qquad (2.10) \\
&= a_t (\mathbf{b}_t - \mathbf{A}_t \boldsymbol{\theta})
\end{aligned}$$

Setting $\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$ to zero yields:

$$\boldsymbol{\theta} = \mathbf{A}_t^{-1} \mathbf{b}_t. \qquad (2.11)$$

2.4.3 Iterative Matrix Inversion

The matrix inversion in Equation 2.11 is computationally expensive (*i.e.* $O(n^3)$). Bradtke and Barto [1996] showed that by updating $\mathbf{A}, \mathbf{b}$, and $\mathbf{A}^{-1}$ iteratively, the per time step computational complexity is reducible to $O(n^2)$ per time step.

$$\begin{aligned}
\mathbf{b}_t &= \mathbf{b}_{t-1} + \underbrace{r_t \boldsymbol{\phi}_t}_{\Delta \mathbf{b}_t}, \qquad (2.12) \\
\mathbf{A}_t &= \mathbf{A}_{t-1} + \underbrace{\boldsymbol{\phi}_t (\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})^T}_{\Delta \mathbf{A}_t}. \qquad (2.13)
\end{aligned}$$

Algorithm 3: LSTD — Complexity

```
0  $s \leftarrow s_0$, $\tilde{\mathbf{A}} \leftarrow \frac{1}{\omega}\mathbf{I}$, $\mathbf{b} \leftarrow \mathbf{0}$
1  Initialize $\boldsymbol{\theta}$ arbitrarily
2  repeat
3    Take action according to $\pi$ and observe $r$, $s'$
4    $\mathbf{b} \leftarrow \mathbf{b} + \boldsymbol{\phi}(s)r$                                    $O(k)$
5    $\mathbf{y} \leftarrow (\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s'))^T$                  $O(k)$
6    $\tilde{\mathbf{A}} \leftarrow \tilde{\mathbf{A}}\Big(I - \Big(\frac{\boldsymbol{\phi}(s)\mathbf{y}}{1+\mathbf{y}\tilde{\mathbf{A}}\boldsymbol{\phi}(s)}\Big)\tilde{\mathbf{A}}\Big)$     $O(n^2)$
7    $\boldsymbol{\theta} \leftarrow \tilde{\mathbf{A}}\mathbf{b}$                                        $O(n^2)$
8    $s \leftarrow s'$
9  end repeat
```

As mentioned by Xu *et al.* [2002] and Lagodakis and Parr [2003], the inverse of a matrix can be computed incrementally:

$$(\mathbf{A} + \mathbf{x}\mathbf{y})^{-1} = \mathbf{A}^{-1}\Big(I - \Big(\frac{\mathbf{x}\mathbf{y}}{1+\mathbf{y}\mathbf{A}^{-1}\mathbf{x}}\Big)\mathbf{A}^{-1}\Big), \tag{2.14}$$

and in our case, $\mathbf{x} = \boldsymbol{\phi}_t$ and $\mathbf{y} = (\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^T$.

2.4.4 LSTD Algorithm

Algorithm 3 shows the LSTD method using an iterative matrix inversion as introduced by Lagodakis and Parr [2003]. It also highlights the computational complexity of the most expensive lines, assuming the sparsity of the features. Line 4 computes **b** iteratively using Equation 2.12, while Lines 5 and 6 update the new estimation of $\mathbf{A}^{-1}$, $\tilde{\mathbf{A}}$, according to Equations 2.13 and 2.14. The algorithm initializes $\tilde{\mathbf{A}}$ with $\frac{1}{\omega}\mathbf{I}$, ($\mathbf{I}$ is the identity matrix) so that the matrix will be full-rank on every time step. As we reduce the value of ω the difference between $\tilde{\mathbf{A}}$ and $\mathbf{A}^{-1}$ can be made arbitrary small. Another approach is to wait for $\mathbf{A}$ to become full-rank, take the inverse once, then use that inverse as a basis for future iterative computations. However, this approach is not suitable for large problems because it demands extensive experience to form the first invertible $\mathbf{A}$.

The LSTD method demonstrates promising results over the TD method [Bradtke and Barto, 1996, Boyan, 1999]. The main advantage is that TD errors are minimized over all past experience on every time step. Therefore, past interactions are not forgotten when updating the value function. The main downside of the LSTD method is its computational cost per step. As Algorithm 3 shows, even by using iterative matrix inversion, LSTD requires $O(n^2)$ operations per time step to compute the new weight vector. Note that using sparse feature vectors does not help the algorithm with its most expensive lines (*i.e.* Lines 6 and 7). This fact prevents LSTD from being applied online in large environments with many features.

2.5 LSTD(λ)

Boyan introduced LSTD(λ), for which LSTD is the special case of $\lambda = 0$ [Boyan, 1999, Boyan, 2002]. The derivation of LSTD(λ) follows naturally when TD updates take into account the effect of eligibility traces as shown in Equation 2.7. Having $\boldsymbol{\mu}_t(\boldsymbol{\theta})$ be the sum of the TD updates through time t, we can add eligibility traces into the LSTD method:

$$\begin{aligned}
\boldsymbol{\mu}_t(\boldsymbol{\theta}) &= \sum_{i=1}^{t} \mathbf{u}_i(\boldsymbol{\theta}) = \sum_{i=1}^{t} z_i \left(r_{i+1} + \gamma V_{\boldsymbol{\theta}}(s_{i+1}) - V_{\boldsymbol{\theta}}(s_i)\right) \\
&= \sum_{i=1}^{t} z_i \left(r_{i+1} + \gamma \boldsymbol{\phi}_{i+1}^T \boldsymbol{\theta} - \boldsymbol{\phi}_i^T \boldsymbol{\theta}\right) \\
&= \underbrace{\sum_{i=1}^{t} z_i r_{i+1}}_{\mathbf{b}_t} - \underbrace{\sum_{i=1}^{t} z_i (\boldsymbol{\phi}_i - \gamma \boldsymbol{\phi}_{i+1})^T}_{\mathbf{A}_t} \boldsymbol{\theta} \qquad (2.15) \\
&= \mathbf{b}_t - \mathbf{A}_t \boldsymbol{\theta}.
\end{aligned}$$

Note that $\mathbf{A}_t$ and $\mathbf{b}_t$ have been updated from their definitions in Equation 2.9 to include eligibility traces. In Boyan's experiments, this extension showed slight improvement in a small synthetic domain [1999].

Algorithm 4 shows the pseudo-code for LSTD(λ) with the computational complexities of important lines. As with TD(λ), Line 4 updates the eligibility trace vector. Lines 5-7 maintain the values of $\mathbf{b}$ and $\tilde{\mathbf{A}}$ (the approximation of $\mathbf{A}^{-1}$) incrementally according to Equations 2.15 and 2.14. Using a threshold to cut off small eligibility traces leads the $\mathbf{z}$ vector to have $O(lk)$ non-zero elements. However, adding eligibility traces to the LSTD algorithm does not increase its per time step computational complexity.

Algorithm 4: LSTD(λ)   Complexity

```
0  $s \leftarrow s_0$, $\mathbf{z} \leftarrow \mathbf{0}$, $\tilde{\mathbf{A}} \leftarrow \frac{1}{\omega}\mathbf{I}$, $\mathbf{b} \leftarrow \mathbf{0}$
1  Initialize $\boldsymbol{\theta}$ arbitrarily
2  repeat
3    Take action according to $\pi$ and observe $r$, $s'$
4    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \boldsymbol{\phi}(s)$                                  $O(lk)$
5    $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{z}r$                                       $O(lk)$
6    $\mathbf{y} \leftarrow (\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s'))$                          $O(k)$
7    $\tilde{\mathbf{A}} \leftarrow \tilde{\mathbf{A}}\left(I - \left(\frac{\mathbf{z}\mathbf{y}^T}{1+\mathbf{y}^T\tilde{\mathbf{A}}\mathbf{z}}\right)\tilde{\mathbf{A}}\right)$        $O(n^2)$
8    $\boldsymbol{\theta} \leftarrow \tilde{\mathbf{A}}\mathbf{b}$                                        $O(n^2)$
9    $s \leftarrow s'$
10 end repeat
```

2.6 Conclusion

In this chapter, we introduced the reinforcement learning framework and the MDP model. We also discussed RL methods for solving MDPs, and how they use function approximators to tackle realistic problems and eligibility traces to propagate the TD error faster. We showed that traditional TD methods are fast and scale easily to larger environments, although data is used inefficiently. On the other hand, LSTD is a powerful algorithm which uses all of the experience at each time step to compute the new value function, but its expensive per time step complexity prevents it from being applied to problems with many features. In the next chapter, we introduce a novel method which seeks to achieve the best of both worlds.

Chapter 3

Incremental Least-Squares Temporal Difference Learning

In this chapter, we present the incremental least-squares temporal difference learning algorithm (iLSTD) and study its per time step complexity, convergence analysis, and its performance on a few problems. This method builds upon the merits of TD and LSTD: if a sparse feature representation is used, the per time step complexity is linear in the number of features, *n*, and each update takes into account all of the past experience. The main idea behind iLSTD is to use the sum of TD updates, $\boldsymbol{\mu}$, over all of the observed trajectories in a gradient fashion to drive it closer to zero. Since iLSTD makes use of all observed trajectories, it is more data efficient than TD. It also avoids an $O(n^2)$ per time step complexity by taking steps in a limited number of dimensions.

3.1 Incremental Computation

iLSTD is based on the iterative computation of the sum of TD updates, $\boldsymbol{\mu}_t(\boldsymbol{\theta})$, as new transitions are observed and the weight vector ($\boldsymbol{\theta}$) changes. As shown in the previous chapter, **A** and **b** can be computed in an incremental fashion given a newly observed reward and transition as:

$$\begin{aligned}\mathbf{b}_t &= \mathbf{b}_{t-1} + \underbrace{r_t\boldsymbol{\phi}_t}_{\Delta\mathbf{b}_t}, \\ \mathbf{A}_t &= \mathbf{A}_{t-1} + \underbrace{\boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^T}_{\Delta\mathbf{A}_t}. \end{aligned} \tag{3.1}$$

Given new updates to $\mathbf{A}$ and $\mathbf{b}$, $\boldsymbol{\mu}_t(\boldsymbol{\theta})$ can be computed incrementally as:

$$\begin{aligned} \boldsymbol{\mu}_t(\boldsymbol{\theta}) &= \mathbf{b}_t - \mathbf{A}_t\boldsymbol{\theta}, \\ &= \boldsymbol{\mu}_{t-1}(\boldsymbol{\theta}) + \Delta\mathbf{b}_t - (\Delta\mathbf{A}_t)\boldsymbol{\theta}. \end{aligned}$$

Finally, given an update to $\boldsymbol{\theta}$, $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t$, $\boldsymbol{\mu}_t(\boldsymbol{\theta}_{t+1})$ is computed as:

$$\boldsymbol{\mu}_t(\boldsymbol{\theta}_{t+1}) = \boldsymbol{\mu}_t(\boldsymbol{\theta}_t) - \mathbf{A}_t(\Delta\boldsymbol{\theta}_t). \tag{3.2}$$

The time complexity of these computations will be discussed in Section 3.3.

3.2 Parameter Update

As shown in chapter 2, maintaining the exact solution of $\boldsymbol{\mu}_t(\boldsymbol{\theta}_{t+1}) = 0$, even with incremental matrix inversion, requires $O(n^2)$ complexity per time step. An alternative is to use $\boldsymbol{\mu}$ as a guiding vector in order to reduce the error on every time step. This approach can be viewed as applying the total change to $\boldsymbol{\theta}$, if batch TD was used over all past transitions. This has the advantage of using all past experiences which can lead to a lower variance than TD's traditional single sample update. Unfortunately, unless $\Delta\boldsymbol{\theta}_t$ has many zero elements, Equation 3.2 demands $O(n^2)$ computation ($A_{n\times n} \cdot \Delta\boldsymbol{\theta}_{n\times 1}$).

The need for many zero elements in $\Delta\boldsymbol{\theta}_t$ suggests the main idea used by iLSTD. Instead of updating all of the components of $\boldsymbol{\theta}$ in one iteration, iLSTD only considers updating a small set of components of $\boldsymbol{\theta}$. More specifically, if only the ith component of $\boldsymbol{\theta}$ is updated, $\boldsymbol{\mu}$ can be updated as follows:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t\boldsymbol{\mu}_t(i)\boldsymbol{e}_i, \\ \boldsymbol{\mu}_t(\boldsymbol{\theta}_{t+1}) &= \boldsymbol{\mu}_t(\boldsymbol{\theta}_t) - \alpha_t\boldsymbol{\mu}_t(i)\mathbf{A}_t\boldsymbol{e}_i, \end{aligned}$$

where $\boldsymbol{\mu}_t(i)$ is the ith component of $\boldsymbol{\mu}_t$ and $\boldsymbol{e}_i$ is the column vector with a single 1 in the ith row, thus $\mathbf{A}_t\boldsymbol{e}_i$ selects the ith column of matrix $\mathbf{A}_t$. On each time step iLSTD updates multiple components by repeatedly selecting a component and performing the one component update above. The algorithm takes a parameter $m \ll n$ that specifies the number of updates performed per time step. The algorithm also takes a feature selection mechanism which determines the choice of i as a function of $\boldsymbol{\mu}_t(\boldsymbol{\theta}_t)$. In Chapter 5, we will address possible feature selection mechanisms in further detail and compare the performance of various methods. For simplicity, throughout the rest of the thesis, unless specified the feature selection method will be non-zero random: we discard dimensions with zero sum TD update and select one uniformly random out of the remaining dimensions.

3.3 Algorithm

Algorithm 5 shows the iLSTD algorithm together with the computational complexity of important lines. After setting the initial values, the agent begins interacting with the environment. $\mathbf{A}$ and $\boldsymbol{\mu}$ are computed incrementally in Lines 5–8 according to Equations 3.1 and 3.2[1]. It is followed by updating m selected features in Lines 9–13. For each of the m updates, one feature is selected through a component selection mechanism. After the update, $\boldsymbol{\mu}$ is recomputed accordingly. This recomputation can affect the next component chosen.

Algorithm 5: iLSTD   Complexity

```
 0  $s \leftarrow s_0$, $\mathbf{A} \leftarrow \mathbf{0}$, $\boldsymbol{\mu} \leftarrow \mathbf{0}$, $t \leftarrow 0$
 1  Initialize $\boldsymbol{\theta}$ arbitrarily
 2  **repeat**
 3    Take action according to $\pi$ and observe $r$, $s'$
 4    $t \leftarrow t + 1$
 5    $\Delta\mathbf{b} \leftarrow \boldsymbol{\phi}(s)r$                                   $O(k)$
 6    $\Delta\mathbf{A} \leftarrow \boldsymbol{\phi}(s)(\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s'))^T$          $O(k^2)$
 7    $\mathbf{A} \leftarrow \mathbf{A} + \Delta\mathbf{A}$                                  $O(k^2)$
 8    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \Delta\mathbf{b} - (\Delta\mathbf{A})\boldsymbol{\theta}$                      $O(k^2)$
 9    **for** $i$ from 1 to m **do**
10      $j \leftarrow$ choose an index of $\boldsymbol{\mu}$ using a *dimension selection mechanism*
11      $\theta_j \leftarrow \theta_j + \alpha\mu_j$                                  $O(1)$
12      $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\mu_j\mathbf{A}\boldsymbol{e}_j$                          $O(n)$
13    **end for**
14    $s \leftarrow s'$
15  **end repeat**
```

3.3.1 Time Analysis of iLSTD

We now examine the time complexity of the iLSTD algorithm.

[1] $\mathbf{b}$ is not computed because it is implicitly included in $\boldsymbol{\mu}$.

Theorem 1. *If there are n features and for any given state s, $\boldsymbol{\phi}(s)$ has at most k non-zero elements, then the iLSTD algorithm is $O(mn + k^2)$ per time step.*

Proof. Lines 6–8 are the most computationally expensive parts of iLSTD outside the inner loop. Because each feature vector does not have more than k non-zero elements, $\boldsymbol{\phi}(s)r$ has only k non-zero elements and the matrix $\boldsymbol{\phi}(s)(\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s'))^T$ has at most $2k^2$ non-zero elements. Therefore lines 6–8 are computable in $O(k^2)$ with a sparse representation of matrices and vectors. Inside the parameter update loop (Line 9), if a feature selection rule with $O(n)$ complexity is applied, then the most expensive line is 12. Because $\mathbf{A}\boldsymbol{e}_j$ is the jth column of $\mathbf{A}$, the parameter update is also $O(n)$. This leads to $O(mn + k^2)$ as the final bound for the algorithm per time step. □

3.4 Convergence Analysis

In this section, we analyze the convergence properties of the iLSTD method. The proof is based on a key lemma proved by Martin Zinkevich [2006]. Here, we present the lemma and show its application to prove the convergence of iLSTD. The proof is similar to Bertsekas and Tsitsiklis' proof for TD(λ) [1996]. We begin by defining an abstract theoretical model. $\forall t \in \mathbb{N}$, $\mathbf{y}_t, \mathbf{d}_t \in \mathbb{R}^n$, $\mathbf{R}_t, \mathbf{C}_t \in \mathbb{R}^{n\times n}$, $\beta_t \in \mathbb{R}$, and:

$$\mathbf{y}_{t+1} = \mathbf{y}_t + \beta_t(\mathbf{R}_t)(\mathbf{C}_t\mathbf{y}_t + \mathbf{d}_t),$$

where $\mathbf{C}_t, \mathbf{d}_t$, and $\mathbf{R}_t$ are random variables. On every time step, $\mathbf{C}_t$ and $\mathbf{d}_t$ are selected first, followed by $\mathbf{R}_t$. Define F_t to be the state of the algorithm on time step t before $\mathbf{R}_t$ is selected. $\mathbf{C}_t$ and $\mathbf{d}_t$ are sequences of random variables.

Lemma 2. *Given assumptions A1 through A6, $\mathbf{y}_t$ converges to $-(\mathbf{C}^*)^{-1}\mathbf{d}^*$ with probability one.*

In order to prove convergence of $\mathbf{y}_t$, we assume that there is a $\mathbf{C}^*$, $\mathbf{d}^*$, v, $\mu > 0$, and M such that:

A1. $\mathbf{C}^*$ *is negative definite,*

A2. $\mathbf{C}_t$ *converges to* $\mathbf{C}^*$ *with probability 1,* *A3.* $\mathbf{d}_t$ *converges to* $\mathbf{d}^*$ *with probability 1,*

A4. (a) $\mathbf{E}[\mathbf{R}_t|F_t] = I$*, and (b)* $\|\mathbf{R}_t\| \leq M$*,*

A5. $\lim_{T\to\infty} \sum_{t=1}^{T} \beta_t = \infty$*, and*

A6. $\beta_t < vt^{-\mu}$.

The complete proof can be found in Zinkevich's original work [2006]. The assumptions deviate from those used by Bertsekas and Tsitsiklis, since they considered $\mathbf{C}_t$ and $\mathbf{d}_t$ that converged in expectation quickly, while we consider $\mathbf{C}_t$ and $\mathbf{d}_t$ that may converge more slowly, but they converge in value not just in expectation. We can now state our main result.

Theorem 3. *If the Markov decision process is finite and an appropriate* α *decay schedule is used, then iLSTD with a uniform random feature selection mechanism converges to the same result as TD.*

Proof. First, we explain how to map iLSTD on to the mathematical model, and then show how iLSTD meets the assumptions of Lemma 2:

1. $\mathbf{y}_t = \boldsymbol{\theta}_t$,
2. $\beta_t = t\alpha/n$,
3. $\mathbf{C}_t = -\mathbf{A}_t/t$,
4. $\mathbf{d}_t = \mathbf{b}_t/t$, and
5. $\mathbf{R}_t$ is a matrix, where there is an n on the diagonal in position (k_t, k_t) (where k_t is uniform random over the set {1,..., n} and i.i.d.) and zeroes everywhere else.

As shown by Bertsekas and Tsitsiklis [1996], $\mathbf{E}[\mathbf{C}_t] = \mathbf{C}^*$ is a negative definite matrix which coincides with Assumption 1. The proof for satisfying Assumptions 2 and 3 are available in Zinkevich's work [2006]. Considering the definition of $\mathbf{R}_t$ above,

$$\begin{aligned}
&\forall t \in \mathbb{N}, \forall i, j \in \{1, \ldots, n\}, \\
&\qquad i \neq j \quad \Rightarrow \quad \mathbf{E}[\mathbf{R}_t^{ij}|F_t] = 0, \\
&\qquad i = j \quad \Rightarrow \quad \mathbf{E}[\mathbf{R}_t^{ij}|F_t] = \frac{1}{n} \times n = 1, \\
&\qquad\qquad \therefore \quad \mathbf{E}[\mathbf{R}_t|F_t] = \mathbf{I},
\end{aligned}$$

and since $||\mathbf{R}_t|| \leq n$, Assumption 4 is satisfied. We only need to introduce an example learning rate that meets Assumptions 5 and 6. We show that having $\alpha_t = t^{-(k+1)}, 0 < k \leq 1$ satisfies Assumptions 5 and 6.

(Assumption 5)

$$\begin{aligned}
\lim_{T\to\infty} \sum_{t=1}^{T} \beta_t &= \lim_{T\to\infty} \sum_{t=1}^{T} \frac{t\alpha_t}{n}, \\
&= \frac{1}{n} \lim_{T\to\infty} \sum_{t=1}^{T} t^{-k}, \\
&= \infty.
\end{aligned}$$

(Assumption 6)

Let $\upsilon = \dfrac{2}{n}$ and $\mu = k$,

$$\begin{aligned}
\beta_t &= \frac{t\alpha_t}{n}, \\
&= \frac{t^{-k}}{n}, \\
&< \upsilon t^{-\mu}.
\end{aligned}$$

□

Notice that one can come up with various dimension selection rules which satisfy the requirements of Lemma 2, of which uniform random selection is the most trivial. In our experimental results, only dimensions with non-zero values are selected. In Chapter 5, we will focus on a few alternatives for dimension selection methods, and investigate their performance and convergence properties using the following lemma.

Theorem 4. *If the Markov decision process is finite and an appropriate α decay schedule is used, iLSTD with any feature selection mechanism which satisfies the following assumption converges to the same result as TD.*

A1. Let $P_t(i)$ be the probability of selecting the ith dimension of $\boldsymbol{\mu}_t$ at time t, then
$\exists \xi \in \mathbb{R}, \forall t \in \mathbb{N}$, *such that* $\forall i \in \{1, \ldots, n\}$ *if* $(\boldsymbol{\mu}_t(i) \neq 0) \Rightarrow 0 < \xi \leq P_t(i) \leq 1$.

Proof. We introduce a new mapping from iLSTD to the theoretical model which differs from our previous mapping by having new definitions for $\mathbf{R}_t$ and β. Thus we merely need to verify the satisfaction of Assumptions 4–6 of Lemma 2, because the rest of the model remains unchanged from the proof of Theorem 3. Let $q_t = \frac{1}{P_t(i)n}$. Define $\mathbf{R}_t$ to be the matrix with an nq_t in position (k_t, k_t) where k_t is the dimension selected at time t, a 1 in position (j, j) for all j such that $\boldsymbol{\mu}(j) = 0$ and zeros everywhere else. Define $\beta_t = \dfrac{t\alpha_t}{nq_t}$ and assume $\alpha_t = q_t t^{-(k+1)}, 0 < k \leq 1$.

(Assumption 4)

$$\begin{aligned}
&\forall t \in \mathbb{N}, \forall i, j \in \{1, \ldots, n\}, \\
i \neq j &\;\Rightarrow\; \mathbf{E}[\mathbf{R}_t^{ij}|F_t] = 0, \\
\boldsymbol{\mu}_t(i) \neq 0, i = j &\;\Rightarrow\; \mathbf{E}[\mathbf{R}_t^{ij}|F_t] = \frac{1}{P_t(i)n} \times P_t(i) \times n = 1, \\
\boldsymbol{\mu}_t(i) = 0, i = j &\;\Rightarrow\; \mathbf{E}[\mathbf{R}_t^{ij}|F_t] = 1, \\
&\;\therefore\; \mathbf{E}[\mathbf{R}_t|F_t] = \mathbf{I}. \\
\forall t \in \mathbb{N}, 0 < \xi \leq P_t(i) \leq 1 &\;\Rightarrow\; q_t \leq \frac{1}{n\xi} \\
&\;\Rightarrow\; ||\mathbf{R}_t||_\infty \leq \frac{1}{\xi}
\end{aligned}$$

(Assumption 5)

$$\begin{aligned}
\lim_{T\to\infty} \sum_{t=1}^{T} \beta_t &= \lim_{T\to\infty} \sum_{t=1}^{T} \frac{t\alpha_t}{nq_t}, \\
&= \frac{1}{n} \lim_{T\to\infty} \sum_{t=1}^{T} t^{-k}, \\
&= \infty.
\end{aligned}$$

(Assumption 6)

$$\begin{aligned}
&\text{Let } \upsilon = \frac{2}{n} \text{ and } \mu = k, \\
\beta_t &= \frac{t\alpha_t}{nq_t}, \\
&= \frac{t^{-k}}{n}, \\
&< \upsilon t^{-\mu}.
\end{aligned}$$

□

Notice that, in the analysis dimensions with zero sum TD update are updated on every time step. Since these updates do not change the weight vector, it is equivalent to completely ignore them as originally described in iLSTD.

Corollary 5. *If the Markov decision process is finite and an appropriate α decay schedule* *is used, then iLSTD with a non-zero random feature selection mechanism converges to the same result as TD.*

Proof. We show that non-zero random selection mechanism satisfies Assumption 1 of Theorem 4. Let m_t be the number of non-zero features at time t, then the probability of selecting a feature is:

$$\begin{aligned} &\forall t \in \mathbb{N}, i \in 1, \ldots, n, \\ P_t(i) &= \begin{cases} \frac{1}{m_t} & \boldsymbol{\mu}_t(i) \neq 0, \\ 0 & \text{Otherwise.} \end{cases}, \\ \max_{t \in \mathbb{N}}\{m_t\} \leq n &\Rightarrow 0 < \frac{1}{n} \leq P_t(i) \leq 1. \end{aligned}$$

□

An interesting corollary to Lemma 2, shown by Zinkevich [2006], is that if we take a step in all of the dimensions at the same time, we can consider the algorithm's rate of convergence.

Corollary 6. *If assumptions A1 through A6 of Lemma 2 hold and* $\mathbf{R}_t = \mathbf{I}$*, then there exists an* $\zeta \in (0, 1)$ *such that for all* $\mathbf{y}_t$*,* $\left\|\mathbf{y}_{t+1} + (\mathbf{C}_t)^{-1}\mathbf{d}_t\right\| < \zeta \left\|\mathbf{y}_t + (\mathbf{C}_t)^{-1}\mathbf{d}_t\right\|$.

This means that if on each iteration iLSTD takes a step in all dimensions, it reduces the error with respect to the LSTD's solution exponentially fast. However, updating all dimensions requires $O(n^2)$ computation, and so is not linear in the number of features. In all of the experimental results in this thesis, we used the iLSTD method as originally defined. However, we still observed that the performance of iLSTD converges quickly to the performance of LSTD.

3.5 Experimental Results

In this section, we study the performance of TD, iLSTD and LSTD methods in two domains: Boyan chain and mountain car. After explaining each domain, we present the performance of these algorithms and compare their experimental running time per step in each domain.

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

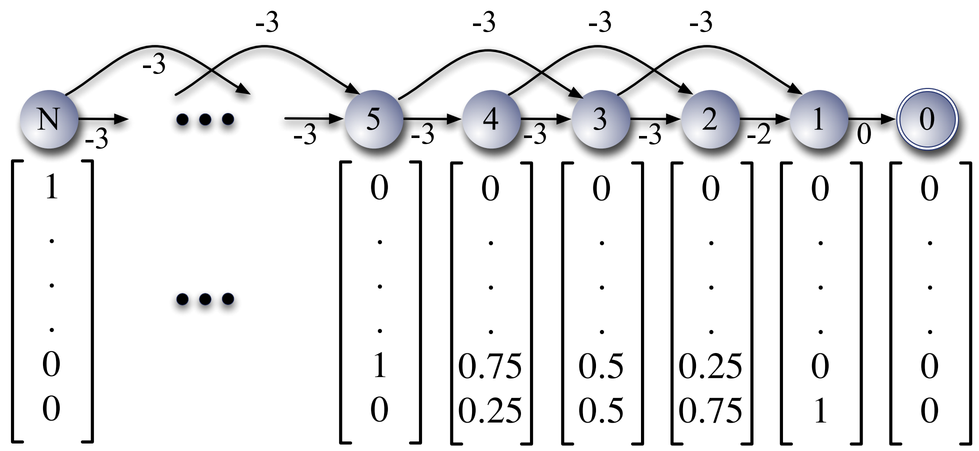

Figure 3.1: Generalized Boyan chain problem

For all of the experiments, the step size α takes a similar form as that used in Boyan's original experiments [Boyan, 1999]. The addition of the exponent on episode# was to make it consistent with the Assumption 6 of Lemma 2. [2]

The choice of α_0 and N_0 for TD and iLSTD was based on experimentally finding the best $\alpha_0 \in \{0.01, 0.1, 1\}$ and $N_0 \in \{100, 1000, 10^6\}$ for each algorithm separately. The parameters used in all experiments are available in the Appendix A. The m parameter for iLSTD was set to one, so on each time step only one non-zero random component was updated. The ω factor for the LSTD method (Algorithm 3) was set to 10^{-6}. The same random seed was used for each method, thus all of the algorithms saw the same set of trajectories.

3.5.1 Boyan Chain Problem

We first considered an extension of the Boyan chain problem [Boyan, 1999]. Figure 3.1 depicts this environment in its general form. The episodic task starts at state N and terminates at state zero. For all states $s > 2$, there is an equal probability of ending up in states $(s-1)$ or $(s-2)$, and the reward for all transitions is -3, except for transitions from states 2 to 1 and 1 to 0 that have rewards -2 and 0, respectively.[3] Vectors on the bottom of the figure illustrate the feature vector corresponding to each state.

[2]In the original work of Boyan, the power of episode# is 1.

[3]In order to have a single solution to the problem, state zero was added to the original problem.

In order to probe the effect of the number of features (n) on the computational complexity of iLSTD, results were conducted with three different problem sizes: 4 (original problem), 25, and 100 features. These will be called the small, medium, and large problems, respectively.

Figure 3.2 depicts the performance of the TD, iLSTD, and LSTD algorithms on the small, medium, and large problems. Results are averaged over 30 runs and the vertical axis shows the average root mean square (RMS) error value of all states in the logarithmic scale. Note that in this domain, the optimal solutions are in the space spanned by the feature vectors: $\boldsymbol{\theta}^* = (8 - 8n, ..., -24, -16, -8, 0)^T$.

In the left column, the horizontal axis shows the number of episodes completed by the algorithms, thus measuring each algorithm's data efficiency. Each point represents the RMS error and its confidence interval every 100 episodes. The confidence bars are shifted slightly to make them more visible. As the complexity of the problem increased the gap between the two least square algorithms and TD got considerably wider. The relative difference was most dramatic in the large problem, highlighting the advantage of least-square methods over TD. In addition, in all cases, iLSTD's performance got close to LSTD's performance fast and then follows it very closely.

In many circumstances, data is free but computation takes time. In such cases, the agent seeks to learn as much as possible in the time allocated while there is no limit on the amount of data. In the right column, the horizontal line is clock time, each point on the graph represents the RMS error, while each cross point shows the completion of 100 episodes. As the size of the problem increases, the least-square methods need more time to complete the same number of episodes computed by TD. Because the run-time complexity of iLSTD is asymptotically faster than LSTD, when the problem got large enough, iLSTD performed better than LSTD based on clock time. Figure 3.2 (bottom-right) shows that before LSTD finished the first 100 episodes in the large environment, iLSTD finished 500 episodes and reached a lower error. By that time, TD finished 900 episodes but could not par with iLSTD, because it does not take into account all of the past experience. Thus in comparison, its

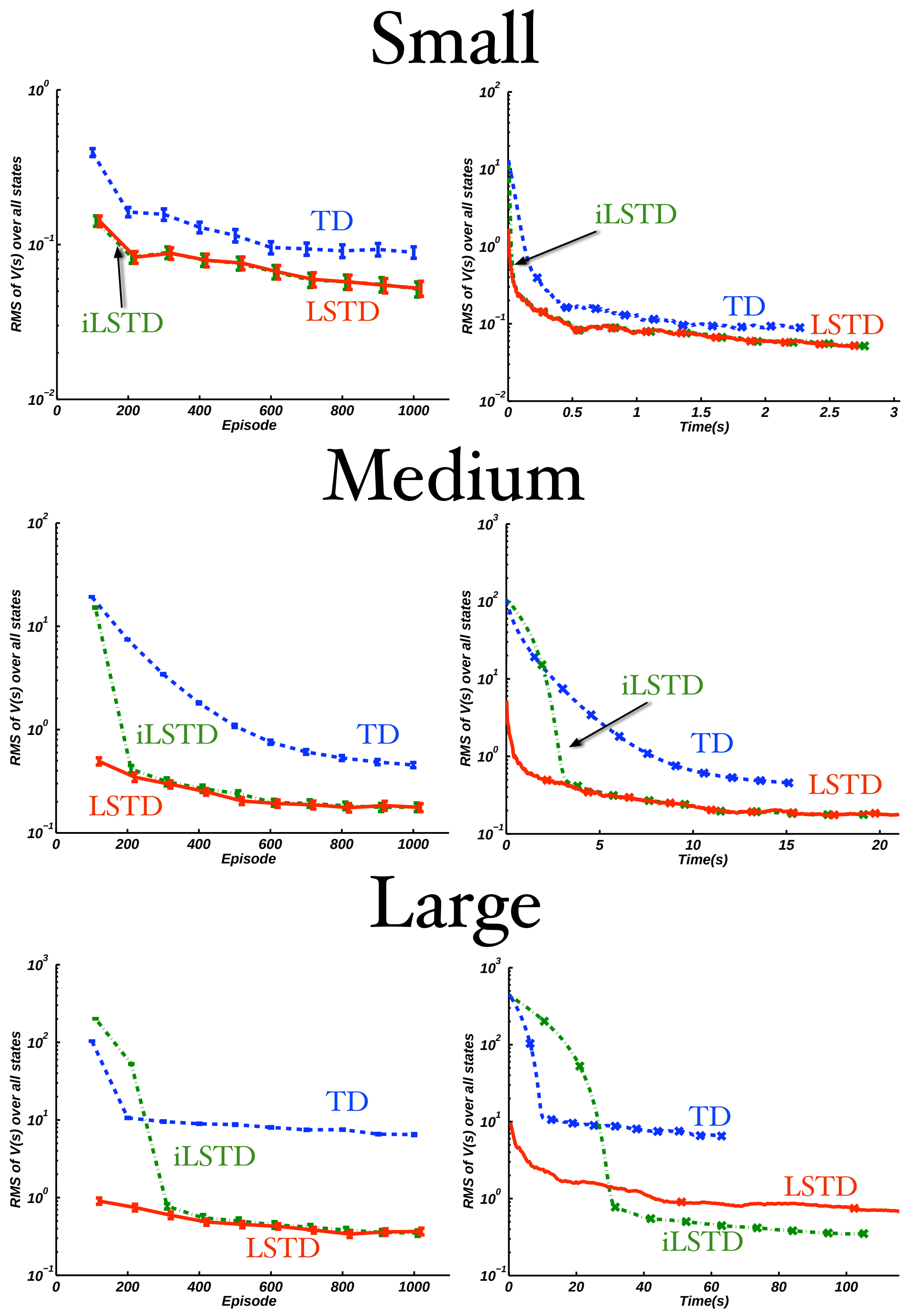

Figure 3.2: Performance of TD, iLSTD, and LSTD algorithms in generalized Boyan chain problem with three different sizes. Results are averaged over 30 trials. From left to right, it shows results based on episode and clock time respectively.

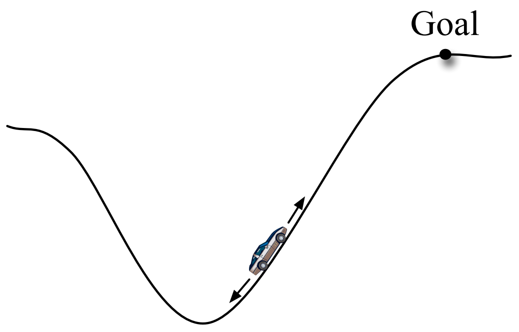

Figure 3.3: Mountain car problem

inefficient use of data was not compensated for by the improved computational efficiency.

3.5.2 Mountain Car Problem

The second test-bed is the mountain car domain [*e.g.*, see Sutton and Barto, 1998]. Illustrated in Figure 3.3, the episodic task for the car is to reach the goal state. Possible actions are accelerate forward, accelerate backward, and coast. The observation is a pair of continuous values: position and velocity. We placed the car in two positions with zero velocity: -1.0 and -0.5, which we call the easy and hard problems, respectively. Although the hard problem is the one that most researchers test their control algorithms based on [Sutton, 1996], we set up the cars in two positions to study the effect of more active features on the speed of different algorithms. Further details about the mountain car problem are available online [RL Library, 2006]. As we are focusing on policy evaluation, the policy was fixed for the car to accelerate in the direction of its current velocity with 90% probability or choose a random action with 10% probability. Tile Coding [*e.g.*, see Sutton, 1996] was selected as the linear function approximator. Ten tilings ($k = 10$) were used over the combination of the two parameter sets and the tilings were hashed into 10,000 features ($n = 10,000$). The rest of the settings were identical to those used for the Generalized Boyan chain problem.

Figure 3.4 depicts the results of the TD, iLSTD, and LSTD methods on the mountain car easy problem (top) and hard problem (bottom). The vertical axis represents the loss function in logarithmic scale. The loss we chose was $||\mathbf{b}^* - \mathbf{A}^*\boldsymbol{\theta}||_2$, and to have an unbiased estimation, $\mathbf{A}^*$ and $\mathbf{b}^*$ were computed based on 200,000 episodes of interaction with

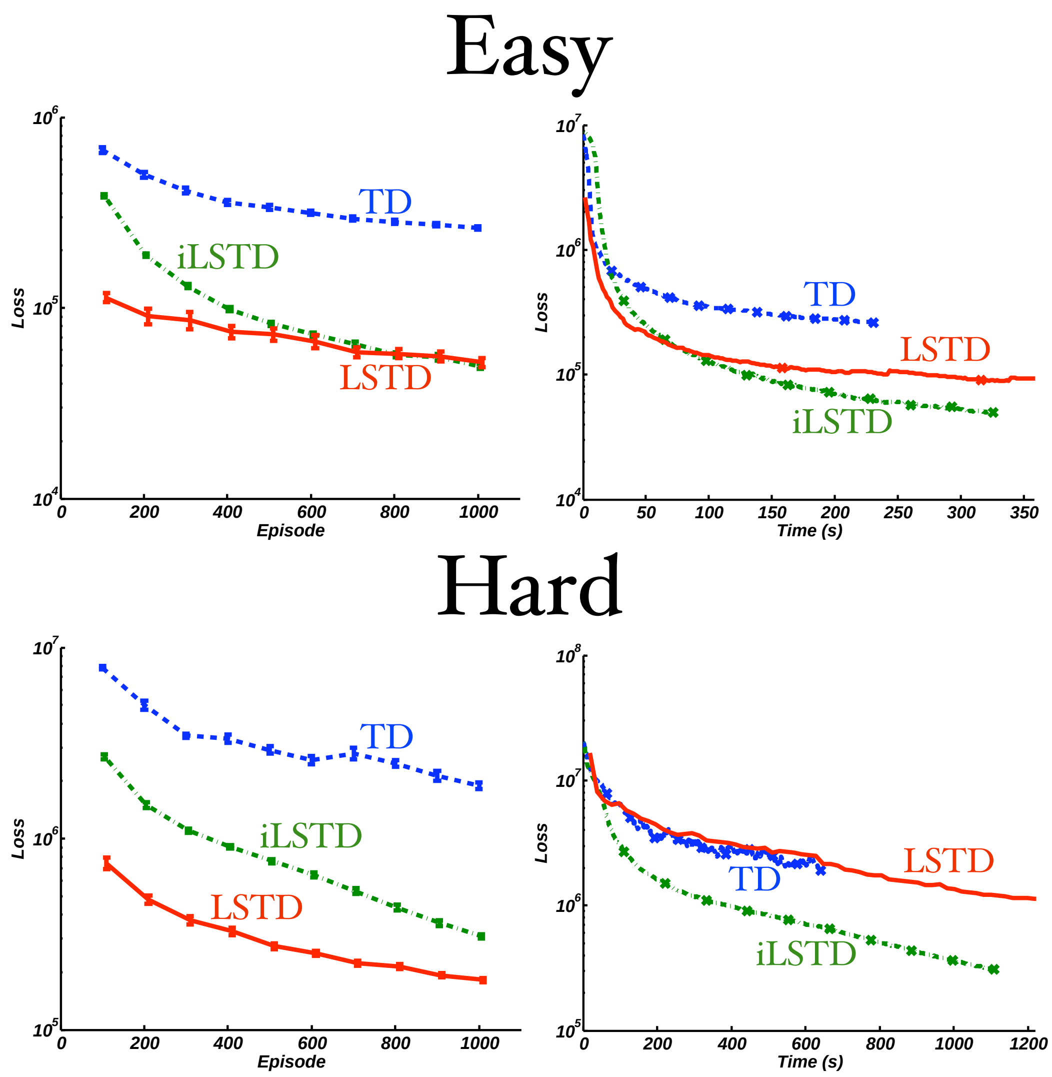

Figure 3.4: Performance of TD, LSTD and iLSTD methods in easy and hard mountain car problems averaged over 30 trials based on episode (left) and clock time (right).

the environment for each problem with $\lambda = 1$. On the left side, the horizontal axis shows the number of episodes, and each point on the graphs shows the loss function after 100 episodes averaged over 30 trials. On the right side, the horizontal axis represents the clock time, and each point on the graph represents the average loss function over 30 trials, while cross points indicate the completion of 100 episodes.

On the easy problem, based on episodes (Figure 3.4, top-left graph), least-squares methods performed considerably better than TD in terms of data efficiency. iLSTD even reached a level competitive with LSTD after 600 episodes. On the other hand, the top-right graph depicts the performance of the same set of methods, based on clock time. On the easy problem, after 100 seconds, iLSTD starts to perform better than LSTD, because it is able to look through more data. By the time that LSTD finished 100 episodes, iLSTD could process about 500 episodes and reach a lower loss. Like before, TD looked through even more data (about 700 episodes), but it could not perform as well.

The bottom part of Figure 3.4 shows the same set of results on the hard problem. Based on episodes, least-squares methods again outperformed TD (bottom-left), although the gap between iLSTD and LSTD is larger. Based on clock time (bottom-right), iLSTD outperformed all other methods, while interestingly LSTD could not outperform TD, and by the time that even iLSTD finished all of the 1000 episodes, LSTD failed to finish the first 100 episodes.

In order to compare the difficulty of these two problems, we examined the number of non-zero elements of the **A** and **b** matrices at the end of the 1000 episodes, which were (8803, 374) for the easy problem and (19038, 575) for the hard one. iLSTD picks one of the active features at each time step and descends in that dimension. Because the number of active features was larger in the hard problem, the effectiveness of each descent (iLSTD) compared to the matrix inversion (LSTD) was reduced, which explains the increased gap between the performances of the iLSTD and LSTD methods based on episodes (Figure 3.4, left graphs). However, at the end of 1000 episodes, iLSTD still reached a performance close to LSTD.

	clock time/step (msec)				
	Boyan chain			Mountain car	
Algorithm	Small	Medium	Large	Easy	Hard
TD	$0.22\pm6.0e\text{-}4$	$0.21\pm1.0e\text{-}4$	$0.21\pm1.0e\text{-}4$	$4.96\pm3.0e\text{-}3$	$5.02\pm3.0e\text{-}3$
iLSTD	$0.27\pm4.0e\text{-}4$	$0.27\pm1.0e\text{-}4$	$0.37\pm1.0e\text{-}4$	$7.01\pm3.0e\text{-}2$	$8.68\pm1.2e\text{-}2$
LSTD	$0.28\pm3.0e\text{-}4$	$0.33\pm1.0e\text{-}4$	$1.88\pm3.0e\text{-}4$	34.01 ± 2.95	$153.80\pm8.0e\text{-}1$

Table 3.1: The averaged clock time per step of the algorithms with eligibility traces used in the Boyan chain and mountain car problems.

Table 3.1 shows the clock time per step results for all of the Boyan chain and mountain car problems. All of the experiment were conducted on an AMD Opteron 250 processor (64 bit, 2.4GHz) with 8 GByte of RAM. As the problem size increased, methods performed slower accordingly. In all cased TD was the fastest method followed by iLSTD, while LSTD could only keep up with the speed of the other methods till the medium Boyan chain problem.

3.6 Alternative Solvers

There are many iterative linear system solvers which might be employed to incrementally solve $\mathbf{b} - \mathbf{A}\boldsymbol{\theta} = \mathbf{0}$. Does the iLSTD method simply take advantage of just one possible method? Might other methods perform better?

When devising the iLSTD update, we considered two facts:

- The algorithm must focus on real-time policy evaluation, thus it must be computationally efficient. In particular, we are interested in methods with linear running time in the number of features[4]. This constraint immediately rules out many methods like: gradient descent [*e.g.* See Avriel, 2003], steepest descent [*e.g.* See Eric W. Weisstein, 2002], conjugate gradient [*e.g.* See Noel *et al.*, 2006], biconjugate gradient [Barrett *et al.*, 1994], Minimal Residual (MINRES) [Paige and Saunders, 1975], Generalized Minimal Residual (GMRES) [Saad and Schultz, 1986], and Quasi-Minimal Residual (QMR) [Freund and Nachtigal, 1991]. Nevertheless, if updating a limited set

[4]Notice that solving the problem in one step through iterative matrix inversion costs $O(n^2)$.

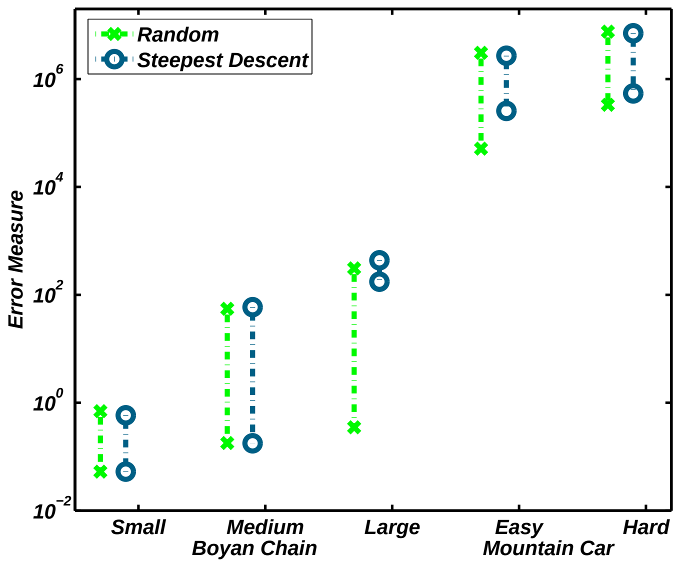

Figure 3.5: Performance of iLSTD using gradient descent in the original problem and steepest descent in the conditioned problem on small, medium, and large Boyan chain and easy and hard cases of mountain car. Both methods picked a dimension with non-zero value of $\boldsymbol{\mu}$ randomly. Each point represents the error averaged over the last 100 episodes after 100 (top) and 1000 (bottom) episodes based on 30 trials.

of components is considered, gradient descent and steepest descent can satisfy this condition.

- As mentioned by Boyan [1999], $\mathbf{A}$ is an asymmetric matrix, while methods like steepest descent, MINRES, and conjugate gradient requires the matrix to be symmetric. In order to make them applicable, the original problem must be conditioned by $\mathbf{A}^T$:

$$\mathbf{A}^T\mathbf{A}\boldsymbol{\theta} - \mathbf{A}^T\mathbf{b} = 0$$

Among all of the available options only two methods satisfy both of these constraints: (1) gradient descent in a limited number of components of the original linear problem (*i.e.* the iLSTD algorithm), and (2) steepest descent in a limited number of components of the conditioned problem. We also would like to highlight this fact that finding the dimension which minimizes the error the most for the conditional steepest descent requires $O(n^2)$ complexity, where as for gradient descent this can be accomplished in $O(n)$.

In order to see whether the disadvantage of conditioning the problem can be compensated by the fact that steepest descent does not require the step size parameter, we put both

	clock time/step (msec)				
	Boyan chain			Mountain car	
Algorithm	Small	Medium	Large	Easy	Hard
Gradient Descent	0.302±5.0e-4	0.304±1.0e-4	0.409±6.0e-4	7.25±3.0e-3	8.97±2.0e-3
Steepest Descent	0.312±5.0e-4	0.311±2.0e-4	0.404±3.0e-4	7.52±3.0e-3	9.97±3.0e-3

Table 3.2: The averaged clock time per step of the iLSTD methods using gradient descent for original problem and steepest descent for conditioned problem in the Boyan chain and mountain car problems.

methods in various environments and compared their performance and running time per step. Figure 3.5 depicts the performance of iLSTD using gradient descent in the original problem and steepest descent in the conditioned problem used in small, medium, and large Boyan chain and easy and hard cases of mountain car. The horizontal axis specifies the environment, while the vertical axis shows the loss measure. The measure was the root mean square error (RMS) of all state values for the Boyan chain problems and $||\mathbf{b}^* - \mathbf{A}^*\boldsymbol{\theta}||_2$ for the mountain car problems as described in Section 3.5. Recall that for the Boyan chain domain, there exists a unique solution for each problem, which was the base for computing the RMS error. After each interaction with the environment, θ was updated only once ($m = 1$), and for gradient descent the best α_0 and N_0 were found empirically according to Section 3.5. Also, dimensions were picked randomly among the non-zero indices of $\boldsymbol{\mu}$ for both methods. Each point on the graph shows the error averaged over the last 100 episodes after 100 (top) and 1000 (bottom) episodes based on 30 trials. Table 3.2 shows the corresponding per time step running time of all of these experiments.

Although on the small and medium Boyan chain problems, steepest descent showed promising results, as the size of the problem increased, it started to suffer from the conditioning, and did not perform as well as the gradient descent method. Also, because conditioning the problem reduces the sparsity of the matrix, time-wise, the steepest descent method was also more computationally expensive in general.

3.7 Conclusion

The computational demands of LSTD make it inapplicable to domains with a large number of features. On the other hand, when the feature representation is sparse, iLSTD can achieve results competitive with LSTD with computational demands that rival the time efficient TD method ($O(n)$ vs. $O(k)$). Based on clock time, when the problem gets large enough, iLSTD can achieve better results than LSTD, because it can look though more data. Although TD is faster than iLSTD, it uses data inefficiently and so does not fully take the advantage of looking through more data. Thus, even based on clock time it does not outperform iLSTD or LSTD. We studied the convergence analysis of the iLSTD method with the uniform random and non-zero random selection mechanisms and proved their convergence to the fixed point solution of TD. We also looked through different alternative methods that can be fused with iLSTD for reducing the error. The steepest descent method on the conditioned problem was the only possible alternative which satisfies our desired properties, and it has the additional advantage of not needing a step size parameter. In our experiments conditioning the problem hurt the performance enough to make the gradient descent method a superior choice.

Chapter 4

iLSTD with Eligibility Traces

One natural extension of iLSTD is to take advantage of eligibility traces. This chapter introduces the iLSTD(λ) algorithm for which iLSTD is the special case where $\lambda = 0$. This extension can potentially harm the per time step complexity, but as we will see, the additive cost is minor and iLSTD(λ)'s running time is still linear in the number of features. We also present a convergence analysis of the algorithm in the general case. Finally we investigate the effect of λ in the small Boyan chain and the hard mountain car problems. Also, we show that the running time of TD(λ), iLSTD(λ), and LSTD(λ) coincide with our theoretical analysis.

4.1 Algorithm

In Chapter 2, we discussed how Boyan added the idea of eligibility traces to the LSTD algorithm. Following that, in Chapter 3, we introduced iLSTD which uses the same matrices as LSTD (**A** and **b**), but instead of computing $\mathbf{A}^{-1}$, iLSTD takes steps to reduce the error gradually. As expected, the same matrices used for LSTD(λ) can be used for the iLSTD(λ)

algorithm:

$$\begin{aligned}
\boldsymbol{\mu}_t(\boldsymbol{\theta}) &= \sum_{i=1}^{t} \mathbf{u}_i(\boldsymbol{\theta}) = \sum_{i=1}^{t} z_i \left(r_{i+1} + \gamma V_{\boldsymbol{\theta}}(s_{i+1}) - V_{\boldsymbol{\theta}}(s_i)\right) \\
&= \sum_{i=1}^{t} z_i \left(r_{i+1} + \gamma \boldsymbol{\phi}_{i+1}^T \boldsymbol{\theta} - \boldsymbol{\phi}_i^T \boldsymbol{\theta}\right) \\
&= \underbrace{\sum_{i=1}^{t} z_i r_{i+1}}_{\mathbf{b}_t} - \underbrace{\sum_{i=1}^{t} z_i (\boldsymbol{\phi}_i - \gamma \boldsymbol{\phi}_{i+1})^T}_{\mathbf{A}_t} \\
&= \mathbf{b}_t - \mathbf{A}_t \boldsymbol{\theta}.
\end{aligned}$$

Following the derivations of LSTD(λ) by Boyan [2002], given a newly observed reward and transition, $\mathbf{A}$ and $\mathbf{b}$ can be computed iteratively as:

$$\begin{aligned}
\mathbf{b}_t &= \mathbf{b}_{t-1} + \underbrace{r_t \mathbf{z}_t}_{\Delta \mathbf{b}_t}, \\
\mathbf{A}_t &= \mathbf{A}_{t-1} + \underbrace{\mathbf{z}_t (\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})^T}_{\Delta \mathbf{A}_t}, \qquad (4.1)
\end{aligned}$$

in which $\mathbf{z}_t$ is the eligibility trace vector, and is computed incrementally as:

$$\mathbf{z}_t = \gamma \lambda \mathbf{z}_{t-1} + \boldsymbol{\phi}_t. \qquad (4.2)$$

With the new definitions of $\Delta \mathbf{A}_t$ and $\Delta \mathbf{b}_t$, the equations for the iterative computation of $\boldsymbol{\mu}_t$ remains the same as in iLSTD:

$$\begin{aligned}
\boldsymbol{\mu}_t(\boldsymbol{\theta}_t) &= \boldsymbol{\mu}_{t-1}(\boldsymbol{\theta}_t) + \Delta \mathbf{b}_t - (\Delta \mathbf{A}_t) \boldsymbol{\theta}_t, \\
\boldsymbol{\mu}_t(\boldsymbol{\theta}_{t+1}) &= \boldsymbol{\mu}_t(\boldsymbol{\theta}_t) - \mathbf{A}_t (\Delta \boldsymbol{\theta}_t), \qquad (4.3)
\end{aligned}$$

and the same method for updating $\boldsymbol{\theta}$ can be used.

Algorithm 6 contains the pseudo-code for iLSTD(λ) and highlights the computational complexity of certain lines of the algorithm. Line 5 updates $\mathbf{z}$ according to Equation 4.2, and Lines 6–9 incrementally compute the $\mathbf{A}_t$ and $\boldsymbol{\mu}_t$ as described in Equations 4.1 and 4.3. As with iLSTD, any feature selection mechanism can be employed in Line 11 to select a dimension of the sum TD update vector ($\boldsymbol{\mu}$). Line 12 then takes a step in the selected dimension, and Line 13 updates the $\boldsymbol{\mu}$ vector accordingly. The iLSTD algorithm (Algorithm 5) can be recovered by simply setting λ to zero. In the next section, we show

Algorithm 6: iLSTD(λ) Complexity

```
0  $s \leftarrow s_0, \mathbf{z} \leftarrow \mathbf{0}, \mathbf{A} \leftarrow \mathbf{0}, \boldsymbol{\mu} \leftarrow \mathbf{0}, t \leftarrow 0$
1  Initialize $\boldsymbol{\theta}$ arbitrarily
2  repeat
3    Take action according to $\pi$ and observe $r$, $s'$
4    $t \leftarrow t + 1$
5    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \boldsymbol{\phi}(s)$                                  $O(lk)$
6    $\Delta\mathbf{b} \leftarrow \mathbf{z}r$                                              $O(lk)$
7    $\Delta\mathbf{A} \leftarrow \mathbf{z}(\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s'))^T$                       $O(lk^2)$
8    $\mathbf{A} \leftarrow \mathbf{A} + \Delta\mathbf{A}$                                      $O(lk^2)$
9    $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \Delta\mathbf{b} - (\Delta\mathbf{A})\boldsymbol{\theta}$                        $O(lk^2)$
10   for $i$ from 1 to m do
11     $j \leftarrow$ choose an index of $\boldsymbol{\mu}$ using some *feature selection mechanism*
12     $\theta_j \leftarrow \theta_j + \alpha\mu_j$                                    $O(1)$
13     $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\mu_j\mathbf{A}\boldsymbol{e}_j$                              $O(n)$
14   end for
15   $s \leftarrow s'$
16 end repeat
```

that adding eligibility traces does not dramatically increase the per time step complexity of the algorithm. The algorithm still requires only linear computation in the number of features per time step.

4.2 Time Analysis

We now examine the time complexity of the iLSTD(λ) algorithm.

Theorem 7. *Assume that the feature selection mechanism takes $O(n)$ computation. If there are n features, and for any given state s, $\boldsymbol{\phi}(s)$ has at most k non-zero elements, then the iLSTD(λ) algorithm requires $O(mn + lk^2)$ computation per time step.*

Proof. Outside of the inner loop, Lines 7–9 are the most computationally expensive steps of iLSTD(λ). Since we assumed that each feature vector has at most k non-zero elements, and the $\mathbf{z}$ vector can have up to lk non-zero elements, the $\mathbf{z}\,(\boldsymbol{\phi}(s) - \gamma\boldsymbol{\phi}(s'))^T$ matrix (Line 7) has at most $2lk^2$ non-zero elements. This leads to $O(lk^2)$ complexity for Lines 4–9. Inside the inner loop, the complexity remains unchanged from iLSTD to iLSTD(λ) with Line 13 as the most expensive line. Because $\boldsymbol{\mu}$ and $\mathbf{A}$ do not have any specific structure, the complexity of this line is $O(n)$.[1] Thus, the final bound for the per time step complexity

[1]Note that $\mathbf{A}\boldsymbol{e}_j$ selects the jth column of $\mathbf{A}$ and so does not require the usual quadratic time for multiplying

of the iLSTD(λ) algorithm is $O(mn + lk^2)$.[2] $\square$

4.3 Convergence Analysis

We studied the iLSTD(λ) algorithm's proof of convergence with $\lambda = 0$ in Chapter 3. This section extends the previous analysis in case of $\lambda \neq 0$.

Theorem 8. *If the Markov decision process is finite, and an appropriate α decay schedule is used, iLSTD(λ) with any feature selection mechanism which satisfies the following assumption converges to the same result as TD(λ).*

A1. Let $P_t(i)$ be the probability of selecting the ith dimension of $\boldsymbol{\mu}_t$ at time t, then
$\exists \xi \in \mathbb{R}, \forall t \in \mathbb{N}$, *such that* $\forall i \in \{1, \ldots, n\}$ *if* $(\boldsymbol{\mu}_t(i) \neq 0) \Rightarrow 0 < \xi \leq P_t(i) \leq 1$.

Proof. Incorporating eligibility traces to iLSTD only affects the definitions of **A** and **b**. This means that Lemma 2 still holds regardless of the value of λ. We incorporate the same mapping used for Theorem 4. In Zinkevich's original work [2006], it was shown that iLSTD satisfies Assumptions 2 and 3 of Lemma 2 regardless of the value of λ. The proof for the rest of the assumptions remains unchanged. $\square$

4.4 Experimental Results

In this section, we examine the effect of λ on the performance of TD(λ) LSTD(λ), and iLSTD(λ) in two problems: the small Boyan chain and hard mountain car. We also discuss the effect of λ on the running time of the methods.

For all of the experiments, the step size α, takes the same form as the set of experiments in Chapter 3. The choice of N_0 and α_0 for TD(λ) and iLSTD(λ) was based on experimentally finding the best $\alpha_0 \in \{0.01, 0.1, 1\}$ and $N_0 \in \{100, 1000, 10^6\}$ for each algorithm and λ value separately. For better use of sparse matrices, traces less than 10^{-4} were set to zero. The rest of the settings remained unchanged.

a vector by a square matrix.

[2]With a very small threshold, **z** is not going to be sparse which translates into $O\big((m + k)n\big)$ per time step complexity for iLSTD(λ).

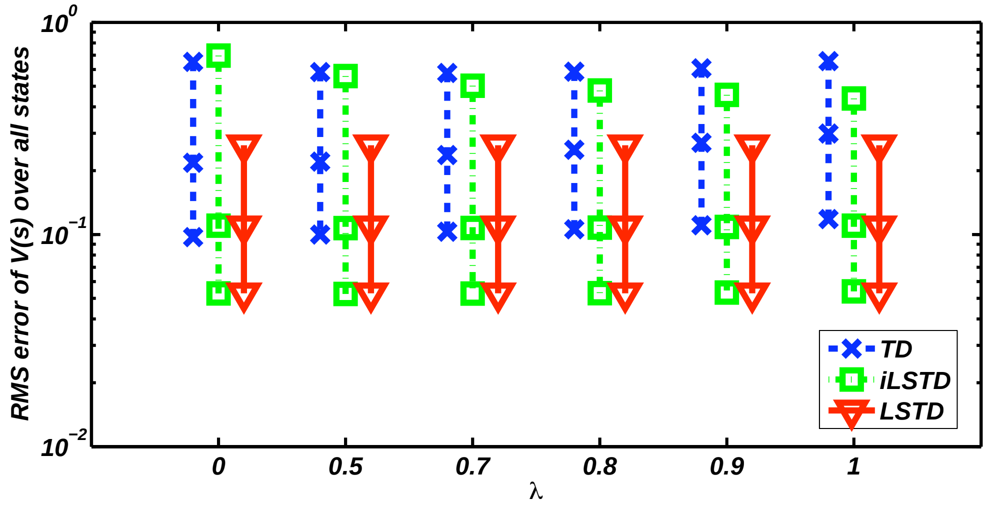

Figure 4.1: Performance of TD(λ), iLSTD(λ), and LSTD(λ) algorithms in the small Boyan chain problem with 6 different lambda values. Each point represents the RMS error after 100 (top), 200 (middle), and 1000 (bottom) episodes averaged over last 100 episodes respectively. Results are also averaged over 30 trials.

4.4.1 Small Boyan Chain Problem

We first examine the small Boyan chain problem. Each of the methods were tested with $\lambda \in \{0, 0.5, 0.7, 0.8, .9, 1\}$. Figure 4.1 shows their performance. The vertical axes represents the root mean square (RMS) error of the values over all states compared to the true values in logarithmic scale,[3] while the horizontal line indicates the λ values. Each point represents the average of performance over the last 100 episodes after 100 (top), 200 (middle), and 1000 (bottom) episodes averaged over 30 trials.

As expected, LSTD(λ) required the least amount of data, obtaining a low error after only 100 episodes. With only 200 episodes, though, iLSTD(λ) performed as well as LSTD(λ), and dramatically outperformed TD(λ). Although, λ did not play a significant role for LSTD(λ), which matches the observation of Boyan [1999], $\lambda > 0$ showed slight improvement in the performance of iLSTD(λ).

[3]As mentioned in Chapter 3, unique solutions are in the space spanned by feature vectors.

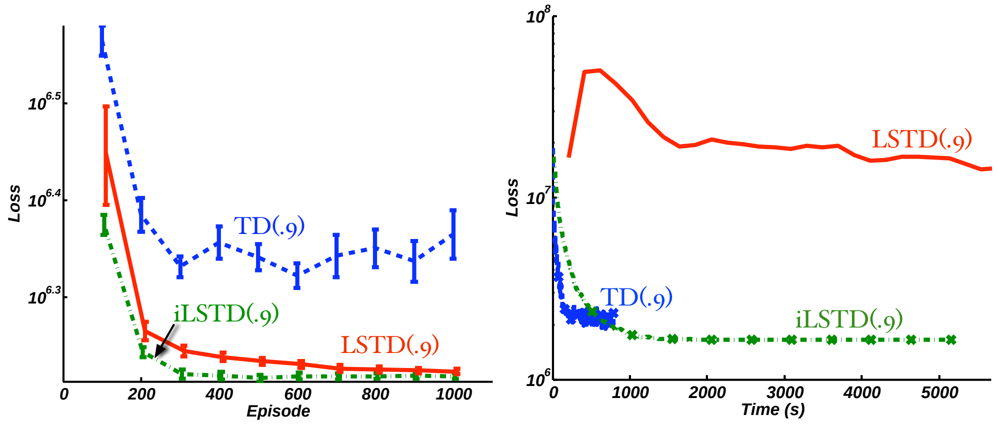

Figure 4.2: Performance of the TD(.9), iLSTD(.9), and LSTD(.9) algorithms in hard mountain car. Results are averaged over 30 trials. From left to right, graphs show results based on episode and clock time, respectively.

4.4.2 Hard Mountain Car Problem

Our second test case is the hard mountain car problem explained in Chapter 3 with the same parameter settings. Figure 4.2 shows the results of the TD(λ), iLSTD(λ), and LSTD(λ) methods with $\lambda = .9$ in this problem. The horizontal axis shows the number of episodes (left) and clock-time (right), while the vertical axis represents our loss function in logarithmic scale. The loss was $||\mathbf{b}^* - \mathbf{A}^*\boldsymbol{\theta}||_2$, where $\mathbf{A}^*$ and $\mathbf{b}^*$ were computed from 200,000 episodes of interaction with the environment. In computing the loss function, we set $\lambda = 1$ in order to get unbiased estimates of $\mathbf{A}$ and $\mathbf{b}$ matrices and made it possible to compare the results of algorithms with different λ values. In this problem, with $\lambda = .9$, LSTD performed worse than iLSTD in terms of data efficiency (Figure 4.2-left), although after 1000 episodes their results are close. Clock-wise, LSTD could not beat either TD or iLSTD (Figure 4.2-right).

Figure 4.3 depicts the results for all of the methods in the hard mountain car problem with two different λ values. Each line represents the loss averaged over the last 100 episodes after 100 (top) and 1000 (bottom) episodes. Based on our loss function, by changing λ from 0 to .9, TD performed better, while iLSTD and LSTD performed worse. iLSTD is still superior to TD in both cases, but its advantage narrowed with the large λ. For TD, having

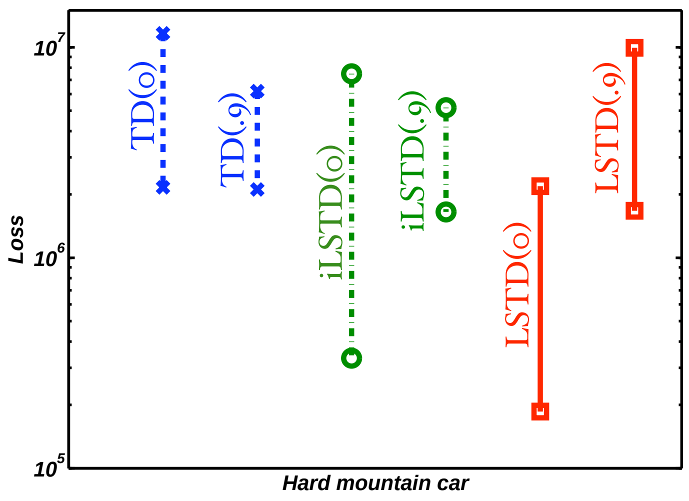

Figure 4.3: Performance of TD(λ), iLSTD(λ), and LSTD(λ) algorithms in hard mountain car problem with $\lambda \in \{0, .9\}$. Each point represents the loss averaged over last 100 episodes and 30 trials after 100 (top) and 1000 (bottom) episodes.

$\lambda = .9$ improved the algorithm by updating weights corresponding to more than one state at each time step, which propagates the TD error faster. However, least-square methods look through the whole experience even with $\lambda = 0$ and so they dont benefit from this feature of eligibility traces. On the other hand, eligibility traces reduces the bias and increase the variance at the same time, which can explain the worse results of least-square methods with $\lambda = .9$.

Table 4.1 shows the averaged running time of the last two experiments. We did not observe any significant difference between the running time of the algorithms with different λ values in small Boyan chain, so we report the average of all runs. However, in the hard mountain car problem the gap is wide, so we report the running times for each λ value separately. In the large problem, the difference between running times became more apparent. By adding eligibility traces, TD's running time increased slightly, although for least-square methods the jump was more noticeable. The slower speed of the algorithms with $\lambda = .9$ can be explained with the fact that adding eligibility traces reduces the sparsity of the update matrices which translates into higher computational cost. In the mountain car environment the $\frac{n}{k}$ is relatively large ($\frac{\sim 500}{10} =\sim 50$), which translates into a significant time improvement of iLSTD(λ) over LSTD(λ).

	clock time/step (msec)		
	Small Boyan chain	Hard mountain car	
Algorithm	λ	λ= 0	λ= .9
TD(λ)	0.305±7.0e-4	5.02±2.0e-3	6.20±2.0e-3
iLSTD(λ)	0.370±7.0e-4	8.68±1.0e-2	40.35±2.0e-1
LSTD(λ)	0.367±7.0e-4	153.77±9.0e-1	1608.40±9.8

Table 4.1: The averaged clock time per step of TD(λ), iLSTD(λ), and LSTD(λ) algorithms used in Small Boyan chain and hard mountain car problems. All results are based on 30 trials.

	Environment				
	Boyan chain			Mountain car	
Parameter	Small	Medium	Large	Easy	Hard
Active features per step (k)	2	2	2	10	10
Total active features (n)	4	25	100	$\sim$400	$\sim$600
Total features	4	25	100	10000	10000

Table 4.2: Feature specification of the environments.

4.4.3 Comparison of Theoretical and Empirical Ratios

So far, we investigated the per time step computational complexities of the iLSTD(λ) and LSTD(λ) methods through theoretical analysis and empirical results. In this section, we show that these results coincide with each other.

Table 4.2 presents the feature specification of all tested environments. To compare the experimental results with our theoretical results, we examined the running time ratios of LSTD and iLSTD algorithms in Figure 4.4. The horizontal axis represents the problem, while the vertical axis shows the ratio value. The ratios indicated by circles show the empirical timing ratios based on Tables 3.1 and 4.1, while those being represented by triangles stand for the theoretical timing ratios, $\frac{O(n^2)}{O(nk)} \approx \frac{n}{k}$,[4] based on Table 4.2 for all environments. The theoretical and empirical results are consistent and they follow the same pattern through all environments.

4.5 Conclusion

This chapter incorporated the idea of eligibility traces with iLSTD and showed that the iLSTD(λ) algorithm still has a linear per time step complexity in the number of features.

[4]Since the cut-off threshold used for the experiments was so small, we considered $O\big((k+m)n\big)$ as the per time step complexity of iLSTD(λ).

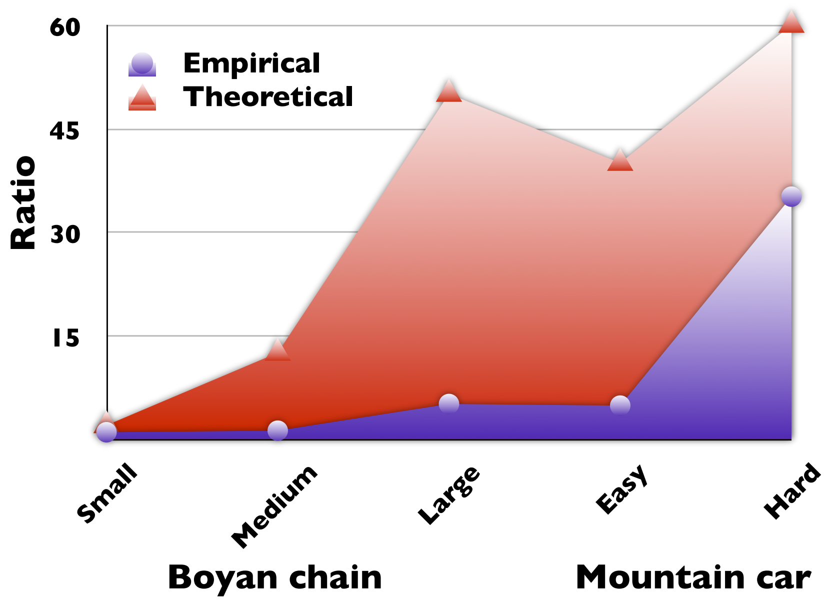

Figure 4.4: The comparison of the $\frac{n}{k}$ ratios and the experimental running time ratios of LSTD and iLSTD in the Boyan chain and mountain car domains. In general, both ratios are coherent with each other.

We showed that theoretical results for iLSTD is valid for iLSTD(λ). We also investigated the effect of λ through empirical experiments with TD(λ), iLSTD(λ), and LSTD(λ) in the small Boyan chain and hard mountain car problems. Setting $\lambda > 0$ slightly improved the performance of iLSTD(λ) in the Boyan chain environment, yet based on our loss function, $\lambda = .9$ hurt the performance of least-square methods in the mountain car compared to $\lambda = 0$. On the other hand, adding the eligibility traces reduces the sparsity of matrices, which increases the running time of all methods. This extra cost mostly affected the LSTD methods.

Chapter 5

Dimension Selection Alternatives

In Chapters 3 and 4, we used non-zero random selection as the feature selection mechanism for the iLSTD algorithm in all experiments. Although, this method is easy to implement, other mechanisms may perform better. In this chapter, we investigate three more feature selection mechanisms: greedy, ϵ-greedy, and Boltzmann. After discussing about the convergence property of each method, we compare the performance and computational demands of the methods in the Boyan chain and the mountain car environments.

5.1 Greedy

One alternative for picking dimensions is to act greedily with respect to the absolute value of each component and break ties randomly. Hence, on each iteration, the element with the highest absolute value of the sum TD update is chosen (*i.e.*, $\mathrm{argmax}_j |\boldsymbol{\mu}_t(j)|$). This approach has a resemblance to prioritized sweeping [Moore and Atkeson, 1993], but rather than updating the *state* with the largest TD update, we choose to update the *parameter component* with the largest TD update. Like prioritized sweeping, iLSTD can tradeoff data efficiency and computational efficiency by increasing m, the number of components updated per time step. In terms of convergence, though, greedy selection does not meet the assumption of Theorem 4,[1] so has no guarantee of convergence. As we will see in the next section, though, greedy selection performs quite well in some cases, despite this lack of asymptotic guarantee.

[1] In the greedy mechanism, a non-zero probability of selecting each dimension with non-zero TD update does not exist. Thus normalizing α can not make it satisfy Assumption 1 of Theorem 4.

5.2 Non-zero ϵ-greedy

By adding an ϵ parameter to the greedy selection rule, both random and greedy methods can be merged into an ϵ-greedy selection mechanism:

$$i \leftarrow \begin{cases} \text{argmax}_j \, |\boldsymbol{\mu}_t(j)| & \text{with } 1-\epsilon \text{ probability} \\ \text{random}(1, n) & \text{with } \epsilon \text{ probability} \end{cases}$$

Setting ϵ to zero and one, one can obtain random and greedy selection methods, respectively. We also restrict our random selection to non-zero elements of $\boldsymbol{\mu}_t$. Suppose that $\boldsymbol{\mu}_t$ has m_t non-zero elements, of which v_t elements are maximal. Given that ties are broken randomly between maximal components, the probability of selecting the ith element of $\boldsymbol{\mu}_t$ is:

$$P_t(i) = \begin{cases} \frac{1-\epsilon}{v_t} + \frac{\epsilon}{m_t} & i = \max_i |\boldsymbol{\mu}_t(i)| \\ \frac{\epsilon}{m_t} & \text{Otherwise} \end{cases}$$

If $\epsilon \neq 0$ and α_t is normalized by $q_t = \frac{1}{P_t(i)n}$, then iLSTD(λ) with the non-zero ϵ-greedy mechanish satisfies the assumption of Theorem 4, therefore it converge to the fixed point solution of TD(λ).

5.3 Boltzman Distribution

Another way of adding stochasticity to the dimension selection rule is to use the Boltzmann distribution function [*e.g.*, see Sutton and Barto, 1998] for selecting non-zero indices of $\boldsymbol{\mu}_t$. In this method, dimensions with higher sum TD updates have more chance of getting selected, but still the selection involves stochasticity. The probability of selecting each dimension is computed by:

$$P_t(i) = \begin{cases} \dfrac{e^{\frac{|\boldsymbol{\mu}_t(i)|}{\tau}}}{\sum_{j=1, \boldsymbol{\mu}(j) \neq 0}^{n} e^{\frac{|\boldsymbol{\mu}_t(j)|}{\tau}}} & \boldsymbol{\mu}(i) \neq 0 \\ 0 & \boldsymbol{\mu}(i) = 0 \end{cases},$$

in which the τ parameter behaves as the temperature. If $\tau \to \infty$, the selection mechanism behaves like non-zero random selection; if $\tau \to 0$, it acts greedily. Compared to previous methods this method is more complicated, thus it is expected to run slower than the other selection rules. Because there exist no lower bound for $P_t(i)$, this mechanism does not meet the assumption of Theorem 4 and therefore has no proof of convergence.

One solution is to mix the Boltzmann and non-zero random selection methods together. Given a parameter $0 < \psi \leq \frac{1}{n}$ and m_t as the number of non-zero elements of $\boldsymbol{\mu}_t$, then each non-zero element has a ψ probability of getting selected, and the rest of $(1 - m_t\psi)$ probability is distributed among non-zero dimensions following Boltzmann distribution. The new mechanism has ψ as a lower bound of $P_t(i)$,[2] thus it is guaranteed to converge to the solution of TD(λ) by using the normalization term $\frac{1}{P_t(i)n}$. The probability function takes the following form:

$$P_t(i) = \begin{cases} \psi + (1 - m_t\psi)\dfrac{e^{\frac{|\boldsymbol{\mu}_t(i)|}{\tau}}}{\sum_{j=1,\boldsymbol{\mu}(j)\neq 0}^{n} e^{\frac{|\boldsymbol{\mu}_t(j)|}{\tau}}} & \boldsymbol{\mu}(i) \neq 0 \\ 0 & \text{Otherwise} \end{cases} .$$

For the empirical results, we used the new defined probability function as the Boltzmann feature selection mechanism.

5.4 Empirical Results

In this section, we study the performance of iLSTD(λ) with the random, greedy, ϵ-greedy, and Boltzmann feature selection mechanisms. For ϵ-greedy, ϵ was set to .1 and for Boltzmann, $\tau = 1$ and $\psi = 10^{-6}$. Experiments were conducted in the small, medium and large Boyan chain domain and easy and hard cases of the mountain car environment. All of the parameter settings remained the same from the previous chapters. For each method α_0 and N_0 were set to the best combination of $\alpha_0 \in \{0.01, 0.1, 1\}$ and $N_0 \in \{100, 1000, 10^6\}$.

Figure 5.1 shows the empirical results. The horizontal axis identifies the problem, while the vertical axis shows the error measure in logarithmic scale as described in Section 3.5.2 of Chapter 3. Each graph shows the averaged performance of each method over last 100 episodes, after 100 (top) and 1000 (bottom) episodes respectively. Results are also averaged over 30 trials, and all of the methods experienced the same set of trajectories.

Aside from the hard mountain car probelm, the greedy mechanism outperformed all of the other methods. This is interesting, as this is the only method which lacks a proof

[2]*i.e.*, $\forall t \in \mathbb{N}, \forall i \in \{1..n\}, 0 < \psi \leq P_t(i)$.

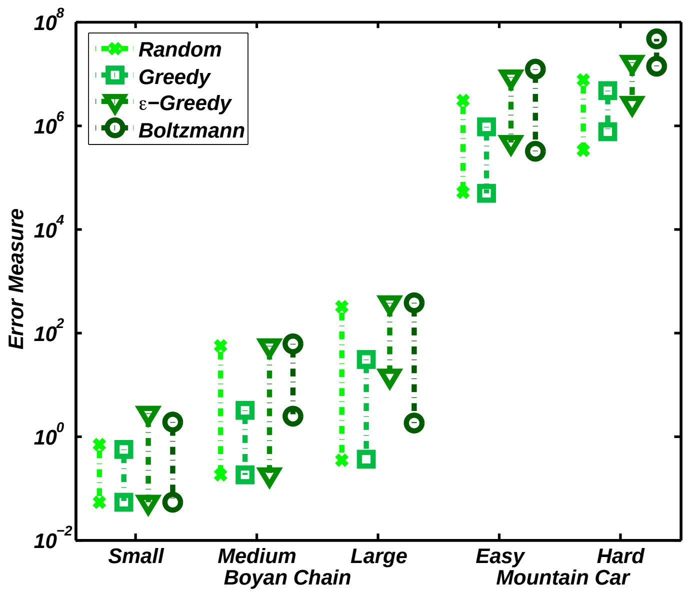

Figure 5.1: Performance of iLSTD with random, greedy, ϵ-greedy, and Boltzmann feature selection methods in small, medium and large Boyan chain and easy and hard cases of mountain car averaged over 30 trials. Each point represents the averaged performance over the last 100 episodes after 100 (top) and 1000 (bottom) episodes respectively.

of convergence, although intuitively, descending in the steepest dimension seems to be a good idea. For the hard mountain car problem, we found a set of parameters, with which the greedy method performed better than random, but that parameter setting was unstable and in rare cases the error would diverge quickly. So, the results of the greedy method on this specific problem are based on the second best parameter setting which was stable on all runs. This instability of the greedy method maybe related to its lack of a convergence guarantee.

The ϵ-greedy and Boltzmann mechanisms, on the other hand, performed worse than the random method in all cases. We suspect that this is due to the normalization factors. Although they are necessary to satisfy the convergence property, they can force large jumps on rarely selected features (*i.e.*, when $P_t(i)$ is close to zero), which is risky at the same time. We observed that in some runs, these methods reached higher error after 200 episodes compared to their performance after 100 episodes. In order to make sure that this phenomenon was because of the normalization factors, we ran the same experiments without the use of these coefficients and in all cases the algorithms reduced the error in a steady fashion, although they were not stable on all runs. The poor performance after 200 episodes is also logical as the $\boldsymbol{\mu}_t$ vector is quite noisy in the beginning and the normalization factors can cause large changes in rarely selected dimensions, but as $\boldsymbol{\mu}_t$ becomes more and more accurate, these jumps become less risky.

By increasing the ϵ and τ parameters, both methods involve more stochasticity and become more similar to the random method. Therefore, it is expected that they can at least perform as well as the random method and maybe better. In order to investigate this issue, we ran all of random, greedy, ϵ-greedy, and Boltzmann mechanisms in the large Boyan chain problem, in which we observed noticeable differences. Also the ϵ and τ parameters were varied according to $\{0.1, 0.3, 0.5, 0.9\}$ and $\{1, 10, 100, 1000\}$ sets, respectively. Figure 5.2 depicts the empirical results averaged over 30 trials. Each point represents the RMS of all values averaged over the last 100 episodes after 100 (top) and 1000 (bottom) episodes. Like before, the best combination of α_0 and N_0 were found throughout the empirical tests individually. As we expected, both ϵ-greedy and Boltzmann mechanisms could eventually reach the performance of the random method, although they did not outperform it. As ϵ increased, ϵ-greedy mechanism behaved more like the random method, but we could not see the same effect for the Boltzmann method. After looking at the results in more detail, we noticed that only after setting τ to 1000, could Boltzmann reach noticeable results with aggressive parameter settings ($\alpha_0 = 1, N_0 = 100$), while with $\tau < 1000$, the best results were obtained with less aggressive parameters ($\alpha_0 = .1, N_0 = 10^6$). Albeit the normalization factor guarantees the convergence of these methods, it also triggers large jumps which potentially can hurt the performance at the same time.

Table 5.1 together with Figure 5.3 show the running time for the experiments described above. All of the numbers for the Boyan chain environment are statistically significant and they match the theoretical complexity of each mechanism, with random having the lowest running time and Boltzmann having the highest one. As the algorithms switched to more complicated environments the extra cost of the Boltzmann method became more apparent.

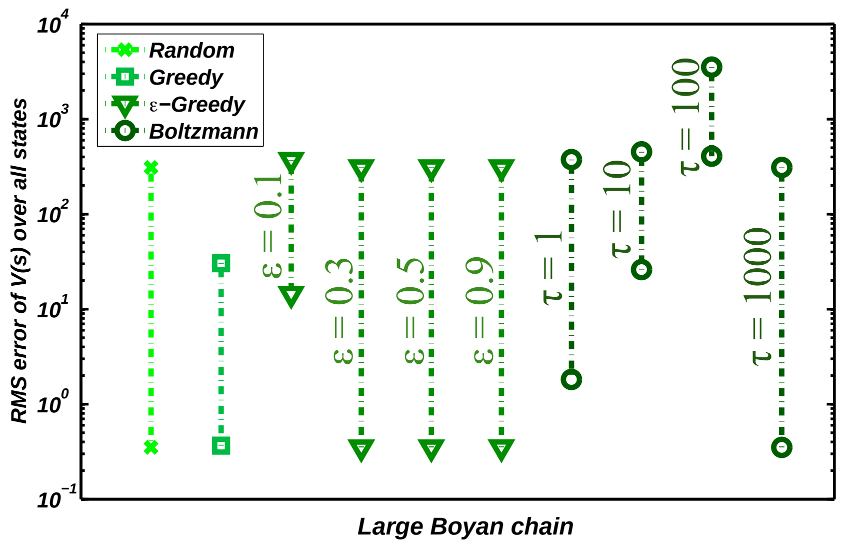

Figure 5.2: Performance of iLSTD with random, greedy, ϵ-greedy, and Boltzmann feature selection methods in the large Boyan chain problem averaged over 30 trials. Each point represents the averaged performance over the last 100 episodes after 100 (top) and 1000 (bottom) episodes respectively. The ϵ and τ parameters for ϵ-greedy and Boltzmann mechanisms were varied according to $\{0.1, 0.3, 0.5, 0.9\}$ and $\{1, 10, 100, 1000\}$ sets, respectively.

Although, in all cases the extra running times are small, but more complicated methods like ϵ-greedy and Boltzmann did not benefit from their extra running time in our experiments.

5.5 Conclusion

In this chapter, we investigated the use of various feature selection mechanisms. The greedy method, despite the lack of a convergence proof, demonstrated promising results. Although in our problems we observed some instability. The ϵ-greedy and Boltzmann mechanisms have convergence proofs using normalization, but based on our experiments, they did not perform as well as the random mechanism, only approaching it as stochasticity in the selection was increased.

	clock time/step (msec)				
	Boyan chain			Mountain car	
iLSTD	Small	Medium	Large	Easy	Hard
Random	0.296±6e-4	0.293±3e-4	0.382±1e-4	7.41±3e-2	8.79±2e-2
Greedy	0.307±5e-4	0.306±2e-4	0.405±1e-4	7.50±4e-2	8.88±2e-2
ϵ-greedy	0.309±3e-4	0.308±2e-4	0.410±2e-4	7.51±4e-2	8.90±2e-2
Boltzmann	0.358±6e-4	0.351±3e-4	0.460±5e-4	8.22±3e-2	9.50±2e-2

Table 5.1: The averaged clock time per step of the iLSTD algorithms with random, greedy, ϵ-greedy, and Boltzmann feature selection rules used in small, medium, and hard Boyan chain and easy and hard cases of mountain car problems.

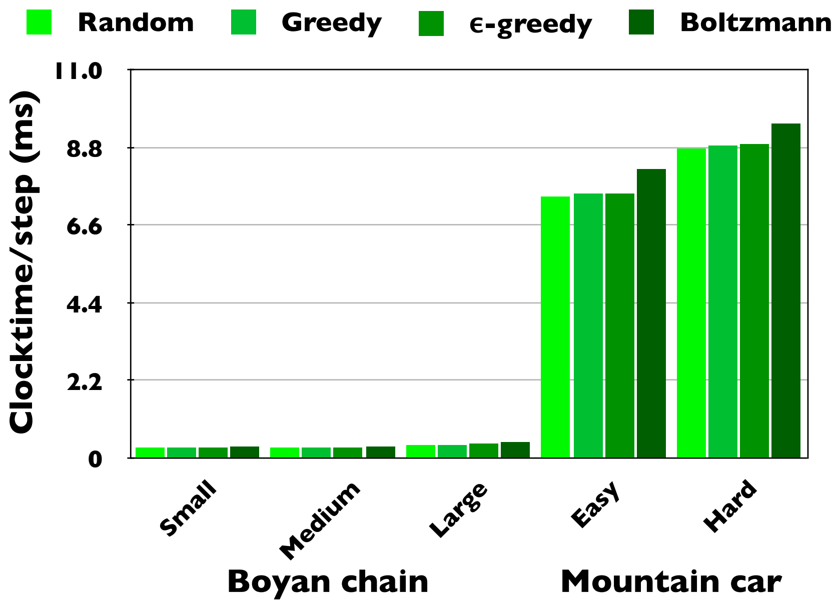

Figure 5.3: The averaged CPU time per step of the iLSTD algorithms with random, greedy, ϵ-greedy, and Boltzmann feature selection rules used in small, medium, and hard Boyan chain and easy and hard cases of mountain car problems.

Chapter 6

Conclusions and Future Work

In this chapter, we highlight the contributions and limitations of this research and finally, we conclude by discussing possible threads for future works.

6.1 Contributions

This dissertation introduced a novel algorithm, called iLSTD, for online policy evaluation. With the use of eligibility traces, this method was extended to iLSTD(λ), which holds the linear running time property. We investigated the running time, convergence, and empirical performance of iLSTD compared to the TD and LSTD methods. Finally, we studied the effect of using different dimension selection mechanisms with iLSTD and their convergence properties.

6.1.1 iLSTD Algorithm

In Chapter 3, we introduced incremental least-squares temporal difference learning (iLSTD) method as a solution to online policy evaluation in large problems when using a sparse feature representation. It has the combined advantages of TD and LSTD: its per time step complexity is linear in the number of features, and it makes use of all experiences. These two properties lead to a fast algorithm which can be applied to large problems with sparse feature representations and achieve comparable results with the LSTD method which is computationally impractical. Although, unlike the LSTD algorithm, iLSTD requires a step size parameter selection, which suggests a thorough search or prior knowledge for optimal performance. Recall that iLSTD can also use steepest descent in the conditioned problem to get rid of the step size parameter, although, based on our experiments, it would not perform as well as the proposed iLSTD method. We also studied the convergence property of iLSTD to the limit point of TD. iLSTD is categorized as model-based reinforcement learning methods, because it builds up a transition and reward model as it interacts with the environment.

6.1.2 iLSTD with Eligibility Traces

In Chapter 4, we integrated eligibility traces with the iLSTD method and proved that this extension will not dramatically harm the per time step computational complexity of the algorithm. We also showed that all convergence analysis results for iLSTD is extendable to the general case of iLSTD(λ). Based on our empirical results, adding eligibility traces showed positive and negative effects to the performance of iLSTD. Although, the loss measure used for the mountain car problem was not accurate, which makes it hard to compare the results. In general, we think that setting $\lambda \neq 0$ for least-squares methods might be harmful, as it adds more variance to the results, while it can improve the performance in less Markovian problems by reducing the bias.

6.1.3 Empirical Results

Through Chapters 3 and 4, results demonstrated the performance of TD, iLSTD and LSTD methods in the Boyan chain and mountain car domains. Both in terms of clock time and data, iLSTD with only a small number of iterations per interaction $(m = 1)$ was superior to TD. While based on clock time, TD can look through more data, its inefficient use of the data hinders its speed advantage. As problems got large enough, iLSTD also overtook the performance lead from LSTD based on clock time, because its linear per-time step complexity allows it to look through more data and achieve a lower error than LSTD in the same amount of time. However, LSTD is still superior if the amount of data is the only restriction.

6.1.4 iLSTD and Dimension Selection Mechanisms

Finally, in Chapter 5, we studied three other dimension selection mechanisms: greedy, ϵ-greedy, and Boltzmann. It was shown that the last two methods have proofs of convergence using normalization factor for the step size parameter. Although, based on our search, their performance could not surpass the random selection method. On the other hand, the greedy method which lacks the proof of convergence, outperformed all of the other methods in most cases. It remains open as to which selection mechanisms satisfy the convergence assumptions and yet perform better than the random method.

6.2 Batch LSTD vs. iLSTD

While this thesis focused on online policy evaluation methods, one might wonder about the performance of iLSTD methods with respect to batch LSTD [1]. This means that LSTD can maintain the $\mathbf{A}$ and $\mathbf{b}$ matrices using incremental updates with $O(k^2)$ complexity and then, once in a while, it computes $\mathbf{A}^{-1}\mathbf{b}$ to obtain the weight vector. If the total number of steps between each inverse is t, then the total complexity of LSTD after T steps is $O(Tk^2+\frac{T}{t}n^3)$, while for iLSTD this complexity is $O(Tk^2+Tmn)$. If $t=O(\frac{n^2}{m})$, then they both have the same asymptotic complexity. This means that if LSTD takes the inverse after each $O(\frac{n^2}{m})$ steps, then it can par with iLSTD in terms of asymptotic running time. Notice that this value is relative to n^2, which means if we double the number of features while LSTD and iLSTD have the same total running time, LSTD must compute the weight vector 4 times less often than before. This is potentially harmful for the LSTD method as it tackles larger problems.

6.3 Future Work

6.3.1 Advances in Theoretical Analysis

For iLSTD, the greedy mechanism performed better than all other methods in most cases of our research, yet we could not find any proof of convergence for this specific algorithm. On the other hand, the other similar methods like ϵ-greedy, and Boltzmann could not out-

[1]Here, we analyze the iLSTD and LSTD methods without eligibility traces. The case with eligibility traces is similar.

perform the random method because of the normalization factors. The question is whether Assumption 4 of Lemma 2 can be relaxed, or changed to a more general assumption, such that the use of normalization factors becomes optional or less influential. In that case we might achieve methods with better performance while maintaining a guarantee of convergence.

6.3.2 More Realistic Problems

Domains used for our empirical results are interesting, yet they are still far from realistic domains. It would be appealing to examine the iLSTD method in yet more challenging tasks such as the keepaway soccer [Stone *et al.*, 2005].

6.3.3 Control

Although policy evaluation is an interesting research domain, ultimately, it is desirable to be combined with policy improvement methods to do control. We think that this extension step is straightforward as several works extended LSTD to the control problems such as RLSTD [Xu *et al.*, 2002], and LSPI [Lagoudakis and Parr, 2003], yet it needs some caution. One concern is that as the policy changes, the distribution of the visited state-action pairs can change dramatically. If the **A** and **b** matrices do not accommodate themselves with these changes, one can end up with a biased policy which might maximize the expected return in irrelevant parts of the state-action space. We suggest two alternatives for solving this issue. The policy improvement could happen only after certain number of time steps, and right after, the matrices should be zeroed. Another option is that the matrices which hold the model of the environment (*i.e.*, **A** and **b**), should decay over time to compensate for the change to the distribution of state-action pairs as done by Xu *et al.* [2002]. We are interested in trying these alternatives as we switch from policy evaluation to control, and see how they can be compared with the LSPI and RLSTD methods.

Bibliography

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

[Alpaydin, 2004] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[Avriel, 2003] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing, 2003.

[Barrett *et al.*, 1994] R. Barrett, M. Berry, Tony F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

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

[Bertsekas and Tsitsiklis, 1996] Dmitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Bowling and Veloso, 2002] M. Bowling and M. Veloso. Scalable learning in stochastic games, 2002.

[Bowling and Veloso, 2003] M. Bowling and M. Veloso. Simultaneous adversarial multi-robot learning, 2003.

[Boyan, 1999] Justin A. Boyan. Least-squares temporal difference learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56. Morgan Kaufmann, San Francisco, CA, 1999.

[Boyan, 2002] Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49:233–246, 2002.

[Bradtke and Barto, 1996] S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

[Eric W. Weisstein, 2002] Eric W. Weisstein. Method of Steepest Descent. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/MethodofSteepestDescent.html`, 2002.

[Freund and Nachtigal, 1991] Ronald W. Freund and Noël M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60:315–339, 1991.

[Geramifard *et al.*, 2006] Alborz Geramifard, Michael Bowling, and Richard S. Sutton. Incremental least-square temporal difference learning. In *The Twenty-first National Conference on Artificial Intelligence (AAAI)*, pages 356–361, 2006.

[Geramifard *et al.*, 2007] Alborz Geramifard, Michael Bowling, Martin Zinkevich, and Richard Sutton. iLSTD: Eligibility traces and convergence analysis. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[Jaakkola *et al.*, 1995] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 345–352. The MIT Press, 1995.

[Jain *et al.*, 1999] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[Kohonen, 1988] Teuvo Kohonen. An introduction to neural computing. *Neural Networks*, 1(1):3–16, 1988.

[Kohonen, 2001] Teuvo Kohonen. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

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

[Lin, 1993] L. J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, 1993.

[Moore and Atkeson, 1993] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[Noel *et al.*, 2006] Black Noel, Shirley Moore, and Eric W. Weisstein. Conjugate gradient method. from mathworld–a wolfram web resource. `http://mathworld.wolfram.com/ConjugateGradientMethod.html`, 2006.

[Paige and Saunders, 1975] Chris C. Paige and Michael A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numerical Analysis*, 12:617–629, 1975.

[RL Library, 2006] RL Library. The University of Alberta reinforcement learning library. `http://rlai.cs.ualberta.ca/RLR/environment.html`, 2006.

[Saad and Schultz, 1986] Y. Saad and Martin H. Schultz. GMRES: A generalized minimal residual method for solving nonsymmetric linear systems. *SIAM*, 1986.

[Sherstov and Stone, 2005] Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In J.-D. Zucker and I. Saitta, editors, *SARA 2005*, volume 3607 of *Lecture Notes in Artificial Intelligence*, pages 194–205. Springer Verlag, Berlin, 2005.

[Stone *et al.*, 2005] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *International Society for Adaptive Behavior*, 13(3):165–188, 2005.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Sutton, 1996] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. The MIT Press, 1996.

[Tham, 1995] Chen K. Tham. Reinforcement learning of multiple tasks using a hierarchical cmac architecture. *Robotics and Autonomous Systems*, 15(4):247–274, 1995.

[Tsitsiklis and Van Roy, 1997] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[Xu *et al.*, 2002] Xin Xu, Han-gen He, and Dewen Hu. Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research*, 16:259–292, 2002.

[Zinkevich, 2006] Martin Zinkevich. The theoretical foundation for incremental least-squares temporal difference learning. Technical Report TR 06-25, University of Alberta, 2006.

Appendix A

Parameter Settings

Parameter	Value
m	1
γ	1
Trace cut-off	$10^{-}4$
ω	$10^{-}6$
ψ	$10^{-}9$
Random seed	$10^{10} - 1$

Table A.1: General parameter settings used for all experiments if applicable.

	Environment				
	Boyan chain			Mountain car	
Algorithm	Small	Medium	Large	Easy	Hard
TD(0)	{1000,.1}	{1000,1}	$\{10^6,1\}$	{1000,.1}	$\{10^6,.01\}$
TD(.5)	{100,1}	N/A	N/A	N/A	N/A
TD(.7)	{100,1}	N/A	N/A	N/A	N/A
TD(.8)	{100,1}	N/A	N/A	N/A	N/A
TD(.9)	{100,1}	N/A	N/A	N/A	$\{10^6,.01\}$
TD(1)	{100,1}	N/A	N/A	N/A	N/A
iLSTD(0)	{100,.1}	{100,1}	{100,1}	{1000,1}	{1000,1}
iLSTD(.5)	{100,.1}	N/A	N/A	N/A	N/A
iLSTD(.7)	{100,.1}	N/A	N/A	N/A	N/A
iLSTD(.8)	{100,.1}	N/A	N/A	N/A	N/A
iLSTD(.9)	{100,.1}	N/A	N/A	N/A	{1000,1}
iLSTD(1)	{100,.1}	N/A	N/A	N/A	N/A

Table A.2: The $\{N_0, \alpha_0\}$ parameter settings used in the experince for TD(λ) and iLSTD(λ) algorithms.

	Environment				
	Boyan chain			Mountain car	
Mechanism	Small	Medium	Large	Easy	Hard
Random	{1000,.1}	{1000,1}	$\{10^6,1\}$	{1000,.1}	$\{10^6,.01\}$
Greedy	{100,.01}	{100,1}	{1000,1}	{100,.1}	{100,.1}
ϵ-greedy(.1)	{100,1}	{100,.1}	{1000,.1}	{100,1}	{1000,.1}
ϵ-greedy(.3)	N/A	N/A	{100,1}	N/A	N/A
ϵ-greedy(.5)	N/A	N/A	{100,1}	N/A	N/A
ϵ-greedy(.9)	N/A	N/A	{100,1}	N/A	N/A
Boltzmann(1)	{100,.1}	{1000,.1}	$\{10^6,.1\}$	$\{10^6,1\}$	$\{10^6,1\}$
Boltzmann(10)	N/A	N/A	$\{10^6,.1\}$	N/A	N/A
Boltzmann(100)	N/A	N/A	$\{10^6,.1\}$	N/A	N/A
Boltzmann(1000)	N/A	N/A	{100,1}	N/A	N/A

Table A.3: The $\{N_0, \alpha_0\}$ parameter settings used for iLSTD algorithm with Random, Greedy, ϵ-Greedy(ϵ), and Boltzmann(τ) dimension selection mechaisms.