

Reinforcement Learning in the Presence of Rare Events

Jordan William Frank

Master of Science

Reasoning and Learning Lab
School of Computer Science

McGill University

Montreal, Quebec

January 2009

A thesis submitted to McGill University
in partial fulfilment of the requirements of
the degree of Master of Science

© Jordan Frank, 2009

ACKNOWLEDGEMENTS

I would like to gratefully acknowledge everyone who inspired and guided me throughout my academic career. Profs. Binay Bhattacharya and Bob Hadley encouraged me to pursue graduate studies, and I thank them for that. I thank Prof. Doina Precup, whose mind and door were always open. I could not imagine a better supervisor, and thank her for her generous support. I thank Prof. Shie Mannor for telling me where to find the best coffee in Montreal, and for being of great assistance during my Master's degree. I am excited and honoured to have the opportunity to continue working with both Doina and Shie. I thank Prof. Luc Devroye for conversations that are always both enlightening and entertaining.

The members of the Reasoning and Learning Lab at McGill University are an exceptional group, and I am lucky to be surrounded by such good friends. I thank Jon, Eric and Mike for getting me out of the house. I would be a lazy hermit without them, and exercise will never be as fun, or occur as regularly, in their absence.

Last, and most important, I thank my family: Mom, Dad, Shaun, Marnie, and the little dog too. I would not be the person I am today without their love and support. I thank my family in Montreal: Leon, Carol, Ryan, Joelle, and Dov, who have made this city seem like home. I thank Zaideh and Roz for the breakfasts on Saturday. I am among a lucky few who can talk shop with their grandfather. Most of all, I thank Gina, without whose love and support I would not have had the courage to do this. This is all for her.

ABSTRACT

Learning agents often find themselves in environments in which rare significant events occur independently of their current choice of action. Traditional reinforcement learning algorithms sample events according to their natural probability of occurring, and therefore tend to exhibit slow convergence and high variance in such environments. In this thesis, we assume that learning is done in a simulated environment in which the probability of these rare events can be artificially altered. We present novel algorithms for both policy evaluation and control, using both tabular and function approximation representations of the value function. These algorithms automatically tune the rare event probabilities to minimize the variance and use importance sampling to correct for changes in the dynamics. We prove that these algorithms converge, provide an analysis of their bias and variance, and demonstrate their utility in a number of domains, including a large network planning task.

ABRÉGÉ

Les agents en phase d'apprentissage se retrouvent souvent dans des environnements dans lesquels les événements rares et signifiants surviennent indépendamment de leur choix d'action. Les algorithmes traditionnels d'apprentissage par renforcement sondent les événements par rapport à leur probabilité de se réaliser, et ont donc tendance à manifester une convergence lente et une variance élevée dans ces environnements. Dans cette thèse, nous supposons que l'apprentissage se déroule dans un environnement simulé dans lequel la probabilité de ces événements rares peut être artificiellement altéré. Nous présentons de nouveaux algorithmes pour l'évaluation d'une politique et pour le contrôle, en utilisant pour les deux d'une part une représentation tabulaire et d'autre part une approximation de la fonction de valeur. Ces algorithmes règlent automatiquement les probabilités d'événements rares pour minimiser la variance et utilisent un échantillonnage d'importance pour corriger les changements dans les dynamiques. Nous prouvons que ces algorithmes convergent et fournissons une analyse de leur biais et de leur variance. Nous démontrons l'utilité de ces algorithmes dans différents domaines, incluant une tâche de planification dans un large réseau.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS		ii
ABSTRACT		iii
ABRÉGÉ		iv
LIST OF FIGURES		vii
LIST OF ALGORITHMS		viii
1	Introduction	1
	1.1 Markov Decision Processes and Reinforcement Learning	2
	1.2 Importance Sampling and Rare Event Simulation	4
	1.3 Contributions	6
	1.4 Outline	6
2	Background	8
	2.1 Reinforcement Learning (MDP) Framework	8
	2.2 Importance Sampling and Rare Event Simulation	22
	2.3 Adaptive Importance Sampling on Discrete Markov Chains	25
3	Learning in the Presence of Rare Events	30
	3.1 Rare Events in MDPs	30
	3.2 Rare Event Adaptive Stochastic Approximation (REASA)	32
	3.3 REASA and Function Approximation	37
4	Theoretical Analysis	42
	4.1 Bias and Variance of Tabular REASA	42
	4.2 Convergence Rates of REASA	47

5	Experiments	49
5.1	Random Markov Chains	49
5.2	Network Planning	51
5.2.1	Policy Evaluation	55
5.2.2	Policy Optimization	58
6	Conclusions and Future Work	65
6.1	Contributions	65
6.2	Future Research Directions	66
	Appendices	69
A	General Notation	69
A.1	Probability and Random Variables	69
A.2	Expectation and Variance	70
A.3	Convergence of Random Variables	71
A.4	Markov Chains	71
B	Proofs	74
B.1	Proof of Theorem 3.2.1	74
B.2	Proof of Theorem 3.2.2	82
B.3	Proof of Theorem 3.3.1	84
	Bibliography	86

LIST OF FIGURES

<u>Figure</u>	LIST OF FIGURES	<u>page</u>
5-1	Value function estimate and Root MSE for state s_0 in Random Markov chains	50
5-2	10-node spanning-tree network used for policy evaluation experiments	57
5-3	Value function estimates and rare event sampling parameter estimates for the tree network	57
5-4	Learning curves and standard deviation for REASA and SARSA on the 10-node network	60
5-5	Example 10-node network built by REASA learning agent	61
5-6	Learning curve for REASA on the 26-node AT&T network	62
5-7	AT&T IP backbone network and network built by REASA learning agent	62

LIST OF ALGORITHMS

<u>Algorithm</u>		<u>page</u>
1	Rare Event Adaptive Importance Sampling	35
2	REASA with Linear Function Approximation	39
3	Network Planning Problem	54

CHAPTER 1

Introduction

Learning agents often find themselves in environments in which rare significant events occur independently of their current choice of action. For example, a stock trading agent may experience a market crash, a network planning agent may experience random link outages that disconnect the network, or a robot exploring a rugged terrain may be caught by a sudden gust of wind that rolls it over. Due to the infrequent nature of these types of events, an agent would require a large number of interactions with the environment in order to observe enough of these events to accurately incorporate them into its learned model of the environment. At the same time, these types of events can have a significant enough impact on the agent that they must be incorporated into the learned model in order for the learning process to be successful.

Traditional reinforcement learning algorithms sample events according to their natural probability of occurring, and therefore tend to exhibit slow convergence and high variance in environments with rare significant events. In their analysis of the bias and variance of a standard model-based reinforcement learning algorithm, Mannor et al. (2007) note that while variance in the reward estimates has a linear effect on the variance in the value function estimates, the value function estimates are nonlinear with respect to the transition probability estimates, and the nonlinearity is substantial. Since the presence of rare events can cause high

variance in the transition probability estimates, this can lead to significant errors in the value function estimates.

In this thesis, we present novel learning algorithms that take into account these rare events in order to converge faster with lower variance. We adapt a method that is commonly used in the rare event simulation community, called adaptive importance sampling, to the problem of learning optimal control strategies in Markov decision processes. We prove that these algorithms converge, provide an analysis of the bias and variance of the algorithms, and demonstrate their utility in a number of domains, including a large network planning task.

1.1 Markov Decision Processes and Reinforcement Learning

Reinforcement learning is a computational approach for studying goal-directed agents that operate in complex stochastic environments. In reinforcement learning, environments are commonly represented as Markov decision processes, which are characterized by a set of states that an agent can find itself in and sets of actions that an agent can take at each state. An agent interacts with the environment by observing its current state, and choosing an action. As a result of the action, the environment responds with a new next state and a reward signal, which is typically a scalar value. The goal of the agent is to learn an optimal policy, or mapping from states to action, such that it maximizes the reward that it accumulates as it moves about the environment. The total expected return that an agent can expect to accumulate if it starts at a specific state and behaves according to a policy is called the value of that state given the policy. The collection of such values for all states is called the value function.

Reinforcement learning has seen successes in many domains, from scheduling tasks such as elevator dispatching (Crites and Barto, 1996), resource-constrained scheduling (Zhang and Dietterich, 2000), job-shop scheduling (Zhang and Dietterich, 1995), and supply chain management (Stockheim et al., 2003), to game playing tasks such as robot soccer (Stone and Veloso, 1999; Stone et al., 2005), creating a world champion backgammon player (Tesauro, 1995), and achieving expert level performance at computer Go (Silver et al., 2007). In all of these cases, and in most applications of reinforcement learning, agents learn in artificial environments, and do not interact with the real world. These artificial environments are realized through software simulators. Reinforcement learning algorithms typically focus on the agent, and treat the simulator as a black box for generating samples.

In this thesis, we shift the focus to the simulator, and look at ways of improving the performance of learning agents by altering the environments such that learning can proceed faster. We focus on a specific case where the states in our MDP are partitioned into two subsets, normal states and rare event states. Normal states represent the states in which the agent finds itself when everything is proceeding as usual. Rare events states are a special subset of the states that are visited infrequently, and only due to some extreme circumstance. For example, in elevator dispatching, the normal states would represent the typical operating conditions of the elevators, and the rare event states would represent the fact that one or more elevators are out of service. A learning agent that learns a policy based on data from a simulator is going to see much more data for the normal states, and may quickly learn a very good policy for the normal operation. On the other hand,

the agent is going to see very few samples of these rare events. In order for the agent to learn how to behave optimally in the rare event states, it must observe a sufficient number of samples. This is inefficient, as the agent will spend most of its time in the normal states, where it has already learned an optimal policy. Most of the agent's time will be wasted waiting for the rare events to occur.

Instead of waiting for the rare events to occur naturally, the simulation community has developed techniques for artificially increasing the rate of rare events, while still obtaining unbiased results. In this thesis, we incorporate some of these techniques into the reinforcement learning framework in order to improve the performance of learning agents in environments with rare events.

1.2 Importance Sampling and Rare Event Simulation

Rare events have been studied extensively in the simulation community (see Asmussen and Glynn, 2007; Bucklew, 2004 for comprehensive reviews), where the main objective is to predict the probabilities of certain events occurring. Many Markov, or Markov-like, models have been studied such as network queues (de Boer et al., 2000, 2002), inventory control problems (Glasserman and Liu, 1996), call centers (Koole and Mandelbaum, 2002), mechanical structures (Bucher, 1988; Au and Beck, 1999), and communication systems (Shanmugam and Balaban, 1980; Devetsikiotis and Townsend, 1993; Stadler and Roy, 1993), where example rare events are buffer overflows, depletion of inventory, excessive wait times, structural failures, or transmission errors, respectively. The underlying idea behind most of the approaches to rare event simulation is to simulate the system under an alternate probability measure, in which the rare

events occur more frequently, and then correct for the change of measure using importance sampling. The search for an optimal change of measure can be done in several ways, including stochastic approximation and the cross-entropy method (Rubinstein and Kroese, 2004; de Boer et al., 2002). The goal of these methods is to find a change of measure such that the variance in the rare event probability estimator is minimized.

Finding an optimal change of measure is a difficult problem. If the frequency of the rare events is increased too much, then the estimator is starved of samples from the normal states, but if the frequency is too low, then we may be no better off than before changing the distribution. Another problem arises from how the change of measure is induced. In some applications, we may be able to directly change the transition distribution, in which case we can calculate exactly how changes in the distribution affect the frequency of the rare events. In most complex systems, however, the transition distribution will be a parameterized function, and by changing the parameter values we indirectly alter the dynamics of the system. In these situations, it is not always straightforward to calculate exactly how changes in the parameters relate to changes in the frequency of the rare events. This leads to difficulties in finding optimal parameter settings, as well as in calculating the importance sampling weights that are needed to correct for the change of measure.

The study of variance reduction is a classical research area in simulation, and the literature is considerable. The most commonly used methods for variance reduction are importance sampling, control variates, and stratification. Work has

been done on using control variates to improve the performance of reinforcement learning algorithms (Baxter and Bartlett, 2001; Greensmith et al., 2004). Importance sampling has also been applied to reinforcement learning (Precup et al., 2000, 2001, 2006). However, in this work importance sampling is not used as a variance reduction technique; instead, the goal is to allow an agent to learn about one policy from samples generated under a different policy.

1.3 Contributions

In this thesis, we present two novel algorithms for learning in environments with rare events: one for learning in environments with a small number of discrete states, and one for learning in environments with large or infinite state spaces. We present proofs that the algorithms converge, and demonstrate the performance advantages on a small domain made up of randomly generated problems, and on a large network planning domain. In both cases, we show that our algorithms outperform the standard approaches. We also investigate the use of our techniques to do policy optimization. Although for the network planning domain, our algorithms do not outperform the standard approach, we gain insight into ways in which our techniques can be improved. Our general approach of using reinforcement learning for optimization in the network planning domain does give excellent results, and to our knowledge, we present the largest network designed by a learning agent with no prior domain knowledge that is currently available in the literature.

1.4 Outline

We begin with a survey of the required background material on reinforcement learning and importance sampling on Markov chains in Chapter 2. In Chapter 3

we describe our rare event framework, and the learning algorithms. Chapter 4 contains an analysis of the bias and variance of our algorithms, and a discussion of the convergence rates. Our experimental results are described in Chapter 5, and we conclude with a further discussion on the contributions of this work, and directions for future work in Chapter 6. Appendix A contains an overview of the notation and concepts from probability theory and stochastic processes that we use throughout this thesis, and Appendix B contains proofs of the theorems from Chapter 3.

CHAPTER 2

Background

In this chapter, we introduce the reinforcement learning framework, and the techniques from stochastic simulation that we use throughout the rest of this thesis. The probability theory and statistical notation generally follows that of Wasserman (2004), and is summarized in Appendix A.

2.1 Reinforcement Learning (MDP) Framework

Reinforcement learning (Sutton and Barto, 1998) is concerned with learning through interaction. A reinforcement learning problem is typically characterized by an environment and a learning agent. The learning agent interacts with the environment at discrete time steps $t = 0, 1, \dots$. The environment is represented by a Markov decision process, which consists of a set of states \mathcal{S} , a set of actions \mathcal{A} , and the one-step dynamics of the environment. At each time step t , the agent observes its current state $s_t \in \mathcal{S}$, chooses to take an action $a_t \in \mathcal{A}$, and then the environment responds with a next state $s_{t+1} \in \mathcal{S}$ as well as a numerical reward signal $r_{t+1} \in \mathbb{R}$. For MDPs with finite states and actions, given any state s and action a , the probability of the next state s' is given by the transition probability distribution

$$p(s'|s, a) = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a),$$

and the rewards are given by a non-negative, bounded, real valued function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, where $r(s, a, s')$ denotes the reward for a transition from state s to s' under action a .

The state and action sets may also be infinite or continuous, in which case p is a density, and the notation changes accordingly. The agent selects its actions according to a policy, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ where $\pi(a|s) = \mathbb{P}(a_t = a | s_t = s)$. The starting states s_0 are drawn from a distribution μ .

Tasks in which the environment contains special terminal states, are called episodic tasks, and an episode is a sequence of states, actions, and rewards starting from some start state and ending in a terminal state. Non-episodic tasks are called continuing tasks. We note that episodic tasks can be treated as continuing tasks, in which the terminal states are considered to be absorbing states that generate 0 reward. That is, once the agent enters a terminal state, it remains there indefinitely, collecting no reward. Therefore, the remainder of this section considers continuing tasks, but the results also apply directly to episodic tasks.

There are two main types of reinforcement learning problems: policy evaluation, and control. In the policy evaluation problem, the agent is given a policy, and the goal is to evaluate the quality of the policy. In the control problem, the goal of the agent is to learn an optimal policy. Most reinforcement learning algorithms are based on estimating some measure of the goodness of states, or state-action pairs. These measures are referred to as value functions. The value of state s under

policy π is defined as

$$V^\pi(s) = \mathbb{E}_\pi (R_t | s_t = s) = \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right), \quad (2.1)$$

where \mathbb{E}_π denotes the expected value given that the agent follows policy π . The discount factor $\gamma \in [0, 1]$ controls the weight of immediate rewards versus future rewards. We use R_t to denote the sum of the discounted future rewards starting from time t . The optimal value function gives the value of every state s under an optimal policy:

$$V^*(s) = \max_{\pi} V^\pi(s).$$

Similarly, the value of taking action a in state s and thereafter following policy π , is defined as:

$$Q^\pi(s, a) = \mathbb{E}_\pi (R_t | s_t = s, a_t = a) = \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right).$$

The optimal state-action value function is defined as:

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a), \forall s, a.$$

For convenience, the state-action values Q are often referred to as Q-values.

For finite state, finite action MDPs, the transition distribution for a given policy π can be written as a $|\mathcal{S}| \times |\mathcal{S}|$ matrix P^π with entries

$$P_{ss'}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) p(s'|s, a),$$

and the expected reward R^π as a $|\mathcal{S}|$ -dimensional vector with entries

$$R_s^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a) r(s, a, s').$$

The value function for a policy π and discount factor γ can then be written as a $|\mathcal{S}|$ -dimensional vector V^π given by:

$$V^\pi = \sum_{n=0}^{\infty} \gamma^n (P^\pi)^n R^\pi.$$

Using the geometric series formula, the value function is given by:

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi,$$

where I is the identity matrix (Bellman, 1957).

While the above equations give the value function in terms of the sum of all future rewards, the Bellman equations for V^π give a recursive definition of the value of each state in terms of the value functions of its successor states:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a) [r(s, a, s') + \gamma V^\pi(s')], \forall s. \quad (2.2)$$

If we treat $V^\pi(s)$ for all $s \in \mathcal{S}$ as unknown values, then the Bellman equations form a set of linear equations whose unique solution is the value function V^π .

Similarly, the Bellman equations exist for the other value functions defined above:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s, a) [r(s, a, s') + \gamma V^*(s')], \quad (2.3)$$

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} p(s'|s, a) \left[r(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s', a') \right], \text{ and} \quad (2.4)$$

$$Q^*(s, a) = \sum_{s' \in \mathcal{S}} p(s'|s, a) \left[r(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right]. \quad (2.5)$$

There are two basic approaches to learning the value functions, Monte Carlo methods and dynamic programming (DP) methods.

Monte Carlo methods for reinforcement learning approximate the value function of a given policy by estimating the expected future returns given by Equation 2.1. For notational convenience, from now on we will drop the superscript π and assume that all value functions are with respect to a given policy π . These methods generate sample trajectories and then approximate the value function for state s (or state-action pair (s, a)) by averaging the sum of the discounted rewards accumulated after first visiting state s (or after the first time action a is taken in state s). This averaging can be done using the standard Robbins-Monro stochastic approximation algorithm (Robbins and Munro, 1951), in which value function estimates are initialized arbitrarily and then, after each observed trajectory, the return R_t is calculated for each state s_t that was visited. The value function estimates are then updated according to:

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)],$$

where $\alpha \in (0, 1)$ is a small positive fraction called the learning rate, or step-size parameter. In some cases, the step-size parameter α_t will depend on the current time step t . A well-known result from stochastic approximation theory (see Kushner and Yin, 2003, for a comprehensive treatment of the subject) tells us that

to ensure convergence with probability 1, the step size parameters α_t must satisfy

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty.$$

In many practical applications, however, we can just take $\alpha_t = \alpha$ to be a small constant value. It is common practice, in theoretical work, to preface a statement with a phrase such as “under standard stochastic approximation conditions” and use α to denote the step-size parameter, where it is implicit that the step-size parameters are chosen to satisfy the required technical conditions to ensure convergence.

Dynamic programming methods are a different approach for estimating value functions, based on the Bellman equations. The value function is initialized to arbitrary values $V_0(s)$ for all s . Then for $t = 0, 1, \dots$ a sweep through all of the states $s \in \mathcal{S}$ is performed and the new values are estimated as:

$$V_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s, a) [r(s, a, s') + \gamma V_t(s')], \forall s.$$

As $t \rightarrow \infty$, the sequence of value functions V_t will converge to the actual value function V . The process of initializing the value functions to an arbitrary value, and then using the current estimates to improve subsequent estimates is called bootstrapping.

DP and Monte Carlo methods differ in a number of important ways. DP methods requires complete knowledge of the model of the transition probability distribution and the reward function. In most applications this is not available. The advantage of the DP method is that it only uses the next state and reward

in calculating the updates, instead of the full future discounted return R_t that is used in the Monte Carlo method. For certain tasks, trajectories may be long, or possibly infinite, so it is advantageous to have a method that does not require waiting until the end of a trajectory in order to calculate the updates.

Temporal difference learning, or TD-learning (Sutton, 1988), combines the bootstrapping and one-step update ideas from DP with the ability to learn from simulated data, rather than a complete model, as in the Monte Carlo method. TD-learning targets the one-step updates given by rewriting Equation 2.2 as an expected value:

$$V(s) = \mathbb{E}_\pi (r_{t+1} + \gamma V(s_{t+1}) | s_t = s).$$

The simplest TD method, TD(0), performs at each time step t the update:

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)].$$

While Monte Carlo methods target the complete return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots,$$

TD methods are based on the idea that the sum of the immediate reward and the value at the subsequent state can be used as an estimate for the return:

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}),$$

where V_t is an estimate of the value function at time t . Clearly, one could also use the n -step return:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}),$$

for $n \geq 1$, and in fact one can use any convex linear combination of n -step returns. This forms the basis of the TD(λ) algorithm, which uses a weighted average of all of the n -step returns. The target for the TD(λ) algorithm is the λ -return, defined by:

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

where $\lambda \in [0, 1]$ is a parameter. When $\lambda = 1$, the λ -return is the same as the Monte Carlo return, and when $\lambda = 0$ it is the same as TD(0). This definition of TD(λ) is called the forward view, as it still requires an entire trajectory to be observed before calculating all of the possible n -step returns and updating the value function estimates. This is inefficient, and in cases where trajectories are long, or infinite, it is not possible.

The most common way to address this problem of having to wait until the end of a trajectory before performing an update is through the use of eligibility traces. Eligibility traces are a method for performing the updates on-line as samples are observed. Instead of performing value function updates by looking forward in time to calculate the returns accumulated after visiting a state, eligibility traces allow propagating updates backwards in time to the states that have been visited along the trajectory, and which are therefore eligible for an update. There are different ways to implement eligibility traces, such as accumulating traces and replacing

traces. We will use the replacing traces method (Singh and Sutton, 1996), as it has been shown to perform better than some other methods, and is simple to implement. The eligibility trace for state s at time step t is denoted $e_t(s) \in [0, 1]$. The eligibility traces are initialized to be $e_0(s) = 0$ for all $s \in \mathcal{S}$, and at each time step t , the eligibility traces are updated as follows:

$$e_t(s) \leftarrow \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ 1 & \text{if } s = s_t, \end{cases}$$

for all $s \in \mathcal{S}$. To perform the value function updates, the TD error at each time step t is computed as:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t),$$

and then for all $s \in \mathcal{S}$, the value function is updated:

$$V(s) \leftarrow V(s) + \alpha \delta e(s).$$

The TD(λ) algorithm solves the policy evaluation task, i.e. it finds the value function for a given policy. It turns out that being able to calculate the value function allows solving the policy improvement, or control problem as well. Suppose that we have an arbitrary policy π , and we computed the value functions V^π and Q^π for that policy. Now suppose that we are at state s , and π would have us choose action a . If instead, we were to choose action $a^* = \operatorname{argmax}_{a' \in \mathcal{A}} Q^\pi(s, a')$, called the greedy action for state s , then $Q^\pi(s, a^*) \geq Q^\pi(s, a)$, and it follows that

$$Q^\pi(s, a^*) \geq V^\pi(s). \tag{2.6}$$

If we let π' be a new policy that picks, for each state s , the greedy action for s , then for all $s \in \mathcal{S}$, $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$. According to the policy improvement theorem (Bellman, 1957), it follows that:

$$V^{\pi'}(s) \geq V^\pi(s). \tag{2.7}$$

Moreover, the policy improvement theorem states that if there is a strict inequality in Equation 2.6 for any state, then there must be a strict inequality in Equation 2.7 for at least one state. Therefore, if there is at least one state in which the greedy policy π' would choose a different action than π , and

$$Q^\pi(s, \pi'(s)) > Q^\pi(s, a),^1$$

then we can expect a strictly greater long-term discounted reward by following π' instead of π , and so π' is a better policy than π . This is the basis of the policy improvement algorithm.

In general the value function is unknown, and so we cannot apply directly the policy improvement algorithm. To solve this problem, policy evaluation steps are interleaved with policy improvement steps, in a framework called generalized policy iteration (GPI). The Sarsa(λ) algorithm (Rummery and Niranjan, 1994) is based on GPI, using TD(λ) for the policy evaluation step. The only modification is that the agent learns the state-action value function Q , rather than just the state

¹ It is important to check that this strict inequality holds, because there may be more than one action with the same Q-value.

value function V . For each state-action pair (s, a) an estimate of $Q(s, a)$ and an eligibility trace $e(s, a)$ are kept. The agent follows a policy by which it chooses the greedy action for the current state with high probability, or some other action otherwise. At each time step t , for the current state s_t , the agent chooses action a_t according to its current policy, and then observes the next state s_{t+1} and reward r_{t+1} . The update becomes

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \delta_t e_t(s, a),$$

for all s, a , where:

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

and

$$e_t(s, a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t; \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t; \\ \gamma \lambda e_{t-1}(s, a) & \text{if } s \neq s_t, \end{cases}$$

and a_{t+1} is the chosen action for state s_{t+1} . Sarsa(λ) is an example of an on-policy reinforcement learning algorithm, as it always selects actions according to the policy derived from the current value function estimates. The name Sarsa(λ) comes from the fact that updates are performed using the tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$.

The only problem that remains is that the policy improvement algorithm requires a good approximation of the value function for all states and all actions. Clearly, if the greedy action is chosen too often, and our approximation of the value function is poor, then the agent might not only pick a suboptimal action, but it will also not learn anything about the other, possibly optimal, actions. GPI

algorithms typically require an assumption that all state-action pairs are visited infinitely often in order for convergence to be guaranteed. In order to guarantee that this assumption is met, an agent is required to occasionally pick actions different from the greedy action. This is referred to as exploration, as instead of exploiting its knowledge of the value function in order to pick what it thinks is an optimal action, the agent is exploring new actions in order to improve its value function estimates.

The exploration/exploitation tradeoff poses an important and challenging problem. While we would like the agent to spend its time taking actions that it knows to be good, its knowledge of the environment is imperfect, and it may have yet to discover that some other action is better than the one that it thinks is best. At the same time, if the agent has found the optimal action, then any time spent exploring other actions is wasted. Therefore, we would like our agent to take more exploratory actions early in the learning process, when its approximation of the value function is poorer, and fewer exploratory actions as the approximation of the value function improves. The simplest means to ensure sufficient exploration is to use exploring starts. That is, each episode begins with a state-action pair (s_0, a_0) that is chosen randomly. This is inefficient, since it only guarantees one exploratory sample on each episode.

Two other simple exploration strategies are ε -greedy and Boltzmann softmax. In ε -greedy exploration, at each time step the greedy action is chosen with probability $1 - \varepsilon$, and with probability $\varepsilon \in [0, 1]$ an action is selected uniformly randomly from the available actions. In general ε is initialized to a large value, which ensures

lots of exploration, and then ε decreases over time. Under the Boltzmann softmax strategy, at each time step t , action a is chosen with probability

$$\frac{e^{Q_t(s_t,a)/\tau}}{\sum_{b \in \mathcal{A}} e^{Q_t(s_t,b)/\tau}},$$

where $\tau > 0$ is a parameter called the temperature. When the temperature is high, actions are selected with nearly the same probability, thus ensuring frequent exploration. As the temperature is lowered, actions with higher Q -values are selected more frequently, and as $\tau \rightarrow 0$, Boltzmann softmax action selection becomes the same as greedy action selection.

The exploration/exploitation tradeoff is well studied, and a number of solutions have been proposed. While the ε -greedy and Boltzmann softmax strategies ensure frequent exploration, they both rely on parameters ε and τ , respectively, that need to be tuned. A more powerful exploration/exploitation strategy, called E^3 (Kearns and Singh, 1998), led to a provably near-optimal polynomial time algorithm for learning in MDPs. The strategy employed by E^3 is to maintain statistics for each state in order to represent the amount of uncertainty the agent has about its approximation for the value function at that state. At states with higher uncertainty, more exploratory actions are chosen, and once the uncertainty at a state has decreased past a certain threshold, no more exploration is performed at that state. The R-MAX algorithm (Brafman and Tennenholtz, 2003) is a simpler strategy that uses the idea of optimism in the face of uncertainty (Kaelbling et al., 1996), and obtains the same near-optimal, polynomial time guarantees as E^3 . The optimism in the face of uncertainty

strategy can be described as assuming that actions which are unknown will obtain the maximum possible reward, denoted R_{\max} . Intuitively, the agent will select suboptimal actions until it knows enough about them to declare them suboptimal. This has the effect of guaranteeing that the policy will either be optimal, or it will ensure quick exploration.

The main problem that we consider in this thesis is that of variance reduction, and our approach is to use importance sampling. In the reinforcement learning literature, work has been done to directly reduce variance in the value function estimates using control variates (Baxter and Bartlett, 2001; Greensmith et al., 2004). Techniques such as hierarchical learning and function approximation can also be viewed as indirectly addressing the problem of variance. Function approximation is typically presented as a technique for generalization (Sutton and Barto, 1998). Generalization allows for learning in large state spaces by exploiting the fact that information gathered in one state can tell us something about similar or nearby states. This generalization has the effect of “smoothing out” the value function estimates over the state space, thus reducing variance in the estimates due to insufficient data and overfitting. Hierarchical reinforcement learning techniques such as MAXQ (Dietterich, 1998), hierarchical abstract machines (Parr and Russell, 1998), and the options framework (Sutton et al., 1999) all address variance in a similar way by decomposing a problem into smaller subproblems. By reasoning at higher levels of abstraction, the problem of variance that arises at the lower levels can be reduced. Importance sampling has also been used in the context of reinforcement learning, albeit for a different purpose. Precup et al.

(2000) used importance sampling for off-policy learning, where the purpose is to learn a value function for a policy different from the policy that was used to generate the data.

2.2 Importance Sampling and Rare Event Simulation

Stochastic simulation is concerned with estimating properties of a system by observing its behaviour. The literature on stochastic simulation is extensive (see Bratley et al., 1986; Asmussen and Glynn, 2007 for comprehensive treatments). The staple of stochastic simulation is the crude Monte Carlo (CMC) method, in which a quantity of interest is approximated by observing multiple samples, or replicates of the quantity, and then taking the average value as the approximate value. More formally, suppose that we have a sequence of independent identically distributed (iid) random variables X_1, X_2, \dots , and let $\bar{X}_n = n^{-1} \sum_{i=1}^n X_i$ be the sample average. Then if $z = \mathbb{E}(|X_1|) < \infty$,² the strong law of large numbers (SLLN) states that $\bar{X}_n \rightarrow z$ almost surely. The central limit theorem (CLT) goes further by saying us that if the X_1, X_2, \dots, X_n are iid with mean $z = \mathbb{E}(X_1)$ and variance $\sigma^2 = \mathbb{V}(X_1)$, then the sample mean \bar{X}_n has a distribution which is approximately Normal, with mean z and variance σ^2/n .

Let X be a random variable, and A a measurable event. Rare event simulation (Bucklew, 2004; Juneja and Shahabuddin, 2006) considers the problem of estimating a probability such as $z = \mathbb{P}(X \in A)$, where z is small (say on the

² Since the random variables are iid, we could define $z = \mathbb{E}(X_i)$ for any i , and so to ease notation we will often write $z = \mathbb{E}(X_1)$.

order of 10^{-3} or less). We can estimate this probability using the CMC method by letting X_1, X_2, \dots, X_n be iid random variables and letting $\bar{X}_n = \sum_{i=1}^n \mathbb{I}(X_i \in A)$, where $\mathbb{I}(B)$ is the indicator function that takes the value 1 if B is true, or 0 otherwise. Since $Z = \mathbb{I}(X_i \in B)$ is a Bernoulli with probability z , it has variance $\sigma_Z^2 = z(1 - z)$. We write $f(x) \approx g(x)$ if $f(x)/g(x) \rightarrow 1$ in some limit like $x \rightarrow 0$ or $x \rightarrow \infty$, and in this notation we have that $\sigma_Z^2 \approx z$. The relative error of this estimator is defined as

$$\frac{\sigma_Z}{z} = \frac{\sqrt{z(1-z)}}{z} \approx \frac{1}{\sqrt{z}} \rightarrow \infty, \text{ as } z \rightarrow 0.$$

To put this into perspective, supposing that we wish to calculate the number of samples N required to ensure a maximum relative error of 10% with respect to the half-width of the 95% confidence interval. Using the central limit theorem, we have that $1.96\sigma_Z/(z\sqrt{N}) = 0.1$, so

$$N = \frac{100 \cdot 1.96^2 z(1-z)}{z^2} \approx \frac{100 \cdot 1.96^2}{z},$$

which increases like z^{-1} as $z \rightarrow 0$. Therefore, if z is small, on the order of 10^{-6} , then N is on the order of 10^8 . Rare events with probabilities on the order of 10^{-6} or even 10^{-9} are not uncommon in many practical applications such as bit-errors in telecommunication systems, overflows in network queues with large buffers, or read errors on computer hard drives. In such applications, CMC methods are infeasible. One approach to overcome this type of problem is importance sampling.

Importance sampling (IS) is a commonly used approach for variance reduction in Monte Carlo simulations (Glynn and Iglehart, 1989; Shahabuddin, 1994;

Heidelberger, 1995). The objective of IS is to modify the sampling distribution so the majority of the sampling is done in areas that contribute the most to z . IS is based on the simple observation that if X is a random variable and h is some function of X , then

$$z = \mathbb{E}_f(h(x)) = \int h(x)f(x)dx = \int \frac{h(x)f(x)}{g(x)}g(x)dx = \mathbb{E}_g(Y),$$

where f and g are probability distribution with g having the same support as f (i.e. for all x , if $f(x) > 0$, then $g(x) > 0$), $Y = h(X)f(X)/g(X)$, and \mathbb{E}_p denotes the expectation with respect to p . Therefore, we can simulate $X_1, \dots, X_n \sim g$, and estimate z by

$$\hat{z} = N^{-1} \sum_{i=1}^N Y_i = N^{-1} \sum_{i=1}^N \frac{f(X_i)}{g(X_i)} h(X_i).$$

The values $f(X_i)/g(X_i)$ are referred to as the importance sampling corrections, or weights. The importance sampling corrections are also often referred to as the likelihood ratios, and the distribution f/g corresponds to the Radon-Nikodym derivative. By the weak law of large numbers, $\hat{z} \rightarrow z$ in probability. That is, for every $\epsilon > 0$,

$$\mathbb{P}(|\hat{z} - z| > \epsilon) \rightarrow 0$$

as $N \rightarrow \infty$. However, with a poor choice of g , the variance of the estimator \hat{z} may be large or even infinite. This typically occurs when the distribution of g has thinner tails than f . Similarly, if $g(x)$ is small over some set A where $f(x)$ is large, then the importance sampling weights could be large, leading to a large variance. Therefore, it is important that g is chosen wisely.

An important result is that for every distribution f and bounded function h , there exists a choice of g that minimizes the variance of \hat{z} , and it is given by

$$g^*(x) = \frac{|h(x)|f(x)}{\int |h(s)|f(s)ds}. \quad (2.8)$$

If $h(x) \geq 0$ under f almost surely, the variance of such an estimator is 0 (Wasserman, 2004). Intuitively, this says that we would like to sample x exactly proportionally to the contribution of $h(x)$ to z . It is important to note that the denominator on the right hand side of Equation 2.8 contains the value that we are trying to approximate, i.e. if we knew how to evaluate $\int |h(s)|f(s)ds$, we would likely be able to calculate z exactly. However, this value can be estimated using stochastic approximation. Provided that we ensure that at each step, the approximation to g has the same support as f , then in general, as the approximation improves, the variance of the estimator decreases. This is the basis of adaptive importance sampling methods. It is noted by Asmussen and Glynn (2007) that even if the approximation is inaccurate, a large reduction in variance can be achieved by sampling outcomes x in rough proportion to $|h(x)|f(x)$.

2.3 Adaptive Importance Sampling on Discrete Markov Chains

One of the main applications that has motivated a large amount of research in Monte Carlo simulation and importance sampling is simulating particle transport through a medium. Particle transport was also one of the original problems addressed through Monte Carlo simulation (Metropolis and Ulam, 1949), and remains an active area of research (Booth, 1985; Kollman, 1993; Kollman et al., 1999; Desai and Glynn, 2001). The general model for studying particle transport

through a medium is to model the path of a particle as a Markov chain. A particle starts from some source emitter, and the states are generally the locations where the particle collides with other particles. Particle motion is inherently stochastic, as are the results of collisions with other atoms in the medium through which it is passing. The Markov chain is transient; it is assumed that at some point the particle will be absorbed by another atom as a result of a collision, or it will escape the medium or some region of interest.

Associated with each transition is a score, typically corresponding to some physical quantity of interest such as the energy released through a collision, or simply an indicator of whether the particle has been absorbed, etc. The notion of a score in particle transport is equivalent to the reinforcement learning notion of a reward. In general, the objective in particle transport simulation is to estimate the expected total score from starting at a state, or set of states; the score would represent, for example, the total energy released per particle or the probability of a particle escaping a region of interest without being absorbed. The notion of expected total score directly corresponds to the reinforcement learning notion of a value function. Due to the nature of the simulated particles, namely their size, simulations that use “nature’s” transition probabilities can be time consuming. Simulating accurate models of complex material geometries can be computationally expensive, and only a small number of simulated trajectories may contribute nonzero scores.

Importance sampling has been proposed as a solution to improve the efficiency of simulators for studying particle transport (Booth, 1985; Kollman, 1993; Kollman

et al., 1999), and we refer to the proposed approach as the AMC algorithm. The general idea of the AMC algorithm is to begin with an initial approximation of the optimal sampling distribution, and then proceed to iteratively improve the approximation from data generated using the sampling distribution, unbiased by the appropriate likelihood values.

The AMC algorithm operates on a Markov chain with a finite state space \mathcal{S} . Let $P = (p_{ss'} : s, s' \in \mathcal{S})$ denote the transition matrix, and let $r : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+$ denote the one step positive reward function associated with each transition. Let $\mathcal{T} \subseteq \mathcal{S}$ denote the terminal states, and $\mathcal{U} = \mathcal{S} \setminus \mathcal{T}$ denote the interior states. We assume that \mathcal{T} is reachable from all states in \mathcal{U} , and define $\tau = \inf \{n : s_n \in \mathcal{T}\}$ as the a.s. finite stopping time for all initial states s_0 . The value function, $V(s)$ is defined as:

$$V(s) = \mathbb{E} \left(\sum_{t=0}^{\tau-1} r(s_t, s_{t+1}) \mid s_0 = s \right),$$

for all interior states $s \in \mathcal{U}$, and $V(s) = 0$ for all terminal states $s \in \mathcal{T}$.

Kollman (1993) shows that the optimal, or minimum variance sampling transition matrix for this Markov chain, denoted P^* , has entries

$$p_{ss'}^* = \frac{p_{ss'}(r(s, s') + V(s'))}{\sum_{s'' \in \mathcal{S}} p_{ss''}(r(s, s'') + V(s''))} = \frac{p_{ss'}(r(s, s') + V(s'))}{V(s)},$$

for all $s, s' \in \mathcal{S}$. It can be easily verified that since r is deterministic, if we were to draw one sample transition from state s , then we could calculate $V(s)$ exactly. Therefore P^* is a zero variance sampling distribution. Of course, it is only of theoretical interest as it requires knowing $V(s)$ for all states $s \in \mathcal{S}$ a priori. Note also that this result only holds for non-negative cost functions.

The AMC algorithm begins by selecting an initial sampling transition matrix $P^{(0)} = (p_{ss'}^{(0)} : s \in \mathcal{U}, s' \in \mathcal{S})$, which may be initialized based on prior knowledge of the problem, results of previous simulation runs, or simply set to be equal to the original transition matrix. The next step of AMC is to choose an initial state $s \in \mathcal{U}$, and simulate a trajectory up to time τ , the time at which a terminal state is reached, using the transition matrix $P^{(0)}$. Let $\langle s_0, s_1, \dots, s_\tau \rangle$ be the set of states visited on the trajectory. Then the likelihood of this trajectory is

$$\mathcal{L} = \prod_{i=0}^{\tau-1} \frac{p_{s_i s_{i+1}}}{p_{s_i s_{i+1}}^{(0)}}, \quad (2.9)$$

and the accumulated reward for the trajectory is

$$R = \prod_{i=0}^{\tau-1} r(s_i, s_{i+1}).$$

AMC proceeds by performing a number of independent simulations from s , and taking the average accumulated reward, weighted by the likelihood ratios, as an estimate $V^{(1)}(s)$. This procedure is repeated for all $s \in \mathcal{U}$, giving estimates $V^{(1)}(s)$ of the solution $V(s)$.

At iteration $n \geq 1$ the AMC algorithm uses the previous estimates $V^{(n)}$ to construct a new sampling distribution $P^{(n)}$ with entries

$$p_{ss'}^{(n)} = \frac{p_{ss'}(r(s, s') + V^{(n)}(s'))}{\sum_{s'' \in \mathcal{S}} p_{ss''}(r(s, s'') + V^{(n)}(s''))}.$$

Then, as for $n = 0$, $P^{(n)}$ is used to simulate trajectories for each state $s \in \mathcal{U}$, the output of which is used to generate new value function estimates $V^{(n+1)}(s)$. This process is repeated over a number of iterations, and it is shown that under certain

conditions the rate of convergence of the estimates to the real value function is exponential as a function of the number of iterations.

The AMC algorithm was later extended to perform updates after each step of the simulation, rather than at the end of a batch of trajectories. This algorithm, called the ASA algorithm (Ahamed et al., 2006), uses stochastic approximation to estimate the optimal sampling transition matrix, allowing it to perform updates on-line. ASA is shown to converge in the limit. However the authors make no claims regarding the convergence rate, and they do not discuss if and when the exponential convergence rates from AMC can be obtained. The empirical results show that for rare event simulation, as the probability of the rare event decreases, ASA performs better than AMC. It is important to note that both AMC and ASA assume knowledge of the the transition probabilities and the reward function. They also assume that samples can be drawn from an arbitrary sampling distribution.

CHAPTER 3

Learning in the Presence of Rare Events

In this chapter, we present the Rare Event Adaptive Stochastic Approximation (REASA) algorithm. REASA is a learning algorithm for MDPs with rare events, and we define algorithms for policy evaluation with both tabular and function approximation representations of the value function. For both cases, we show that the algorithms converge. We also provide an analysis of the bias and variance of tabular REASA, based on the analysis by Mannor et al. (2007). We note that a similar analysis for the function approximation case is not available, even for standard TD-learning without importance sampling.

3.1 Rare Events in MDPs

In this thesis, we are concerned with problems involving rare, significant events that occur as a result of environmental factors, and which are independent of the current action taken by the agent. We model such a problem as an MDP whose states \mathcal{S} are partitioned into two disjoint subsets, the so-called “rare event” states \mathcal{T} , and the “normal” states $\mathcal{U} = \mathcal{S} \setminus \mathcal{T}$. We describe the setup for the finite state, finite action case, and note that the infinite case can be defined similarly. We define the transition probability distribution as a mixture of two separate transition probability distributions: $f(s'|s, a)$, which captures the environment dynamics during “normal” operating conditions, and $g(s'|s)$, which captures

the transitions into our “rare event” states. We assume that for all states s and actions a , $f(s'|s, a) = 0$ for all $s' \in \mathcal{T}$, and similarly, $g(s'|s) = 0$ for all $s' \in \mathcal{U}$.

For each state $s \in \mathcal{S}$, there is a small probability $\varepsilon(s) \in [0, 1]$ that a “rare event” will occur from this state, and we call $\varepsilon(s)$ the rare event probability for state s . We define the transition probability distribution p as:

$$p(s'|s, a) = (1 - \varepsilon(s))f(s'|s, a) + \varepsilon(s)g(s'|s), \quad (3.1)$$

and note that since \mathcal{T} and \mathcal{U} are disjoint with $\mathcal{S} = \mathcal{T} \cup \mathcal{U}$, the transition probability distribution can be rewritten as:

$$p(s'|s, a) = \begin{cases} (1 - \varepsilon(s))f(s'|s, a) & \text{if } s' \notin \mathcal{T}; \\ \varepsilon(s)g(s'|s) & \text{if } s' \in \mathcal{T}. \end{cases}$$

We are concerned with rare events that have a significant impact on the variance of the value function for a given policy. We therefore need to make a distinction between events that occur rarely but have little or no effect on the variance, and those that lead to high variance. We define the rare event states sets as follows.

Definition 3.1.1. *A subset of states $\mathcal{T} \subseteq \mathcal{S}$ is called a rare event state set if the following three properties hold:*

1. *For all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $s' \in \mathcal{T}$, $f(s'|s, a) = 0$ (i.e., transitions into \mathcal{T} cannot depend on the agent’s current action).*
2. *There exists $s \in \mathcal{S}$ and $s' \in \mathcal{T}$ such that $g(s'|s) > 0$ (i.e., transitions into \mathcal{T} can be forced by the environment).*

3. Let V_f^π denote the value function obtained by replacing p with f in (2.2).

Then, for the given policy π ,

$$\exists s \in S \text{ s.t. } |V_f^\pi(s) - V^\pi(s)| \gg 0.$$

The last condition means that states in the rare event set must (collectively) have a large impact on the value function. We define rare events to be transitions into the rare event state set.

We note that we use the term “rare event” loosely from the point of view of the simulation community (Bucklew, 2004) in that our definition is not based solely on the probability of the event. We require this distinction because it is likely that in any environment, there will be events that occur infrequently, but are not of interest in terms of the value function estimates.

3.2 Rare Event Adaptive Stochastic Approximation (REASA)

The ASA algorithm (Ahamed et al., 2006) combines stochastic approximation with adaptive importance sampling in order to approximate a zero-variance change of measure for a finite state Markov chain. ASA makes two strong assumptions: (1) they assume that both the transition probabilities and the reward function, collectively referred to as the model, is completely known, (2) they assume that the distribution from which the samples are drawn can be arbitrarily modified. Both of these assumptions are difficult to achieve in practical applications. In many problems, the model is not given, and must also be learned through simulation. In addition, the dynamics of practical problems are typically complex and making arbitrary changes to the precise transition probabilities is not possible.

The REASA algorithm relaxes both of these assumptions. We assume a rare event MDP, as described in Section 3.1, and we assume no knowledge of the model other than the true rare event probabilities $\varepsilon(s)$ for all $s \in \mathcal{S}$. We also assume that we have access to a simulation environment in which the rare event probability can be artificially modified, and we assume that the rare event state set \mathcal{T} is known. We define $\hat{\varepsilon} : \mathcal{S} \rightarrow [0, 1]$ to be the rare event sampling parameter, and assume access to a simulation environment that can generate samples from the sampling distribution

$$q(s'|s, a, \hat{\varepsilon}) = (1 - \hat{\varepsilon}(s))f(s'|s, a) + \hat{\varepsilon}(s)g(s'|s), \quad (3.2)$$

where f and g are unknown, and remain unchanged. By considering that the state space \mathcal{S} is separated into disjoint normal and rare event subsets, \mathcal{T} and \mathcal{U} respectively, we note that the likelihood of any transition under the sampling distribution can be computed by

$$\mathcal{L}(s, a, s') = \begin{cases} \varepsilon(s)/\hat{\varepsilon}(s) & \text{if } s \in \mathcal{T}; \\ (1 - \varepsilon(s))/(1 - \hat{\varepsilon}(s)) & \text{if } s \notin \mathcal{T}. \end{cases} \quad (3.3)$$

Previous work on importance sampling on Markov chains has considered only the state value function V , and so we now present the zero-variance importance sampling distribution in terms of the action-value function Q .

Theorem 3.2.1. *Given an MDP with states \mathcal{S} , actions \mathcal{A} , one-step transition probability distribution p , and reward function r , the zero-variance importance*

sampling distribution p^* is given by

$$p^*(s'|s, a) = \frac{p(s'|s, a) \sum_{a' \in \mathcal{A}} \pi(a'|s') [r(s, a, s') + \gamma Q^\pi(s', a')]}{Q^\pi(s, a)} \quad (3.4)$$

Proof: See Appendix B.1 ■

We can further simplify Equation 3.4 to be in terms of both V and Q :

$$p^*(s'|s, a) = \frac{p(s'|s, a) [r(s, a, s') + \gamma V^\pi(s')]}{Q^\pi(s, a)}.$$

Furthermore, if we use the fact that $V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q(s, a)$, then we get

$$p^*(s'|s) = \sum_{a \in \mathcal{A}} \pi(a|s) p^*(s'|s, a) = \frac{\sum_{a \in \mathcal{A}} p(s'|s, a) [r(s, a, s') + \gamma V^\pi(s')]}{V^\pi(s)},$$

which is form used by Ahamed et al. (2006). We prefer the form of Equation 3.4 because when we extend REASA to solve control tasks, it is useful to have expressions in terms of state-action pairs.

Note that the existence of a zero-variance importance sampling distribution p^* implies that $R_{p^*} = Q^\pi(s, a)$ if $s_0 = s$ and $a_0 = a$. Therefore, if we calculate total reward for a single trajectory starting from s and a under p^* , then we immediately have the value function for that state-action pair. However, as has been previously noted, the zero-variance importance sampling contains the value function Q , and therefore is only of theoretical interest.

Given policy π , we define our optimal value for $\hat{\varepsilon}(s)$ as:

$$\varepsilon^*(s) = \varepsilon(s) \frac{\sum_{s' \in \mathcal{T}} g(s'|s) [\sum_{a \in \mathcal{A}} \pi(a|s) r(s, a, s') + \gamma V^\pi(s')]}{V^\pi(s)}. \quad (3.5)$$

for all $s \in \mathcal{S}$. We approximate the values $\varepsilon^*(s)$ on-line using samples.

Algorithm 1 Rare Event Adaptive Importance Sampling

Input: Rare event set $\mathcal{T} \subset \mathcal{S}$, true rare-event probabilities $\varepsilon(s)$, learning rates α , α_T , and α_U , and parameter $\delta > 0$.

1. Initialize \hat{V}^π arbitrarily, $\hat{\varepsilon}(s) \leftarrow 1/|\mathcal{S}|$, $\hat{T}(s) \leftarrow 0$, and $\hat{U}(s) \leftarrow 0$ for all states s .
2. Select the initial state s_0 .
3. Initialize eligibility traces: $e(s_0) = 1$ and $e(s) = 0, \forall s \neq s_0$.
4. Repeat for $t = 0, 1, \dots$:

- (a) Select an action $a_t \sim \pi(s_t, \cdot)$.
- (b) Select whether a rare event happens, according to $\hat{\varepsilon}(s_t)$, and sample s_{t+1} from f or g accordingly. Observe the reward r_{t+1} .
- (c) Compute the importance sampling weight:

$$w_t = \begin{cases} \varepsilon(s_t)/\hat{\varepsilon}(s_t) & \text{if } s_{t+1} \in \mathcal{T}; \\ (1 - \varepsilon(s_t))/(1 - \hat{\varepsilon}(s_t)) & \text{if } s_{t+1} \notin \mathcal{T}. \end{cases}$$

- (d) Compute the importance-sampling TD-error:

$$\Delta_t = w_t(r_{t+1} + \gamma\hat{V}^\pi(s_{t+1})) - \hat{V}^\pi(s_t).$$

- (e) Update the value estimates for all $s \in \mathcal{S}$:

$$\hat{V}^\pi(s) \leftarrow \hat{V}^\pi(s) + \alpha e(s)\Delta_t.$$

- (f) If $s_{t+1} \in \mathcal{T}$, then:

$$\hat{T}(s_t) \leftarrow (1 - \alpha_T)\hat{T}(s_t) + \alpha_T\varepsilon(s_t)(r_{t+1} + \hat{V}^\pi(s_{t+1})),$$

else

$$\hat{U}(s_t) \leftarrow (1 - \alpha_U)\hat{U}(s_t) + \alpha_U(1 - \varepsilon(s_t))(r_{t+1} + \hat{V}^\pi(s_{t+1})).$$

- (g) Update the rare event probabilities:

$$\hat{\varepsilon}(s_t) \leftarrow \min \left(\max \left(\delta, \frac{|\hat{T}(s_t)|}{|\hat{T}(s_t)| + |\hat{U}(s_t)|} \right), 1 - \delta \right).$$

- (h) Update eligibility traces for all $s \in \mathcal{S}$:

$$e(s) \leftarrow \begin{cases} 1 & \text{if } s = s_{t+1}; \\ \gamma\lambda w_t e(s) & \text{if } s \neq s_{t+1}. \end{cases}$$

Algorithm 1 is our proposed approach for learning in the presence of rare events for finite state MDPs where the value function is represented in tabular form. It is based on the observation that we can rewrite Equation 3.5 as follows:

$$\varepsilon^* = \frac{T(s)}{T(s) + U(s)}, \quad (3.6)$$

where

$$T(s) = \varepsilon(s) \sum_{s' \in \mathcal{T}} g(s'|s) \left[\sum_{a \in \mathcal{A}} \pi(a|s) r(s, a, s') + \gamma V^\pi(s') \right]$$

is the contribution to the value function at state s from the states in the rare event state set \mathcal{T} , and

$$U(s) = (1 - \varepsilon(s)) \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{U}} f(s'|s, a) [r(s, a, s') + \gamma V^\pi(s')]$$

is the contribution to $V^\pi(s)$ from the states in the set $\mathcal{U} = \mathcal{S} \setminus \mathcal{T}$ of normal states.

For all $s \in \mathcal{S}$, let

$$\tilde{\varepsilon}(s) = \frac{\min(\max(\delta, \varepsilon^*(s)), 1 - \delta)}{\sum_{s' \in \mathcal{S}} \min(\max(\delta, \varepsilon^*(s')), 1 - \delta)}.$$

Note that this equals $\varepsilon^*(s)$ if $\delta < \varepsilon^*(s) < 1 - \delta$. The parameter $\delta > 0$ is used to bound the importance sampling weights, which keeps the variance of the estimator bounded. In general, we want δ to be smaller than the smallest non-zero value in our transition matrix, and we have found that setting

$$\delta = \min_{s \in \mathcal{S}} \{\varepsilon(s)/10\}$$

works well in practice.

In Algorithm 1, $\hat{T}(s)$ and $\hat{U}(s)$ are unbiased estimators for $T(s)$ and $U(s)$ respectively. It follows from Equations 3.4 and 3.6, that as $t \rightarrow \infty$, if $\delta < \varepsilon^*(s) < 1 - \delta$, then

$$\hat{\varepsilon}(s) = \frac{\hat{T}(s)}{\hat{T}(s) + \hat{U}(s)} \rightarrow \varepsilon^*(s),$$

for all states $s \in \mathcal{S}$. These results are summarized in the following proposition.

Theorem 3.2.2. *Using Algorithm 1 with the usual step-size conditions on α , α_T , and α_U , for any $\lambda \in [0, 1]$ that does not depend on the action a_t , and assuming that the MDP is unichain for $\varepsilon = \delta$,¹ we have that:*

$$\hat{V}^\pi(s) \rightarrow V^\pi(s) \text{ almost surely.}$$

Moreover, for all $s \in \mathcal{S}$, $\hat{\varepsilon}(s) \rightarrow \tilde{\varepsilon}(s)$ almost surely.

Proof: See Appendix B.2 ■

It should also be noted that while the treatment above is for non-negative rewards (for ease of notation), the algorithm is formulated for the general case in which the reward function may be positive or negative, which is an extension of the original ASA algorithm.

3.3 REASA and Function Approximation

If the state space is large or continuous, then function approximation must be used to estimate the value function. Here, we are concerned with value function

¹ The unichain assumption is needed to invoke the stochastic approximation argument; see Bertsekas and Tsitsiklis (1996). Also note that if the MDP is unichain for one value of $\varepsilon \in (\delta, 1 - \delta)$ it is unichain for all values.

estimates that are linear in a set of feature vectors $\{\phi_s\}, s \in \mathcal{S}$:

$$V^\pi(s) \approx \theta^T \phi_s = \sum_{i=1}^n \theta(i) \phi_s(i),$$

where $\theta \in \mathbb{R}^n$ is the learned parameter vector. We represent our eligibility traces as a vector \vec{e} of the same size as θ . We assume that the rare event states and normal states are represented by disjoint features, and so we are able to determine whether a state s is in the rare event state set \mathcal{T} , or the normal state set \mathcal{U} . We no longer have a state-dependent rare event probability, and we assume that the rare event probability ε is a constant over the state space. We discuss possible extensions to this in Section 6.2.

Algorithm 2 presents our approach for learning in MDPs with rare events where the value function is represented using linear function approximation. Instead of keeping a state-dependent rare event probability estimate, we treat the normal state set \mathcal{U} and the rare event state set \mathcal{T} as two states in a high-level MDP. In this high level MDP, we only concern ourselves with transitions from a normal state \mathcal{U} to a rare event state \mathcal{T} , and we estimate a parameter $\hat{\varepsilon}$ that gives us the probability of transitioning from \mathcal{U} to \mathcal{T} under our sampling distribution, just as in the tabular version of REASA. As in the tabular case, we assume that we can sample from the transition probability distribution

$$q(s'|s, a, \hat{\varepsilon}) = (1 - \hat{\varepsilon})f(s'|s, a) + \hat{\varepsilon}g(s'|s),$$

where $\hat{\varepsilon}$ is a parameter that we control.

Algorithm 2 REASA with Linear Function Approximation

Input: Rare event set $\mathcal{T} \subset \mathcal{S}$, true rare-event probability ε , learning rates α , α_T , and α_U , and parameter $\delta > 0$.

1. Initialize parameter vector θ_0 arbitrarily, $\hat{\varepsilon}_0 \leftarrow 1/2$, $\hat{T}_0 = 0$, and $\hat{U}_0 = 0$.
2. Initialize the total importance sampling trajectory weight: $c_0 = 1$.
3. Select the initial state s_0 .
4. Initialize eligibility vector: $\vec{e}_0 = c_0\phi(s_0)$.
5. Repeat for $t = 0, 1, \dots$:
 - (a) Select an action $a_t \sim \pi(s_t, \cdot)$.
 - (b) Select whether a rare event happens, according to $\hat{\varepsilon}_t$, and sample s_{t+1} from f or g accordingly. Observe the reward r_{t+1} .
 - (c) Compute the importance sampling weight:

$$w_t = \begin{cases} \varepsilon/\hat{\varepsilon}_t & \text{if } s_{t+1} \in \mathcal{T}; \\ (1 - \varepsilon)/(1 - \hat{\varepsilon}_t) & \text{if } s_{t+1} \notin \mathcal{T}. \end{cases}$$

- (d) Compute the importance-sampling TD-error:

$$\Delta_t = w_t(r_{t+1} + \gamma\theta\phi(s_{t+1})) - \theta\phi(s_t).$$

- (e) If $s_{t+1} \in \mathcal{T}$, then:

$$\hat{T}_{t+1} = (1 - \alpha_T)\hat{T}_t + \alpha_T\varepsilon(r_{t+1} + \gamma\theta_t\phi(s_{t+1})),$$

else

$$\hat{U}_{t+1} = (1 - \alpha_U)\hat{U}_t + \alpha_U(1 - \varepsilon)(r_{t+1} + \gamma\theta_t\phi(s_{t+1})).$$

- (f) Update the parameter vector: $\theta_{t+1} = \theta_t + \alpha\vec{e}_t\Delta_t$.
- (g) Update the rare event probabilities:

$$\hat{\varepsilon}_{t+1} = \min \left(\max \left(\delta, \frac{|\hat{T}_{t+1}|}{|\hat{T}_{t+1}| + |\hat{U}_{t+1}|} \right), 1 - \delta \right).$$

- (h) Update the trajectory weight: $c_{t+1} = c_t w_t$.
- (i) Update eligibility traces:

$$\vec{e}_{t+1} = \gamma\lambda w_t \vec{e}_t + c_{t+1}\phi(s_{t+1}).$$

We denote the rare event sampling probability at time t as $\hat{\varepsilon}_t$, and note that it is a function of the entire trajectory up time t . We define an indicator variable I_t which takes the value 1 if state $s_{t+1} \in \mathcal{T}$ or 0 otherwise, which indicates whether a rare event occurred at time t or not. We define our importance sampling ratio at time t as:

$$w_t = \frac{\varepsilon}{\hat{\varepsilon}_t} I_t + \frac{1 - \varepsilon}{1 - \hat{\varepsilon}_t} (1 - I_t).$$

Let $\Omega_t(s) = \{\langle s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t \rangle | s_0 = s\}$ denote the set of all trajectories of length t beginning at state s , and denote by $\rho_p(\omega)$ the probability of trajectory $\omega \in \Omega_t(s)$ occurring when transitions are generated according to the original transition probability distribution p . The likelihood of a trajectory $\omega \in \Omega_t(s)$ generated under sampling distribution q is $\prod_{i=0}^{t-1} w_i$.

Under conventional TD(λ), the cumulative updates to the parameter vector θ following step t are given by the forward-view equations:

$$\begin{aligned} \Delta\theta_t &= \alpha(R_t^\lambda - \theta^T \phi_t) \phi_t, \\ R_t^\lambda &= (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}, \\ R_t^{(n)} &= r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{n-1} r_{t+n} + \gamma^n \theta^T \phi_{t+n}, \end{aligned}$$

where ϕ_t is shorthand for ϕ_{s_t} .

Algorithm 2 computes the parameter updates as:

$$\Delta\theta_t = \alpha(\bar{R}_t^\lambda - \theta^T \phi_t) \phi_t w_0 w_1 \dots w_t, \tag{3.7}$$

where \bar{R}_t^λ is defined similar to R_t^λ above, except in terms of the weighted n -step return:

$$\bar{R}_t^{(n)} = r_{t+1}w_t + \gamma r_{t+2}w_t w_{t+1} + \cdots + \gamma^{n-1} r_{t+n} w_t \cdots w_{t+n-1} + \gamma^n w_t \cdots w_{t+n} \theta^T \phi_{t+n}.$$

Theorem 3.2.2 tells us that $\mathbb{E}_p \left(R_t^{(n)} | s_t \right) = \mathbb{E}_q \left(\bar{R}_t^{(n)} | s_t \right)$ for all n , and the following result extends the idea to the case of linear function approximation.

Theorem 3.3.1. *Under standard stochastic approximation conditions, Algorithm 2 converges in the limit, with probability 1, to the same estimates as the on-policy TD-learning algorithm.*

Proof: See Appendix B.3. ■

CHAPTER 4

Theoretical Analysis

In this chapter, we consider the theoretical aspects of REASA. We analyze the bias and variance in Section 4.1, and follow up with a brief discussion on the convergence rates in Section 4.2.

4.1 Bias and Variance of Tabular REASA

Theorem 3.2.1 tells us that the minimum variance estimator comes from sampling the next state-action pair in proportion to its contribution to the value function for the current state-action pair. Therefore, without making any additional assumptions on f and g , ε^* , as defined in Equation 3.5, is the optimal rare event sampling parameter. Since we do not optimize individual transitions from states in the normal state set to other states in the normal state set, our estimator will exhibit variance. In fact, by oversampling the rare events, we decrease the number of observed transitions between states in the normal state set, and so we would expect the error in our estimates of these normal transitions to increase.

For REASA to improve the performance, it must be the case that the contribution to the variance of our estimator from the transitions to rare event states is much larger than the contribution to the variance by transitions to normal states. In this case, given a fixed number of samples, increasing the number of samples of rare event transitions will lead to a greater decrease in the sample variance than

the increase in the sample variance introduced by decreasing the number of normal transitions. This follows from the fact that the sample variance is equal to the variance divided by the number of samples.

Additionally, because the rare event state set is disjoint from the normal state set, we can estimate the variance due to the individual transition probability distributions, f and g , separately. We show this through an analysis in the same spirit as the analysis presented by Mannor et al. (2007), which derives bias and variance results for standard temporal difference algorithms.

For simplicity, let us assume that $\varepsilon(s) = \varepsilon$ for all states $s \in \mathcal{S}$ (all the analysis can be done without this assumption, but becomes more tedious). Let R^π denote the vector of immediate rewards for every state, with entries:

$$R_s^\pi = \sum_{a \in A} \sum_{s' \in \mathcal{S}} \pi(a|s) p(s'|s, a) r(s, a, s'),$$

and P^π be an $|\mathcal{S}| \times |\mathcal{S}|$ transition matrix under π , with entries:

$$P_{ss'}^\pi = \sum_a \pi(a|s) p(s'|s, a).$$

From (3.1), we can re-write P^π as:

$$P^\pi = (1 - \varepsilon)F^\pi + \varepsilon G,$$

where F^π is the transition matrix corresponding to staying in the normal states, and G is the matrix corresponding to transiting into the rare event states. Note that according to our assumptions, G does not depend on π . Similarly, the reward vector can be decomposed into two components, R_F^π and R_G^π .

We use two sequences, $\{X_k\}_{k=1}^\infty$ and $\{Y_k\}_{k=1}^\infty$, of geometrically distributed random variables, with means $(1 - \varepsilon)^{-1}$ and ε^{-1} respectively, to represent the amount of time between transitions from the normal states and the rare event states respectively. We also assume that the initial state is a normal state. Hence, the simulation starts in some normal state and stays in the set of normal states for X_1 time steps, at which point it transitions to a state in the rare event set, where it stays for Y_1 time steps, then transitions back to the normal set for X_2 time steps, etc.

We make two further simplifications. First, we assume that after each excursion into the rare event state set, the system “jumps back” to the normal state in which it was before entering; that is, $(F^\pi)^i G^j (F^\pi)^k \approx (F^\pi)^{i+k}$. Second, we assume that the rewards for transitioning to states in the normal set are similar regardless of the origin, that is that $GR_F^\pi \approx F^\pi R_F^\pi$. The analysis can be done without these assumptions, but it becomes more tedious. These assumptions are reasonable because in general the rare events model failures in the system, such as a failed link in a network, and when the failure is no longer present, the system resumes from the state prior to the failure. We define $\tau(k) = \sum_{i=1}^{k-1} X_i$ and $v(k) = \sum_{i=1}^{k-1} Y_i$. The value function estimate, V^π , can be re-written as:

$$\begin{aligned} V^\pi &= \mathbb{E} \left(\sum_{k=1}^{\infty} \gamma^{\tau(k)+v(k)} \prod_{i=1}^{k-1} ((F^\pi)^{X_i} G^{Y_i}) \left(\sum_{i=0}^{X_k-1} \gamma^i (F^\pi)^i R_F^\pi + \gamma^{X_k} (F^\pi)^{X_k-1} \sum_{i=0}^{Y_i-1} \gamma^i G^i R_G^\pi \right) \right) \\ &\approx \mathbb{E} \left(\sum_{k=1}^{\infty} \gamma^{\tau(k)+v(k)} (F^\pi)^{\tau(k)-1} \left(\sum_{i=0}^{X_k-1} \gamma^i (F^\pi)^i R_F^\pi + \gamma^{X_k} (F^\pi)^{X_k-1} \sum_{i=0}^{Y_i-1} \gamma^i G^i R_G^\pi \right) \right). \end{aligned}$$

When the value function is estimated from data using TD-learning, we can analyze the bias and variance of this estimate by considering that all the model component estimates are affected by noise components. The estimate of the value function, \hat{V}^π can be broken up into two components; the first component ignores rare events, and the second takes rare events into account. The first component is:

$$\begin{aligned}\mathbb{E}\left(\hat{V}_F^\pi\right) &= \mathbb{E}\left(\sum_{k=1}^{\infty} \gamma^{\tau(k)+v(k)} (F^\pi + \tilde{F}^\pi)^{\tau(k)-1} \sum_{i=0}^{X_k-1} \gamma^i (F^\pi + \tilde{F}^\pi)^i (R_F^\pi + \tilde{R}_F^\pi)\right) \\ &= \sum_{k=1}^{\infty} \mathbb{E}\left(\gamma^{\tau(k)+v(k)} (F^\pi + \tilde{F}^\pi)^{\tau(k)-1} \sum_{i=0}^{\infty} (1-\varepsilon)^i \varepsilon \gamma^i (F^\pi + \tilde{F}^\pi)^i (R_F^\pi + \tilde{R}_F^\pi)\right),\end{aligned}$$

where \tilde{R}_F^π and \tilde{F}^π represent noise in our estimates of R_F^π and F^π respectively.

The bias and variance of this estimate can be derived directly as in Mannor et al. (2007), noting that $\tau(k)$ and $v(k)$ are sums of independent geometrically distributed variables, and are therefore distributed according to a negative binomial distribution.

The second component is:

$$\begin{aligned}\mathbb{E}\left(\hat{V}_G^\pi\right) &= \mathbb{E}\left(\sum_{k=1}^{\infty} \gamma^{\tau(k+1)+v(k)} (F^\pi + \tilde{F}^\pi)^{\tau(k+1)-1} \sum_{i=0}^{Y_k-1} \gamma^i (G + \tilde{G})^i (R_G^\pi + \tilde{R}_G^\pi)\right) \\ &= \sum_{k=1}^{\infty} \mathbb{E}\left(\gamma^{\tau(k+1)+v(k)} (F^\pi + \tilde{F}^\pi)^{\tau(k+1)-1} \sum_{i=0}^{\infty} (1-\varepsilon)^i \varepsilon \gamma^i (G + \tilde{G})^i (R_G^\pi + \tilde{R}_G^\pi)\right),\end{aligned}$$

where \tilde{R}_G^π and \tilde{G} represent noise in our estimates of R_G and G respectively.

Note that the noise components \tilde{G} , \tilde{F}^π , \tilde{R}_G^π depend on the number of transitions observed in the environment. If we observe N transitions, then the expected number of observed transitions according to F^π is $(1-\varepsilon)N$ and the expected

number of transitions according to G is εN . Hence, we assume that the noise components \tilde{F}^π and \tilde{R}_F^π are negligible compared to \tilde{G} , and \tilde{R}_G^π .

To establish bias-variance estimates for \hat{V}_G^π , we need to look at

$$\mathbb{E} \left((G + \tilde{G})(R_G^\pi + \tilde{R}_G^\pi) \right).$$

Similarly to Mannor et al. (2007), we assume that $\mathbb{E}(\tilde{G}) = 0$ and $\mathbb{E}(\tilde{R}_G^\pi) = 0$. Hence, the remaining term which will determine the bias and variance is $\mathbb{E}(\tilde{G}\tilde{R}_G^\pi)$, which captures the correlations between the transition and model estimates, due to the fact that they are estimated from the same samples. This expectation can be derived directly from the formulas in Mannor et al. (2007).

Note that we could also have applied the analysis of Mannor et al. (2007) directly to P^π . However, this would lead to very loose bounds, because their results depend on the inverse of the minimum number of samples obtained for any transition, and we expect that there will be very few transitions into the rare event set. In our analysis, only the second term depends on numbers of transitions into the rare events states, so we can focus our analysis on the effect of the rare events on the bias and variance in our estimates. Therefore, we can expect tighter bounds, and better confidence intervals by estimating the variance in the estimates of F^π and G separately.

Also, as previously noted, the purpose of the algorithm is to sample rare events proportionately to their contribution to the value function for all states. Hence, intuitively, it will reduce bias and variance in the second component by oversampling the rare events, and thus decrease the noise components \tilde{G} and \tilde{R}_G^π .

Given the same amount of data, the errors in \tilde{F}^π and \tilde{R}_F^π will increase, but not by much.

4.2 Convergence Rates of REASA

Without making further assumptions on f and g , very little can be said in general about the rate of convergence of REASA. The previous section showed that in MDPs with rare events, sampling according to the optimal sampling distribution decreases the variance of our estimator, and in general this implies that we can expect our convergence rates to improve.

The difficulty is that in standard adaptive importance sampling, the sampling distribution has the Markov property, that is the sampling distribution at a time step only depends on the sampling distribution at the previous time step. This is due to the fact that the sampling distribution is a function of the current value function estimates, and the current value function estimates are updated at each time step by generating a trajectory according to the current sampling distribution. As such, the sampling distribution forms a Markov chain with an absorbing state (the zero-variance distribution). Therefore it has an invariant measure 0, and under a few basic assumptions, it can be shown that this invariant measure is unique. In general, Markov chains on probability distributions converge to their invariant measure at an exponential rate, and so in general, it can be expected that the sampling distribution will converge to the zero-variance sampling distribution at such a rate. This is the typical argument behind the exponential convergence rates for adaptive importance sampling schemes (Kollman, 1993; Kollman et al., 1999).

However, our rare event parameter does not have the Markov property. The current rare event parameter is a function of the current value function estimates, but our updates to the value function at each step depend not only on the rare event parameter, also on the current state. We could modify REASA to only do Monte Carlo updates at the end of each episode, and our rare event parameter would have the Markov property, but we then lose the advantages of TD-learning, and we can no longer consider infinite-horizon problems.

Even if we did have a rare event parameter that evolved according to a Markov chain, due to variance in the value function estimates for f and g , which remains even when sampling with the optimal rare event parameter, there is no absorbing state. However, if we assume that the variance of our value function estimates are bounded, which is a fairly modest assumption, then the rare event parameter would have a stationary distribution.

In practice, we observe that the rare event parameter tends to converge to the optimal value very quickly, and then oscillates around the optimal value. Quantifying this convergence rate is, however, nontrivial.

CHAPTER 5

Experiments

We evaluate the performance of REASA on two different domains, small randomly generated Markov chains, and a large network planning domain. For the random Markov chain domain, we compare the performance of tabular REASA to ASA and on-line TD(λ), and show that REASA and ASA both outperform TD(λ), and REASA performs as well as ASA despite not having knowledge of the model. The network planning domain is based on a problem posed by Hobbs and Bell (2002), where we are given a set of cities and a set of traffic demands, and the goal is to build a network to deliver the traffic.

5.1 Random Markov Chains

We first compare the performance of REASA to on-line TD(λ) and ASA on a testbed of randomly generated Markov chains. Each environment contains 10 regular states s_0, \dots, s_9 and one rare event state δ . Each regular state can transition to δ with probability $\varepsilon = 0.001$, and to seven other regular states (chosen randomly) with probabilities drawn from a uniform distribution, normalized to sum to $1 - \varepsilon$. The rewards for transitioning between the regular states and from the rare event state to the regular states are drawn from a normal distribution with mean 1.0 and standard deviation 0.5, with negative values being discarded so that we can run ASA. The rewards for transitioning to the rare event state are drawn from a

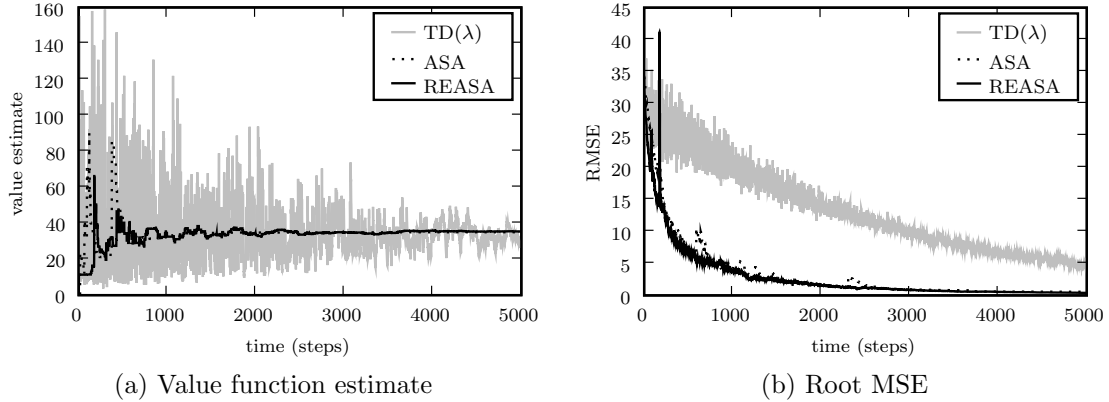


Figure 5–1: Value function estimate and Root MSE for state s_0 in Random Markov chains

normal distribution with mean $10/\varepsilon$ and standard deviation $1/\varepsilon$. The initial state is s_0 , and the discount factor is $\gamma = 0.7$.

In the following results, a step is considered to be one transition for both ASA and REASA, but for $\text{TD}(\lambda)$, a “step” actually consists of 2300 real time steps. We chose this number of steps so that the probability of observing at least one rare event transition in each episode is approximately 0.9. Therefore, we put $\text{TD}(\lambda)$ at a significant advantage in terms of the number of samples that it is provided. In Figure 5–1a we plot the estimate for the value function at the initial state over time, averaged across 70 independent runs. We use a value of $\lambda = 0.7$. The learning rates are on a decreasing schedule where we initially set $\alpha = \alpha_0$ and then halve the value of α at time steps $T, 2T, 4T, 8T, \dots$. The values of α_0 and T were tuned individually for each algorithm. Figure 5–1b shows the root mean squared error for the value function estimate at the initial state, again averaged across 70 independent runs.

The learning and error curves for REASA and ASA are nearly indistinguishable, and both outperform $TD(\lambda)$. We note that in the case of ASA, the original transition probability distribution is needed, and the algorithm has full control over the transition probabilities that are used in the simulation (an unlikely case in many practical applications). We observe that despite the fact that REASA can only know and control the rare event probability, it performs nearly as well as ASA.

5.2 Network Planning

In order to demonstrate REASA in a practical setting with a large state space, we use a network planning task in which a reinforcement learning agent has to build and maintain a telecommunications network linking a number of North American cities. Each pair of cities has a certain traffic demand, ranging from 3GBs¹ to 60GBs initially, and this demand grows stochastically at a rate of approximately 3% per year. The goal is to design and build a network topology in order to meet the traffic demands by placing links between pairs of cities. Links consist of bundles of fiber optic cables, and each fiber can carry a specific unit of bandwidth. Building a link between a pair of cities incurs a large one-time cost of \$500k/mile. Once a link has been built, the capacity of the link can be increased by activating fibers, in units of 25GBs; this incurs a cost of \$30k/mile.

¹ We use GBs to represent an average sustained traffic rate of 1 gigabyte per second; because the time interval under consideration is always roughly the same, we also use it as a unit of traffic, with an abuse of notation.

The revenue from traffic is generated daily: traffic delivered generates a reward of \$1k/GBs/mile, and undelivered traffic is penalized at a rate of \$200k/GBs/mile every hour. Each episode consists of a 10 year period, with discrete time steps representing a single day, for a total of 3650 steps per episode (we ignore leap years)² .

Link failures occur with a small probability, completely severing a link for a short period of time. Without considering link failures, a minimum spanning tree (MST) could be built, with enough activated fibers to carry the traffic. However, in such a network, any link failure would disconnect the network, which would lead to undelivered traffic and a high penalty. Hence, link failures in a network that lacks robustness are rare events according to our definition. On each day, each link goes down with probability $1/1460$, or approximately once every four years. When a link fails, it remains down for a random amount of time that is normally distributed with a mean of 12 hours and standard deviation of 2 hours. In a tree network with 10 nodes and 9 links, this is equivalent to seeing at least one link fail with probability of approximately 0.00896 each day during a 10 year simulation period; this is our rare event probability.

Algorithm 3 contains the high-level description of the steps in the networking simulation. The algorithm begins by initializing the network, which consists of

² As we are using data provided by an American company, the units are in miles. In order to maintain consistency, we will continue to use miles for quantities based on these data, and note that 1 mile is approximately 1.61 km.

setting the locations of the nodes, any initial links, and the initial traffic. The algorithm then iterates a sequence of steps for each day in the simulation. Our simulations last 10 virtual years, or 3650 virtual days, as we ignore leap years. The **CalculateLinkFailures** method simulates the link failures by taking the network as input and returning a copy of the network with link failures that are determined according to a given link failure probability. After the link failures are determined, the agent is presented with the current state of the network, and chooses its actions. This process is encapsulated in the **GetAgentAction** method. Next, the **SimulateTraffic** method is called, which simulates the network traffic for a single day, restoring the links whose failure times have run out, and returns the reward r_1 that represents the profit, or loss, from the network. After the network has been simulated for one day, the agent's actions are performed by the **PerformAction** method, which returns the new network updated with any upgrades or new links that the agent has chosen, as well as the cost r_2 of performing the actions. We calculate the total reward by adding r_1 and r_2 , and then provide the agent with the reward using the method **AgentHandleReward**, which allows the agent to observe the updated state of the network, and the reward generated on the current day. The **AgentHandleReward** method contains the necessary logic for updating the value function of the agent based on its performance. Finally, at the end of each iteration, the traffic demands are increased by the **UpdateNetworkTraffic** method.

The instance of the network planning problem that we consider was first presented by Hobbs and Bell (2002) and reinforcement learning approaches to the

Algorithm 3 Network Planning Problem

Initialization: Initialize network N .

For each day from $0, \dots, 3650$:

1. $N \leftarrow \mathbf{CalculateLinkFailures}(N)$.
 2. $a \leftarrow \mathbf{GetAgentAction}(N)$.
 3. $(N, r_1) \leftarrow \mathbf{SimulateTraffic}(N)$.
 4. $(N, r_2) \leftarrow \mathbf{PerformAction}(N, a)$.
 5. $r \leftarrow r_1 + r_2$.
 6. $\mathbf{AgentHandleReward}(N, r)$.
 7. $N \leftarrow \mathbf{UpdateNetworkTraffic}(N)$.
-

problem were studied by Precup (2002) and Vigeant (2007). The network planning task is similar the problem of network formation games that has been studied in the context of game theory. A good overview of general network formation games was presented by (Jackson, 2003), and Altman et al. (2006) studies network games in the specific context of telecommunications networks. What makes the network planning task difficult is that the problem is combinatoric; the number of segments of the network increases proportional to the square of the number of cities. The problem is further compounded by the addition of a time component, and by the fact that link failures lead to high variance for simulation-based techniques. The action space can be infinite, and is difficult to represent in a manner that is simple to implement, yet rich enough to build arbitrary networks.

Both Hobbs and Bell (2002) and Precup (2002), consider a simplified version of the task without link failures. In their work, the focus is on ways to reduce the dimensionality so that the approaches could scale up to larger networks. Nodes are clustered geographically, using fuzzy clusters, such that every node has non-zero membership in every cluster. The states are represented by the amount of

bandwidth in each cluster, calculated by taking the average bandwidth on each segment, weighted by the membership of its endpoint nodes in the cluster. The actions are to increase the capacity in a specific cluster, and the result of taking an action on a cluster is that a pair of cities within the cluster is selected randomly, with probability related to the membership of the cities in the cluster, and a link was built between the cities, or if a link existed, it was upgraded. Therefore, the effect of an action is stochastic, and so even under a fixed, deterministic policy, the resulting network topology could vary greatly from run to run. This led to high variance in the value function estimates, and the results were quite poor. Vigeant (2007) presented a number of RL-based approaches, allowing the agent to make more precise changes to the network. These methods performed well on the 10 node network, but required large amounts of computation time. For our experiments, we re-implemented the simulator using slightly different parameters, and so our results cannot be directly compared with previous results. In addition, for the control task we also consider a larger 26-node network in order to demonstrate the scalability of our algorithm.

5.2.1 Policy Evaluation

First, we tested the REASA algorithm for policy evaluation on a network with 10 nodes. The traffic demands are based on real data provided by a telecommunications company (Hobbs and Bell, 2002). We implemented a network planning agent with a simple heuristic policy, which starts with a spanning-tree network, shown in Figure 5.2.1, and then monitors the links, adding capacity when the utilization of a link reaches 90%. The agent does not build new links, and so the

network structure remains the same throughout the course of the simulation. We use REASA and TD(λ) to estimate the value of this policy. Clearly, in a tree network, link failures will be catastrophic, and so we expect TD(λ) will require much more data in order to provide a good estimate for the policy, as the time required to observe a sufficient number of link failures is large. REASA, on the other hand, has the ability to increase the rate of the link failures, and so we expect that it will provide a better estimate in fewer steps. The state space consists of information regarding the existing links, the number of fibers active on each link, the traffic demand between each pair of cities, the utilization of each link, any link failures that are occurring, and other properties. Because the state space is large, we represent the network state as a vector of binary features, and use linear function approximation to represent the value function. In order to cope with the high dimensionality, we use a fairly coarse state representation, consisting of: 45 binary features indicating whether each of the possible links have been built; 45 binary features indicating, for each possible link, whether the link is in a failure state; and the percentage utilization of each link, partitioned into 4 bins: $[0]$, $(0, 0.6]$, $(0.6, 0.9]$, and $(0.9, 1.0]$. For our 10 node network, this corresponds to 270 binary features plus an additional bias feature.

We use a discount factor of 0.95 and we set $\lambda = 1.0$. We use a decaying schedule for the learning rate parameter α , starting with a value of $\alpha_0 = 2^{-15}$ for $T = 100$ episodes, then using $\alpha_0/2$ for $2T$ episodes, $\alpha_0/4$ for $4T$ episodes, etc. We note that α_0 is small due to the fact that the rewards often have large magnitude and can vary between -10^7 and 10^5 . These parameters were specifically tuned to

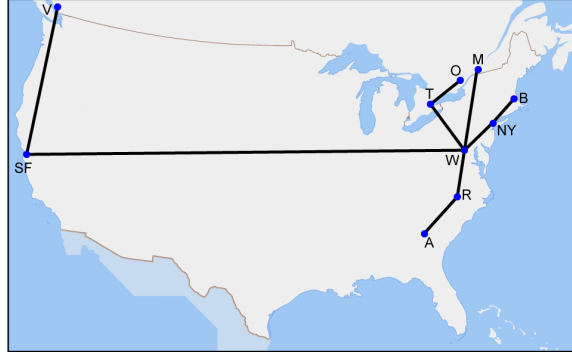


Figure 5–2: 10-node spanning-tree network used for policy evaluation experiments

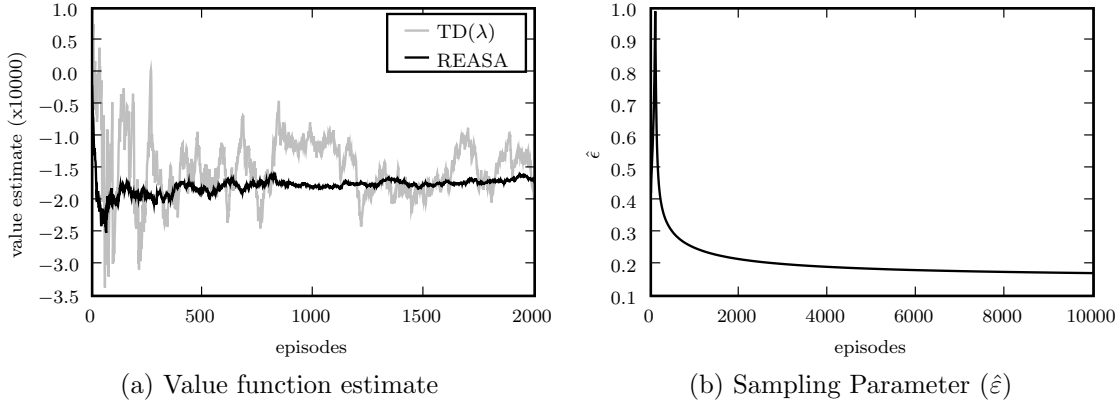


Figure 5–3: Value function estimates and rare event sampling parameter estimates for the tree network

this task. In the following results, an episode consists of a simulated 10-year time span, with each time step corresponding to 1 day.

In Figure 5–3a, we show the value estimates for the initial state, over the first 2000 episodes. We see that REASA converges quickly, while the $\text{TD}(\lambda)$ estimates have high variance and converge quite slowly. On longer runs, the $\text{TD}(\lambda)$ estimates do converge to the same value as REASA. In Figure 5–3b, we show the rare event sampling parameter estimates $\hat{\epsilon}$ over the first 10000 episodes. After 10000 episodes,

REASA estimates the optimal failure probability to be 0.155, which in a tree network with 9 links corresponds to each link going down approximately every 54 days; this is quite far from the original failure probability of once every 1460 days³.

The rate of convergence is crucial for applications such as the network task. Each episode corresponds to a simulated 10 year period, and these simulations are computationally expensive to run, because on each day, a routing algorithm has to be run to determine the reward. Hence, the gains obtained by REASA are significant.

5.2.2 Policy Optimization

For the control task, we integrated REASA into the standard SARSA(λ) algorithm (Sutton and Barto, 1998), which for the purpose of this section we will still refer to as REASA. We evaluated the performance of REASA on two networks, the 10-node network from Hobbs and Bell (2002) that was used in the previous section, and a 26-node network based on publicly available data on the AT&T IP backbone network from 2Q2000, which was studied by Maxemchuk et al. (2005).

The state was represented using linear function approximation with the same features as in the previous section. This corresponds to 271 binary features for

³ Although the value function estimates effectively converged after the first 2000 episodes, we allowed the agent to continue to run in order to allow the $\hat{\epsilon}$ parameter estimate to converge as well.

the 10-node network, and 1950 for the 26-node network. For the actions, we chose to have one action for each possible link, where an action corresponds to either building the link if it does not exist, or upgrading the link if it does exist, and one additional action which represents making no change to the network. This corresponds to 46 actions for the 10-node network, and 326 actions for the 26-node network. Therefore, at each step, the agent is only able to make a single change to the network. We would therefore expect the network to be built slowly, and so a large negative reward will be accumulated early in the episode, before the network is connected. Ideally, then, our agent will not only learn to design a network that is robust to link failures, but it will learn a policy to build a network from scratch such that important traffic is delivered early in the building process. Such a strategy would be important for, say, a telecommunication company that is entering a new market. The company may have limited resources, and only have the ability to build up its network incrementally, and such an agent would be able to choose where to place links to generate the highest revenue the fastest.

With such a large action space, exploration becomes a challenge. Also in early experiments we found that since taking actions early in the episode are so important, the agent initially assigns a much lower value to the action which makes no change to the network than to all other actions. This leads to the agent always wanting to make changes to the network, even once a good network has been built. To assist the agent in learning when to stop building, we consider two higher-level actions, one which makes some change to the network, the “do something” action, and the second, which makes no change to the network, the

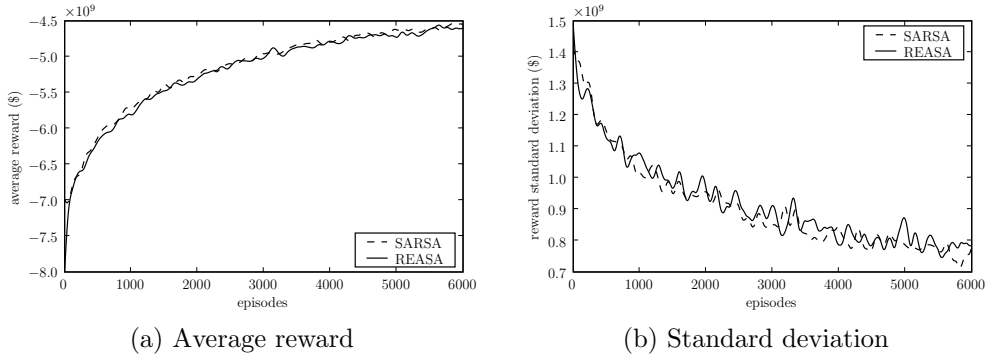


Figure 5-4: Learning curves and standard deviation for REASA and SARSA on the 10-node network

“do nothing” action. On each time step, the agent first decides with to “do something”, in which case it then chooses a primitive action from the set of actions that make changes to the network, or “do nothing”, in which case it chooses the primitive action that makes no changes to the network. We found that adding this high-level decision greatly improved the performance of the agent. For both the primitive and high-level actions, we use Boltzmann exploration, with the temperature schedule tuned for the specific network.

For the 10-node network, we use the decaying learning rate schedule described in the previous section with $\alpha_0 = 2^{-5}$ and $T = 125$. We set $\lambda = 0.9$ and $\gamma = 0.995$. We use the same temperature schedule for the Boltzmann exploration for the high-level and primitive actions, with temperature $\tau = 50.0/\sqrt{t+1}$, where t is the current episode. We compared REASA to standard SARSA, using the same parameters. The learning curves are shown in Figure 5.2.2. What should be immediately apparent from the learning curves is that the performance of REASA and SARSA are nearly identical. REASA exhibits only slightly lower variance.

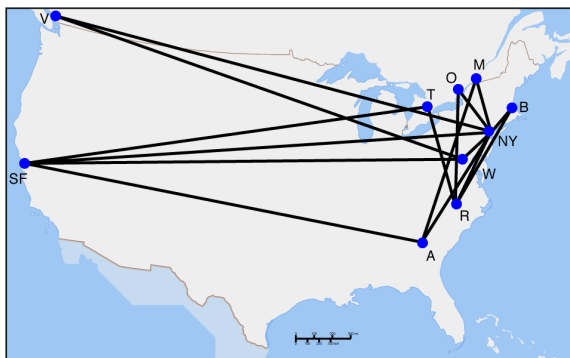


Figure 5–5: Example 10-node network built by REASA learning agent

The reason for this is that in the control problem, the learning agent spends very little time considering networks in which failures are rare events. Learning seems to proceed in two stages, first the agent learns to grow very large networks, and then it learns to prune them down to a reasonable size. When the network is highly connected, network failures do not result in traffic loss, and therefore have no noticeable effect on performance. Therefore, tuning our rare event parameter has no effect on the learning rate. An example of a network built by REASA is shown in Figure 5.2.2, and we see that the agent has learned to build networks with redundant routes to avoid traffic loss in the event of link failures.

For the 26-node network, we use parameters $\lambda = 0.9$, $\gamma = 0.995$, $\alpha_0 = 2^{-5}$, and $T = 100$. For the Boltzmann exploration temperature, we use $\tau = 101.0/(t + 1)^{0.7}$, where t is the current episode, for both the high-level and primitive actions. For a baseline, we use the AT&T network (shown in Figure 5–7a), and created an agent that would assemble the network one piece per time step, choosing the next link to build uniformly randomly from the links that have not yet been built. Since this network represents the actual backbone network that AT&T employed,

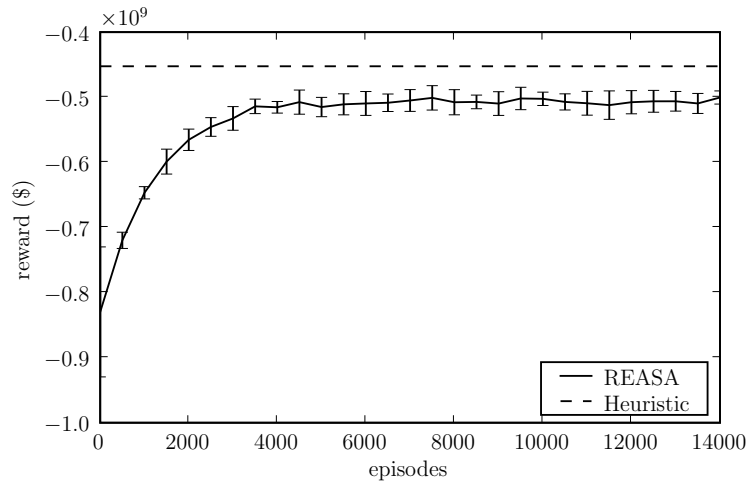
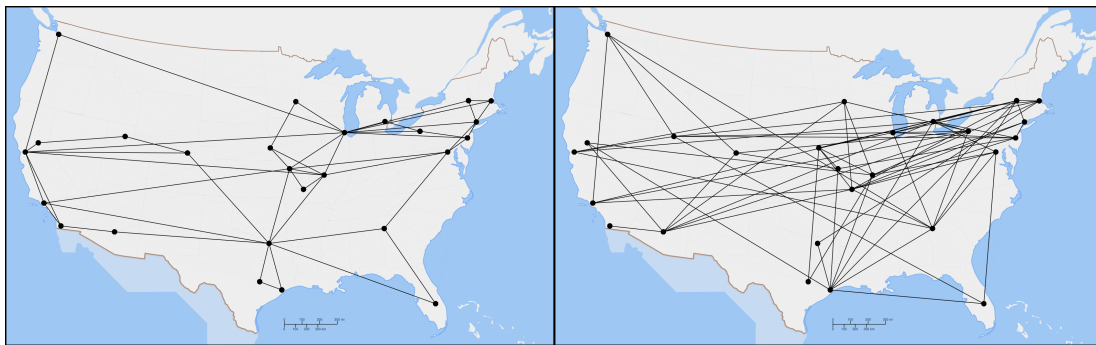


Figure 5-6: Learning curve for REASA on the 26-node AT&T network



(a) AT&T IP Backbone Network

(b) Example network built by REASA

Figure 5-7: AT&T IP backbone network and network built by REASA learning agent

it is reasonable to assume that this network was designed by experts, and so this represents as near an optimal network as we could expect to see. Because of the stochastic nature of the heuristic agent, we ran 1000 independent 10-year simulations, and took the average value for the baseline. For REASA, we use the reward averaged over 10 independent 10-year simulations. Figure 5.2.2 shows the learning curve and error bars, and Figure 5–7b. REASA does not quite achieve the same performance as the heuristic, but it comes fairly close. This is a significant result, in that to our knowledge, this is the largest network that has been designed completely by a learning agent equipped with no domain knowledge, and the performance of the network is nearly 90% of that of a network designed by domain experts. This performance statistic was calculated using the average of the last 1000 episodes of each of the 10 independent runs. The time required for each individual run on a 3.0GHz Pentium 4 was approximately 716.5 hours, or just under 30 days.

We note that the algorithm is fairly robust to parameter settings. The most important, and most difficult parameter to tune is the exploration temperature. The parameter settings that we used were fairly naive, and so it is likely that even better performance could be obtained by tuning these parameters.

It is unfortunate that REASA did not lead to significant performance gains over SARSA. As we noted earlier, this is due to fact that during the learning process, the events that we originally identified to be rare events, link failures, do not turn out to fit our definition or rare events because they do not significantly affect the value function estimates. It is likely that there are other domains in

which rare events play a factor during the learning process, but we leave such investigation for future work.

CHAPTER 6

Conclusions and Future Work

We conclude with a review of the contributions made in this work, and a discussion of future research directions.

6.1 Contributions

In this thesis, we deviate from the typical approach taken by researchers in RL. In most applications of RL, the simulated environment is treated as a black box that generates samples, and the focus is on improving the performance of the agent. However, the simulated environment is typically a software application, and there are usually parameters that can be modified, leading to different behaviours. If we are aware of the effect of changing these parameters, then we can do so and still obtain agents that perform well on the original problem. In certain scenarios, such as the rare events scenario that we consider in this work, changing the parameters can improve the performance of the learning agent. We present a novel algorithm, based on adaptive importance sampling, that improves the performance of reinforcement learning agents in environments where rare significant events occur. The performance improvements have been empirically demonstrated for policy evaluation, but we found that for the specific network planning task that we considered, there were no improvements in the control case.

This should not be seen as implying that our methods cannot improve performance for policy optimization problems, only that the network planning task

is not an appropriate task for these kinds of methods. As we noted in Section 5.2.2, for the network planning task, most of the learning is done in a part of the state space where there are no rare events, and so it is unsurprising that our algorithm does not lead to improved performance. On the other hand, our method of splitting the actions into a high-level action and primitive actions, and performing action selection in two phases led to an algorithm that could design very good networks with a reasonable number of nodes¹. To the best of our knowledge of the available literature, we have presented the largest network that has been built from scratch by a learning agent with no prior domain knowledge.

6.2 Future Research Directions

As we noted in the previous section, this work focuses on improving the performance of a reinforcement learning agent by exploiting properties of the simulated environment. We focused on the specific problem of rare events, and incorporated techniques from the stochastic simulation community, where rare events have been widely investigated, into a learning algorithm. By assuming that we have some control over the simulator, which is generally the case, this opens the door to a whole new area of research. We looked at rare event simulation, but there are many other variance reduction techniques from the stochastic simulation literature that can be applied.

¹ This approach was suggested by Doina Precup, based on her previous work on the network planning task.

One obvious extension to this work would be to consider other types of parameterized transition distributions. We looked at the case where the parameter at our disposal controlled the probability of rare events occurring in a system. However, there are many other cases of parameterized transition probabilities. For example, in the standard mountain-car problem (Sutton and Barto, 1998), we could take the slope of the valley as a parameter, and then consider whether an agent may be able to improve its performance by altering this parameter. The idea of learning on easier problems and then transferring the knowledge to other, harder problems is an area of active research (Singh, 1992; Madden and Howley, 2004; Taylor and Stone, 2005), called transfer learning, and our approach may be useful in that area.

Our approach also assumes that rewards are deterministic, as is consistent with most work on adaptive importance sampling. Without this assumption, it cannot be assumed that it is always best to sample transitions proportionally to their contribution to the value function. When the rewards are stochastic, it may also be important to consider the variance of the rewards. This is similar to the work on upper confidence bounds (UCB) and the multi-armed bandit problem (Auer, 2002; Even-Dar et al., 2006). It is likely that much of the work on UCB can easily be adapted to our approach, allowing the sampling procedure to incorporate a measure of uncertainty.

Finally, we note that for the function approximation case, we are restricted to having only one estimate for our rare event parameter, which is used independent of the current state. If we are able to represent the rare event probability as a

linear function of our state features, then we would be able to lift this assumption. Gradient descent could then be used to learn a set of weights for representing not only the value function, but the optimal rare event parameter as well, similar to what is done in the tabular case. This is likely to greatly improve the performance of the algorithm when used with linear function approximation.

APPENDIX A General Notation

A.1 Probability and Random Variables

Let Ω denote a possibly infinite sample space, whose elements ω are called sample outcomes or realizations. We denote the complement of a set A as A^c , and the empty set as \emptyset . Subsets of Ω are called events, and we restrict ourselves to a set of events called a σ -field, which is a class \mathcal{A} that satisfies: $\emptyset \in \mathcal{A}$, if $A_1, A_2, \dots \in \mathcal{A}$ then $\cup_{i=1}^{\infty} A_i \in \mathcal{A}$, and $A \in \mathcal{A}$ implies that $A^c \in \mathcal{A}$. Elements of \mathcal{A} are said to be measurable. A function \mathbb{P} that assigns a real number $\mathbb{P}(A)$ to each measurable event A a probability distribution or probability measure if it satisfies: $\mathbb{P}(A) \geq 0$ for every A , $\mathbb{P}(\Omega) = 1$, and if A_1, A_2, \dots are disjoint, then

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mathbb{P}(A_i).$$

We call (Ω, \mathcal{A}) a measurable space, and if \mathbb{P} is a probability measure defined on \mathcal{A} , then $(\Omega, \mathcal{A}, \mathbb{P})$ is called a probability space.

A random variable X is a measurable map $X : \Omega \rightarrow \mathbb{R}$, where measurable means that for every x , $\{\omega : X(\omega) \leq x\} \in \mathcal{A}$. X is discrete if it takes countably many values, and continuous otherwise. For a discrete random variable X , we define the probability function, or probability mass function $f_X(x) = \mathbb{P}(X = x)$. For a continuous random variable X , we define the probability function or probability density function (PDF) as a function f_X such that $f_X(x) \geq 0$ for all x ,

$\int_{-\infty}^{\infty} f_X(x)dx = 1$, and for every $a \leq b$,

$$\mathbb{P}(a < X < b) = \int_a^b f_X(x)dx.$$

When it is clear what random variable a probability function is defined for, we drop the subscript X to ease our notation. In addition, since getting bogged down in precise measure theoretic details will only serve to make this thesis less readable, we will often resort to sloppy notation, and refer to f as a probability distribution, or law, regardless of whether X is continuous or discrete.

A.2 Expectation and Variance

The expected value or mean of a random variable X is defined to be

$$\mathbb{E}(X) = \begin{cases} \sum_x xf(x) & \text{if } X \text{ is discrete;} \\ \int xf(x)dx & \text{if } X \text{ is continuous,} \end{cases}$$

where f is a probability function. The variance of a random value with mean μ is defined to be

$$\sigma_X^2 = \mathbb{E}((X - \mu)^2),$$

assuming this expectation exists. We also write $\mathbb{V}(X)$ to denote the variance of X , or σ^2 when it is obvious which random variable we are referring to. The standard deviation is $\text{sd}(X) = \sqrt{\mathbb{V}(X)}$, and is also denoted by σ_X or σ . For random variables X and Y with means μ_X and μ_Y and standard deviations σ_X and σ_Y , we define the covariance between X and Y as

$$\text{Cov}(X, Y) = \mathbb{E}((X - \mu_X)(Y - \mu_Y)).$$

The covariance between X and Y gives a measure of how strong the linear relationship is between X and Y .

A.3 Convergence of Random Variables

Let X_1, X_2, \dots be a sequence of random variables, and let X be another random variable. We say that X_n converges to X (written $X_n \rightarrow X$) in probability if, for every $\epsilon > 0$,

$$\lim_{n \rightarrow \infty} \mathbb{P}(|X_n - X| > \epsilon) = 0.$$

We say that $X_n \rightarrow X$ almost surely (a.s.), or with probability 1 (w.p.1), if

$$\mathbb{P}\left(\lim_{n \rightarrow \infty} X_n = X\right) = 1.$$

A.4 Markov Chains

A Markov chain is a stochastic processes possessing the Markov property, which says that given the present state, the future states are independent of the past states. We represent a Markov chain as a sequence of random variable X_1, X_2, \dots , sometimes written as $(X_i)_{i=1}^{\infty}$, or $(X_i)_{i=1}^n$ if the sequence is finite. The set of possible values of X_i is referred to as the state space, which we denote by \mathcal{X} , and the process evolves according to a single step transition probability distribution p , where the probability of going from state x to state y is given by

$$p_{xy} = \mathbb{P}(X_{n+1} = y | X_n = x).$$

We define the probability of going from state x to state y in n steps as

$$p_{xy}^{(n)} = \mathbb{P}(X_n = y | X_0 = x).$$

A Markov chain is time-homogeneous if p does not change over time. For Markov chains with countable state spaces, or finite state spaces of size N , it is often useful to label the states using the natural numbers, or elements of the set $\{1, 2, \dots, N\}$ respectively. For Markov chains with finite state spaces, we can represent the transition probability distribution by a matrix $P = (p_{xy} : x, y \in \mathcal{X})$, called the transition matrix, whose (x, y) th entry is p_{xy} . For time-homogeneous finite Markov chains, the k -step transition probabilities can be calculated as P^k .

A state y is said to be accessible from x if there exists an integer $n \geq 0$ such that $p_{xy}^{(n)} > 0$. Two states x and y are said to be communicating if y is accessible from x and x is accessible from y . A communicating class $\mathcal{C} \subseteq \mathcal{X}$ is a set of states where every pair of states in \mathcal{C} communicates and no state in \mathcal{C} communicates with a state outside of \mathcal{C} . A Markov chain is said to be irreducible if its states \mathcal{X} form a communicating class.

The period of a state is defined as $k = \gcd \{n : \mathbb{P}(X_n = x | X_0 = x) > 0\}$, where \gcd is the greatest common divisor. This means that if a state x has period k , then the number of steps that occur from one visit to x to the next must be a multiple of k . If the period k of a state is 1, then that state is said to be aperiodic, otherwise if $k > 1$ the state is said to be periodic with period k . A finite state irreducible Markov chain is said to be ergodic if its states are aperiodic.

A state x is said to be transient if, given that we start in x , there is a non-zero probability that we will never return to x . A state is called recurrent if it is not transient. A state x is called absorbing if it is impossible to leave that state, that is that $p_{xx} = 1$ and $p_{xy} = 0$ for $y \neq x$.

A vector π is called the stationary distribution, or equilibrium distribution, for a time-homogeneous Markov chain if its entries π_y sum to 1 and satisfy

$$\pi_y = \sum_{x \in \mathcal{X}} \pi_x p_{xy}.$$

For a finite state Markov chain, the stationary distribution π is a row vector which satisfies $\pi = \pi P$. Since P is a stochastic matrix, π always exists, but is not necessarily unique. However, if the Markov chain is irreducible and aperiodic, then the Perron-Frobenius theorem ensures that the stationary distribution π is unique and satisfies

$$\lim_{k \rightarrow \infty} P^k = \mathbf{1}\pi,$$

where $\mathbf{1}$ is the column vector with all entries equal to 1. This can be interpreted as saying that as time goes by, the Markov chain forgets where it began, and converges to its stationary distribution.

APPENDIX B

Proofs

B.1 Proof of Theorem 3.2.1

Theorem: *Given an MDP with states \mathcal{S} , actions \mathcal{A} , one-step transition probability distribution p , and reward function r , the zero-variance importance sampling distribution p^* is given by*

$$p^*(s'|s, a) = \frac{p(s'|s, a) \sum_{a' \in \mathcal{A}} \pi(a'|s') [r(s, a, s') + \gamma Q^\pi(s', a')]}{Q^\pi(s, a)}$$

Proof: This proof follows a similar form as the derivation of the zero-variance importance sampling distribution for discrete Markov chains by Kollman (1993). However, they consider transient Markov chains and undiscounted rewards. We also fill in a few details that were left out of their proof (namely Lemma B.1.1). Although a large amount of this proof is nearly identical to the cited proof, we include it for completeness.

Given a policy π , we consider the discrete Markov chain $(X_t)_{t=0}^\infty$ of state-action pairs, where $X_t = (s_t, a_t)$, $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$ for all $t \geq 0$. s_0 is drawn from an initial state distribution μ , a_0 is generated by the policy $\pi(\cdot|s_0)$ and the state-action to state-action transitions occur according to

$$p(s', a'|s, a) = \mathbb{P}(s_{t+1} = s', a_{t+1} = a' | s_t = s, a_t = a) = p(s'|s, a)\pi(a'|s').$$

The discounted return for the chain becomes

$$R_p = \sum_{t=1}^{\infty} \gamma^{t-1} r(s_{t-1}, a_{t-1}, s_t),$$

where the subscript p denotes the usage of the distribution p for generating transitions. The action-value function becomes

$$Q_p^\pi(s, a) = \mathbb{E}_p(R_p | s_0 = s, a_0 = a),$$

The state-action transitions can be generated by an alternative transition probability distribution q having the same support as p , in which case

$$q(s', a' | s, a) = \mathbb{P}(s_{t+1} = s', a_{t+1} = a' | s_t = s, a_t = a) = q(s' | s, a) \pi(a' | s').$$

We denote our discounted return for this auxiliary chain

$$R_q = \sum_{t=1}^{\infty} \gamma^{t-1} r(s_{t-1}, a_{t-1}, s_t) \mathcal{L}_t,$$

where \mathcal{L}_t is the likelihood of a trajectory under the alternative dynamics, given by

$$\mathcal{L}_t = \prod_{i=1}^t \frac{p(s_i, a_i | s_{i-1}, a_{i-1})}{q(s_i, a_i | s_{i-1}, a_{i-1})}.$$

We denote the action-value function under the dynamics q as

$$Q_q^\pi(s, a) = \mathbb{E}_q(R_q | s_0 = s, a_0 = a).$$

Lemma B.1.1. *Let Q_q and R_q be the state-action value function and expected return, respectively, under our importance sampling distribution q , and let Q_p and R_p be the state-action value function and expected return, respectively, under the*

original transition distribution p . If the likelihood ratios are finite, that is $\mathcal{L}_t < \infty$ for all t , then

$$Q_q^\pi(s, a) = \mathbb{E}_q(R_q | s_0 = s, a_0 = a) = \mathbb{E}_p(R_p | s_0 = s, a_0 = a) = Q_p^\pi(s, a).$$

Proof: Let $R_p^{(n)}$ denote the n -step return under p , given by

$$R_p^{(n)} = \sum_{i=1}^n \gamma^{i-1} r(s_{i-1}, a_{i-1}, s_i),$$

and the $R_q^{(n)}$ denote the weighted n -step return under q , given by

$$R_q^{(n)} = \sum_{i=1}^n \gamma^{i-1} r(s_{i-1}, a_{i-1}, s_i) \mathcal{L}_i.$$

We show by induction on n , that

$$\mathbb{E}_q(R_q^{(n)} | s_0 = s, a_0 = a) = \mathbb{E}_p(R_p^{(n)} | s_0 = s, a_0 = a)$$

for all $n > 0$, $s \in \mathcal{S}$, and $a \in \mathcal{A}$, thus establishing our result.

For $n = 1$, let $s_0 = s$ and $a_0 = a$, then

$$\begin{aligned} \mathbb{E}_q(R_q^{(1)} | s_0 = s, a_0 = a) &= \sum_{s' \in \mathcal{S}} q(s' | s, a) r(s, a, s') \frac{p(s' | s, a)}{q(s' | s, a)} \\ &= \sum_{s' \in \mathcal{S}} p(s' | s, a) r(s, a, s') \\ &= \mathbb{E}_p(R_p^{(1)}). \end{aligned}$$

Define the n step transition probabilities under p and q respectively as

$$p^{(n)}(s_t, a_t | s_0, a_0) = \sum_{\substack{s_1, \dots, s_t \\ a_1, \dots, a_t}} \prod_{i=1}^n \pi(a_i | s_i) p(s_i | s_{i-1}, a_{i-1}),$$

and

$$q^{(n)}(s_t, a_t | s_0, a_0) = \sum_{\substack{s_1, \dots, s_t \\ a_1, \dots, a_t}} \prod_{i=1}^n \pi(a_i | s_i) q(s_i | s_{i-1}, a_{i-1}).$$

The expected likelihood can then be written as

$$\begin{aligned} \mathbb{E}_q(\mathcal{L}_n | s, a) &= \sum_{\substack{s' \in \mathcal{S} \\ a' \in \mathcal{A}}} \frac{p^{(n)}(s', a' | s, a)}{q^{(n)}(s', a' | s, a)} \\ &= \sum_{\substack{s' \in \mathcal{S} \\ a' \in \mathcal{A}}} \frac{p^{(n-1)}(s', a' | s, a)}{q^{(n-1)}(s', a' | s, a)} \sum_{s'' \in \mathcal{S}} \frac{p(s'' | s', a')}{q(s'' | s', a')}, \end{aligned}$$

where the last line follows from the Markov property.

Now suppose that for $k > 1$,

$$\mathbb{E}_q(R_q^{(k-1)} | s_0 = s, a_0 = a) = \mathbb{E}_p(R_p^{(k-1)} | s_0 = s, a_0 = a),$$

then by the Markov property and our induction hypothesis we have:

$$\begin{aligned} \mathbb{E}_q(R_q^{(k)} | s_0 = s, a_0 = a) &= \mathbb{E}_q(R_q^{(k-1)} | s_0 = s, a_0 = a) \\ &\quad + \gamma^{k-1} \sum_{\substack{u \in \mathcal{S} \\ v \in \mathcal{A}}} q^{(k-1)}(u, v | s, a) \sum_{\substack{s' \in \mathcal{S} \\ a' \in \mathcal{A}}} q(s' | u, v) r(u, v, s') \\ &\quad \cdot \frac{p^{(k)}(s', a' | s, a)}{q^{(k)}(s', a' | s, a)} \\ &= \mathbb{E}_p(R_p^{(k-1)} | s_0 = s, a_0 = a) \\ &\quad + \gamma^{k-1} \sum_{\substack{s' \in \mathcal{S} \\ a' \in \mathcal{A}}} p^{(k-1)}(s', a' | s, a) \sum_{s'' \in \mathcal{S}} p(s'' | s', a') r(s', a', s'') \\ &= \mathbb{E}_p(R_p^{(k)} | s_0 = s, a_0 = a). \end{aligned}$$

■

Since the value functions do not depend on the transition distribution, we can drop the subscript, and write $Q^\pi(s, a)$. Let $v_{sa} = \mathbb{V}(R_q | s_0 = s, a_0 = a)$ denote the variance of the expected reward R_q .

Lemma B.1.2. *By the Markov property,*

$$\mathbb{E}(R_q | s_0 = s, a_0 = a, s_1 = s', a_1 = a') = \frac{p(s'|s, a)}{q(s'|s, a)} (r(s, a, s') + \gamma Q^\pi(s', a')),$$

and

$$\mathbb{V}(R_q | s_0 = s, a_0 = a, s_1 = s', a_1 = a') = \gamma^2 \left(\frac{p(s'|s, a)}{q(s'|s, a)} \right)^2 v_{s'a'}.$$

Proof:

$$\begin{aligned} \mathbb{E}(R_q | s_0, a_0, s_1, a_1) &= \mathbb{E} \left(\sum_{t=1}^{\infty} \gamma^{t-1} r(s_{t-1}, a_{t-1}, s_t) \mathcal{L}_t | s_0, a_0, s_1, a_1 \right) \\ &= \mathbb{E} \left(r(s_0, a_0, s_1) \mathcal{L}_1 \right. \\ &\quad \left. + \gamma \mathcal{L}_1 \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \frac{\mathcal{L}_{t+1}}{\mathcal{L}_1} | s_0, a_0, s_1, a_1 \right) \\ &= \frac{p(s_1, a_1 | s_0, a_0)}{q(s_1, a_1 | s_0, a_0)} [r(s_0, a_0, s_1) + \gamma Q^\pi(s_1, a_1)], \\ \mathbb{V}(R_q | s_0, a_0, s_1, a_1) &= \mathbb{V} \left(\sum_{t=1}^{\infty} \gamma^{t-1} r(s_{t-1}, a_{t-1}, s_t) \mathcal{L}_t \middle| s_0, a_0, s_1, a_1 \right) \\ &= \mathbb{V} \left(r(s_0, a_0, s_1) \mathcal{L}_1 \right. \\ &\quad \left. + \gamma \mathcal{L}_1 \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \frac{\mathcal{L}_{t+1}}{\mathcal{L}_1} \middle| s_0, a_0, s_1, a_1 \right) \\ &= \gamma^2 \left(\frac{p(s_1, a_1 | s_0, a_0)}{q(s_1, a_1 | s_0, a_0)} \right)^2 \mathbb{V}(R_q | s_1, a_1). \end{aligned}$$

■

Lemma B.1.3. *The variance of the expected reward under the importance sampling distribution q is*

$$v_{sa} = \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} \frac{\pi(a'|s')p(s'|s, a)^2 [r(s, a, s') + \gamma Q^\pi(s', a')]^2}{q(s'|s, a)} - Q^\pi(s, a)^2 \\ + \gamma^2 \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} \frac{\pi(a'|s')p(s'|s, a)^2}{q(s'|s, a)} v_{s'a'},$$

Proof:

$$\begin{aligned} v_{sa} &= \mathbb{V}(R_q | s_0 = s, a_0 = a) \\ &= \mathbb{V}(\mathbb{E}(R_q | s_0, a_0, s_1) | s_0 = s, a_0 = a) + \mathbb{E}(\mathbb{V}(R_q | s_0, a_0, s_1) | s_0 = s, a_0 = a) \\ &= \mathbb{V}\left(\frac{p(s_1, a_1 | s_0, a_0)}{q(s_1, a_1 | s_0, a_0)} [r(s_0, a_0, s_1) + \gamma Q^\pi(s_1, a_1)] \Big| s_0 = s, a_0 = a\right) \\ &\quad + \mathbb{E}\left(\gamma^2 \left(\frac{p(s_1, a_1 | s_0, a_0)}{q(s_1, a_1 | s_0, a_0)}\right)^2 \mathbb{V}(R_q | s_1, a_1) \Big| s_0 = s, a_0 = a\right) \\ &= \mathbb{E}\left(\left(\frac{p(s_1, a_1 | s_0, a_0)}{q(s_1, a_1 | s_0, a_0)}\right)^2 [r(s_0, a_0, s_1) + \gamma Q^\pi(s_1, a_1)]^2 \Big| s_0 = s, a_0 = a\right) \\ &\quad - \mathbb{E}\left(\frac{p(s_1, a_1 | s_0, a_0)}{q(s_1, a_1 | s_0, a_0)} [r(s_0, a_0, s_1) + \gamma Q^\pi(s_1, a_1)] \Big| s_0 = s, a_0 = a\right)^2 \\ &\quad + \mathbb{E}\left(\gamma^2 \left(\frac{p(s_1, a_1 | s_0, a_0)}{q(s_1, a_1 | s_0, a_0)}\right)^2 \mathbb{V}(R_q | s_1, a_1) \Big| s_0 = s, a_0 = a\right) \\ &= \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} \frac{(p(s'|s, a)\pi(a'|s'))^2 [r(s, a, s') + \gamma Q^\pi(s', a')]^2}{q(s'|s, a)\pi(a'|s')} - Q^\pi(s, a)^2 \\ &\quad + \gamma^2 \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} \frac{(p(s'|s, a)\pi(a'|s'))^2}{q(s'|s, a)\pi(a'|s')} v_{s'a'}. \end{aligned}$$

■

Let $N = |\mathcal{S} \times \mathcal{A}|$, and denote as \mathbf{v} the N element vector with entries v_{sa} . If we define \mathbf{f} to be the N element vector with entries defined by

$$f_{sa} = \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} \frac{\pi(a'|s')p(s'|s,a)^2 [r(s,a,s') + \gamma Q^\pi(s',a')]^2}{q(s'|s,a)} - Q^\pi(s,a)^2,$$

and define \mathbf{R} to be the $N \times N$ matrix whose $(sa, s'a')$ th element is given by

$$r_{sas'a'} = \gamma^2 \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} \frac{\pi(a'|s')p(s'|s,a)^2}{q(s'|s,a)}$$

(with $0/0 = 0$ is defined), then in matrix form the variance equation becomes

$$\mathbf{v} = \mathbf{f} + \mathbf{R}\mathbf{v}. \tag{B.1}$$

We now show that Equation B.1 has a unique solution.

Lemma B.1.4. $\mathbf{v} = \sum_{n=0}^{\infty} \mathbf{R}^n \mathbf{f}$.

Proof: First we note that any \mathbf{v} that satisfies Equation B.1 must also satisfy

$$\mathbf{v} = \sum_{i=0}^{n-1} \mathbf{R}^i \mathbf{f} + \mathbf{R}^n \mathbf{v},$$

for all $n > 0$, and so $\hat{\mathbf{v}} = \sum_{n=0}^{\infty} \mathbf{R}^n \mathbf{f}$ is the minimal solution to Equation B.1.

Let

$$R_q^{(n)} = \sum_{i=0}^{n-1} \gamma^i r(s_i, a_i, s_{i+1}) \mathcal{L}_i + \gamma^n Q(s_n, a_n) \mathcal{L}_n,$$

and note that $\mathbb{E}_q \left(R_q^{(n)} | s_0 = s, a_0 = a \right) = Q(s, a)$.

Let $\mathbf{v}^{(n)}$ denote the vector with elements $v_{sa}^{(n)} = \mathbb{V} \left(R_q^{(n)} | s_0 = s, a_0 = a \right)$, then using an analogous argument as before, we see that

$$\mathbf{v}^{(n)} = \mathbf{f} + \mathbf{R}\mathbf{v}^{(n-1)},$$

for all $n > 0$.

By induction,

$$\mathbf{v}^{(n)} = \sum_{i=0}^{n-1} \mathbf{R}^i \mathbf{f},$$

and by Fatou's Lemma and the fact that $R_q = \lim_{n \rightarrow \infty} R_q^{(n)}$, we have that

$$\mathbf{v} \leq \sum_{n=0}^{\infty} \mathbf{R}^n \mathbf{f} = \hat{\mathbf{v}}.$$

Since $\hat{\mathbf{v}}$ is the minimal solution to Equation B.1, we must have that $\mathbf{v} = \hat{\mathbf{v}}$. ■

Finally, we show that for a specific choice of sampling distribution q , we get $\mathbf{f} = \mathbf{0}$.

Lemma B.1.5. *If we let*

$$q(s'|s, a) = \frac{p(s'|s, a) [r(s, a, s') + \gamma V^\pi(s')]}{Q^\pi(s, a)},$$

then if $\gamma > 0$, we get $\mathbf{f} = \mathbf{0}$.

Proof: If we let

$$\begin{aligned} q(s', a'|s, a) &= q(s'|s, a)\pi(a'|s') = \frac{p(s'|s, a)\pi(a'|s') [r(s, a, s') + \gamma Q^\pi(s', a')]}{\sum_{\substack{u \in \mathcal{S} \\ v \in \mathcal{A}}} p(u|s, a)\pi(v|u) [r(s, a, u) + \gamma Q^\pi(u, v)]} \\ &= \frac{p(s'|s, a)\pi(a'|s') [r(s, a, s') + \gamma Q^\pi(s', a')]}{Q^\pi(s, a)}, \end{aligned}$$

then for all $s \in \mathcal{S}, a \in \mathcal{A}$

$$\begin{aligned}
f_{sa} &= \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} \frac{(p(s'|s, a)\pi(a'|s'))^2 [r(s, a, s') + \gamma Q^\pi(s', a')]^2}{q(s'|s, a)\pi(a'|s')} - Q^\pi(s, a)^2 \\
&= Q^\pi(s, a) \sum_{\substack{a' \in \mathcal{A} \\ s' \in \mathcal{S}}} p(s'|s, a)\pi(a'|s') [r(s, a, s') + \gamma Q^\pi(s', a')] - Q^\pi(s, a)^2 \\
&= Q^\pi(s, a)^2 - Q^\pi(s, a)^2 = 0.
\end{aligned}$$

■

Therefore, this choice of q gives us $\mathbf{f} = \mathbf{0}$, and then from Lemma B.1.4 we have that $\mathbf{v} = 0$, giving us a zero-variance importance sampling scheme, and thus completing our proof. ■

B.2 Proof of Theorem 3.2.2

Theorem: *Using Algorithm 1 with the usual step-size conditions on α , α_T , and α_U , for any $\lambda \in [0, 1]$ that does not depend on the action a_t , and assuming that the MDP is unichain for $\varepsilon = \delta$, we have that:*

$$\hat{V}^\pi(s) \rightarrow V^\pi(s) \text{ almost surely.}$$

Moreover, for all $s \in \mathcal{S}$, $\hat{\varepsilon}(s) \rightarrow \tilde{\varepsilon}(s)$ almost surely.

Proof: We start by noting that for a given stationary policy π , REASA without eligibility traces is a special case of the ASA algorithm, which is known to converge (see Ahamed et al., 2006, Theorem 1). Although the sampling probabilities in REASA do not necessarily converge to the zero-variance distribution, as they do in ASA, this property of the sampling distribution is not used in the ASA

convergence proof. The only requirement is that the importance sampling weights are bounded, and the parameter δ ensures that they are. Therefore, we can use their result without modification.

We therefore need to show that adding eligibility traces allows us to retain our convergence guarantees. This follows from a straightforward adaptation of the second part of the proof of Theorem 2 of Precup et al. (2000).

The weighted n -step return under REASA without eligibility traces is given by:

$$R_t^{(n)} = \sum_{i=1}^n \gamma^{i-1} r_{t+i} \prod_{l=t+1}^{t+i-1} w_l + \gamma^n V(s_{t+n}) \prod_{l=t+1}^{t+n-1} w_l,$$

and the eligibility trace for state s at time t can be re-written as:

$$e_t(s) = \sum_{i=0}^t (\gamma\lambda)^{t-t_m} \prod_{l=i}^{t-1} w_l,$$

where t_m is time step at which we last visited state s , given by

$$t_m = \sup_i \{i < t | s_i = s_t\}.$$

If we have never visited state s at any time step prior to t , then $e_t(s) = 0$. From our unichain assumption, the parameter δ , and the use of replacing traces, we ensure that the eligibility traces remain bounded.

Supposing that at time t , we visit state s , and let τ be the next time at which we visit state s , given by

$$\tau = \inf_i \{i > t | s_i = s\}.$$

We now show that applying the updates of Algorithm 1 for $n \leq \tau$ successive steps, we perform the same update as by using the n -step return $R_t^{(n)}$.

$$\begin{aligned}
\sum_{i=1}^n e_{t+i-1}(s) \Delta_{t+i-1}(s) &= \sum_{i=1}^n \gamma^{i-1} \left(\prod_{l=t}^{t+i-2} w_l \right) \\
&\quad \cdot \left(w_{t+i-1} (r_{t+i} + \gamma V(s_{t+i})) - V(s_{t+i-1}) \right) \\
&= \sum_{i=1}^{\tau} \gamma^{i-1} r_{t+i} \prod_{l=t}^{t+i-1} w_l + \gamma^n V(s_{t+n}) \prod_{l=t}^{t+n-1} w_l - V(s_t) \\
&= R_t^{(n)} - V(s_t).
\end{aligned}$$

The second claim follows from our definition of $\hat{\varepsilon}$ and our choice of δ . ■

B.3 Proof of Theorem 3.3.1

Theorem: *Under standard stochastic approximation conditions, Algorithm 2 converges in the limit, with probability 1, to the same estimates as the on-policy TD-learning algorithm.*

Proof: The bulk of this proof is a straightforward adaptation of the proofs of Theorems 1 and 4 from Precup et al. (2001). First we show that the expected cumulative weighted forward-view updates (Equation 3.7) are equal to the expected cumulative forward-view updates for conventional on-policy TD(λ). We then show that the backward-view updates performed in Algorithm 2 are equivalent to the weighted forward-view updates.

Let $\Delta\theta$ and $\Delta\hat{\theta}$ be the cumulative updates to the parameter vector under on-policy TD(λ) and our weighted TD(λ) respectively. We begin by showing that $\mathbb{E}_p(\Delta\theta|s_0 = s) = \mathbb{E}_q(\Delta\hat{\theta}|s_0 = s)$ for all $s \in \mathcal{S}$. To simplify the notation, we

henceforth take it as implicit that expectations are conditioned on s_0 . Following the same arguments as in the proof of Theorem 1 from Precup et al. (2001), it suffices to show that for any n ,

$$\mathbb{E}_q \left(\sum_{t=0}^{\infty} (\bar{R}_t^{(n)} - \theta^T \phi_t) \phi_t w_0 \cdots w_t \right) = \mathbb{E}_p \left(\sum_{t=0}^{\infty} (R_t^{(n)} - \theta^T \phi_t) \phi_t \right).$$

Given a policy π , the probability of a trajectory $\omega \in \Omega_t(s)$ under transition probability p is given by

$$\begin{aligned} \rho_p(\omega) &= \prod_{i=0}^{t-1} p(s_{i+1}|s_i, a_i) \pi(a_i|s_i) \\ &= \prod_{i=0}^{t-1} (\varepsilon g(s_{i+1}|s_i) I_t + (1 - \varepsilon) f(s_{i+1}|s_i, a_i) (1 - I_t)) \pi(a_i|s_i), \end{aligned}$$

and similarly, the probability of ω under transition probability q is given by

$$\begin{aligned} \rho_q(\omega) &= \prod_{i=0}^{t-1} q(s_{i+1}|s_i, a_i, \hat{\varepsilon}_i) \pi(a_i|s_i) \\ &= \prod_{i=0}^{t-1} (\hat{\varepsilon}_i g(s_{i+1}|s_i) I_i + (1 - \hat{\varepsilon}_i) f(s_{i+1}|s_i, a_i) (1 - I_i)) \pi(a_i|s_i), \end{aligned}$$

where I_t is the indicator function which takes the value 1 if state $s_{t+1} \in \mathcal{T}$ or 0 otherwise.

Our expected n -step weighted return can be rewritten as

$$\begin{aligned}
& \mathbb{E}_q \left(\sum_{t=0}^{\infty} (\bar{R}_t^{(n)} - \theta^T \phi_t) \phi_t w_0 \cdots w_t \right) \\
&= \sum_{t=0}^{\infty} \sum_{\omega \in \Omega_t} \rho_q(\omega) \phi_t \prod_{k=0}^t w_k \mathbb{E}_q \left(\bar{R}_t^{(n)} - \theta^T \phi_t | s_t \right) \\
&\quad \text{(given the Markov property)} \\
&= \sum_{t=0}^{\infty} \sum_{\omega \in \Omega_t} \prod_{j=1}^t (\hat{\varepsilon}_j g(s_{j+1} | s_j) I_j + (1 - \hat{\varepsilon}_j) f(s_{j+1} | s_j, a_j) (1 - I_j)) \pi(a_j | s_j) \phi_t \\
&\quad \cdot \left(\prod_{k=1}^t \frac{\varepsilon}{\hat{\varepsilon}_k} I_k + \frac{1 - \varepsilon}{1 - \hat{\varepsilon}_k} (1 - I_k) \right) \left(\mathbb{E}_q \left(\bar{R}_t^{(n)} | s_t \right) - \theta^T \phi_t \right) \\
&= \sum_{t=0}^{\infty} \sum_{\omega \in \Omega_t} \prod_{j=1}^t (\varepsilon g(s_{j+1} | s_i) I_j + (1 - \varepsilon) f(s_{j+1} | s_j, a_j) (1 - I_j)) \pi(a_j | s_j) \phi_t \\
&\quad \cdot \left(\mathbb{E}_q \left(\bar{R}_t^{(n)} | s_t \right) - \theta^T \phi_t \right) \\
&\quad \text{(using the fact that } I_t^2 = I_t \text{ and } I_t(1 - I_t) = 0) \\
&= \sum_{t=0}^{\infty} \sum_{\omega \in \Omega_t} \rho_p(\omega) \phi_t \left(\mathbb{E}_p \left(R_t^{(n)} | s_t \right) - \theta^T \phi_t \right) \\
&\quad \text{(using the result from Proposition 3.2.2)} \\
&= \mathbb{E}_p \left(\sum_{t=0}^{\infty} (R_t^{(n)} - \theta^T \phi_t) \phi_t \right).
\end{aligned}$$

That the backward-view description given in Algorithm 2 is equivalent to the forward-view definition in Equation 3.7 follows directly from the proof of Theorem 4 in Precup et al. (2001) where we let $g_0 = 1$ and $g_t = 0, \forall t > 0$. ■

Bibliography

- Ahamed, T. P. I., Borkar, V. S., and Juneja, S. (2006). Adaptive importance sampling technique for markov chains using stochastic approximation. *Operations Research*, 54(3):489–504.
- Altman, E., Boulogne, T., El-Azouzi, R., Jiménez, T., and Wynter, L. (2006). A survey on networking games in telecommunications. *Computers and Operations Research*, 33(2):286–311.
- Asmussen, S. and Glynn, P. (2007). *Stochastic Simulation: Algorithms and Analysis*. Springer.
- Au, S. K. and Beck, J. (1999). A new adaptive importance sampling scheme for reliability calculations. *Structural Safety*, 21(2):135–158.
- Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422.
- Baxter, J. and Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15(4):319–350.
- Bellman, R. (1957). Dynamic programming.
- Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.
- Booth, T. E. (1985). Exponential convergence for Monte Carlo particle transport. *Transactions of the American Nuclear Society*, 50:267–268.

- Brafman, R. I. and Tenenbholz, M. (2003). R-max: A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.
- Bratley, P., Fox, B. L., and Schrage, L. E. (1986). *A guide to simulation (2nd ed.)*. Springer-Verlag New York, Inc., New York, NY, USA.
- Bucher, C. G. (1988). Adaptive sampling - an iterative fast Monte Carlo procedure. *Structural Safety*, 5(2):119–126.
- Bucklew, J. (2004). *Introduction to Rare Event Simulation*. Springer.
- Crites, R. H. and Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1017–1023. The MIT Press.
- de Boer, P. T., Kroese, D. P., and Rubinstein, R. Y. (2002). Rare event simulation and combinatorial optimization using cross entropy: estimating buffer overflows in three stages using cross-entropy. In *Proc. of the 2002 Winter Simulation Conf.*, pages 301–309. Winter Simulation Conference.
- de Boer, P. T., Nicola, V. F., and Rubinstein, R. Y. (2000). Adaptive importance sampling simulation of queueing networks. In *Proc. of the 2000 Winter Simulation Conf.*, volume 1.
- Desai, P. Y. and Glynn, P. W. (2001). A Markov chain perspective on adaptive Monte Carlo algorithms. *Simulation Conference Proceedings, 2001. Winter*, 1.
- Devetsikiotis, M. and Townsend, J. K. (1993). An algorithmic approach to the optimization of importance sampling parameters in digital communication

- system simulation. *IEEE Transactions on Communications*, 41(10):1464–1473.
- Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *Proc. of the 15th Intl. Conf. on Machine Learning*, pages 118–126. Morgan Kaufmann.
- Even-Dar, E., Mannor, S., and Mansour, Y. (2006). Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, 7:1079–1105.
- Glasserman, P. and Liu, T.-W. (1996). Rare-event simulation for multistage production-inventory systems. *Management Science*, 42(9):1292–1307.
- Glynn, P. W. and Iglehart, D. L. (1989). Importance sampling for stochastic simulations. *Management Science*, 35(11):1367–1392.
- Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530.
- Heidelberger, P. (1995). Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simulation*, 5(1):43–85.
- Hobbs, C. and Bell, J. (2002). Wisenet: First network planning investigation. Technical report, Nortel Networks.
- Jackson, M. O. (2003). A survey of models of network formation: Stability and efficiency. *Group Formation in Economics: Networks, Clubs and Coalitions*.
- Juneja, S. and Shahabuddin, P. (2006). *Rare-event simulation techniques*, volume 13 of *Handbooks in Operations Research and Management Science*, chapter 11, pages 291–350. Elsevier.

- Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.
- Kearns, M. J. and Singh, S. P. (1998). Near-optimal reinforcement learning in polynomial time. In *Proc. of the 15th Intl. Conf. on Machine Learning*, pages 260–268, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Kollman, C. (1993). *Rare event simulation in radiation transport*. PhD thesis, Stanford University.
- Kollman, C., Baggerly, K., Cox, D., and Picard, R. (1999). Adaptive importance sampling on discrete Markov chains. *Annals of Applied Probability*, 9(2):391–412.
- Koole, G. and Mandelbaum, A. (2002). Queueing models of call centers: An introduction. *Annals of Operations Research*, 113(1):41–59.
- Kushner, H. J. and Yin, G. (2003). *Stochastic Approximation and Recursive Algorithms and Applications*. Springer.
- Madden, M. G. and Howley, T. (2004). Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21(3):375–398.
- Mannor, S., Simester, D., Sun, P., and Tsitsiklis, J. N. (2007). Bias and variance in value function estimation. *Management Science*, 53(2):308–322.
- Maxemchuk, N. F., Ouveysi, I., and Zukerman, M. (2005). A quantitative measure for telecommunications networks topology design. *IEEE/ACM Transactions on Networking*, 13(4):731–742.
- Metropolis, N. and Ulam, S. (1949). The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341.

- Parr, R. and Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems*, 10:1043–1049.
- Precup, D. (2002). Report on the application of reinforcement learning to network planning tasks. Technical report, McGill University.
- Precup, D., Sutton, R. S., and Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. *Proc. of the 18th Intl. Conf. on Machine Learning*, pages 417–424.
- Precup, D., Sutton, R. S., Paduraru, C., Koop, A., and Singh, S. P. (2006). Off-policy learning with recognizers. In *Advances in Neural Information Processing Systems*, volume 18.
- Precup, D., Sutton, R. S., and Singh, S. P. (2000). Eligibility traces for off-policy policy evaluation. In *Proc. of the 17th Intl. Conf. on Machine Learning*, pages 759–766, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Robbins, H. and Munro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.
- Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer.
- Rummery, G. A. and Niranjan, M. (1994). On-line q-learning using connectionist systems. Technical report, Engineering Department, Cambridge University.
- Shahabuddin, P. (1994). Importance sampling for the simulation of highly reliable markovian systems. *Management Science*, 40(3):333–352.

- Shanmugam, K. S. and Balaban, P. (1980). Modified Monte-Carlo simulation technique for the evaluation of error rate in digital communication systems. *IEEE Transactions on Communications*, 28(11):1916–1923.
- Silver, D., Sutton, R. S., and Muller, M. (2007). Reinforcement learning of local shape in the game of Go. *Proc. of the 20th Intl. Conf. on Artificial Intelligence*, pages 1053–1058.
- Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3):323–339.
- Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158.
- Stadler, J. S. and Roy, S. (1993). Adaptive importance sampling. *IEEE Journal on Selected Areas in Communications*, 11(3):309–316.
- Stockheim, T., Schwind, M., and Koenig, W. (2003). A reinforcement learning approach for supply chain management. In *1st European Workshop on Multi-Agent Systems*, St Catherine’s College, Oxford, UK.
- Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188.
- Stone, P. and Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. In Asada, M. and Kitano, H., editors, *RoboCup-98: Robot Soccer World Cup II*, volume 1604 of *Lecture Notes in Artificial Intelligence*, pages 261–72. Springer Verlag, Berlin.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

- Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.
- Sutton, R. S., Precup, D., and Singh, S. P. (1999). Between MDPs and Semi-MDPs: Learning, planning, and representing knowledge at multiple temporal scales. *Artificial Intelligence*, 112:181–211.
- Taylor, M. E. and Stone, P. (2005). Behavior transfer for value-function-based reinforcement learning. In *AAMAS '05: Proc. of the 4th Intl. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, USA. ACM.
- Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.
- Vigeant, E. (2007). Application of reinforcement learning to the network planning task. Unpublished.
- Wasserman, L. (2004). *All of Statistics: A Concise Course in Statistical Inference*. Springer.
- Zhang, W. and Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence*.
- Zhang, W. and Dietterich, T. G. (2000). Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Research*, 1(1):1–38.