*It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.*

– Richard P. Feynman, American theoretical physicist, 1918-1988.

**University of Alberta**

# REINFORCEMENT LEARNING IN ENVIRONMENTS WITH INDEPENDENT DELAYED-SENSE DYNAMICS

by

**Masoud Shahamiri** ©

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

Department of Computing Science

Edmonton, Alberta
Fall 2008

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

*Your file Votre référence*
*ISBN: 978-0-494-47408-2*
*Our file Notre référence*
*ISBN: 978-0-494-47408-2*

*To my beautiful fiancé Neda,*
*for her love and support.*

# Abstract

This thesis is a detailed investigation into applying reinforcement learning to environments with *independent delayed-sense dynamics* (IDSD), where some of state variables evolve independently of both agent's actions and other state variables, and can be sensed only after a delay. These independent state variables are analogous to disturbances, since they are independent of control actions and are not observable before the agent commits a course of action.

In this thesis, we first formalize IDSD problems and then develop four reinforcement learning algorithms that exploit the structure of IDSD problems to achieve better efficiency. Two of the algorithms are partially model-based and two are model-free. We discuss that for the same amount of experiments the quality of the policy learned by the proposed algorithms is better than that of learned by conventional reinforcement learning algorithms.

We demonstrate the effectiveness of our algorithms by applying them to traffic grid-world problems and to a hybrid vehicle problem, in which the traffic and driver acceleration play the role of the independent state variable respectively. We show experimentally that our algorithms evaluate a given policy more accurately than the corresponding TD(0). We also show that in the case of control, the learning speeds of our algorithms are substantially higher than the learning speed of conventional reinforcement learning algorithms that do not use the knowledge of the IDSD structure.

# Acknowledgements

I would like to thank my supervisor Dr. Richard Sutton, for guiding me through each step of my research and also for spending his precious time. He showed me different ways to approach a research problem and the need to be precise to accomplish any goal. He taught me how to ask questions and express my ideas. Thank you Rich. I would also to thank Martin Jägersand my former supervisor and my current advisor for his fabulous support. Martin was always there to listen and to give me great advice. Without his guidance and support, I could not have made it this far. Thank you Martin.

I want to recognize the insightful comments of Brian Tanner, Alborz Geramifard and Adam White on my thesis, and thank them for helping me through my research. I want to also thank Anna Koop who volunteered for proofreading my thesis with so much enthusiasm. Finally, special thanks to all other members of the RLAI lab at the University of Alberta for their support.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

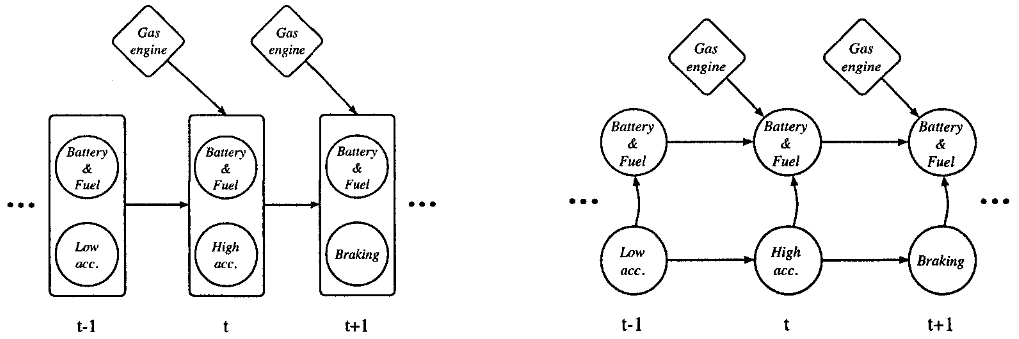
# Introduction

Markov decision processes (MDPs) are a popular framework for modeling decision-making problems in the presence of uncertainty. MDPs characterize a control process by a set of states and a set of actions for each state which from an agent must choose. Reinforcement learning is one of the well understood approaches for control in MDPs. In this approach the agent learns which action to perform in each state through trial-and-error interaction with the system to be controlled. The learning is based on the state transition and a numerical reward signal provided by the system.

In many control applications, the system to be controlled includes a component whose state transition cannot be controlled by agent's actions. Considering this independent component as a part of state may cause problem because agent's actions may be penalized or rewarded for the transition of the uncontrolled component. On the other hand, removing the independent component from the state may cause the agent to make poor decisions because the agent loses an important knowledge of the system.

For instance, consider a hybrid vehicle problem, in which the vehicle runs either on a conventional gas engine or an electric motor powered by a rechargeable battery. The agent's goal is to minimize the fuel consumption of the vehicle by switching between these two power sources. Assume each power sources can respond to any acceleration request, and the driver requests for acceleration autonomously. Therefore, the acceleration request is not under control of the agent.

Conventional reinforcement learning algorithms consider the acceleration request as a part of the state which result in penalizing or rewarding the agent's actions depending on



(a) Conventional representation of state transition. Acceleration request, gas-level and battery-level are parts of one state.

(b) New representation of state transition. The acceleration request is represented by a separate state which is independent of the agent's action.

Figure 1.1: Transitions of state variables for the hybrid car problem. The state variables and actions are shown in the circles and diamond shapes respectively.

the transition of the acceleration request. This variance in assigning credit to the agent's actions can increases the time to learn an effective policy. In contrast, the agent can exploit the knowledge of independency between the acceleration request and the agent's actions to factor out the credit caused by the transition of the acceleration request.

Figure 1.1(a) shows a situation that causes high variance in the credit assignment of conventional reinforcement learning algorithms. The agent's action is penalized for the transition of the acceleration request from low to high because it will increase gas consumption, and rewarded for the transition of the acceleration request from high to braking because it will charge the battery. Figure 1.1(b) shows a new representation of the same situation. The lack of the arc from the gas engine to the acceleration request represents the independency between the acceleration request and the agent's action. The arc from acceleration request to power sources within one time-step represent the influence of the acceleration request on the fuel consumption or battery depletion. This representation enables the agent to separate the transition of the acceleration request from the transition of battery and gas, and factor out the credit caused by the transition of the acceleration request.

The independencies between state variables have received attention in factored MDPs (Boutilier et al., 1999). Nevertheless, the problems with the structure described in the above example, called *independent delayed-sense dynamics* (IDSD) problems, are not explored in the artificial intelligence literature. Furthermore, factored MDPs are introduced to represent large problems compactly using their structures and do not address, in general, how these structures can be exploited to achieve better efficiency (see chapter 2 for more details).

The particular focus of this thesis is in finding specialized reinforcement learning algorithms that exploit the structured of IDSD problems to reduce the variance of the conventional credit assignment, and thus reduce the time to learn an effective policy.

## 1.1 Contributions

The contributions of this thesis comprise four parts:

- Introducing a reinforcement learning framework to model the agent-environment interaction in IDSD problems, and deriving the standard definitions of reinforcement learning under that framework.
- Proposing two partially model-based algorithms and two model-free algorithms for prediction and control in IDSD problems.
- Deriving asymptotic convergence bounds of the proposed algorithms in the case of control, and discussing the relationship between these bounds and the asymptotic bound of Sarsa(0).
- Presenting three new IDSD problems for comparing the proposed algorithms to conventional reinforcement learning algorithms. These problems can be used as a benchmark for evaluating new algorithms under the IDSD structure.

## 1.2 Overview

Chapter 2 covers the background material necessary to understand the contributions made to IDSD problems. The chapter contains a formal definition of Markov decision processes, reinforcement learning, and its preliminaries. It also reviews some of the work that is most directly related to IDSD problems.

Chapter 3 introduces a reinforcement learning framework for modeling IDSD problems. Based on this framework, we propose four algorithms that exploit the structure of IDSD problems in order to speed up the learning procedure. The first two algorithms rely on having a model of the independent component, and thus we refer to them as partially model-based algorithms. The other two algorithms are model-free algorithms.

Chapter 4 analyzes the time complexity of the proposed algorithms. In the case of control, the asymptotic convergence bounds of the proposed algorithms will be presented. Finally, the chapter discusses the similarity between the bounds of the proposed algorithms and Sarsa(0), and how these similarities imply the advantages of the proposed algorithms.

Chapter 5 presents the experimental results. We first introduce two episodic and one infinite horizon IDSD problems. We then use these problems to compare the proposed algorithms to conventional reinforcement learning algorithms in the cases of prediction and control.

Chapter 6 discusses the limitations of the partially model-based algorithms presented in Chapter 3. In particular, we investigate one possible adjustment to the algorithms. We experimentally show that this adjustment is able to resolve the limitations of the algorithms.

In the final chapter, we summarize the contributions presented in this work. Finally, we conclude with some discussion of the remaining challenges, and speculate on future trends in IDSD problems.

# Chapter 2

# Background

In this chapter, we present the background work in reinforcement learning (RL), although for further detailed information, we direct the reader to (Sutton & Barto, 1998) and (Baird, 1993). We start with presenting Markov decision processes (MDPs) followed by reinforcement learning in MDPs. We briefly review the temporal difference learning algorithm, the well-known RL algorithm for evaluating states under a fixed policy. Then, we present two RL algorithms that have been used widely for control tasks in MDPs: Sarsa(0), and the advantage updating algorithm. Afterward, we present some of the work that is most directly related to IDSD problems. In particular, we present factored MDPs, in which MDPs are represented compactly by exploiting the structure of problems. We discuss how this representation can help address the planning problem, and how it can be related to IDSD problems.

## 2.1 Markov Decision Processes

Markov decision processes have been considered extensively as a basic framework for the problem of planning under uncertainty. An MDP represents a problem as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'} \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}^a_{ss'}$ is the probability of reaching state $s'$ after taking action $a$ in state $s$, and $\mathcal{R}^a_{ss'}$ is the received reward. If the set of states and actions are finite, then the problem is called a finite MDP. Furthermore, the problem is called *episodic* if it has natural endpoints, and *infinite horizon* if it continues forever.

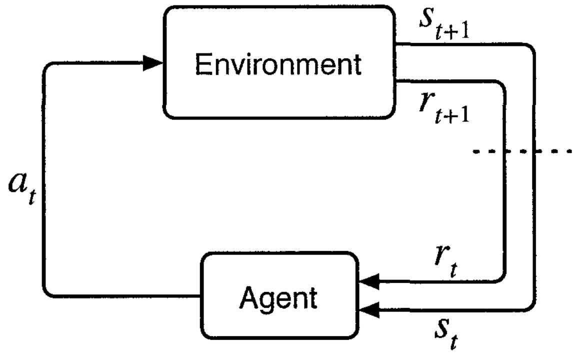


Figure 2.1: The agent-environment interaction in reinforcement learning.

## 2.2 Reinforcement Learning

Reinforcement learning is the task of learning which action to perform in each state in order to maximize a numerical reward signal (Sutton & Barto, 1998). The learner or decision-maker is called the agent, and the system that generates the states and rewards is called the environment. The agent solves the learning task through trial-and-error interaction with the environment (Moriarty et al., 1999).

Figure 2.1 shows the agent-environment interaction in a reinforcement learning problem. At any time $t$, the agent takes an action $a_t \in \mathcal{A}$ based on the current state $s_t$. One time-step later, the environment responds by changing the state into $s_{t+1}$, and giving the agent a numerical reward, $r_{t+1} \in \mathcal{R}$, as a consequence of its action. In this work, we assume that problems satisfy Markov property; *i.e.,* $s_{t+1}$ is only a function of $s_t$ and $a_t$.

A policy is a mapping $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0\ 1]$, where $\pi(s, a)$ denotes the probability of performing action $a$ whenever the agent is in state $s$. The agent goal is to adopt an optimal policy, $\pi^*$, that maximizes the *expected return.* The return, $R_t$, is the total discounted reward.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

where $\gamma \in [0, 1]$ is a discount factor. The value of state $s$ under policy $\pi$, denoted by $V^\pi(s)$, is the return that agent will receive by starting from state $s$ and following policy $\pi$ thereafter.

$$V^{\pi}(s) = E_{\pi}\left[R_t \middle| s_t = s\right] \tag{2.2}$$

Similarly, the value of taking action $a$ in state $s$ under policy $\pi$, called the *action-value function*, is defined by:

$$Q^{\pi}(s,a) = E_{\pi}\left[R_t \middle| \begin{array}{c} s_t = s \\ a_t = a \end{array}\right] \tag{2.3}$$

Considering equation (2.2) and (2.3), one can derive the state-value function from the action-value function by:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) Q^{\pi}(s,a) \tag{2.4}$$

Note that in the case of optimal policy, $\pi^*(s,a)$ is zero for all actions except the optimal action. Therefore:

$$V^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a) \tag{2.5}$$

The *advantage* of a state-action pair $(s,a)$ under policy $\pi$, denoted by $A^{\pi}(s,a)$, represents the degree to which the return is increased by performing that action rather than the current best action.

$$\begin{aligned} A^{\pi}(s,a) &= E_{\pi}\left[R_t \middle| \begin{array}{c} s_t = s \\ a_t = a \end{array}\right] - E_{\pi}\left[R_t \middle| \ s_t = s \ \right] \\ &= Q^{\pi}(s,a) - V^{\pi}(s) \end{aligned} \tag{2.6}$$

Equations (2.5) and (2.6) imply that the advantage of the optimal policy is zero if $a$ is the optimal action, and is negative for any suboptimal action.

Given these definitions, there are two families of algorithms for learning the optimal policy: (1) policy iteration and (2) value iteration. Policy iteration algorithms consist of two parts. In the first part, called prediction, the value function of a fixed policy is estimated. Then in the second part, called control, the estimate of the value function is used to improve the policy by finding for each state some action that is better than the current best action. The algorithms alternate between these two parts until the optimal policy is found; *i.e.,* the policy becomes stable (LaValle, 2006).

Value iteration algorithms combine prediction and control steps in a simple backup operation. These algorithms use the estimate of $Q$ or $A$ directly, and select at each time-step the action that has the highest value (Schuurmans & Patrascu, 2001).

### 2.2.1 Temporal Difference Learning

Sutton (Sutton, 1988) introduced the temporal difference learning algorithm (TD), the most commonly used algorithm for prediction in MDPs. The algorithm is based on the error signal defined at every time-step by the temporal difference between the estimate of the value of two consecutive states. If $V(s_t)$ and $V(s_{t+1})$ denote the estimate values of two consecutive states in iteration $t$, and $r_{t+1}$ denote the observed reward at that iteration, then the simplest version of TD, known as TD(0), updates $V(s_t)$ by:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big] \tag{2.7}$$

where $\alpha$ is a constant step-size parameter.

### 2.2.2 Sarsa(0)

The Sarsa(0) algorithm is an on-policy TD(0) that uses the estimate of the action-value function for control. This algorithm learns the action-value function by considering transitions from state-action pair to state-action pair (Sutton & Barto, 1998). Despite of its simplicity, Sarsa(0) is one the most effective algorithm for control in MDPs.

Formally, after seeing trajectory fragment $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$, Sarsa(0) updates the estimate of the action-value function, $Q(s_t, a_t)$, by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big] \tag{2.8}$$

Given any state, the action can be selected by a greedy policy or by a $\epsilon$-greedy policy; *i.e.,* the action with the highest $Q$ value is selected most of the time, but once in a while, with the probability of $\epsilon$, an action is selected at random uniformly.

### 2.2.3 Advantage Updating Algorithm

The advantage updating algorithm is a control algorithm that learns the estimate of the advantage function (Baird, 1993). This algorithm stores two types of information: (1) the estimate of the state-value function, and (2) the estimate of the advantage function. Both functions are required for the learning procedure, but the policy is extracted directly from the estimate of the advantage function.

At each iteration $t$, the algorithm uses equation (2.7) to update the estimate of the value function. Afterward, it updates the estimate of the advantage function, $A(s_t, a_t)$, by:

$$\delta_t = \max_{a'} A(s_t, a') + [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] - A(s_t, a_t) \tag{2.9}$$

$$A(s_t, a_t) \leftarrow A(s_t, a_t) + \beta\delta_t \tag{2.10}$$

where $\beta$ is a step-size parameter. Subsequently, the algorithm normalizes the estimate of the advantage function to ensure that after convergence $\max_{a'} A(s_t, a') = 0$ for all states. The normalization is done by picking a state-action pair $(s, a)$ at uniform random, and updating $A(s, a)$ by:

$$A(s, a) \leftarrow A(s, a) - \omega \max_{a'} A(s, a') \tag{2.11}$$

where $\omega$ is a step-size parameter. Note that the normalization does not require interaction with the environment. Therefore, the normalizing update can be performed multiple times per time-step.

## 2.3 Dynamic Bayesian Network

Dynamic Bayesian networks (DBN), introduced by Dean and Kanazawa (Dean & Kanazawa, 1989), are used to reflect implicit independencies among the state variables of MDPs. A DBN is a two-layer directed acyclic graph, in which vertices correspond to state variables, and an edge between two vertices indicates a direct probabilistic dependency between them. The edges are divided into two categories: (1) diachronic arcs, and (2) synchronic arcs. Diachronic arcs are those directed from state variables at time $t$ to state variables at time $t+1$; synchronic arcs are directed between state variables within a time-step (Boutilier, Dean & Hanks, 1999). Figure 2.2 shows a DBN that contains only diachronic arcs.

The lack of a diachronic arc from a state variable at the current time, $x_i$, to another state variable at the next time-step, $x'_j$, means that the knowledge of $x_i$ is irrelevant to the prediction of $x'_j$. These independencies have a strong effect on the number of parameters that must be supplied to compute the distribution over resulting states.

A DBN is quantified by specifying a probability for each state variable conditioned on all possible values of its immediate parents (Boutilier, Dean & Hanks, 1999). The network also must have a marginal distribution; *i.e.,* an unconditional probability for each state variable that has no parents. These quantifications are captured by associating a conditional probability table (CPT) with each state variable in the network (Boutilier, Dean & Hanks, 1999).

## 2.4 Related Work

Using the problem structure to speed up the learning process has a long history in the prediction and control problems. Boutilier, Dearden and Goldszmidt introduced factored MDPs in order to reduce the problem complexities that grows exponentially with the number of state variables (Boutilier, Dearden & Goldszmidt, 1995). Factored MDPs rely on the fact that many large MDPs have significant internal structure, and can be modeled compactly if that structure is exploited by the representation (Koller & Parr, 1999). For instance, consider a simple robotic task, in which a robot navigates in a building and delivers coffee and mail. In this example, the location of the robot at time $t+1$ may depend on its position, velocity,

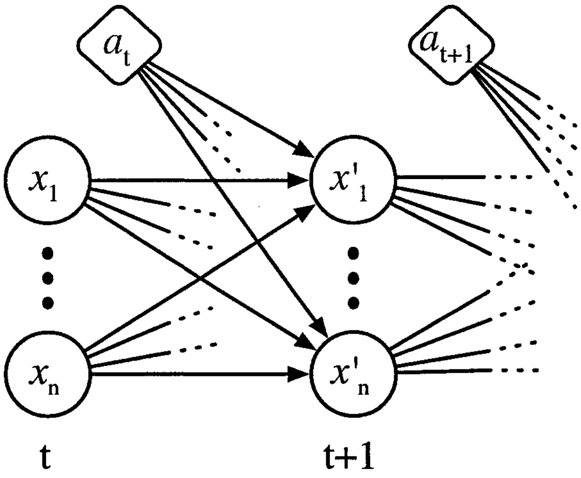


Figure 2.2: A dynamic Baysian network. The nodes represent the state variables at time $t$ and $t + 1$. The edges represent the conditional dependencies between state variables from time $t$ to time $t + 1$.

and acceleration at time $t$, but not on what it is carrying.

In factored MDPs, a state is described by a set of state variables, where each state variable is in some finite domain. Then a set of DBNs, one per action, is used to represent a compact transition model by exploiting the fact that under one particular action the transition of a state variable depends only on a small number of other variables (Note that in some factored MDPs the DBNs of actions are the same). Unfortunately, this representation of MDP does not help the decision-making problem, since certain regularities and structures in a MDP do not guarantee, in general, any type of structure in the value function (Koller & Parr, 1999). Nevertheless, it has been shown that in many factored MDPs, the optimal policy also has a certain structure (as does the value function).

Boutilier, Dearden and Goldszmidt introduced the structured policy iteration (SPI) algorithm for constructing optimal policies in factored MDPs (Boutilier, Dearden & Goldszmidt, 1995). The algorithm is based on the intuition that if a problem can be represented by exploiting certain regularities and structures, one can expect that the optimal policy and the value function also have a certain structure. The algorithm constructs tree-structured representations of the value function and policies. The nodes of each tree are labeled with state variables, and the edges are labeled with the values of state variables. The leaves of the value tree denote the value of any state consistent with the labeling of the corresponding branch. In the case of policy improvement, the leaves indicate the action to be performed at any state that is consistent with the labeling of the corresponding branch. Given these structured trees, SPI performs the policy evaluation by exploiting the fact that if the effects of an action on two states are only different in the state variables that are not relevant to the tree-structured representation of the value function, then the value of those states must be identical.

The key operation of the algorithm assumes that the effects of actions on the value of successive state variables are independent (Boutilier, Dearden & Goldszmidt, 2000), which does not hold in IDSD and many other problems. Nevertheless, (Boutilier, Dearden & Goldszmidt, 1995) proposed that such dependencies in decision trees can be replaced by modified subtrees containing only the parents of the successive state variables. However, this is a complex operation that makes the algorithm feasible only on MDPs with localized dependencies between the values of successive state variables within the same time-step (Boutilier, 1997; Teichteil-Knigsbuch & Fabiani, 2004).

Kearns and Koller proposed a model-based RL algorithm for factored MDPs (Kearns & Koller, 1999). The algorithm learns an approximate model from experience, and then exploits (or explores) it by $T$-step planning given the approximate model. The algorithm estimates the model by learning the parameters of a CPT for each action. It then uses a recursive sparse sampling method to compute near-optimal actions from any state. They proved that "a randomly sampled look-ahead tree that covers only a vanishing fraction of the full look-ahead tree is sufficient for calculating near-optimal policy" (Kearns, Mansour & Ng, 1999).

Koller and Parr proposed that the value function of a policy in factored MDPs can be approximated closely by a linear combination of local basis functions, each of which depends only on a restricted set of state variables (Koller & Parr, 1999). They argued that these additive value functions can be estimated by iterating through a dynamic programming, and finding the closest value function in the restricted set. The algorithm finds the closest value function based on the probability of visiting different states in the stationary distribution. This method is restricted to policy evaluation, and cannot guarantee overall policy improvement, since the states that are visited infrequently may have very poor value estimates. Later, Koller and Parr removed the correlation of the distance metric on the stationary distribution and used the algorithm as a subroutine in a policy iteration process (Koller & Parr, 2000). They showed that the one-step greedy policy can be represented compactly by a decision list, where each element of the list is the difference between the value of a state-action pair, and the value of a default action in that particular state.

(Schuurmans & Patrascu, 2001) and (Poupart, Boutilier, Patrascu & Schuurmans, 2002) proposed a direct linear programming approach to determining optimal policies in factored MDPs. This approach is also based on the assumption that the optimal policy can be approximated by a compact linear form of some basis functions. The approach exploits the structure of the factored MDP to find the best linear fit to the optimal value function. The issues of selecting basis functions and accuracy were addressed in (Poupart, Boutilier, Patrascu & Schuurmans, 2002).

Although these algorithms are focusing on independencies between state variables given a particular action, but one can use a similar idea for IDSD problems. In chapter 3, we show that the value function of a policy in IDSD problems has a specific structure. In particular, we introduce partially model-based algorithms, in which the state-value or the action-value function is calculated by a linear combination of some values, called advanced values.

# Chapter 3

# Algorithms for IDSD Problems

In this chapter, we first formally define IDSD problems. Then we introduce a reinforcement leaning framework for modeling IDSD problems. The preliminaries of an IDSD problem will be presented. Afterward, we propose two partially model-based and two model-free algorithms for prediction and control in IDSD problems.

## 3.1 IDSD Problems

In an IDSD problem, the state variables are divided into two sets: the *autonomous state* and the *controllable state*. The transition of the autonomous state is independent of the controllable state and agent's actions, and can be sensed after a delay. However, the transition of the controllable state is influenced by agent's actions and by the value of the autonomous state.

## 3.2 IDSD Framework

We model an IDSD problem by using two separate processes. We denote the controllable state by vector $X$, and model it as signals coming from *controllable dynamics*. Similarly, we model the autonomous state, denoted by vector $Y$, as signals produced by *autonomous dynamics*. Further, we assume that both processes have the Markov property, and both signals are in a discrete, finite set; $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$.

Figure 3.1 shows a typical agent-environment interaction in an IDSD problem. At any time $t$, the agent takes an action $a_t \in \mathcal{A}$ based on the current representation of state $s_t = (X_t, Y_t)$. One time-step later, the autonomous dynamics produce $Y_{t+1}$ based on $Y_t$.

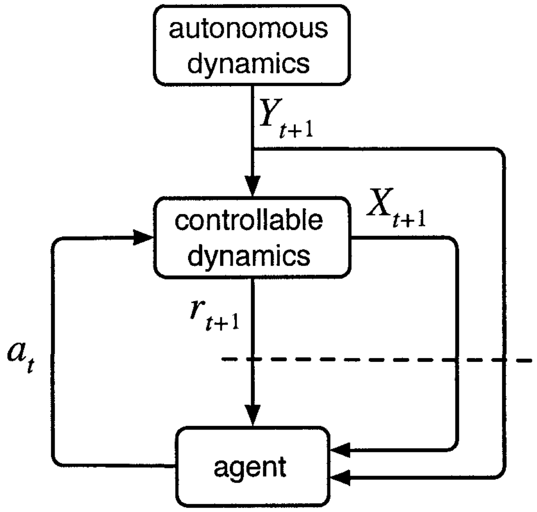


Figure 3.1: The agent-environment interface in an IDSD problem.

Then the controllable dynamics produce $X_{t+1}$ based on tuple $\langle X_t, Y_{t+1}, a_t\rangle$. Afterward, the agent receives a new state $s_{t+1} = (X_{t+1}, Y_{t+1})$, and a numerical reward, $r_{t+1} \in \mathcal{R}$, as a consequence of its action. $r_{t+1}$ is a function of $\langle X_t, X_{t+1}, Y_{t+1}, a_t\rangle$.

We define the value of a state $(x, y)$ under policy $\pi$ as the return that the agent will receive by starting from that state and following policy $\pi$ thereafter:

$$V^{\pi}(x,y) = E_{\pi}\left[R_t \middle| \begin{array}{c} X_t = x \\ Y_t = y \end{array}\right] \tag{3.1}$$

Note that at any time-step, the return depends on the current value of controllable state and the next value of the autonomous state, called the *advanced state*. Thus, we define a new value function, called *advanced value function* as:

$$\overline{V}^{\pi}(x,y) = E_{\pi}\left[R_t \middle| \begin{array}{c} X_t = x \\ Y_{t+1} = y \end{array}\right] \tag{3.2}$$

The value of a state-action pair $(x, y, a)$ under policy $\pi$, is the return that the agent will receive for taking action $a$ in state $(x, y)$.

$$Q^\pi(x,y,a) = E_\pi \left[ R_t \middle| \begin{array}{l} X_t = x \\ Y_t = y \\ a_t = a \end{array} \right] \tag{3.3}$$

Similarly, the *advanced action-value function* of a policy $\pi$, denoted by $\overline{Q}^\pi$, is the return that the agent will receive for taking action $a$, given the current value of the controllable state and the next value of the autonomous state.

$$\overline{Q}^\pi(x,y,a) = E_\pi \left[ R_t \middle| \begin{array}{l} X_t = x \\ Y_{t+1} = y \\ a_t = a \end{array} \right] \tag{3.4}$$

It is easy to show that if the dynamics of the autonomous state is known, then $V^\pi$ and $Q^\pi$ can be derived from $\overline{V}^\pi$ and $\overline{Q}^\pi$ respectively. In order to show these relationships, consider the following lemma:

**Lemma 3.1.** *Let $A \in \mathcal{U}$ and $B \in \mathcal{V}$ be discrete random variables. Then*

$$E[A] = \sum_{r \in \mathcal{V}} Pr(B = r) E[A \mid B = r]$$

*where $Pr(B = r)$ is the probability of $B$ equals to $r$.*

Using the Markov property of the autonomous process and lemma 3.1, we have:

$$\begin{aligned} V^\pi(x,y) &= E_\pi \left[ R_t \middle| \begin{array}{l} X_t = x \\ Y_t = y \end{array} \right] \\ &= \sum_{y' \in \mathcal{Y}} Pr(Y_{t+1} = y') E_\pi \left[ R_t \middle| \begin{array}{l} X_t = x \\ Y_t = y \\ Y_{t+1} = y' \end{array} \right] \\ &= \sum_{y' \in \mathcal{Y}} Pr(Y_{t+1} = y' \mid Y_t = y) E_\pi \left[ R_t \middle| \begin{array}{l} X_t = x \\ Y_{t+1} = y' \end{array} \right] \\ &= \sum_{y' \in \mathcal{Y}} Pr(Y_{t+1} = y' \mid Y_t = y) \overline{V}^\pi(x, y') \end{aligned} \tag{3.5}$$

Similarly, one can derive $Q^\pi$ from $\overline{Q}^\pi$ by:

$$
\begin{aligned}
Q^{\pi}(x,y,a) &= E_{\pi}\left[R_t \middle| \begin{array}{l} X_t = x \\ Y_t = y \\ a_t = a \end{array}\right] \\
&= \sum_{y' \in \mathcal{Y}} Pr(Y_{t+1} = y') E_{\pi}\left[R_t \middle| \begin{array}{l} X_t = x \\ Y_t = y \\ Y_{t+1} = y' \\ a_t = a \end{array}\right] \\
&= \sum_{y' \in \mathcal{Y}} Pr(Y_{t+1} = y' \mid Y_t = y) E_{\pi}\left[R_t \middle| \begin{array}{l} X_t = x \\ Y_{t+1} = y' \\ a_t = a \end{array}\right] \\
&= \sum_{y' \in \mathcal{Y}} Pr(Y_{t+1} = y' \mid Y_t = y) \overline{Q}^{\pi}(x, y', a) \qquad (3.6)
\end{aligned}
$$

## 3.3 Partially Model-Based Algorithms

In this section, we propose two partially model-based algorithms that learn the model of the autonomous dynamics and use that model for prediction or control in IDSD problems. The first algorithm, called the *autonomous-based prediction algorithm* (AB-P), is a prediction algorithm that uses the model of the autonomous dynamics to derive the value function from the advanced value function. The second algorithm, called *autonomous-based control algorithm* (AB-C), is a control algorithm that exploits the model of the autonomous dynamics to derive the action-value function from the advanced action-value function. The following sections describe the algorithms in more detail.

### 3.3.1 Autonomous-Based Prediction Algorithm

Autonomous-Based Prediction algorithm (AB-P) learns the value function and the advanced value function simultaneously, one from the other. Given a trajectory fragment $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$, the algorithm uses the following equation to update the estimate of the advanced value function, denoted by $\overline{V}(X_t, Y_{t+1})$:

$$
\overline{V}(X_t, Y_{t+1}) \leftarrow \overline{V}(X_t, Y_{t+1}) + \alpha \left[ r_{t+1} + \gamma V(X_{t+1}, Y_{t+1}) - \overline{V}(X_t, Y_{t+1}) \right] \qquad (3.7)
$$

Afterward, the algorithm computes the estimate of the value of state $(X_t, Y_t)$ by:

$$V(X_t, Y_t) = \sum_{y' \in \mathcal{Y}} P(y', Y_t)\overline{V}(X_t, y') \tag{3.8}$$

where $P(y', Y_t)$ is the estimate of $Pr(Y_{t+1} = y' \mid Y_t)$. Note that $Pr(Y_{t+1} = y' \mid Y_t)$ is independent of the agent's policy, and thus can be estimated by any time-series prediction methods such as the Kalman filter or the maximum likelihood methods. In this work, we consider the simple form of the maximum likelihood method for the sake of simplicity.

At each time-step $t$, AB-P updates $P(y', Y_t)$ by:

$$P(y', Y_t) = \frac{n_t(y', Y_t)}{n_t(Y_t)} \tag{3.9}$$

where $n_t(y', Y_t)$ is the number of times that $y'$ has been seen immediately after seeing $Y_t$, and $n_t(Y_t)$ is the number of times that $Y_t$ has been visited. Algorithm 1 shows the general form of AB-P.

### 3.3.2 Autonomous-Based Control Algorithm

Autonomous-Based Control Algorithm (AB-C) stores the estimate of the action-value function and the advanced action-value function. Both estimates are required for learning, but the policy is derived from the estimate of the action-value function.

At each time-step $t$, AB-C updates the estimate of the advanced action-value function, $\overline{Q}(X_t, Y_{t+1}, a_t)$, by:

$$\begin{aligned}\overline{Q}(X_t, Y_{t+1}, a_t) \leftarrow{}& \overline{Q}(X_t, Y_{t+1}, a_t) + \\ & \alpha\Big[r_{t+1} + Q(X_{t+1}, Y_{t+1}, a_{t+1}) - \overline{Q}(X_t, Y_{t+1}, a_t)\Big]\end{aligned} \tag{3.10}$$

Afterward, the algorithm computes the estimate of the action-value function, denoted by $Q(X_t, Y_t, a_t)$, based on equation (3.6):

$$Q(X_t, Y_t, a_t) = \sum_{y' \in \mathcal{Y}} P(y', Y_t)\overline{Q}(X_t, y', a_t) \tag{3.11}$$

**Algorithm 1** Autonomous-Based Prediction Algorithm

```
Initialize $V(x, y)$ and $\overline{V}(x, y)$ arbitrarily
Initialize $\pi$ to the policy to be evaluated
Initialize $m(y, y)$ and $n(y)$ to zero
Repeat (for each episode):
    Initialize $(x, y)$
    Repeat (for each step of episode):
        $a \leftarrow$ action given by $\pi$ for $(x, y)$
        Take action a; observe reward $r$, and next state $(x', y')$
        $\overline{V}(x, y') \leftarrow \overline{V}(x, y') + \alpha \left[r + \gamma V(x', y') - \overline{V}(x, y')\right]$
        $n(y) \leftarrow n(y) + 1$
        $m(y', y) \leftarrow m(y', y) + 1$
        $V(x, y) \leftarrow \sum_{z \in \mathcal{Y}} \frac{m(z,y)}{n(y)} \overline{V}(x, z)$
        $(x, y) \leftarrow (x', y')$
    until $(x, y)$ is terminal
Return $V$ as the estimate of the value function
```

where $P(y', Y_t)$ is calculated by the method described in the previous section. Algorithm 2 shows the general form of AB-C.

## 3.4 Model-Free Algorithms

In this section, we propose model-free algorithms for prediction and control in IDSD problems. The algorithms estimate the advantage function and use that estimate to evaluate or improve a policy. The algorithms are based on the intuition that, in assigning a credit to an action, the agent should consider only the state variables that either are the outcomes of that action or influenced the outcomes of that action. As indicated in figure 3.2, in IDSD problems, $Y_t$ evolves independently of the agent's action and does not influence $X_{t+1}$. Therefore, $Y_t$ should be excluded from assigning a credit to the agent's action as shown in figure 3.2.

**Algorithm 2** Autonomous-Based Control Algorithm

```
Initialize $Q(x,y,a)$ and $\overline{Q}(x,y,a)$ arbitrarily
Initialize $m(y,y)$ and $n(y)$ to zero
Repeat (for each episode):
    Initialize $(x,y)$
    Choose $a$ for $(x,y)$ using the policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action a; observe reward $r$, and next state $(x',y')$
        Choose $a'$ for $(x',y')$ using the policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $\overline{Q}(x,y',a) \leftarrow \overline{Q}(x,y',a) + \alpha\left[r + \gamma Q(x',y',a') - \overline{Q}(x,y',a)\right]$
        $m(y',y) \leftarrow m(y',y) + 1$
        $n(y) \leftarrow n(y) + 1$
        $Q(x,y,a) \leftarrow \sum_{z\in\mathcal{Y}} \frac{m(z,y)}{n(y)}\overline{Q}(x,z,a)$
        $(x,y) \leftarrow (x',y')$; $a \leftarrow a'$
    until $(x,y)$ is terminal
```

### 3.4.1 Model-Free Prediction Algorithm

Model-free prediction algorithm (MF-P) uses the estimate of a new function, called the *mean advanced advantage function*, to exclude the current value of the autonomous state from the credit assignment. Given policy $\pi$, MF-P uses two types of information to approximate the mean advanced advantage function of the policy: (1) the estimate of the value function, and (2) the estimate of the advanced value function.

At each iteration $t$, MF-P updates the estimate of the value function, $V(X_t, Y_t)$, by the simplest TD error, as shown in equation (2.7). Then, the algorithm updates the estimate of the advanced value function by:

$$\overline{V}(X_t, Y_{t+1}) \leftarrow \overline{V}(X_t, Y_{t+1}) + \alpha\Big[r_{t+1} + \gamma V(X_{t+1}, Y_{t+1}) - \overline{V}(X_t, Y_{t+1})\Big] \qquad (3.12)$$

Afterward, the algorithm uses the following equation to update the estimate of the mean

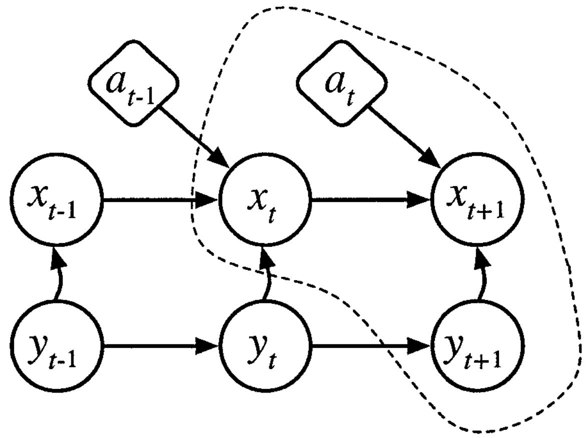


Figure 3.2: The dynamic Bayesian network of IDSD problems. The dotted line shows the state variables that participate in assigning credit to the agent action in the model-free algorithm.

advanced advantage function, denoted by $\widetilde{A}(X_t, Y_t, a_t)$:

$$\begin{aligned} &\widetilde{A}(X_t, Y_t, a_t) \leftarrow \widetilde{A}(X_t, Y_t, a_t) + \\ &\quad \beta\Big[r_{t+1} + \gamma V(X_{t+1}, Y_{t+1}) - \overline{V}(X_t, Y_{t+1}) - \widetilde{A}(X_t, Y_t, a_t)\Big] \end{aligned} \tag{3.13}$$

Subsequently, whenever it is needed, MF-P extracts the estimate of the advantage function, $A(X_t, Y_t, a_t)$, by:

$$A(X_t, Y_t, a_t) = \widetilde{A}(X_t, Y_t, a_t) - \sum_{a \in \mathcal{A}} \pi(X_t, Y_t, a) \widetilde{A}(X_t, Y_t, a) \tag{3.14}$$

In the following, we show mathematically that the mean advanced advantage function excludes the current value of the autonomous dynamics from the credit assignment. We also prove that one can derive the advantage function from the mean advanced advantage function by using equation (3.14).

First, we define a new advantage function, called the *advanced advantage function*, that does not depend on the current value of the autonomous dynamics. The advanced advantage of action $a$ in state $(x, y)$ is the advantage of performing that action rather than the current best action, assuming the agent knows the next value of the autonomous state. Formally, we define the advanced advantage function by:

$$\begin{aligned} \overline{A}^{\pi}(x,y,a,y') &= E_{\pi}\left[R_t \middle| \begin{array}{l} X_t = x \\ Y_t = y \\ a_t = a \end{array}\right] - E_{\pi}\left[R_t \middle| \begin{array}{l} X_t = x \\ Y_{t+1} = y' \end{array}\right] \\ &= Q^{\pi}(x,y,a) - \overline{V}^{\pi}(x,y') \end{aligned} \tag{3.15}$$

The following equation shows that the advantage function of policy $\pi$ can be derived from its advanced advantage function:

$$\begin{aligned} A^{\pi}(x,y,a) &= Q^{\pi}(x,y,a) - V^{\pi}(x,y) \\ &= \left(Q^{\pi}(x,y,a) - \overline{V}^{\pi}(x,y')\right) - \left(V^{\pi}(x,y) - \overline{V}^{\pi}(x,y')\right) \\ &= \overline{A}^{\pi}(x,y,a,y') - \left(\sum_{a'\in\mathcal{A}} \pi(x,y,a')Q^{\pi}(x,y,a') - \overline{V}^{\pi}(x,y')\right) \\ &= \overline{A}^{\pi}(x,y,a,y') - \left(\sum_{a'\in\mathcal{A}} \pi(x,y,a')Q^{\pi}(x,y,a') - \overline{V}^{\pi}(x,y') \sum_{a'\in\mathcal{A}} \pi(x,y,a')\right) \\ &= \overline{A}^{\pi}(x,y,a,y') - \sum_{a'\in\mathcal{A}} \pi(x,y,a')\left(Q^{\pi}(x,y,a') - \overline{V}^{\pi}(x,y')\right) \\ &= \overline{A}^{\pi}(x,y,a,y') - \sum_{a'\in\mathcal{A}} \pi(x,y,a')\overline{A}^{\pi}(x,y,a',y') \end{aligned} \tag{3.16}$$

Note that $y'$ is arbitrary. Therefore, we can write equation 3.16 for all possible values of $y'$. Adding all of these equations, we have:

$$\begin{aligned} \sum_{y'\in\mathcal{Y}} A^{\pi}(x,y,a) &= \sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a,y') - \sum_{y'\in\mathcal{Y}} \left(\sum_{a'\in\mathcal{A}} \pi(x,y,a')\overline{A}^{\pi}(x,y,a',y')\right) \\ |\mathcal{Y}|A^{\pi}(x,y,a) &= \sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a,y') - \sum_{a'\in\mathcal{A}} \left(\sum_{y'\in\mathcal{Y}} \pi(x,y,a')\overline{A}^{\pi}(x,y,a',y')\right) \\ A^{\pi}(x,y,a) &= \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a,y') - \frac{1}{|\mathcal{Y}|}\sum_{a'\in\mathcal{A}} \left(\pi(x,y,a') \sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a',y')\right) \end{aligned} \tag{3.17}$$

If we define the mean advanced advantage function as:

$$\widetilde{A}^{\pi}(x,y,a) = \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a,y') \tag{3.18}$$

then equation (3.17) becomes as:

$$\begin{aligned} A^{\pi}(x,y,a) &= \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a,y') - \sum_{a'\in\mathcal{A}}\left(\pi(x,y,a')\frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a',y')\right) \\ &= \widetilde{A}^{\pi}(x,y,a) - \sum_{a'\in\mathcal{A}} \pi(x,y,a')\widetilde{A}^{\pi}(x,y,a') \end{aligned} \tag{3.19}$$

Equation 3.19 confirms that the advantage function can be derived directly from $\widetilde{A}^{\pi}(x,y,a)$. The following equation shows that $\widetilde{A}^{\pi}(x,y,a)$ can be estimated independently of the current value of the autonomous state:

$$\begin{aligned} \widetilde{A}^{\pi}(x,y,a) &= \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{A}^{\pi}(x,y,a,y') \\ &= \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \left(Q^{\pi}(x,y,a) - \overline{V}^{\pi}(x,y')\right) \\ &= Q^{\pi}(x,y,a) - \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{V}^{\pi}(x,y') \\ &= E_{\pi}\left[ r_{t+1} + \sum_{k=1}^{\infty}\gamma^{k} r_{t+k+1} \middle| \begin{array}{c} X_t = x \\ Y_t = y \\ a_t = a \end{array} \right] - \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{V}^{\pi}(x,y') \\ &= E_{\pi}\left[ r_{t+1} + V^{\pi}(x',y') \middle| \begin{array}{c} X_t = x \\ Y_t = y \\ a_t = a \end{array} \right] - \frac{1}{|\mathcal{Y}|}\sum_{y'\in\mathcal{Y}} \overline{V}^{\pi}(x,y') \end{aligned} \tag{3.20}$$

where $x'$ and $y'$ are the values of the successive controllable and autonomous states respectively.

### 3.4.2 Model-Free Control Algorithm

Model-free control algorithm (MF-C) is a control algorithm for IDSD problems. The algorithm is based on the fact that the orders of $A^{\pi}(x,y,a)$ and $\widetilde{A}^{\pi}(x,y,a)$ are the same for any state $(x, y)$ (see equation 3.19). Therefore, in the control case, one can extract the policy directly from the mean advanced advantage function.

MF-C algorithm stores the estimate of the mean advanced advantage function as described in the previous section, and uses that estimate to extract the policy. Algorithm 4 shows the general form of MF-C.

**Algorithm 3** Model-Free Prediction Algorithm

```
Initialize $V(x, y)$ and $\overline{V}(x, y)$ arbitrarily
Initialize $\widetilde{A}(x, y, a)$ and $A(x, y, a)$ arbitrarily
Initialize $\pi$ to the policy to be evaluated
Repeat (for each episode):
        Initialize $(x, y)$
        Repeat (for each step of episode):
                $a \leftarrow$ action given by $\pi$ for $(x, y)$
                Take action a; observe reward $r$, and next state $(x', y')$
                $V(x, y) \leftarrow V(x, y) + \alpha \left[r + \gamma V(x', y') - V(x, y)\right]$
                $\overline{V}(x, y') \leftarrow \overline{V}(x, y') + \alpha \left[r + \gamma V(x', y') - \overline{V}(x, y')\right]$
                $\widetilde{A}(x, y, a) \leftarrow \widetilde{A}(x, y, a) + \beta \left[r + \gamma V(x', y') - \overline{V}(x, y') - \widetilde{A}(x, y, a)\right]$
                $(x, y) \leftarrow (x', y')$
        until $(x, y)$ is terminal
For each state-action pair $(x, y, a)$
        $A(x, y, a) \leftarrow \widetilde{A}(x, y, a) - \sum_{a \in \mathcal{A}} \pi(x, y, a) \widetilde{A}^{\pi}(x, y, a)$
Return $A$ as the estimate of the advantage function
```

**Algorithm 4** Model-Free Control Algorithm

```
Initialize $V(x, y)$ and $\overline{V}(x, y)$ arbitrarily
Initialize $\widetilde{A}(x, y, a)$ arbitrarily
Repeat (for each episode):
    Initialize $(x, y)$
    Repeat (for each step of episode):
        Choose $a$ for $(x, y)$ using the policy derived from $\widetilde{A}$ (*e.g.*, $\epsilon$-greedy)
        Take action a; observe reward $r$, and next state $(x', y')$
        $V(x, y) \leftarrow V(x, y) + \alpha \left[r + \gamma V(x', y') - V(x, y)\right]$
        $\overline{V}(x, y') \leftarrow \overline{V}(x, y') + \alpha \left[r + \gamma V(x', y') - \overline{V}(x, y')\right]$
        $\widetilde{A}(x, y, a) \leftarrow \widetilde{A}(x, y, a) + \beta \left[r + \gamma V(x', y') - \overline{V}(x, y') - \widetilde{A}(x, y, a)\right]$
        $(x, y) \leftarrow (x', y')$; $a \leftarrow a'$
    until $(x, y)$ is terminal
```

# Chapter 4

# Analysis of the Proposed Algorithms

## 4.1 Partially Model-Based Algorithms

In the following sections we analysis the time complexity of the partially model-based algorithms. We also investigate the asymptotic convergence bound of AB-C for greedy action selection. Furthermore, we discuss that for the same amount of experiments, this asymptotic convergence bound is lower than that of Sarsa(0).

### 4.1.1 Time Complexity

Both of the partially model-based algorithms take a weighted average over the space of the autonomous state at every time-step (lines 12 in algorithms 1 and 2). Therefore, the time complexity of the partially model-based algorithms is polynomial in the size of the autonomous state's space, $O(|\mathcal{X}|\cdot|\mathcal{Y}|^2)$. This is a drawback for the algorithms, since the time complexities of TD(0) and Sarsa(0) are linear in the size of state space, $O(|\mathcal{X}|\cdot|\mathcal{Y}|)$.

### 4.1.2 Asymptotic Convergence Bound

(Bertsekas, 1987) and (Singh & Yee, 1994) showed that under $\gamma$-discounted setting, the quality of the policy learned by being greedy with respect to the estimate of the action-value function, $Q$, is bounded by the max norm error of $Q$, defined by $||Q - Q^*||_\infty = \arg\max_{(s,a)} |Q(s,a) - Q^*(s,a)|$:

**Theorem 4.1.** *Assume $||Q - Q^*|| \leq \epsilon$, and let $\pi$ be the greedy policy with respect to $Q$. Then for all $s$,*

$$V^\pi(s) \geq V^*(s) - \frac{2\epsilon}{1-\gamma}$$

Similarly, we can show that for a greedy update in AB-C, the quality of the learned policy is bounded by the max norm error in the estimate of the advanced action-value function, denoted by $\overline{Q}$.

**Theorem 4.2.** *Assume* $||\overline{Q} - \overline{Q}^*||_\infty \leq \epsilon'$*. Let* $Pr(y'|y)$ *denotes the probability of the next value of the autonomous state given the current value of the autonomous state, and let* $\pi$ *be the greedy policy with respect to* $Q(s) = Q(x,y) = \sum_{y' \in \mathcal{Y}} Pr(y'|y)\overline{Q}(x,y',a)$*. Then for all s,*

$$V^\pi(s) \geq V^*(s) - \frac{2\epsilon'}{1-\gamma}$$

*Proof.* By construction of $\pi$, $Q(s,\pi(s)) \geq Q(s,\pi^*(s))$:

$$\begin{aligned}
V^*(s) - Q^*(s,\pi(s)) &= V^*(s) - Q(s,\pi(s)) + Q(s,\pi(s)) - Q^*(s,\pi(s)) \\
&\leq V^*(s) - Q(s,\pi^*(s)) + \sum_{y'} Pr(y'|y)\overline{Q}(x,y',\pi(s)) - \sum_{y'} Pr(y'|y)\overline{Q}^*(x,y',\pi(s)) \\
&\leq V^*(s) - Q(s,\pi^*(s)) + \sum_{y'} Pr(y'|y)\Big(\overline{Q}(x,y',\pi(s)) - \overline{Q}^*(x,y',\pi(s))\Big) \\
&\leq Q^*(s,\pi^*(s)) - Q(s,\pi^*(s)) + \epsilon' \\
&\leq \sum_{y'} Pr(y'|y)\overline{Q}^*(x,y',\pi^*(s)) - \sum_{y'} Pr(y'|y)\overline{Q}(x,y',\pi^*(s)) + \epsilon' \\
&\leq \sum_{y'} Pr(y'|y)\Big(\overline{Q}^*(x,y',\pi^*(s)) - \overline{Q}(x,y',\pi^*(s))\Big) + \epsilon' \\
&\leq 2\epsilon'
\end{aligned}$$

Since $V^\pi(s) = Q^\pi(s,\pi(s))$, it follows

$$\begin{aligned}
V^*(s) - V^\pi(s) &= V^*(s) - Q^*(s,\pi(s)) + Q^*(s,\pi(s)) - V^\pi(s) \\
&\leq 2\epsilon' + Q^*(s,\pi(s)) - Q^\pi(s,\pi(s)) \\
&\leq 2\epsilon' + \gamma E_{s'\ p(.|s,\pi(s))}\Big(V^*(s') - V^\pi(s')\Big) \qquad (4.1)
\end{aligned}$$

where $s'$ denotes all possible successive states for taking action $\pi(s)$ in state $s$, and $p(.|s,\pi(s))$

is the probability of going to that state. The result follows from recursing inequality 4.1 and using linearity of expectation (Kakade, 2003). □

Theorem 4.1 and 4.2 show that for a greedy update in Sarsa(0) and AB-C, the policy does not get worse by more than a factor related to the worst-case error of $Q$ and $\overline{Q}$ respectively (Kakade, 2003). We now discuss that for the same amount of experiments, the worst-case error of $\overline{Q}$ is lower than the worst-case error of $Q$.

Recall that the advanced action-value function is the return given the current value of the controllable state and the next value of the autonomous state. Therefore, the transition of the autonomous state does not play any role in the value of the advanced action-value function. In contrast, any update to the value of a particular state-action pair makes the value greater or lesser depending on the next value of the autonomous state. This causes high variance in the estimate of $Q$ and increases the amount of experiment required to learn.

## 4.2 Model-Free Algorithms

In the following sections, we analyze the model-free algorithms in term of time complexity. We also discuss that for the same amount of experiment the asymptotic bound of MF-C is lower than the asymptotic bound of Sarsa(0).

### 4.2.1 Time Complexity

MF-P has two stages. First, it estimates the mean advanced advantage function which is linear in the size of state-action pairs (lines 4-12 in algorithm 3). Second, it derives $A(x, y, a)$ from $\widetilde{A}(x, y, a)$ by going through all state-action pairs which is again linear in the size of state-action pairs (lines 13-15 in algorithm 3). This leads the algorithm to have a linear time complexity in the size of state-action pairs, $O(|\mathcal{X}|\cdot|\mathcal{Y}|\cdot|\mathcal{A}|)$. In the case of control, MF-C, the estimate of $\widetilde{A}(x, y, a)$ is used directly to choose an action. Thus, the time complexity of MF-C is $O(|\mathcal{X}|\cdot|\mathcal{Y}|\cdot|\mathcal{A}|)$.

### 4.2.2 Asymptotic Convergence Bound

The following theorem shows that the policy learned by being greedy with respect to the estimate of the mean advanced advantage function, $\widetilde{A}$, does not get worse by more than a factor related to the max norm error of $\widetilde{A}$:

**Theorem 4.3.** *Assume* $||\widetilde{A} - \widetilde{A}^*||_\infty \leq \epsilon''$ *and let* $\pi$ *be the greedy policy with respect to* $\widetilde{A}$*. Then for all s,*

$$V^\pi(s) \geq V^*(s) - \frac{2\epsilon''}{1-\gamma}$$

*Proof.* By construction of $\pi$, $\widetilde{A}(s,\pi(s)) \geq \widetilde{A}(s,\pi^*(s))$:

$$\begin{aligned}
V^*(s) - Q^*(s,\pi(s)) &= V^*(s) - \widetilde{A}(s,\pi(s)) + \widetilde{A}(s,\pi(s)) - Q^*(s,\pi(s)) \\
&\leq V^*(s) - \widetilde{A}(s,\pi^*(s)) + \widetilde{A}(s,\pi(s)) - Q^*(s,\pi(s)) \\
&\leq V^*(s) - \widetilde{A}(s,\pi^*(s)) + \widetilde{A}(s,\pi(s)) - \widetilde{A}^*(s,\pi(s)) - \overline{V}^*(s) \\
&\leq V^*(s) - \widetilde{A}(s,\pi^*(s)) + \epsilon'' - \overline{V}^*(s) \\
&\leq Q^*(s,\pi^*(s)) - \widetilde{A}(s,\pi^*(s)) + \epsilon'' - \overline{V}^*(s) \\
&\leq \widetilde{A}^*(s,\pi^*(s)) + \overline{V}^*(s) - \widetilde{A}(s,\pi^*(s)) + \epsilon'' - \overline{V}^*(s) \\
&\leq 2\epsilon''
\end{aligned}$$

Since $V^\pi(s) = Q^\pi(s,\pi(s))$, it follows:

$$\begin{aligned}
V^*(s) - V^\pi(s) &= V^*(s) - Q^*(s,\pi(s)) + Q^*(s,\pi(s)) - V^\pi(s) \\
&\leq 2\epsilon'' + Q^*(s,\pi(s)) - Q^\pi(s,\pi(s)) \\
&\leq 2\epsilon'' + \gamma E_{s'\ p(.|s,\pi(s))}\Big(V^*(s') - V^\pi(s')\Big) \qquad (4.2)
\end{aligned}$$

where $s'$ denotes all possible successive states for taking action $\pi(s)$ in state $s$, and $p(.|s,\pi(s))$ is the probability of going to that state. The result follows from recursing inequality (4.2) and using linearity of expectation (Kakade, 2003). □

Figure 4.1, shows the main difference between MF-C and Sarsa(0). MF-C considers only the transition of the controllable state for assigning credit to the agent's action. Intuitively, this exclusion eliminates any variance caused by the dependency between controllable and autonomous state within a time-step, since (1) the agent is not penalized for the

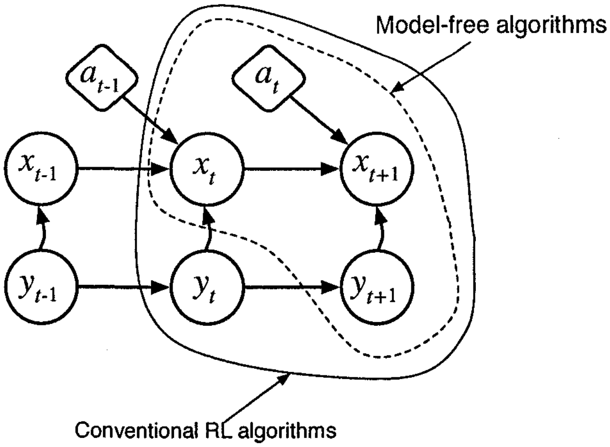


Figure 4.1: The state variables that participate in assigning credit to the agent's action in the model-free and conventional RL algorithms. The dotted line shows the state variables that participate in assigning credit to the agent action in the model-free algorithms. The solid line shows the state variables that participate in assigning credit to the agent action in the conventional RL algorithms.

evolution of the autonomous state, and (2) only the autonomous state that is involved in the transition of controllable state participate in the credit assignment.

In contrast, Sarsa(0) considers both the transitions of the autonomous and controllable states for assigning credit to the action-value function. Therefore, the credit that the agent receives for executing one particular action is greater or lesser depending on the next value of the autonomous state. This causes high variance in the estimate of $Q$ and increases the amount of experiment required to learn an accurate action-value function.

# Chapter 5

# Experimental Results

In this chapter, we first introduce three problems that match the conditions of IDSD problems: (1) traffic gridworld, (2) noisy traffic gridworld, and (3) hybrid vehicle. For each of the problem, we show the comparison between the results of the proposed algorithms and conventional reinforcement learning algorithms in the case of prediction and control.

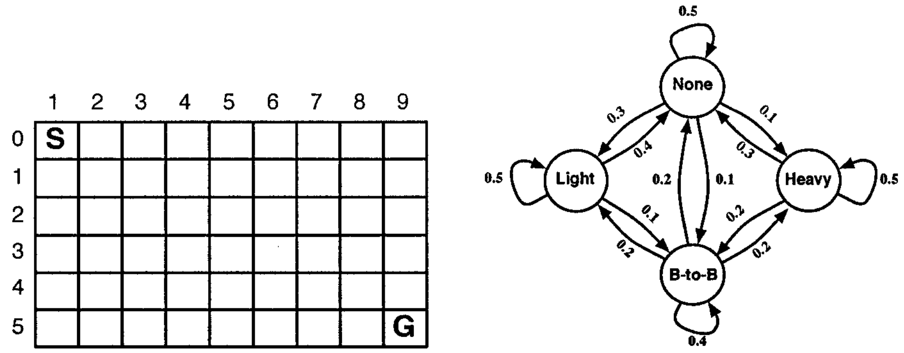
In the prediction case, we compared the proposed algorithms with TD(0). Our comparison is based on the quality of the estimate of the advantage function, because the advantage function is the only function that can be estimated by all algorithms.

In the case of control, we compared the quality of the policy learned by the proposed algorithms to Sarsa(0) and the advantage updating algorithm. The comparison is based on learning speed, as well as performance in the long run.

## 5.1 Traffic Gridworld Problem

Traffic gridworld is a $6 \times 9$ standard gridworld, in which the reward is a function of a traffic system (see figures 5.1(a)). The traffic system has four levels: none, light, heavy, and bumper to bumper. The agent has four actions—up, down, left, and right— and receives a cost (reward) for performing each action with respect to the next level of the traffic system. The next traffic level is a function of the current traffic level. Figure 5.1(b) shows the state diagram of the traffic system. The agent's objective is to find a policy that minimizes the cost of going to the goal.

At each time-step, the agent takes an action based on the current state $s_t = (x_t, y_t)$, in which $x_t = Row(x_t) \times 9 + Column(x_t)$ is the current cell number and $y_t$ is the current



(a) Standard gridworld with start and goal state. (b) State diagram of the traffic system.

Figure 5.1: Traffic gridworld problem.

traffic level. Afterward, the agent receives a new state and a reward as a consequence of its action. The cost is 1, 5, 10 or 20 when the next traffic level, $y_{t+1}$, is none, light, heavy or bumper-to-bumper respectively.

The traffic gridworld problem matches the conditions that define an IDSD problem. First, the traffic level is independent of cells and the action selected by the agent. Second, the agent is not able to sense the next traffic level until the time-step is completed.

### 5.1.1 Prediction

We evaluated the autonomous-based prediction algorithm (AB-P), the model-free prediction algorithm (MF-P), and TD(0) in the traffic gridworld problem. We chose a uniform random policy as the behavior policy. We set all the initial values of states and state-action pairs to 0, and the discount factor to 0.9. In the case of MF-P, we used the simplest setting for the step-size parameters; *i.e.,* we set $\alpha = \beta$. This setting has been used in all of the experiments in this study.

We created 100 different trajectories of the traffic system. We then ran the algorithms for 100 000 time-steps over each of these 100 trajectories. After each run, we measured the prediction error by:

$$error = A(x, y, a) - A^*(x, y, a) \tag{5.1}$$

where $A(x, y, a)$ denotes the estimated advantage of the state-action pair at 100 000 time-

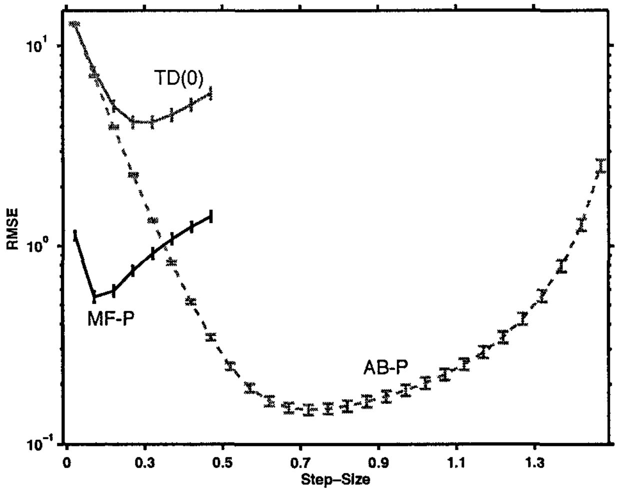


Figure 5.2: Prediction in the traffic gridworld problem. Root mean-square error for TD(0), the autonomous-based prediction algorithm (AB-P), and the model-free prediction algorithm (MF-P).

steps, and $A^*(x, y, a)$ denotes the true advantage of that state-action pair. Afterward, we measured the root mean-square error (RMSE) of each algorithm over 100 runs by:

$$RMSE = \sqrt{\frac{1}{100}\sum_{i=1}^{100}\left(error_i\right)^2} \tag{5.2}$$

where $error_i$ denotes the prediction error of the $i$-th run.

Figure 5.2 shows the RMSE of each algorithm for a wide range of the step-size parameter. The results show that MF-P and AB-P performed better than TD(0) for many values of the step-size parameter. Moreover, AB-P worked better than MF-P.

### 5.1.2 Control

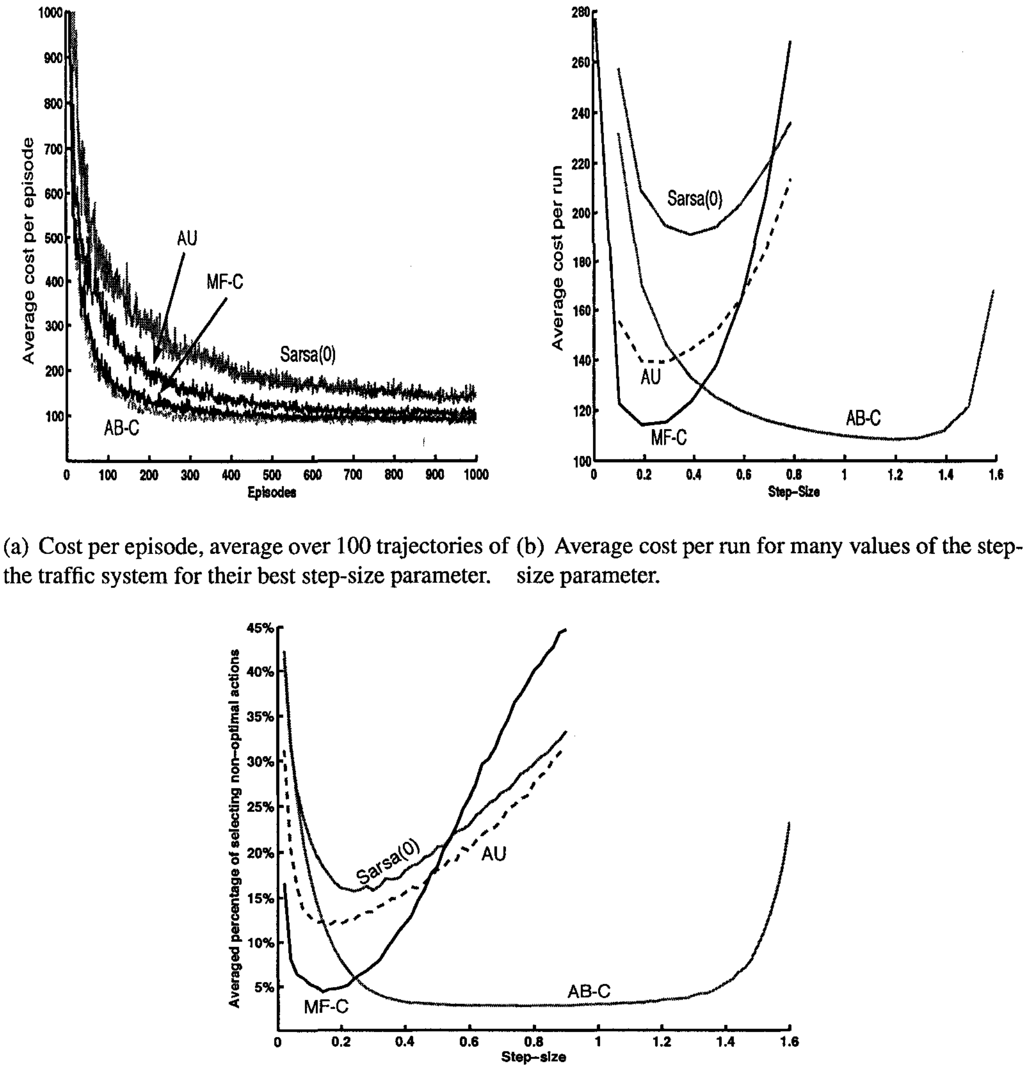
We performed two set of experiments to compare the quality of the policy learned by the proposed algorithms to Sarsa(0) and the advantage updating algorithm (AU).

In the first set of experiments, we compared the learning speed of the algorithms. We created 100 different trajectories of the traffic system. We ran the algorithms for 2000 episodes over each of these trajectories of the traffic system, and repeated the experiment for a wide range of step-size parameter. All algorithms used an $\epsilon$-greedy policy with the exploration factor set to 5%. The initial values of states and state-action pairs were 0, and the discount factor was 0.9. In the case of the advantage updating algorithm, we used $\beta = 0.5\alpha$, one of the best settings for its step-size parameters. In the case of MF-C, we used the simplest setting for the step-size parameters; *i.e.,* $\alpha = \beta$. In all of the experiments in this work, the advantage updating algorithm and MF-C used the same setting for their step-size parameters.

Figure 5.3(a) shows the cost per episode of each algorithm, averaged over 100 runs for the best step-size parameter. The results show that the learning speeds of MF-C and AB-C algorithms were substantially better than learning speeds of Sarsa(0) and AU. Moreover, although MF-C did not use any model, its learning speed was nevertheless almost the same as that of AB-C (Note that in this particular experiment the model of the autonomous dynamics was almost exact after 50 episodes.) Figure 5.3(b) shows the average cost per run for many values of the step-size parameter. The results confirmed that the proposed algorithms outperformed the conventional algorithms in the long run.

In the second set of experiments, we measured the effect of delays in sensing the traffic levels on the behavior of the algorithms. We divided the first set of experiments into two parts. The first part was a learning part, in which we ran the algorithms for 1500 episodes over the 100 different trajectories of the traffic system. By the end, all algorithms nearly converged to their final solutions. The second part was an evaluation part. For each algorithm, we initialized the values of states and state-action pairs to the values learned in the first part. We then ran the algorithms for 500 episodes over the 100 different trajectories of the traffic system.

Figure 5.3(c) shows the average percentage of selecting non-optimal actions per runs for many values of the step-size parameter. The results show that MF-C and AB-C converged to the optimal policy (Note that the optimal percentage of selecting non-optimal actions is equal to the exploration factor.) In contrast, Sarsa(0) and AU failed to behave optimally. This is due to the fact that the conventional credit assignment can decrease or increase the action-value function depending on the next value of the traffic. The results also show that the AB-C algorithm was less prone to the step-size changes than the other algorithms.



(a) Cost per episode, average over 100 trajectories of the traffic system for their best step-size parameter.

(b) Average cost per run for many values of the step-size parameter.

(c) Average percentage of selecting non-optimal actions per run for many values of the step-size parameter. The results are averaged over 100 trajectories of the traffic system.

Figure 5.3: Control in the traffic gridworld problem using Sarsa(0), the autonomous-based control algorithm (AB-C), the model-free control algorithm (MF-C), and the advantage updating algorithm (AU).

## 5.2 Noisy Traffic Gridworld Problem

Noisy traffic gridworld is a traffic gridworld like the one described in section 5.1, but with one difference: the number of the current cell that the agent receives from the gridworld is polluted with respect to the traffic level. More precisely, if $s_t = (x_{t+1}, y_{t+1})$ is the real successive state for taking action $a_t$ in state $s_t = (x_t, y_t)$, then the observation that the agent receives is $o_t = (\widehat{x}_{t+1}, y_{t+1})$, where $\widehat{x}_{t+1}$ is:

$$\widehat{x}_{t+1} = \begin{cases} x_{t+1} - d & \text{if } x_{t+1} - d \geq 1 \\ 1 & \text{otherwise} \end{cases} \tag{5.3}$$

where $d$ is 0, 1, 2 or 3 when $y_{t+1}$ is none, light, heavy or bumper-to-bumper respectively.

At each time-step $t$, the agent takes an action based on the current observation $o_t = (\widehat{x}_t, y_t)$. Afterward, the agent receives a new observation $o_t = (\widehat{x}_{t+1}, y_{t+1})$, and a cost as a consequence of its action. The cost is 1, 5, 10 or 20 when the next traffic level, $y_{t+1}$, is none, light, heavy or bumper-to-bumper respectively.

### 5.2.1 Policy Evaluation

We used the noisy traffic gridworld problem to compare the proposed algorithm to TD(0) in the case of prediction. We set the behavior policy to a uniform random policy, and used the same parameter setting as described in section 5.1.1.

We created 100 different trajectories of the traffic system, and ran the algorithms for 100 000 time-steps over each of these trajectories. We then repeated the experiment for a wide range of step-size parameters. We then used equation (5.2) to measure the RMSE of each algorithm over 100 runs.

Table 5.1 shows the best RMSE of each algorithm, the variance of the best RMSE and the step-size parameter that the algorithms used to achieve that RMSE. The results show that the proposed algorithm outperformed TD(0). Moreover, MF-P worked slightly better than AB-P.

| Algorithm | RMSE | Var. | Step-size |
|---|---|---|---|
| TD(0) | 5.8820 | 4.3018 | 0.29 |
| AB-P | 3.9023 | 3.6218 | 0.6 |
| MF-P | 3.8283 | 3.3380 | 0.06 |

Table 5.1: Prediction in the noisy traffic gridworld problem. Best root mean-square error, its variance and its respective step-size parameter for TD(0), the autonomous-based prediction algorithm (AB-P), and the model-free prediction algorithm (MF-P).

### 5.2.2 Control

We compared the learning speed of the proposed algorithms to Sarsa(0) and AU in the noisy traffic gridworld problem under the same parameter setting as described in section 5.1.2.

We ran the algorithms for 2000 episodes over each of the 100 different trajectories of the traffic system, and repeated the experiment for many values of the step-size parameter.

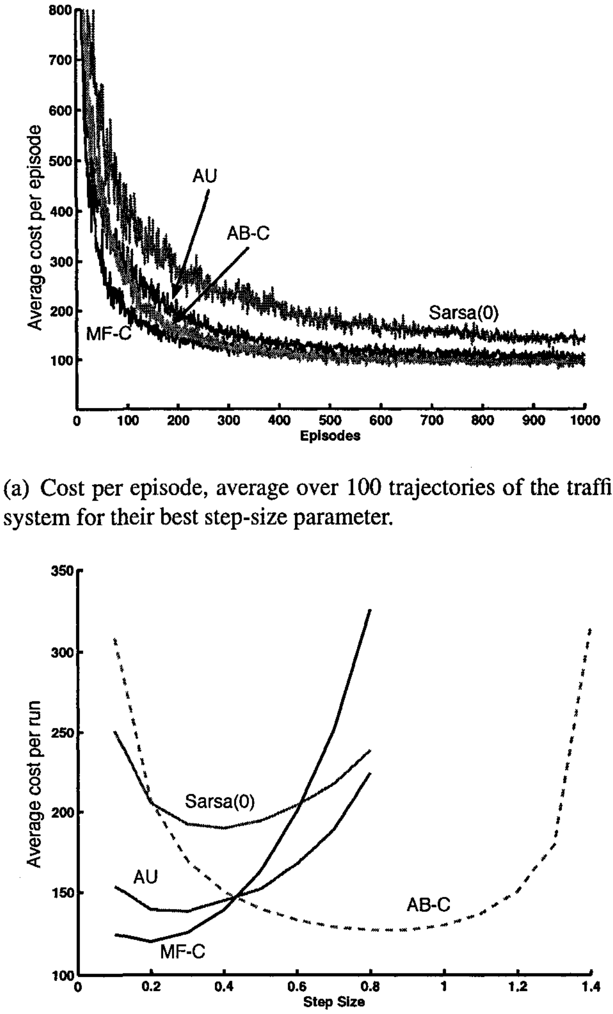
Figure 5.4(a) shows the cost per episode, averaged over 100 runs. The results show that the proposed algorithms learned faster than Sarsa(0) and AU. Moreover, MF-C outperformed AB-C. This is a surprising result, since model-based algorithms, in general, work better than model-free algorithms. In section 5.4, we discuss some of the possible reasons for this result.

Figure 5.4(b) shows the average cost per run for many values of the step-size parameter. The results confirmed that MF-C worked better than the other algorithms in the long run.

## 5.3 Hybrid Vehicle Problem

We explored the proposed algorithms in relation to a hybrid vehicle problem, where the driver plays the role of the autonomous dynamics. A hybrid electric vehicle combines a conventional propulsion system with a rechargeable energy storage system to achieve better fuel economy. In the hybrid vehicle problem, the vehicle uses an internal gas engine and a rechargeable electrical battery to power an electric motor. The agent objective is to minimize gas consumption by switching between the gas engine and the electrical motor.

In our simplified simulation, the battery is discretized into six levels. If the agent selects the electric motor, the battery-level is depleted by one or more units depending on the next value of driver acceleration. When the gas engine is used, the vehicle consumes



(a) Cost per episode, average over 100 trajectories of the traffic system for their best step-size parameter.

(b) Average cost per run for many values of the step-size parameter.

Figure 5.4: Control in the noisy traffic gridworld problem using Sarsa(0), the autonomous-based control algorithm (AB-C), the model-free control algorithm (MF-C), and the advantage updating algorithm (AU).

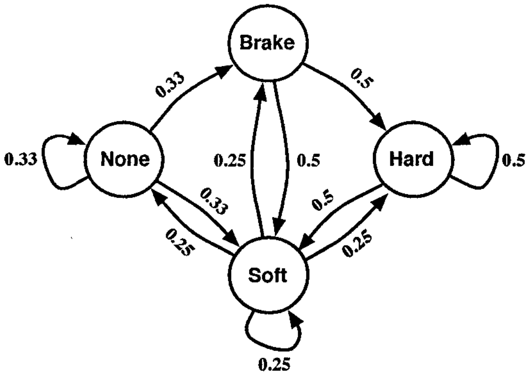


Figure 5.5: The state diagram of the driver acceleration in the hybrid vehicle problem.

some amount of gas with respect to the next value of driver acceleration, and the battery is replenished by one level. Table 5.2 shows the transition of the battery-level and gas as a function of driver acceleration and agent's actions. The transition of driver acceleration is presented in figure 5.5.

At any time-step $t$, the agent takes an action based on the current state, the current values of the battery-level and driver acceleration. Afterward, the agent receives a new state and a reward as a consequence of its action. The reward is equal to the gas consumption. If the battery-level drops to 0, the agent receives a reward of -20 and the battery-level is reset to its maximum level.

The hybrid vehicle problem is an infinite horizon task that satisfies the IDSD conditions. First, driver acceleration is independent of agent's actions and battery-levels. Second, the transition of the driver acceleration can be sensed only after a delay.

| Driver Acceleration | Gas Engine | | Electric Motor | |
|---|---|---|---|---|
| | Gas Consumption | Battery Change | Gas Consumption | Battery Change |
| None | 1 | +1 | 0 | -1 |
| Soft | 1.5 | +1 | 0 | -2 |
| Hard | 3 | +1 | 0 | -5 |
| Brake | 1 | +1 | 0 | +1 |

Table 5.2: Battery-level transitions and gas consumption with respect to the driver acceleration and agent actions.

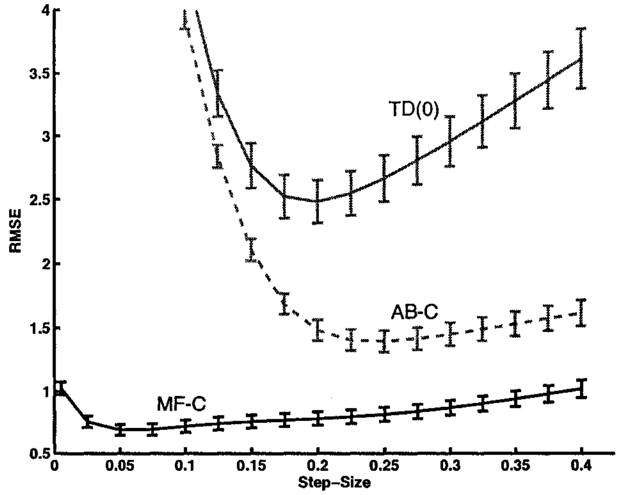


Figure 5.6: Prediction in the hybrid vehicle problem. Root mean-square error for TD(0), the autonomous-based prediction algorithm (AB-P), and the model-free prediction algorithm (MF-P).

### 5.3.1 Prediction

We compared the prediction speed of the proposed algorithms to TD(0) in the hybrid vehicle problem. We chose a uniform random policy as the behavior policy, and used the same parameter setting as described in section 5.1.1.

We created 100 different trajectories of the driver acceleration, and ran the algorithms for 10 000 time-steps over each of these trajectories. We then repeated the experiment for a wide range of step-size parameter. We then measured the RMSE of each algorithm by using equation (5.2).

Figure 5.6 shows the RMSE of each algorithm for many values of the step-size parameter. The results show that MF-P and AB-P performed substantially better than TD(0). Moreover, the RMSE of the proposed algorithms was less sensitive to the choice of the step-size parameter. Finally, as with the noisy traffic gridworld, MF-P outperformed AB-P.

Table 5.3 compares the best RMSE of each algorithm after 1000, 10 000 and 100 000 time-steps. As it can be seen, the initial learning speed of MF-C was substantially higher than the other algorithms. Nevertheless, the difference between the RMSE of the algorithms decreased over time.

| Algorithm | RMSE | | |
|---|---|---|---|
| | 1000 Time-steps | 10000 Time-steps | 100000 Time-steps |
| TD(0) | 3.1349 | 2.4781 | 1.1539 |
| AB-P | 3.3284 | 1.3912 | 1.0531 |
| MF-P | 1.0873 | 0.6867 | 0.6461 |

Table 5.3: Prediction in the hybrid vehicle problem. Best root mean-square error, its variance and its respective step-size parameter for TD(0), the autonomous-based prediction algorithm (AB-P), and the model-free prediction algorithm (MF-P).

### 5.3.2 Control

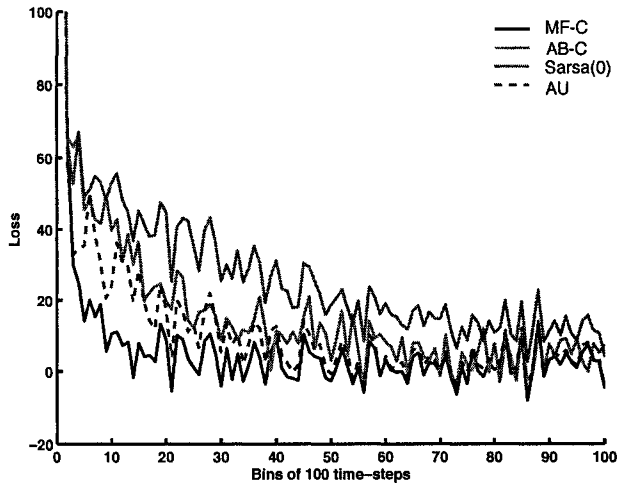
We compared the quality of the policy learned by the proposed algorithms to that of Sarsa(0) and AU in the hybrid vehicle problem.

We ran the algorithms for 100 000 time-steps over the 100 different trajectories of the driver acceleration for many values of the step-size parameter. All algorithms followed an $\epsilon$-greedy policy with the exploration factor set to 5%. The initial values of states and state-action pairs were set to 0, and the discount factor was 0.9.

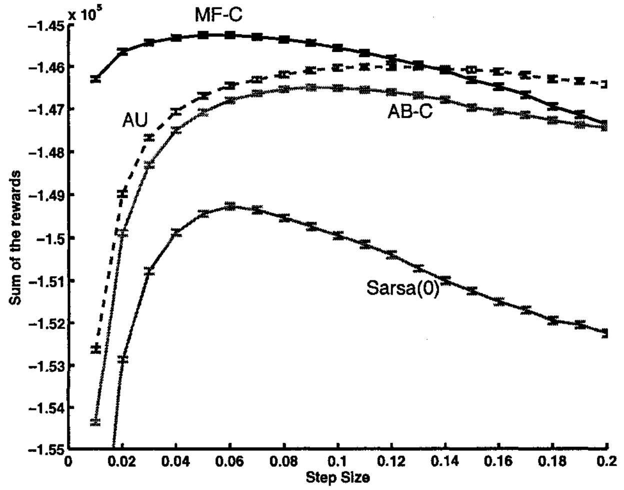
We divided each run into 1000 intervals, called *bins*, each of which includes 100 consecutive time-steps. For each bin, we define the loss of an algorithm as the difference between the sum of the rewards received by that algorithm and the $\epsilon$-soft optimal policy. Figure 5.7(a) shows the loss of each algorithm in the first 100 bins, averaged over 100 runs for the best step-size parameter. The results show that MF-C learned substantially faster than the other algorithms. Figure 5.7(b) shows the total received rewards (after 100 000 time-steps) for each algorithm, averaged over 100 runs for many values of the step-size parameter. The results confirm that MF-C outperformed the other algorithms in the long run.

## 5.4 Discussion

The model-free algorithms learned faster than the partially model-based algorithms in the noisy traffic gridworld and hybrid vehicle problems. These results suggest that the sample complexity of the partially model-based algorithms is higher than that of the model-free algorithms. In the following, we will investigate some of the properties of the partially model-based algorithm in the case of prediction (AB-P), that could affect its sample com-



(a) Loss per bin of algorithms averaged over 100 runs for their best step-size parameter.



(b) Sum of the received rewards after 100 000 time-steps, averaged over 100 runs for many values of the step-size parameter.

Figure 5.7: Control in the hybrid vehicle problem using Sarsa(0), the autonomous-based control algorithm (AB-C), the model-free control algorithm (MF-C), and the advantage updating algorithm (AU).

plexity. The same argument holds for AB-C.

Recall that AB-P uses two functions: (1) the estimate model of the autonomous dynamics, and (2) the estimate of the advanced value function. The algorithm uses both functions to derive the value function (equation (3.8)), and uses that value function to make a target for the advanced value function (equation 3.7). We can combine these two procedures to form the following updating rule for the advanced value function:

$$\overline{V}(X_t, Y_{t+1}) = \overline{V}(X_t, Y_{t+1}) + \alpha\Big[r_{t+1} + \gamma \sum_{y' \in \mathcal{Y}} P(y', Y_t)\overline{V}(X_{t+1}, y') - \overline{V}(X_t, Y_{t+1})\Big] \tag{5.4}$$

Equation (5.4) implies that any estimated error in each of $\overline{V}(X_{t+1}, y')$ propagates into $\overline{V}(X_t, Y_{t+1})$ with respect to $P(y', Y_t)$. These errors again propagate into other advanced values whenever $\overline{V}(X_t, Y_{t+1})$ is used to form a target. In other words, the estimated error in $\overline{V}(X_t, Y_{t+1})$ is related to the estimated error of all possible $P(y', Y_t)\overline{V}(X_{t+1}, y')$, and the estimated error in each of $\overline{V}(X_{t+1}, y')$ is related to the estimated error of all possible $P(y', Y_t)\overline{V}(X_{t+2}, y')$ and so forth. This propagation of errors in the advanced value function increases the sample complexity of the algorithm, especially in the case that the model of the autonomous dynamics can be learned before the agent experiences enough trajectories to have a good estimate of the advanced value function.

For instance, in the noisy traffic gridworld problem, AB-P had a perfect model of the autonomous dynamics by episode 50, but at that time it had not experienced enough trajectories to have a good estimate of the values of advanced states. Therefore, making a target by taking a weighted average over all values of possible successive advanced states makes the updating less effective and slows down the learning procedure. In the next chapter, we present some solutions that can prevent the propagation of error from the estimate value of one advanced state into the estimate values of other advanced states.

# Chapter 6

# Modified Partially Model-Based Algorithms

As discussed in section 5.4, the partially model-based algorithms are prone to the propagation of error in the estimate value of one advanced state into the estimate values of other advanced states. In this chapter, we will investigate some simple ways to prevent this error propagation in the partially model-based algorithms.

Recall that the error propagation in the partially model-based algorithms is due to the fact that all possible successive advanced states, $(X_{t+1}, y')$, participate in making a target with respect to $P(y', Y_t)$ calculated by the model of the autonomous state. Therefore, the first solution is to initially bias the model of the autonomous dynamics towards those advanced states that have more accurate estimates. Unfortunately, determining which advanced state have a better estimate is a difficult task, and is not feasible without modeling both the controllable and autonomous dynamics together.

The second solution is to learn a value function separately, and use that value function to form a target for the advanced value function. In this case, the error in the estimate value of one advanced state is limited to the error in the estimates values of its immediate successive states, and does not propagate into other estimate values of advanced states.

We call the algorithms that has been improved in this way the *modified autonomous-based prediction algorithm* (MAB-P) in the case of prediction and *modified autonomous-based control algorithm* (MAB-C) in the case of control. Algorithms 5 and 6 show the general form of MAB-P and MAB-C respectively. Note that in the case of prediction, taking the weighted average over the space of the autonomous state is done only once at

| Algorithm | RMSE | Var. | Step-size |
|---|---|---|---|
| MAB-P | 3.5505 | 3.0478 | 0.2 |
| AB-P | 3.9023 | 3.6218 | 0.6 |
| MF-P | 3.8283 | 3.3380 | 0.06 |

Table 6.1: Prediction in the noisy traffic gridworld problem. Best root mean-square error, its variance and its respective step-size parameter for the modified autonomous-based prediction algorithm (MAB-P), the autonomous-based prediction algorithm (AB-P), and the model-free prediction algorithm (MF-P).

the end of the algorithm (lines 14–16). Therefore, the time complexity of MAB-P is much lower than that of AB-P (algorithm 1).

## 6.1 Experimental Results

We evaluated the modified partially model-based algorithms by performing prediction and control on the noisy traffic gridworld and hybrid vehicle problems. In all of the experiments, we used the same problem settings and measurements as we used in chapter 5. We also used the simplest setting for the step-size parameters of the modified partially model-based algorithms; *i.e.*, $\alpha = \beta$.

### 6.1.1 Noisy Traffic Gridworld Problem

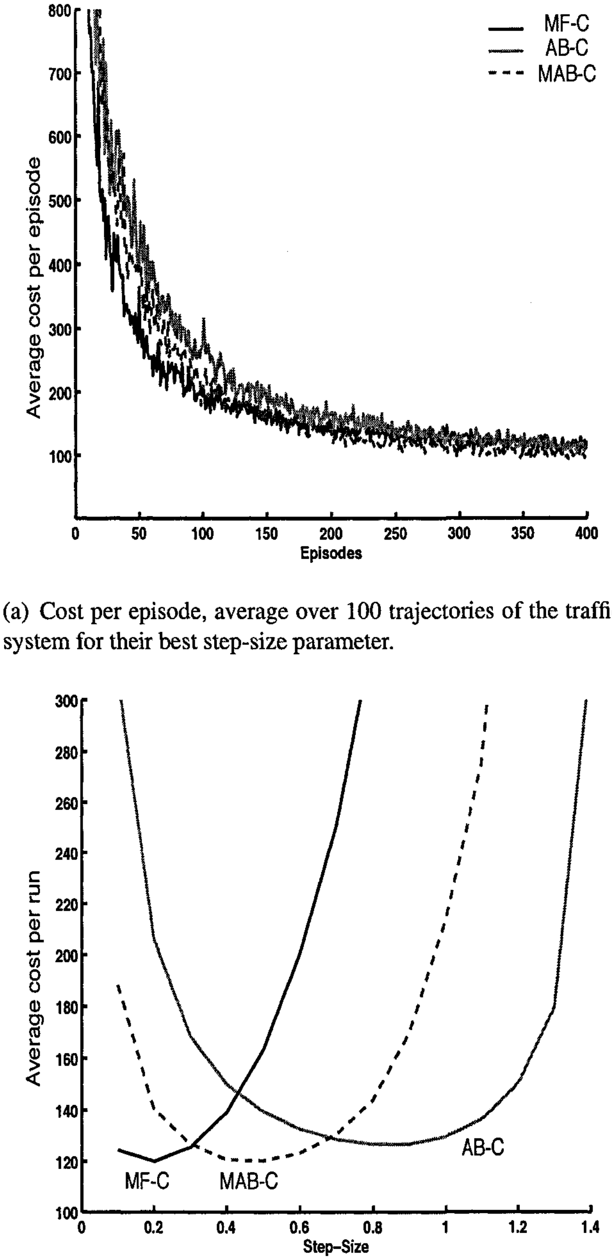
Table 6.1 compares the best RMSE of MAB-P, AB-P and MF-C in the case of prediction. The results show that the MAB-P worked better than the other algorithms.

Figure 6.1(a) shows the cost per episode, averaged over 100 runs in the case of control. The results confirm that MAB-C learned faster than AB-C, but still slower than MF-C. Figure 6.1(b) shows the average cost per run for many values of the step-size parameter. The results show that the long run performance of MAB-C was slightly better than that of the AB-C.

### 6.1.2 Hybrid Vehicle Problem

Figure 6.2 compares MAB-P to AB-P and MF-P in the case of prediction. The results show that MAB-P performed better than AB-P, but still worse than MF-P.

Figure 6.3(a) shows the loss of the algorithms, average over 100 runs for their best step-



(a) Cost per episode, average over 100 trajectories of the traffic system for their best step-size parameter.

(b) Average cost per run for many values of the step-size parameter.

Figure 6.1: Control in the noisy traffic gridworld problem using the modified autonomous-based control algorithm (MAB-C), the autonomous-based control algorithm (AB-C), and the model-free control algorithm (MF-C).

size parameters. The results show that MAB-C learned slightly faster than AB-C. Figure 6.3(b) confirms that the long-run performance of MAB-C was better than that of AB-C.

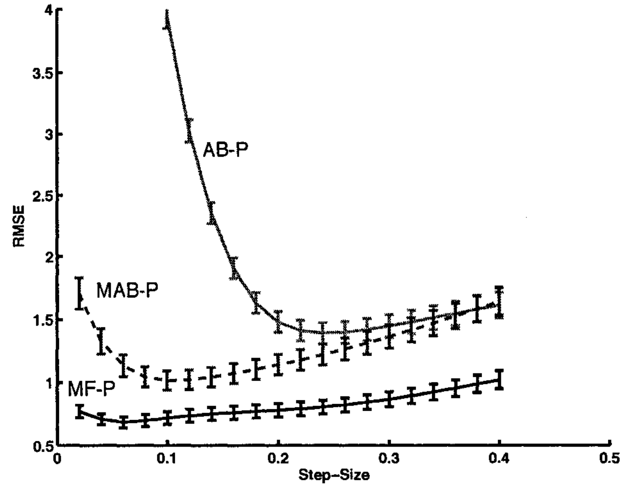
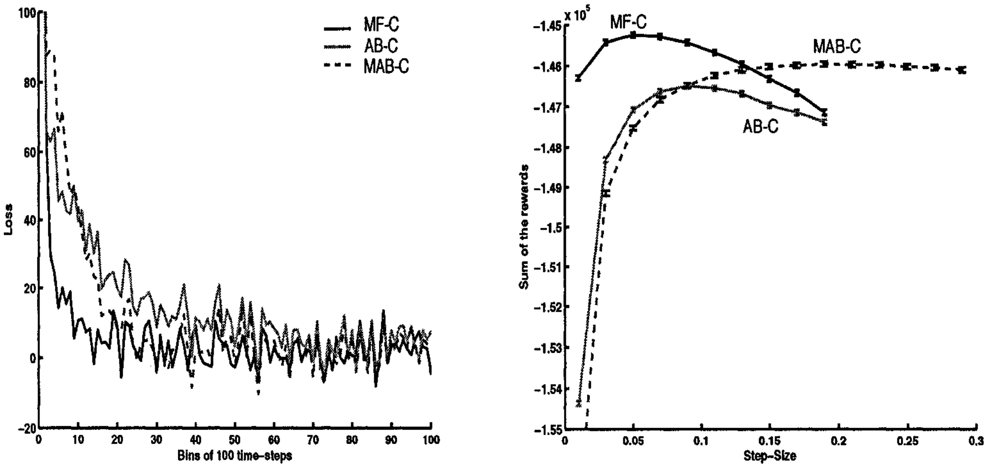


Figure 6.2: Prediction in the hybrid vehicle problem. Root mean-square error for the modified autonomous-based prediction algorithm (MAB-P), the autonomous-based prediction algorithm (AB-P), and the model-free prediction algorithm (MF-P).



(a) Average loss over 100 runs of algorithms for their best step-size parameter.

(b) Sum of the received rewards after 100 000 time-steps for many values of the step-size parameter.

Figure 6.3: Control in the hybrid vehicle problem using the modified autonomous-based control algorithm (MAB-C), the autonomous-based control algorithm (AB-C), and the model-free control algorithm (MF-C).

**Algorithm 5** Modified Autonomous-Based Prediction Algorithm

```
Initialize $V_{temp}(x,y)$, $\overline{V}(x,y)$ and $V(x,y)$ arbitrarily
Initialize $\pi$ to the policy to be evaluated
Initialize $m(x,y)$ and $n(y)$ to zero
Repeat (for each episode):
    Initialize $(x,y)$
    Repeat (for each step of episode):
        $a \leftarrow$ action given by $\pi$ for $(x,y)$
        Take action $a$; observe reward $r$, and next state $(x',y')$
        $V_{temp}(x,y) \leftarrow V_{temp}(x,y) + \beta\,[r + \gamma V_{temp}(x',y') - V_{temp}(x,y)]$
        $\overline{V}(x,y') \leftarrow \overline{V}(x,y') + \alpha\,\left[r + \gamma V_{temp}(x',y') - \overline{V}(x,y')\right]$
        $n(y) \leftarrow n(y) + 1$
        $m(y',y) \leftarrow m(y',y) + 1$
        $(x,y) \leftarrow (x',y')$
    until $(x,y)$ is terminal
For each $(x,y)$
    $V(x,y) \leftarrow \sum_{z\in\mathcal{Y}} \frac{m(z,y)}{n(y)}\overline{V}(x,z)$
Return $V$ as the estimate of the value function
```

**Algorithm 6** Modified Autonomous-Based Control Algorithm

```
Initialize $V(x,y)$, $Q(x,y,a)$ and $\overline{Q}(x,y,a)$ arbitrarily
Initialize $m(x,y)$ and $n(y)$ to zero
Repeat (for each episode):
    Initialize $(x,y)$
    Choose $a$ for $(x,y)$ using the policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $a$; observe reward $r$, and next state $(x',y')$
        Choose $a'$ for $s'$ using the policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $V(x,y) \leftarrow V(x,y) + \beta\left[r + \gamma V(x',y') - V(x,y)\right]$
        $\overline{Q}(x,y',a) \leftarrow \overline{Q}(x,y',a) + \alpha\left[r + \gamma V(x',y') - \overline{Q}(x,y',a)\right]$
        $n(y) \leftarrow n(y) + 1$
        $m(y',y) \leftarrow m(y',y) + 1$
        $Q(x,y,a) \leftarrow \sum_{z\in\mathcal{Y}} \frac{m(z,y)}{n(y)}\overline{Q}(x,z,a)$
        $(x,y) \leftarrow (x',y')$; $a \leftarrow a'$
    until $(x,y)$ is terminal
```

# Chapter 7

# Conclusion

We have introduced a reinforcement learning framework for modeling environments with independent delayed-sense dynamics. Under this framework, we have presented the fundamental material of reinforcement learning. We have shown that the value of a state (or state-action pair) is a linear combination of the values of all possible advanced states (or state-action pairs).

We have proposed two partially model-based algorithms for prediction and control in IDSD problems. The algorithms learn a model of the autonomous dynamics from experiments, and use that model to compute the state-value or the action-value function. We have proved that for greedy action selection, the policy learned by AB-C does not get worse by more than a factor related to the worst-case error (in terms of the max norm regression error) in the estimate of the advanced action-value function. We then discussed that for the same amount of experiment, the worst-case error in the estimate of the advanced action-value function learned by AB-C is lower than the worst-case error in the estimate of the action-value function learned by Sarsa(0).

We have also proposed two model-free algorithms, in which the value of the current autonomous state is excluded from assigning a credit to agent's actions. We have shown that this exclusion can be done by calculating the mean advanced advantage function for state-action pairs. We have shown that the quality of the policy learned by being greedy with respect to the estimate of the mean advanced advantage function is bounded by the worst-case error (in terms of the max norm regression error) in the estimate of the mean advanced advantage function. We also discussed that for the same amount of experiment, the worst-case error in the estimate of the mean advanced advantage function learned by MF-C is lower than the worst-case error in the estimate of the action-value function learned by Sarsa(0).

## 7.1 Empirical Evaluations

We have shown the performance of the proposed algorithms on three IDSD problems. The first two problems were episodic, in which a traffic system plays the role of the autonomous dynamics. The third problem was infinite horizon, where driver acceleration plays the role of the autonomous state variable.

The results confirmed that in all of the problems, the proposed algorithms outperformed the conventional RL algorithms in terms of prediction and control. The results also showed that the model-free algorithms outperformed the partially model-based algorithms in more complicated tasks such as the hybrid vehicle problem. We discussed some of the reasons of this limitation, and presented an adjustment to the updating rule of the partially model-based algorithms to resolve the limitation.

Finally, we conclude this thesis by emphasizing two advantages of the model-free algorithms: (1) their time complexity are linear in the size of the state space, and (2) their learning speeds on the selected problem were higher than that of the partially model-based algorithms, Sarsa(0), and the advantage updating algorithm. We expect that the model-free algorithms exhibit the same behavior (in terms of learning speed) in most IDSD problems.

## 7.2 Future Work

This thesis provides a foundation for future work in environments with independent delayed-sense dynamics. One area of future work is in using knowledge of autonomous dynamics in a more appropriate way resulting in better performance. Another area is in applying the algorithms studied here to various real-world problems that have an IDSD structure.

### 7.2.1 Applying Knowledge of Autonomous Dynamics

While the modification to the partially model-based algorithm improved the performance of the algorithm, the fact that the model-free algorithms were still superior implies that the model of the autonomous dynamics has not been perfectly exploited. We have tried a simple regularization method to improve the performance of the algorithms, but the results were not significant. In our simple regularization method, the weight of each advanced state was related to a regularization factor, the value of the advanced state, and the number of times that the advanced state had been visited. Therefore, future research should look at more complex methods of exploiting the model of the autonomous dynamics in a more efficient way.

### 7.2.2 Applying the Proposed Algorithms to Real-World Problems

The ultimate goal of this thesis is to provide reinforcement learning algorithms for real applications that have an IDSD structure. There are two issues that have to be solved before applying the proposed algorithms in real-world applications. First, usually real-world applications have a continuous state space. Therefore, future research should look at using the proposed algorithms with function approximation. This extension seems to be straightforward in the model-free algorithms.

In contrast, using function approximation in the partially model-based algorithms would be problematic. The reason for this is that the partially model-based algorithms rely on predicting the next value of the autonomous state. In function approximation, the whole state space is partitioned into certain binary features, and it is not clear how a conventional prediction method could predict these binary features based on the experiments.

Second, in real-world IDSD problems, the autonomous dynamics do not, in general, have the Markov properties. For instance, in the real-world hybrid vehicle problem, driver acceleration is determined by factors such as the driver's goal, the road's condition, the time at which the driver is traveling, various unexpected events, and so on. Therefore, assuming that driver acceleration is a function of its previous value is not realistic. Thus, future research should move beyond looking at IDSD problems with Markov properties and consider IDSD problems with Markovian controllable dynamics and non-Markovian autonomous dynamics. We believe that the model-free algorithm is a good start for working on these types of problems, since at any time-step only the successive autonomous state participates in the credit assignment.

# Bibliography

Baird, L. C. (1993). Advantage updating. *Advantage updating. Tech. rep. WL-TR-93-1146, Wright-Patterson Air Force Base.*

Bertsekas, D. P. (Ed.). (1987). *Dynamic programming: Deterministic and stochastic models.* NJ: Prentice-Hall.

Boutilier, C. (1997). Correlated action effects in decision theoretic regression. *Uncertainty in Artificial Intelligence* (pp. 30–37).

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research, 11*, 1–94.

Boutilier, C., & Dearden, R. (1996). Approximating value trees in structured dynamic programming. *Proceedings of the Thirteenth International Conference on Machine Learning.*

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1104–1111). San Francisco: Morgan Kaufmann.

Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence, 121*, 49–107.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence, 5*, 142–150.

Harmon, M. E., Baird III, L. C., & Klopf, A. H. (1995). Advantage updating applied to a differrential game. *Advances in Neural Information Processing Systems* (pp. 353–360). The MIT Press.

Kakade, S. M. (2003). *On the sample complexity of reinforcement learning.* Doctoral dissertation, University College London, UK.

Kearns, M. J., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *IJCAI* (pp. 740–747).

Kearns, M. J., Mansour, Y., & Ng, A. Y. (1999). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *IJCAI* (pp. 1324–1231).

Koller, D., & Parr, R. (1999). Computing factored value functions for policies in structured MDPs. *IJCAI* (pp. 1332–1339).

Koller, D., & Parr, R. (2000). Policy iteration for factored MDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (pp. 326–334).

LaValle, S. M. (2006). *Planning algorithms.* Cambridge University Press.

Moriarty, D. E., Schultz, A. C., & Grefenstette, J. J. (1999). Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research, 11*, 199–229.

Patrascu, R., Poupart, P., Schuurmans, D., Boutilier, C., & Guestrin, C. (2002). Greedy linear valueapproximation for factored markov decision processes. *In Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)* (pp. 285–291).

Poole, D. (1995). Exploiting the rule structure for decision making within the independent choice logic. *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence* (pp. 454–463).

Poupart, P., Boutilier, C., Patrascu, R., & Schuurmans, D. (2002). Piecewise linear value function approximation for factored MDPs. *In Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)* (pp. 292–299).

Schuurmans, D., & Patrascu, R. (2001). Direct value-approximation for factored MDPs. *Proceedings of NIPS-14* (pp. 1579–1586).

Singh, S., & Yee, R. C. (1994). An upper bound on the loss from approximate optimal-value functions. *Machine Learning, 16:227.*

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Massachusetts: MIT Press, Cambridge.

Teichteil-Knigsbuch, F., & Fabiani, P. (2004). Probabilistic reachability analysis for structured markov decision processes. *The Probabilistic Planning track of the International Planning Competition.*

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*, 58–68.