# Reinforcement Learning- Results in Approximate Dynamic Programming and Stochastic Approximation

A Thesis

Submitted for the Degree of

**Doctor of Philosophy**

in the **Faculty of Engineering**

by

**Chandrashekar Lakshminarayanan**


Computer Science and Automation
INDIAN INSTITUTE OF SCIENCE
BANGALORE – 560 012, INDIA

JUNE 2015

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Markov Decision Processes

Markov Decision Process (MDP) is an useful framework to cast problems involving optimal sequential decision making under uncertainity. Such problems often occur in science, engineering and economics, with planning by autonomous agents, control of large queuing systems and inventory control being some specific examples. Informally, the decision maker/planner wishes to maximize its objective, and in order to achieve this end, needs to perform an appropriate *action/control* at each decision epoch based on the *state* of the underlying *system*. The sequence of actions/control is also known as *policy* and the planner is interested in computing the *optimal* policy. In most cases, in order to compute the optimal policy, the planner also needs to *predict* the value of each state of any given control. Thus, with any given MDP, there are two natural sub-problems namely the problem of prediction and the problem of control.

Dynamic Programming (DP) is a general principle to solve MDPs, the core of which constitutes the *Bellman* equation (BE). The Bellman equation relates the optimal policy to its corresponding value known as the optimal *value function*, i.e., it relates the prediction and control problems of a given MDP. A host of analytical and iterative methods exists to solve the Bellman equation for MDPs. These methods are known as exact dynamic programming methods since they spell out the optimal control and its value

function exactly.

The three important exact DP methods are

- Value Iteration.
- Policy Iteration.
- Linear Programming.

Of these methods, value iteration and linear programming are termed as value function based DP methods since they compute the optimal value function first and the optimal policy is obtained by substituting it in the Bellman Equation. On the other hand, policy iteration predicts the value of a policy at each iterative step and then improves it to yeild a better policy in the next step eventually converging to the optimal policy. It is important to emphasize that these exact DP methods solve both the control and prediction problems.

## 1.2 Practical Issues

MDPs arising in practical problems are faced with two important issues. First issue is due to the *curse-of-dimensionality* (or simply the *curse*), a term which signifies the fact that the number of states of the MDP grows exponentially in the number of state variables. As the number of state variables increase, it becomes difficult to solve the MDP employing exact DP methods due to the computational overhead involved. Moreover, it also difficult to store and retrieve the exact value function and policy when the number of states are large.

The second issue is the lack of model information, wherein, the the model parameters of the MDP are not available explicitly. However, the the underlying system can be simulator or sample trajectories via direct interaction with the system. This scenario is known as the *reinforcement learning* setting since the model parameters have to be *learned* by using the feedback obtained via direct interaction with the environment.

A wide range of methods/algorithms exist in literature to address the issues of the curse and the lack of model information. In particular, the methods that tackle the curse

are broadly termed as approximate dynamic programming methods and methods that handle the case of lack of model information are called reinforcement learning algorithms.

## 1.3 Approximate Dynamic Programming

Approximate Dynamic Programming (ADP) is the name given to a class of approximate solution methods that tackle the curse. A major bottleneck the ADP methods face in the case of MDPs having a large number of states is that the value corresponding to each and every state cannot be stored, retrieved and computed. Most often ADP methods adopt an approximation architecture that enables compact representation of the value function in higher dimension so as to ease the storage and computation. Once the approximation architecture is fixed it is also important to devise a scheme that will choose the right candidate function that approximates the value function well. Thus central to any ADP method are

1. Approximation architecture.
2. Scheme to compute the right function within the chosen approximation architecture.

In most cases, ADP methods are obtained by combining exact DP methods with a given approximation architecture. A given ADP scheme can be said to belong to either of two approaches based on the way the prediction and the control problems are addressed. The two distinct approaches are:

1. The *value function* based approach, wherein a direct approximation to the optimal value function is obtained and a *greedy/sub-optimal* policy is computed by substituing the approximate value function in the Bellman equation.
2. The *approximate policy iteration* based approach, wherein, the critic evaluates the current policy, i.e., computes an approximation of the value function of the current policy. The actor on the other hand, makes use of the approximate value function to imporve the current policy.

Further, the performance of a given ADP method can be quantified by the following metrics namely

- Prediction Error, i.e., the difference between the exact value function and the approximate value function.
- Control Error, i.e., the loss in performance due to the sub-optimal policy as compared to the optimal policy.

It is a known result that if the prediction error is bounded in the max-norm (or $L_\infty$-norm), the control error can be bounded as well. The necessacity of $L_\infty$-norm arises due to the fact that the Bellman operator is only a contraction map in the $L_\infty$ norm.

An ADP method is said to address both the prediction and control problems if it can offer guaranteed performance levels measured in terms of the aforementioned metrics. Given an ADP method, it a major theoretical challenge to come up with an analytical expression to bound the aforementioned performance metrics and it is in particlar difficult to bound the control error in many ADP schemes.

## 1.4 Linear Function Approximation

The first step in any ADP method is the choice of approximation architecture. In particular, a compact way of representing an approximation for the value function is necessary in most cases. A paramtererized function class is an useful approximation architecture since every function can be specified by the parameter it is enough to store only the parameters. Computing the right function that best approximates the exact value function then boils down to computations involving only the parameters and the curse can be tackled by choosing the number of parameters to be much lesser than the number of states. The most widely used parameterized class is the linear function approximator (LFA). Under a LFA, each function is written as a linear combination of pre-selected *basis* functions. The exact value function is then approximated by computing/learning the weights of the various basis functions in the linear combination.

The method used to compute the right weights of the linear combination affects the quality of the approximation. The projected Bellman equation and the approximate linear programming are two different ways of computing the weights. These two differing approaches have their advantages/disadvantages and provide interesting research problems.

### 1.4.1 Projected Bellman Equation

Once the basis functions of the LFA have been fixed, the right candidate function from the LFA has to be picked as the approximate value function. A candidate for the approximate value function could be that function which is at the least distance from the exact value function. Minimizing the distance between the linear combination of basis functions and the exact value function is similar to the idea of conventional *linear regression* wherein, the target function is *projected* onto the linear sub-space spanned by the basis functions. However, the idea of linear regression cannot be applied in a straightforward manner to compute the approximate value function because the exact value function (which is the target function in this case) is not known.

The Projected Bellman Equation (PBE) combines the idea of linear least squares projection and the Bellman equation. The central idea underlying the PBE is to find a fixed point in the linear sub-space spanned by the basis functions. However, existence of such a fixed point can be ensured only under a restricted setting. In particular, the Bellam operator is *non-linear* and is a contraction map in the max-norm ($L_\infty$-norm). On the other hand, the linear least squares projection operator is non-expansive only in the $L_2$-norm. Hence the Bellman operator cannot be combined as such with the least squares projection operator. Neverthless, this issue can be side stepped by considering the Bellman operator restricted to a given policy which can be shown to be a contraction map in a generalized $L_2$-norm. The Bellman operator restricted to a given policy can be combined with the linear least squares projection operator to find a fixed point. However, such a fixed point can approximate only the value function of the particular policy (with respect to which the Bellman operator has been restricted) and not the optimal value

function. As a consequence, the PBE can only be used for approximate policy evaluation, i.e., it can be used to compute an approximation to the value function of a given policy. This means that the PBE based methods fall into the category of approximate policy iteration approach, i.e., the approximate policy evaluation should be followed by a policy improvement step. However, there is a 'norm-mismatch' between the projection operator that minimizes the error in $L_2$-norm and the $\max/L_\infty$-norm required to guarantee policy improvement. As a result, the sub-optimal policy obtained from the approximate value function computed by the PBE is not guaranteed to be an improvement.

The major disadvantage of the PBE based methods is that they do not address the control problem completely. In fact, there are simple MDPs for which the PBE based solution methods are known to produce a sequence of policies that oscillate within a set of bad policies. This phenomenon is called as *policy-chattering* and is a direct consequence of the norm mismatch.

**Problem 1: Projected Bellman Equation in the $(\min, +)$ linear basis**

It is known that **?** the issue of norm-mismatch can be alleviated if the projection operator preserves *monotonicity*, i.e., if given two functions with one of them greater (component-wise) that the other, then the same holds for their projections as well. It is also known that the projection operator arising in the $(\min, +)$ linear algebra preserves this monotonicity property. The first part of the thesis deals with developing convergent ADP methods based on $(\min, +)$ linear algebra. In particular, the monotonicity property helps in

- Building convergent value function based ADP methods. Since the optimal value function is approximated directly, the issue of policy-chattering is absent.
- Providing performance guarantees for the greedy/sub-optimal policy. This is possible since there is no issue of 'norm-mismatch'.

### 1.4.2 Approximate Linear Programming

The approximate linear programming (ALP) formulation is obtained by introducing linear function approximation in the linear programming formulation (LP) of the given MDP. In

contrast to the PBE, the ALP does not rely on the linear least squares projection operator, but instead, computes the approximate value function by optimizing a linear objective over a set of linear inequality constraints. In particular, the ALP restricts its search to a space of linear functions that upper bound the optimal value function. The ALP is a value function method as it computes an approximation to the optimal value function. Since, the corresponding greedy policy can be derived in a straightforward manner, the ALP does not suffer from issues of policy-chattering. Further, the ALP also offers good performance guarantees for both the prediction as well as the control problems, and thus addresses both the problems.

A significant shortcoming of the ALP formulation is that the number of constraints is of the order of the state space. Most MDPs arising in practice have a large number of constraints and hence it is always not possible to solve the ALP with all the constraints. However, there are some special cases of factored MDPs with factored value function representations that enable elimination of variables to come up with a tractable number of constraints. A general approach is to handle the large number of constraints is constraint sampling, wherein a Reduced Linear Program (RLP) is formulated by sampling fewer number of constraints from the orginial constraints of the ALP. Though the RLP is known to perform well in experiments, the theoretical analysis is available only for a specific RLP formulated under idealized settings.

**Problem 2: Framework to analyse constraint reduction in ALP:**

The second part of the thesis deals with developing a novel theoretical framework to analyse constraint reduction/approximation in the ALP. The framework is built around studying a Generalized Reduced Linear Program (GRLP) whose constraints are obtained as positive linear combinations of the original constraints of the ALP. The salient features of the framework are

1. The analysis is based on two novel contraction maps and the error bounds are provided in a modified $L_\infty$-norm. Both the prediction and control error are bounded, thus making the GRLP a complete ADP scheme.

2. Justification of linear function approximation of the Lagrange multipliers associated

with the constraints of the ALP. This is a desirable outcome, since both the primal and dual variables have linear function approximation.

## 1.5 Reinforcement Learning

Reinforcement Learning (RL) algorithms can be viewed as 'on-line'/sample trajectory based solution methods for solving MDPs. In the case of MDPs with a large number of states, RL algorithms are obtained as on-line versions of ADP methods. In order to handle the noise in the sample trajectory RL algorithms make use of stochastic approximation (SA). Typically, RL algorithm employing stochastic approximation are iterative schemes which take a small *step* towards the desired value at each iteration. By making the right choice of the *step-size* schedule effect of noise can be nullified and convergence to the desired value can be guaranteed.

Actor-Critic algorithms form an important sub-class of RL algorithms, wherein, the critic is responsible for policy evalutaion and the actor is responsible for policy improvement. In order to tackle the curse, the actor-critic schemes parameterize the policy and the value function. In an actor-critic algorithm, the critic forms the inner-loop and the actor constitutes the outer-loop. This is due to the fact that the actor has to needs the critic to evaluate a given policy before the actor updates the policy parameters. This effect of having two seperate loops can instead by achieved in practice by adopting different *step-size* schedules for the actor and the critic. Specifically, the step-sizes used by the actor updates have to be much smaller than those used in the critic updates.

Stochastic approximation schemes that use different step-size schedules for different sets of iterates are known as multi timescale stochastic approximation schemes. The conditions under which the iterates of a multi timescale SA schemes converge to the desired value have been studied in literature. One of the conditions required to ensure the convergence of the iterates of a multi timescale SA scheme is that the iterates need to be stable, i.e., they should be bounded. However, the conditions that *imply* the stability of the iterates in a multi timescale SA scheme have not been understood.

**Problem 3: Stability Criterion for Two Timescale Stochastic Approximation Schemes:**

The third part of the thesis deals with providing conditions under which the stability of iterates in a two timescale stochastic approximation scheme follows. Salient features of the contribution are as follows:

1. The analysis is based on the ODE approach to stochastic approximation.

2. Stability of iterates of important actor-critic algorithms follow for the result.

# Chapter 2

# Markov Decision Processes (MDPs)

## 2.1 Definition

An MDP is a 4-tuple $\langle S, A, P, g \rangle$, where $S$ is the state space, $A$ is the action space, $P$ is the probability transition kernel and $g$ is the reward function. The theoretical results developed in this thesis assume the setting as under.

**Setting and Notation**:

1. The MDP has finite number of states and actions.
2. Without the loss of generality, the state space is given by $S = \{1, 2, \ldots, n\}$ and the action space is given by $A = \{1, 2, \ldots, d\}$.
3. It is also assumed that all the actions are feasible in all the states.
4. The probability transition kernel $P$ specifies the probability $p_a(s, s')$ of transitioning from state $s$ to state $s'$ under the action $a$.
5. The reward obtained for performing action $a \in A$ in state $s \in S$ is denoted by $g_a(s)$.

## 2.2 Inifinte Horizon Discounted Reward

By a policy, we mean a sequence $\pi = \{\pi_0, \ldots, \pi_n, \ldots\}$ of functions $\pi_i, i \geq 0$ that describe the manner in which an action is picked in a given state at time $i$. The most general class of policies is the class of *history-dependent randomized policies* denoted by $\pi$ and is defined below.

**Definition 1** *Let* $h_n = \{s_0, \ldots, s_n, \pi_0, \ldots, \pi_{n-1}\}$ *denote the* history. *A policy* $\pi$ *is said to be a* history-dependent randomized *policy if* $\pi_n, n \geq 0$ *is such that for every* $s \in S$, $\pi_n(s, \cdot|h_n)$ *is a probability distribution over the set of actions.*

We denote the class of policies by $\Pi$. The infinite horizon discounted reward corresponding to state $s$ under a policy $\pi$ is denoted by $J_\pi(s)$ and is defined by

$$J_\pi(s) \stackrel{\Delta}{=} \mathbf{E}[\sum_{n=0}^{\infty} \alpha^n g_{a_n}(s_n)|\pi],$$

where $a_n \sim \pi_n(s, \cdot)$ and $\alpha \in (0, 1)$ is a given discount factor. Here $J_\pi(s)$ is known as the value of the state $s$ under the policy $\pi$, and the vector quantity $J_\pi \stackrel{\Delta}{=} (J_\pi(s))_{s \in S} \in R^n$ is called the value-function corresponding to the policy $\pi$. The optimal value function $J^*$ is defined as $J^*(s) \stackrel{def}{=} \max_{\pi \in \Pi} J_\pi(s)$.

A stationary deterministic policy (SDP) is one where $\pi_n \equiv u$ for all $n \geq 0$ for some $u \colon S \to A$. By abuse of notation we denote the SDP by $u$ itself instead of $\pi$. In the setting that we consider, one can find an SDP that is optimal **??**, i.e., there exists an SDP $u^*$ such that $J_{u^*}(s) = J^*(s), \forall s \in S$. In this paper, we restrict our focus to the class $U$ of SDPs and without loss of generality we assume that there exists a unique SDP $u^*$ that is optimal. Under a stationary policy $u$ (or $\pi$), the MDP is a Markov chain and we denote its probability transition kernel by $P_u = (p_{u(i)}(i, j), i, j = 1, \ldots, n)$ (or $P_\pi = (p_{\pi(i)}(i, j), i, j = 1, \ldots, n)$, where $p_{\pi(i)}(i, j) = \sum_{a \in A} \pi(i, a) p_a(i, j)$ and $\pi(i) = (\pi(i, a), a \in A)$).

## 2.3 Bellman Equation

Given an MDP, our aim is to find the optimal value function $J^*$ and the optimal policy SDP $u^*$. The optimal policy and value function obey the Bellman equation (BE): for all $s \in S$,

$$J^*(s) = \max_{a \in A} \big(g_a(s) + \alpha \sum_{s'} p_a(s, s') J^*(s')\big), \tag{2.1a}$$

$$u^*(s) = \arg\max_{a \in A} \big(g_a(s) + \alpha \sum_{s'} p_a(s, s') J^*(s')\big). \tag{2.1b}$$

If $J^*$ is computed first, then $u^*$ can be obtained by substituting $J^*$ in (2.1b). The Bellman operator $T$ is defined using the model parameters of the MDP as follows:

**Definition 2** *The Bellman operator $T : \mathrm{R}^n \to \mathrm{R}^n$ is defined by*

$$(TJ)(s) = \max_{a \in A} \left(g_a(s) + \alpha \sum_{s'} p_a(s, s') J(s')\right), \tag{2.2}$$

*where $J \in \mathrm{R}^n$ and $J_s = J(s)$, $s \in S = \{1, \ldots, n\}$. Similarly one can define the Bellman operator $T_u : \mathrm{R}^n \to \mathrm{R}^n$ restricted to an SDP $u$ as follows:*

$$(T_u J)(s) = g_{u(s)}(s) + \alpha \sum_{s'} p_{u(s)}(s, s') J(s')\,. \tag{2.3}$$

Given $J \in \mathrm{R}^n$, $TJ$ is the 'one-step' greedy value function. It is also useful to define the notion of a one-step greedy policy as below:

**Definition 3** *A policy $\tilde{u}$ is said to be greedy with respect to $\tilde{J}$ if*

$$\tilde{u}(s) = \arg\max_{a \in A} \big(g_a(s) + \alpha \sum_{s'} p_a(s, s') \tilde{J}(s')\big) \tag{2.4}$$

We also define the Bellman operator $H$ for action values **?**:

**Definition 4** *Let $H : \mathrm{R}^n \to \mathrm{R}^{nd}$ be defined as follows: For $J \in \mathrm{R}^n$,*

$$HJ = \begin{bmatrix} H_1 J \\ \vdots \\ H_d J \end{bmatrix} \in \mathrm{R}^{nd}, \textit{where}$$

$$(H_a J)(s) = g_a(s) + \alpha \sum_{s'} p_a(s, s') J(s'), \qquad s \in S, a \in A. \tag{2.5}$$

We now state without proof the important properties related to the Bellman operator. Though in these results, we make use of the Bellman operator $T$, the results also trivially hold for $T_u$ as well.

## 2.4 Bellman Operator

**Lemma 5** *$T$ is a* max*-norm contraction operator, i.e., given $J_1, J_2 \in \mathbf{R}^n$,*

$$||TJ_1 - TJ_2||_\infty \leq \alpha ||J_1 - J_2||_\infty. \tag{2.6}$$

**Corollary 1** *$J^*$ is a unique fixed point of $T$, i.e., $J^* = TJ^*$.*

The Bellman operator $T$ exhibits two more important properties presented in the following lemmas (see **?** for proofs):

**Lemma 6** *$T$ is a monotone map, i.e., given $J_1, J_2 \in \mathrm{R}^n$ such that $J_2 \geq J_1$, we have $TJ_2 \geq TJ_1$. Further, if $J \in \mathrm{R}^n$ is such that $J \geq TJ$, it follows that $J \geq J^*$.*

**Lemma 7** *Given $J \in \mathrm{R}^n$, $t \in \mathrm{R}$ and $\mathbf{1} \in \mathrm{R}^n$, a vector with all entries 1, we have*

$$T(J + t\mathbf{1}) = TJ + \alpha t \mathbf{1}. \tag{2.7}$$

Note that Lemmas 5-7 also hold for the $Q$-Bellman operator $H$. Note that Lemmas 5-7 also hold for the Bellman operator $H$ defined for the action values in Definition 4.

Solving an MDP involves handling two sub-problems namely the problem of *control* and the problem of *prediction*. The problem of *control* deals with coming up with a good

(and if possible the optimal) policy. Often, in order to solve the problem of *control*, one needs to solve the problem of *prediction*, which deals with computing the value function $J_u$ of the policy $u$. The fact that the two problems are related is reflected in the Bellman equation in (2.1), where $J^*$ from (2.1a) is used in (2.1b) to obtain the optimal policy $u^*$. Thus, the Bellman equation is at the heart of the solution methods to MDPs. Any solution method to MDP is said to be complete only if it satisfactorily (with provable performance guarantees) addresses both the prediction and the control problems.

The basic solution methods namely value iteration, policy iteration and linear programming (LP) formulation **?** solve both the control and prediction problems. Of the three basic methods, in this paper, we are interested in the LP formulation given by

$$\begin{aligned} &\min_{J \in \mathrm{R}^n} c^\top J \\ &\quad \text{s.t } J(s) \geq g_a(s) + \alpha \sum_{s'} p_a(s,s') J(s'), \qquad s \in S, a \in A, \end{aligned} \tag{2.8}$$

where $c \in \mathrm{R}^n_+$ is any vector whose components are all non-negative. One can show that $J^*$ is the solution to the LP formulation (2.9) **?**. Also, of the three methods, value iteration and LP formulation are value function based methods, i.e., they compute $J^*$ directly and then $u^*$ is obtained by plugging $J^*$ in (2.1b).

While the basic methods (i.e., VI, PI and LP) can be used to compute exact values $J^*$ and $u^*$ for MDPs with a small number of states, they are computationally expensive in the case of MDPs with a large number of states.

## 2.5 Basic Solution Methods

**Algorithm 1** Value Iteration (VI)

1: Initialize $J_0$.
2: **for** $n = 0, 1, 2, \ldots$ **do**
3: $\quad J_{n+1} = TJ_n$.
4: **end for**

One can show that the iterates $J_n$ in Algorithm 1 converge to $J^*$, i.e., $J_n \to J^*$ as $n \to \infty$.

**Algorithm 2** Policy Iteration (PI)

1: Initialize a policy $u_0$.
2: **for** $n = 0, 1, 2, \ldots$ **do**
3: Policy Evaluation Step: Compute $J_{u_n}$ such that $J_{u_n} = T_{u_n} J_{u_n}$.
4: Policy Improvement Step: $u_{n+1}(s) = \arg\max_{a \in A}(g_a(s) + \sum_{s'} p_a(s, s') J_{u_n}(s'))$.
5: **end for**

The policy iteration algorithm (Algorithm 2) has two important steps namely *evaluation/prediction* and *improvement/control*. PI is different from VI and LP in that it computes a sequence of policies $u_n$ and their corresponding value functions $J_{u_n}$. One can show that in Algorithm 2 as $n \to \infty$ $u_n \to u^*$ and $J_{u_{n+1}} \geq J_{u_n}, \forall n$.
The linear programming (LP) formulation can be obtained by unfurling the max operator in the Bellman equation into a set of linear inequalities and is given by:

$$\begin{aligned} &\min_{J \in \mathrm{R}^n} c^\top J \\ &\text{s.t } J(s) \geq g_a(s) + \alpha \sum_{s'} p_a(s, s') J(s'), \ \forall s \in S, a \in A, \end{aligned} \tag{2.9}$$

where $c \in \mathrm{R}^n_+$ is any vector whose components are all non-negative. One can show that $J^*$ is the solution to the LP formulation (2.9) **?**. Also, note that the value iteration and linear programming formulations are value function based methods, i.e., they compute $J^*$ directly and then $u^*$ is obtained by plugging $J^*$ in (2.1b).
While the basic methods (i.e., VI, PI and LP) can be used to compute exact values $J^*$ and $u^*$ for MDPs with a small number of states, they are computationally expensive in the case of MDPs with a large number of states.

## 2.6 Curse of Dimentionality

*Curse-of-Dimensionality* is a term used to denote the fact that the number of states grows exponentially in the number of state variables. Most MDPs arising in practical applications suffer from the curse, i.e., have large number of states and it is difficult to compute $J^*/J_u \in \mathrm{R}^n$ exactly in such scenarios. Approximate dynamic programming (ADP) **????** methods compute an approximate value function $\tilde{J}$ instead of $J^*/J_u$. In order to tackle the curse, ADP methods resort to dimensionality reduction by parameterizing the value function and/or the policy. Value-function based ADP methods form an important subclass of ADP methods whose brief overview is given below.

## 2.7 Value Function Based Approximate Dynamic Programming

In the value-function based ADP schemes a parameterized family is chosen and the approximate value-function $\tilde{J}$ is picked from the chosen parameterized class. Once the approximate value function $\tilde{J}$ is computed, a sub-optimal policy $\tilde{u}$ can be obtained as the one-step greedy policy with respect to $\tilde{J}$ by making use of (2.4). The following lemma characterizes the degree of sub-optimality of the greedy policy $\tilde{u}$.

**Lemma 8** *Let $\tilde{J}$ be the approximate value function and $\tilde{u}$ be as in* (2.4)*, then*

$$||J^* - J_{\tilde{u}}||_\infty \leq \frac{2}{1-\alpha}||J^* - \tilde{J}||_\infty. \tag{2.10}$$

**Proof:** We know that

$$(T_{\tilde{u}})\tilde{J}(s) = g(s) + \alpha \sum_{s'} p_{\tilde{u}(s)}(s, s')\tilde{J}(s'), \tag{2.11}$$

$$J_{\tilde{u}}(s) = g(s) + \alpha \sum_{s'} p_{\tilde{u}(s)}(s, s')J_{\tilde{u}}(s'). \tag{2.12}$$

Hence we can write by subtracting (2.11) from (2.12)

$$\begin{aligned} J_{\tilde{u}} - \tilde{J} &= T_{\tilde{u}}\tilde{J} - \tilde{J} + \alpha P_{\tilde{u}}(J_{\tilde{u}} - \tilde{J}), \text{ or,} \\ J_{\tilde{u}} - \tilde{J} &= (I - \alpha P_{\tilde{u}})^{-1}(T_{\tilde{u}}\tilde{J} - \tilde{J}), \text{ hence} \\ ||J_{\tilde{u}} - \tilde{J}||_\infty &\leq \frac{1}{1-\alpha}||T_{\tilde{u}}\tilde{J} - \tilde{J}||_\infty. \end{aligned}$$

We know from (2.4) that $T_{\tilde{u}}\tilde{J} = T\tilde{J}$. Also from the fact that $J^* = TJ^*$ and the contraction property of $T$, we know $||T\tilde{J} - J^*||_\infty \leq \alpha||\tilde{J} - J^*||_\infty$ and $||T_{\tilde{u}}\tilde{J} - \tilde{J}||_\infty \leq (1+\alpha)||\tilde{J} - J^*||_\infty$. Hence we have

$$\begin{aligned} ||J_{\tilde{u}} - J^*||_\infty &= ||J_{\tilde{u}} - J^* + \tilde{J} - \tilde{J}||_\infty \\ &\leq ||J_{\tilde{u}} - \tilde{J}||_\infty + ||\tilde{J} - J^*||_\infty \\ &\leq \frac{1}{1-\alpha}||T_{\tilde{u}}\tilde{J} - \tilde{J}||_\infty + ||J^* - \tilde{J}||_\infty \\ &\leq \frac{1+\alpha}{1-\alpha}||J^* - \tilde{J}||_\infty + ||J^* - \tilde{J}||_\infty \\ &\leq \frac{2}{1-\alpha}||J^* - \tilde{J}||_\infty. \end{aligned}$$

The quality of any ADP method depends on the approximation guarantees it offers for the quantities $||J^* - \tilde{J}||$ and $||J^* - J_{\tilde{u}}||$, where $||\cdot||$ is an appropriate norm. The term $||J^* - \tilde{J}||$ denotes the error in prediction and $||J^* - J_{\tilde{u}}||$ denotes the loss in performance resulting from the sub-optimal policy $\tilde{u}$ with respect to the optimal policy $u^*$. Of the two error terms, $||J^* - J_{\tilde{u}}||$ is more important because ultimately we are interested in coming up with a useful policy. In the context of ADP methods, the control and prediction problems are said to be addressed when the error terms $||J^* - \tilde{J}||$ and $||J^* - J_{\tilde{u}}||$ are bounded by "small" constants.

## 2.8 Linear Function Approximation

The most widely used parameterized class to approximate the value-function is the linear function approximator (LFA). Under LFA, the value-function is approximated as $\tilde{J} = \Phi r^*$, with $\Phi = [\phi_1 | \dots | \phi_k]$ being an $n \times k$ feature matrix and $r^*$ is the parameter to be learnt. There are two important approaches to value function approximation. Both the approaches start out with a basic solution method and appropriately introduce function approximation in it. The two approaches are:

1. The Projected Bellman Equation (PBE) which combines the BE and the linear least squares projection operator to project high dimensional quantities onto the subspace of the LFA.

2. The approximate linear programming formulation which introduces the LFA in the linear programming formulation.

# Chapter 3

# Approximate Dynamic Programming in $(\min, +)$ linear basis

A host of ADP methods are based on the PBE and have been found to be useful in practice. The main application of the PBE is for approximate policy evaluation, i.e., to compute $\tilde{J}_u$, an approximation to the value function $J_u$ of policy $u$. Due to the mismatch in the norms, i.e, the linear least squares projection operator based on the $L_2$-norm and the $L_\infty$-norm of the Bellman operator $T$, one cannot use the PBE to obtain a direct approximation to $J^*$. Thus in order to solve the problem of control, $\tilde{J}_u$ is used to compute a one-step greedy policy. However, again due to the mismatch in the norms, i.e., $L_2$-norm of the linear least squares projection and the $L_\infty$ norm required for policy improvement (Lemma 8), the one-step greedy policy need not necessarily be an improvement. This leads to a phenomenon called *policy-chattering* **?** where looping within of bad policies can occur. Further, such policy-chattering can be demonstrated in simple examples as well **?**. Thus, though the approximate value function obtained by solving the PBE offers guarantees for prediction it does not offer any guarantees for the control problem, a significant shortcoming of the PBE based methods.

## 3.1 Projected Bellman Equation

A large class of ADP methods compute $r^*$ to be the solution to the projected Bellman equation (PBE) given below:

$$\Phi r^* = \Phi(\Phi^\top D_u \Phi)^{-1}\Phi^\top T_u \Phi r^*, \tag{3.1}$$

where $T_u$ is the Bellman operator corresponding to an SDP $u$ and $D_u$ is a $n \times n$ diagonal matrix whose diagonal entries are the stationary probabilities of the states under the SDP $u$. The PBE can be written in short as

$$\Phi r^* = \Pi T_u \Phi r^*, \tag{3.2}$$

where $\Pi = \Phi(\Phi^\top D_u \Phi)^{-1}\Phi^\top$ is the projection operator. Notice that approximate value function $\tilde{J}_c = \Phi r^*$ obtained by solving the PBE (3.2) is an approximation only to $J_u$ and not $J^*$. This scenario is unavoidable because $T_u$ cannot be replaced by $T$ in (3.2), since $\Pi T$ cannot be guaranteed to be a contraction map. As a result, an equation of the form $\Phi r = \Pi T \Phi r$ need not have a solution. Thus, the PBE can only be used for approximate policy evaluation, i.e., to compute $\tilde{J}_u \approx J_u$. Nevertheless, the solution to the PBE offers error bounds for the prediction problem and is given below:

$$||J_u - \Phi r^*||_D \leq \frac{1}{\sqrt{1-\alpha^2}}||J_u - \Pi J_u||_D. \tag{3.3}$$

In order to address the problem of control, one resorts to approximate policy improvement (API) wherein a one-step look ahead policy is derived. The API scheme is given in Algorithm 3

**Algorithm 3** Approximate Policy Iteration (API)

1: Start with any policy $u_0$
2: **for** $i = 0, 1, \ldots, n$ **do**
3: Approximate policy evaluation $\tilde{J}_i = \Phi r_i^*$, where $\Phi r_i^* = \Pi T_{u_i} \Phi r_i^*$.
4: Improve policy $u_{i+1}(s) = \arg\max_a (g_a(s) + \alpha \sum_{s'} p_a(s, s') \tilde{J}_i(s'))$.
5: **end for**
6: **return** $u_n$

The performance guarantee of API can be stated as follows:

**Lemma 9** *If at each step $i$ one can guarantee that $||\tilde{J}_i - J_{u_i}||_\infty \leq \delta$, then one can show that $\lim_{n\to\infty} ||J_{u_i} - J^*|| \leq \frac{2\delta\alpha}{(1-\alpha)^2}$.*

Note that the error bound required by Lemma 9 is in the $L_\infty$ norm, whereas (3.3) is only in the $L_2$ norm. So API cannot guarantee an approximate policy improvement at each step which is a shortcoming. Also, even-though each evaluation step (line 3 of Algorithm 3) converges, the sequence $u_n, n \geq 0$ is not guaranteed to converge. This is known as policy *chattering*. Thus the problem of *control* is only partially addressed by API, i.e, suffers in both convergence and performance guarantees.

In short, whilst the approximate value function obtained by the solving the PBE offers guarantees for prediction it does not offer any guarantees for the control problem.

In the next section, we discuss the approximate linear programming (ALP) formulation, the basic results and present prior results in literature as well as motivate the problem that we address in this paper.

## 3.2 $(\min, +)$ linear functions

The $\mathbf{R}_{\min}$ semiring is obtained by replacing multiplication ($\times$) by $+$, and addition ($+$) by min.

**Definition 10**

$$\begin{aligned} &\textit{Addition:} && x \oplus y = \min(x, y). \\ &\textit{Multiplication:} && x \otimes y = x + y. \end{aligned}$$

Henceforth we use, $(+, \times)$ and $(\oplus, \otimes)$ to respectively denote the conventional and $\mathbf{R}_{\min}$ addition and multiplication respectively. A Semimodule over a semiring can be defined in a similar manner to vector spaces over fields. In particular we are interested in the semimodule $\mathcal{M} = \mathbf{R}_{\min}^n$ (since $J^* \in \mathbf{R}_{\min}^n$). Given $u, v \in \mathbf{R}_{\min}^n$, and $\lambda \in \mathbf{R}_{\min}$, we define addition and scalar multiplication as follows:

**Definition 11**

$$\begin{aligned} (u \oplus v)(i) &= \min\{u(i), v(i)\} = u(i) \oplus v(i), \forall i = 1, 2, \ldots, n. \\ (u \otimes \lambda)(i) &= u(i) \otimes \lambda = u(i) + \lambda, \forall i = 1, 2, \ldots, n. \end{aligned}$$

Similarly one can define the $\mathrm{R}_{\max}$ semiring which has the operators max as addition and $+$ as multiplication.

It is a well known fact that deterministic optimal control problems with cost/reward criterion are $(\min, +)/(\max, +)$ linear (**????** ). However, it is straightforward to show that the Bellman operator $T$ (as well as $H$) in (**??**) corresponding to infinite horizon discounted reward MDPs is neither $(\min, +)$ linear nor $(\max, +)$ linear. Nevertheless, the motivation behind developing ADP methods based on $(\min, +)$ LFAs is to explore them as an alternative to the conventional basis representation.

Given a set of basis functions $\{\phi_i, i = 1, \ldots, k\}$, we define the $(\min, +)$ linear span to be $\mathcal{V} = \{v | v = \Phi \otimes r \stackrel{def}{=} \min(\phi_1 + r(1), \ldots, \phi_k + r(k)), r \in \mathbf{R}_{\min}^k\}$. Thus $\mathcal{V}$ is a subsemimodule. In the context of value function approximation, we would want to project quantities in $\mathbf{R}_{\min}^n$ onto $\mathcal{V}$. The $(\min, +)$ projection operator $\Pi_M$ works as follows (**???**):

$$\Pi_M u = \min\{v | v \in \mathcal{V}, v \geq u\}, \forall u \in \mathcal{M}. \tag{3.4}$$

We can write the PBE in the $(\min, +)$ basis:

$$\begin{aligned} v &= \Pi_M T v, v \in \mathcal{V} \\ \Phi \otimes r^* &= \min\{\Phi \otimes r^* \in \mathcal{V} | \Phi \otimes r^* \geq T\Phi \otimes r^*\}. \end{aligned} \tag{3.5}$$

Thus by making use of $(\min, +)$ LFAs and the projection operator $\Pi_M$ we aim to find the minimum upper bound to $J^*/Q^*$.

## 3.3 Fenchel Dual and Projection on Subsemimodules

In this section, we demonstrate the connections between the *Fenchel-Legendre* transform (FLT) and the $(\min, +)$ projection defined in (3.4). Given a function $f\colon \mathrm{R}^n \to \mathrm{R}$, its FLT is defined by $f^*\colon \mathrm{R}^n \to \mathrm{R}$, with

$$f^*(y) = \sup_{x \in \mathrm{R}^n} (y^\top x - f(x)), y \in \mathrm{R}^n. \tag{3.6}$$

If $f$ is convex, then it can be recovered as $f = f^{*^*}$, i.e.,

$$f(x) = f^{*^*}(x) = \sup_{y \in \mathrm{R}^n} (x^\top y - f^*(y)), x \in \mathrm{R}^n. \tag{3.7}$$

We can rewrite (3.6) as below

$$f^*(y) = \sup_{x \in \mathrm{R}^n} (f_y(x) - f(x)), y \in \mathrm{R}^n, \text{ where } f_y(x) = y^\top x. \tag{3.8}$$

Now instead of considering functions $f_y(x)$ indexed by $y \in \mathrm{R}^n$, we consider the sequence $\{\phi_j\}, j \in \mathcal{J} = \{1, 2, \ldots, k\}, \phi_j\colon \mathrm{R}^n \to \mathrm{R}$. Then (3.8) can be modified as below:

$$f^*(j) = \sup_{x \in \mathrm{R}^n} (\phi_j(x) - f(x)), j \in \mathcal{J}. \tag{3.9}$$

We call (3.9), the sup-Transform or the max-Transform. It is easy to check that $\phi_j(x) - f^*(j) < f(x), \forall x \in \mathrm{R}^n, j \in \mathcal{J}$. Since our index set in (3.9) is finite (as opposed to $\mathrm{R}^n$