Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Xindi Gong April 12, 2022

Linear Programming Approach for Q-Learning

by

Xindi Gong

Lars Ruthotto
Adviser

Department of Mathematics

Lars Ruthotto

Adviser

Zhiyun Gong

Committee Member

Jeremy Jacobson

Committee Member

2022

Linear Programming Approach for Q-Learning

By

Xindi Gong

Lars Ruthotto

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2022

Abstract

Linear Programming Approach for Q-Learning
By Xindi Gong

Reinforcement learning has made into the headlines for its success in games such as Alpha Go and in scientific applications such as protein folding and automation. These successful stories have inspired a better mathematical understanding of reinforcement learning, which has become a very important and active area of research.

In a reinforcement learning system, the agent's goal is to learn an optimal policy that minimizes the total running costs of actions over all successive steps. Given enough time and training trials, the agent will eventually learn the optimal policy. But within a complex system, tracking the action sequence and the feedback of each action from the environment may be difficult and time-consuming. In order to learn the optimal policy without observing numerous training trials, we consider the reinforcement learning technique Q-Learning and investigate a convex formulation of a training problem.

In this thesis, we apply Q-Learning to a deterministic and discrete-time reinforcement learning problem with discrete state and action space. To obtain an algorithm that solves the problem numerically, we formulate the learning problem as a linear program based on the Bellman equation. Although, linear programming approach solves our problem successfully, we believe that linear programming approach has limitations in systems with large state and action space and randomness. Our approach will become infeasible very quickly when the number of states and actions grows. Despite the limitations, linear programming approach provides insights in solving reinforcement learning problems, and further studies on testing limitations and improving the approach for complex systems will be included in the future.

Linear Programming Approach for Q-Learning

By

Xindi Gong

Lars Ruthotto

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2022

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support and assistance. This project would not have been possible without the support of many people.

Many thanks to Emory University for providing resourceful courses and numerous research opportunities for undergraduate students, providing me with a broader view of mathematical research and scientific applications in real life.

I would like to thank all my committee members, Dr. Lars Ruthotto, Dr. Zhiyun Gong, and Dr. Jeremy Jacobson for their support and understanding over the past year. Many thanks to my adviser, Dr. Ruthotto, who answered all my questions and motivated me to sharpen my thinking with insightful feedback. Without his assistance and dedicated involvement in every step throughout the project, this thesis would have never been accomplished. Also, many thanks to Dr. Gong and Dr. Jacobson, who offered noble guidance and support with full encouragement.

Finally, I would also like to thank my parents who guided me in the right direction and gave me valuable advice. I am also especially grateful to my dearest friends Mildred Ouyang and Kaiyuan Li, who accompanied me through this long process and always encouraged and supported me when I needed it.

Contents

Chapter 1

Introduction

Reinforcement learning is a type of machine learning training that captures the learning agent's optimal behaviors in an uncertain and possibly complex environment to achieve the goal, obtaining either maximum rewards or minimum costs. This optimal behavior is learned through the agent's interactions over time with the environment and the sequence of decisions made based on its observations, which are very similar to children exploring the world and learning behaviors that help them to accomplish their goals.

In this game-like environment, the agent gets either rewards or penalties for the decisions of actions it makes. And with the aim of achieving maximum cumulative rewards or minimum costs in an uncertain environment, the agent learns from the trials based on the feedback of its previous experiences and actions [2]. Although the game designer, the algorithm, sets the reward or cost policy, which can be considered as the game rule, and gives the agent some hints on how to maximize the total rewards or minimize the costs, the decision making depends on how the agents perform the task, starting from random trials and finishing with specific sequences of actions. This reward and cost policy programs the agent to seek a long-term and overall goal, allowing the agent to learn to avoid the negative-reward actions and to seek the positive-reward actions.

Other types of machine learning includes supervised learning and unsupervised learning [2]. While both supervised learning and reinforcement learning map inputs and outputs, reinforcement learning uses rewards and penalties as signals for actions, rather than having the agent receive the correct set of actions when performing a task. Also, reinforcement learning and unsupervised learning have different goals. Unlike unsupervised learning, which aims to determine the patterns in data sets, reinforcement learning intends to find the optimized action model.

A common example of reinforcement learning is dog training. In the training, dogs learn by rewards for desired responses [3]. In this case, the dog is the learning agent that is exposed to the environment, the house. The action states could be the dog sitting, fetching, or shaking hands. Based on the dog's action, it may get a reward or penalty in return.

Reinforcement learning algorithms offer tremendous capabilities in systems that require decision making based on thousands of parallel game rounds. By leveraging the power of trying and searching on many trials, reinforcement learning becomes the most effective way in fields like engineering design and artificial captivities. As reinforcement learning helps to find the suitable actions under certain situation and attain largest rewards, in more complex situations, it has been used in robotics for industrial automation, business strategic investment, and personalized recommendations.

1.1 Background of Q-learning

As the ultimate goal of reinforcement learning is for the agent to learn an optimal policy that accomplishes the maximum rewards or minimum cost, one may wonder if it is possible to calculate the optimal policy and determine the best action to take at a given state. In order to study the optimal policy, the concept of Q-learning was firstly introduced in 1989 by Chris Watkins in his thesis *Learning from Delayed Rewards* [5]. Different but similar to the memory matrix introduced in Watkins's thesis, the present Q-learning studies the state-action-value function Q to find the expected rewards for an agent's action in a specific state. Thus, the Q-learning has always been an important algorithm in reinforcement learning, and many approaches have been proposed.

1.2 Contributions and Outline

The linear programming approach for a deterministic dynamic system has an ancient history. Mehta and Meyn have proposed a new type of convex

Q-learning algorithms, the dynamic programming linear program (DPLP), for deterministic system and then generalized it to Markov decision process in their pre-print *Convex Q-Learning, Part 1: Deterministic Optimal Control* [1].

Built upon Mehta and Meyn's formulations for their proposition, we utilize simpler notations in this thesis and aim to examine the linear programming approach in depth using an intuitive training example. As the linear programming approach quickly becomes infeasible as the number of states and actions increases, we intent to discuss this algorithm's limitations and hopefully provide some insights on connecting linear programming to reinforcement learning problems.

This thesis begins with an introduction to the finite dynamic system and the framework of reinforcement learning in Chapter 2. Fundamental concepts such as state-value function, state-action-value function, and Bellman equation are also introduced with notations. In Chapter 3, we formulate Q-learning as a linear program in a deterministic, discrete-time RL system with discrete state and action space. In Chapter 4, in order to obtain an algorithm that solves the problem exactly, we formulate the learning problem as a linear program and solve it numerically. Since the LP approach quickly becomes infeasible when the number of states and actions grows, in Chapter 5, we survey recently proposed approximation schemes and discuss the limitations of the linear program-based solution.

Chapter 2

Background

In order to formulate problems in the reinforcement learning problems and to study the optimal reward policy, the basic reinforcement learning system is modeled based on the Markov decision process. To simplify the system and problem, we will only deal with a discreet and finite system. In this chapter we introduce the basic framework of reinforcement learning with a training example to illustrate key components within the environment. In the second half of this chapter, we also introduce the Bellman equation, which solves for the value functions.

2.1 The Basic Reinforcement Learning Framework

The formal framework of reinforcement learning is driven by finding the optimal control of finite Markov decision process (finite MDP), which provides us a way to formulate sequential decision making [4].

In a finite MDP, we have an *agent*, the decision maker that interacts with the *environment.* At each time step, the agent will choose an *action* to take. The environment is then transitioned into a new *state*, and the agent is given a *reward* or a *cost* as a consequence of the previous action.

The main elements of a MDP Systems includes:

- Agent
- Environment − the physical world in which the agent operates
- State − current situation of the agent
- Action
- Reward or Punishment − the feedback of the agent's action

This process of selecting an action from a give state, transitioning to a new state, and receiving a reward happens sequentially over and over again. Throughout this process, it is the agent's goal to maximize the cumulative amount of rewards (or minimizes the cumulative costs) that are received from taking a series of actions. Although the agent can both receive reward or punishment based on its action, for numerical simplification, we now assume the agents only receive punishment, *cost*, as its feedback from the environment, and thus its ultimate goal is to minimizes the cumulative costs.

Suppose in a MDP, we have a discreet set of states X, a discreet set of actions U, and a set of rewards C. With the assumption that all of these sets are finite and discrete.

At each time step t, the agent receives some representation of the environment's state $x_t \in X$. Based on this state, the agent selects an action $u_t \in U$. This gives us the state-action pair (x_t, u_t). Time is then incremented to the next time step $t + 1$, and the environment is transitioned to a new state $x_{t+1} \in X$. At this time, the agent receives a numerical cost $c_{t+1} \in C$ for the action u_t taken from state x_t. Thus we can think of the process of receiving a cost as an running *cost* function $c(x_t, u_t)$ that maps state-action pairs to costs. At time t, we have the cost function as

$$c(x_t, u_t) = c_{t+1}. \tag{2.1}$$

What drives a reinforcement learning agent in a MDP is the expected return. The expected return can be think of as the sum of future cost. Thus mathematically, we can define the return G at time t as

$$G_t = c_{t+1} + c_{t+2} + \cdots + c_T, \tag{2.2}$$

where T is the final time step. Thus, it is the agent's goal to minimize the expected return of costs G_t.

For all possible actions an agent might take in all possible states of the environment, one might wonder what is the probability that an agent will choose an action in a particular state? In order to answer the question, the notion of *Policy* is introduced. A *Policy*, ψ, is a function that defines how an agent will act from a certain state.

For a stochastic policy, $\psi : X \times U \to [0, 1]$ maps a given state to a probability of selecting possible action at that state. Thus, if an agent follows policy ψ at time t, then

$$\psi(u_t|x_t) = Pro(u_t = u|x_t = x), \tag{2.3}$$

indicating the probability of taking action u at state x at time t.

However, in a deterministic setting, which is our focus, the policy $\psi : X \to U$ is a sequence of actions u taken starting at state x for each time step t

$$\psi(x_t) = u_t, \tag{2.4}$$

meaning that at state x_t, the agent can only take the action u_t under policy $\psi(x)$, unless the policy changes.

Thus, our goal to find the optimal policy becomes to find the best action for the agent to take at a certain state in order to minimize the total expected cost G,

$$\min_{\psi} \mathbb{E}(G). \tag{2.5}$$

In the finite MDP, the space of states, actions, and costs all have finite number of elements. Therefore, we have a well defined *transition probability distribution*. If the agent start in state $x_t \in X$, and after taking action $u_t \in U$, the agent ends up in new state $x_{t+1} \in X$ with an action cost $c_t \in C$, the transition probability distribution is then given by

$$p(x_{t+1}, c_t | x_t, u_t). \tag{2.6}$$

Also, probability distribution of each choice of x and u should add up to 1. Thus,

$$\sum_{x_{t+1} \in X} \sum_{c \in C} p(x_{t+1}, c_t | x_t, u_t) = 1. \tag{2.7}$$

2.2 Example

To better illustrate the basic reinforcement learning framework, a deterministic, discrete-time reinforcement learning training problem is provided. Suppose there is a reinforcement learning agent in the environment, with given state space $X = \{0, 1, 2, 3, 4, 5, 6\}$ and action space $U = \{-1, 0, 1\}$ and the model is

$$f(x, u) = x + u. \tag{2.8}$$

The running cost is also given by:

$$c(x,u) = \begin{cases} 0 & x = 0 \text{ or } x = 6 \\ \\ 1 & \text{else.} \end{cases} \tag{2.9}$$

At time t_0, suppose the agent is at initial state $x_0 = 0$. If the agent's desire is to move to state $x = 1$, what series of actions, or policies, can the agent follow?

To better understand the system, we can think of the state space as a line segment, and action space $x = 0$ as "staying", $x = 1$ as "going right", and $x = -1$ as "going left". One possible action state sequence is shown below using the diagram 2.1. In the diagram, we picture the agent pictured as a blue circle, and list out action, state, and cost at each time step based on the training problem above. The agent takes a series of actions $u_0 = 1$, $u_1 = 1$, and $u_2 = -1$ to reach the desired state with a total cost $G = 2$.

There are many other policies the agent can follow to reach state $x = 1$, but note that the agent will not be able to stay at $x = 1$ after it reaches the state because the agent does not have the option to act $u = 0$, which is equivalent to "staying", at $x = 1$.

Although the agent can move along the line following different policies, its total cost of actions increases as it takes more action. Remember that the goal of reinforcement learning is to find the action sequence that allow the agent to receive the minimum total expected cost, we then introduce the

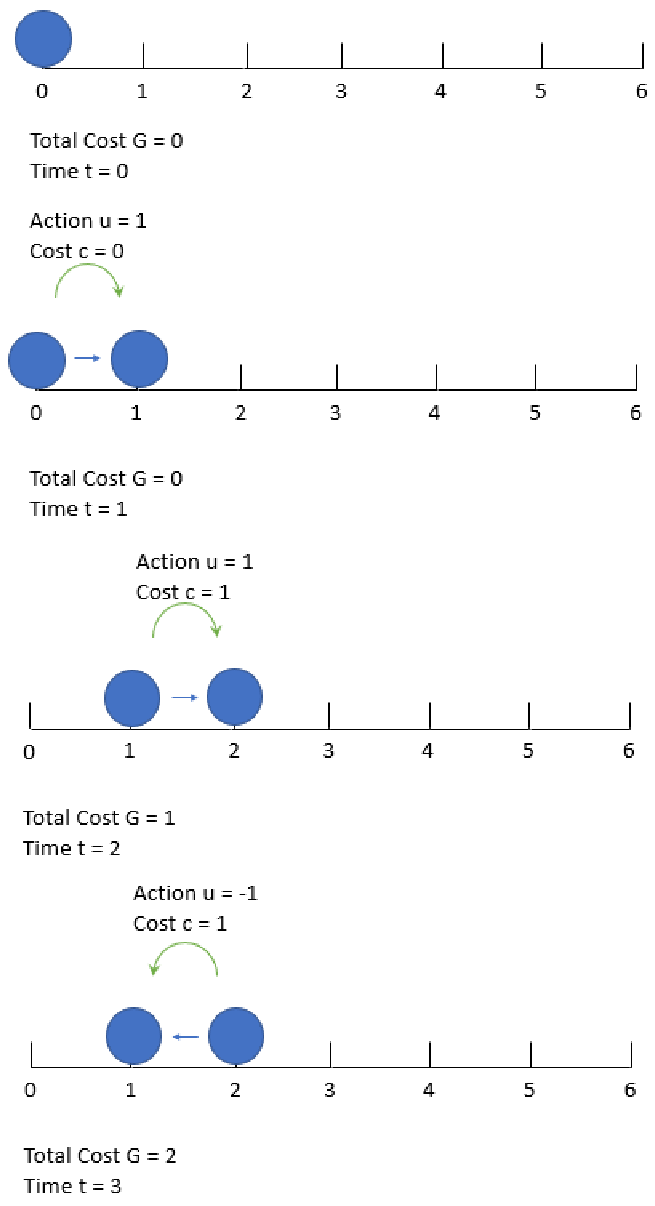

Figure 2.1: Example diagram of the agent's action sequence

concepts of value function and state-value function to discuss how to find the optimal action sequence, or we call it the optimal policy.

2.3 Value Function and State-value Function

In terms of cost, selecting one action over another in a given state may increase or decrease the agent's cumulative costs. Therefor in addition to understanding how to select an action, questions like how good a given action or a given state is for the agent need to be answered. The notion of "how good" is defined in terms of future expected cost. To determine the value of certain state and/or action, *value function* and *action-value function* are introduced. With these two functions, we are then able to study the agent's best decision on which actions to take at certain states.

- Value Function $v_\psi(x) : X \to \mathbb{R}$

  The *value function* $v_\psi(x)$, also known as state-value function, indicates the total costs that an agent can receive from state x to the end of the sequence following policy ψ. By averaging overall costs, value function allows us to determine the quality of the policy.

- Action-value Function $Q_\psi(x, u) : X \times U \to \mathbb{R}$

  The *action-value function* $q_\psi(x, u)$, also known as state-action-value function, indicates the total expected costs that an agent can receive by taking an action u from state x to the end of the sequence under policy ψ. By definition, $Q(x,u)$ function take in state-action pairs and estimate how good it is for an agent to perform a given action at a given state. Conventionally, the state-action-value function $Q_{\psi}(x,u)$ is refereed as the Q-function.

As we mentioned before, a policy specifies the agent's decision of actions in the environment. Our goal in reinforcement learning system is to find the optimal policy that gives the agent the least costs. Since we are considering discrete states, one policy may perform better than other policies. We denote the optimal policy $\psi^*(x)$ as

$$\psi^*(x) \geq \psi_i(x). \tag{2.10}$$

By the definition of value function, one policy $\psi_1(x)$is considered better than $\psi_2(x)$ if and only if the value under $\psi_1(x)$ is greater than the value under $\psi_2(x)$:

$$\psi_1(x) > \psi_2(x) \Leftrightarrow v_{\psi 1}(x) > v_{\psi 1}(x) \quad \text{for all} \quad x \in X. \tag{2.11}$$

Thus the optimal value function $\Phi(x)$ has the lowest cost in every state:

$$\Phi(x) = \min_{\psi} v_{\psi}(x) \quad \text{for all} \quad x \in X. \tag{2.12}$$

Similarly, the optimal action-value function Q is the minimum for every state action pair:

$$Q^*(x, u) = \min_{\psi} Q_{\psi}(x, u) \quad \text{for all} \quad x \in X, u \in U. \tag{2.13}$$

The definition of the optimal value function $\Phi(x)$ and optimal action-value function $\min Q(x, u)$ is introduced with more detail in section 3.1. With the optimal value function and optimal action-value function, we can connect the current and future values of the states and actions, without waiting the agent to receive all future costs.

2.4 Relevant Work

The calculation and approximation of value function and action-value function has been an essential approach to understand the agent's decisions in a reinforcement learning system. As the value functions and action-value functions summarize total costs of all actions in the future under the policy, we are then able to judge the quality or how good different policies are. If we are able to calculate or approximate the optimal value function $\Phi(x)$ and action-value function $\min Q(x, u)$, we can not only estimate the minimum total costs but more importantly determine the best policy that are not immediately available before long-term observations.

The formulation of optimal value functions are introduced in Chapter 3.7 and 3.8 in book [2]. By pointing out the recursive relationships, Sutton et al. decomposed the value function into immediate reward and the future values. Instead of computing the summation over multiple time steps, such simplification allows us to find the optimal solution in each sub-problem. However Sutton et al. also points out that under stochastic policies, the probability of taking action u at state x and landing in state x' is uncertain. Thus, it's not possible to apply bellman equation directly. Therefore, Sutton et al. proposed value function $v_{\psi}(x)$ and $Q_{\psi}(x, u)$ can be estimated through *Monte Carlo methods*, which averages over random samples of actual returns.

Chapter 3

Q-learning Algorithm

In this chapter, the design of reinforcement learning algorithms is for non-linear, deterministic state space models. The setting is in discrete time and action space. The problem setup follows the recent work of [1] and the book [2], but with a simpler and easier notation.

Consider a reinforcement learning problem with discrete state space X and action space U, and the action and state are related based on the dynamical system

$$x_{t+1} = f(x_t, u_t), t \geq 0, x_0 \sim \rho, \tag{3.1}$$

where $f : X \times U$ denotes the model of the environment and t denotes the discrete definite time step. In practical situations, f is more likely to be unknown, and only observations (x_t, u_t, x_{t+1}) are given. For simplicity, we will deal with a training problem that f is known to us. The initial state $x_0 \sim \rho$ and the running *cost* function $c : X \times U \to \mathbb{R}$ are also given in the system.

It is also assumed that an equilibrium (x^e, u^e) is achieved

$$x^e = f(x^e, u^e). \tag{3.2}$$

3.1 Bellman Equation

Based on Chapter 3.7 in book [2] we define $v(x)$ as the value function $v : X \to \mathbb{R}$ under the policy ψ:

$$v_\psi(x) = \mathbb{E}_\psi[G_t | x_t \in X]. \tag{3.3}$$

Based on the Bellman equation of vale function, we have

$$v_\psi(x) = \sum_x \psi(u|x) \sum_{x',c} p(x', c|x, u)[c + v_\psi(x')], \tag{3.4}$$

where x' is the next state from x, and $p(x', c|x, u)$ shows the weight or probability of landing on state x' with cost c after taking action u at state x.

But under deterministic policy ψ, we don't need to take expected value of the total costs as there is no uncertainty. Also, as the deterministic policy matches the state to a specific action with no uncertainty,

$$p(x', c|x, u) = 1. \tag{3.5}$$

Thus, the Bellman equation of value faction becomes

$$v_\psi(x) = c(x, \psi(x), x') + v_\psi(x'). \tag{3.6}$$

We also define $Q(x, u)$ as the action-value function $Q : X \times U \to \mathbb{R}$ under the deterministic policy ψ:

$$Q_\psi(x) = \mathbb{E}_\psi[G_t | x_t \in X, u_t \in U]. \tag{3.7}$$

After simplify the expectation operator, we we derive the Q-value Bellman equation

$$Q_\psi(x) = c(x, u, x') + v_\psi(x'), \tag{3.8}$$

where x' is the next state from x, and $c(x, u, x')$ shows the cost of landing on state x' with after taking action u at state x.

Detailed derivation for Bellman equation of value function and action-value function following a deterministic policy can be found in **??**.

3.2 Q-Learning: an Introduction

Given the initial state $x_0 \sim \rho$, and the running *cost* function $c : X \times U \to \mathbb{R}$, our goal is to find a control sequence $\boldsymbol{u}^*(x_0) = \{u_t\}_{t=0}^{\infty}$ that minimizes a given cost functional of the form

$$J(x_0, \boldsymbol{u}) = \sum_{t=0}^{\infty} c(x_t, u_t), \tag{3.9}$$

where $c : X \times U \to \mathbb{R}$ is a *running cost*. When a mathematical description of the model f and the running cost c are known, solving (3.9) is a dynamic programming (DP) problem.

We define $\Phi(x)$ as the optimal value function $\Phi : X \to \mathbb{R}$ that assigns the optimal cost-to-go to every state. The dynamic programming equations associate with the control sequence $\boldsymbol{u}^*(x)$ is then:

$$\Phi(x) = J(x, \boldsymbol{u}^*(x)) = \min_{\boldsymbol{u}} J(x, \boldsymbol{u}). \tag{3.10}$$

We then define the optimal state-action-value function, also known as the Q-function, $Q : X \times U \to \mathbb{R}$ as

$$Q(x, u) = c(x, u) + \Phi(f(x, u)). \tag{3.11}$$

Thus the Bellman equation is equivalent to:

$$\Phi(x) = \min_{u} Q(x, u), \tag{3.12}$$

which can be used for finding the value function Φ and/or Q. Q is very important to us as the Q-function allows us to extract the optimal action at every given state via *policy* function $\psi : X \to U$

$$\psi(x) = \arg\min_{u \in U} Q(x, u). \tag{3.13}$$

3.3 Q-Learning as a Linear Program

Suppose Φ is continuous and vanishes only at equilibrium x^e, then we can find the pair of (Φ, Q) that solves the following linear program:

$$\min_{\Phi, Q} \mathbb{E}_{x \sim \mu} [\Phi(x)] \tag{3.14a}$$

$$\text{subject to} \quad Q(x, u) = c(x, u) + \Phi(f(x, u)) \tag{3.14b}$$

$$Q(x, u) \geq \Phi(x), \quad \forall (x, u) \in X \times U \tag{3.14c}$$

$$\Phi(x^e) = 0, \tag{3.14d}$$

where μ is some probability distribution.

Since we assumed that the state and action space are finite, this problem is indeed a finite-dimensional linear program. Its size grows very rapidly though as the number of state and actions grow.

A few remarks and explanation about this formulation:

1. The first constraint (3.14b) is exactly the same as the definition of Q in (3.11).

2. The second constraint (3.14c) is the optimality condition in (3.11).This

relaxation is very essential for the algorithms because the minimization operator in the constraint will make the linear program hard to solve.

3. Similar to equation (23) in [1], we can also relax (3.14b) to inequality constraints.

4. In the above formulation, although Q can be eliminated, we choose not to because Q is the main thing we want to learn in Q-learning.

Chapter 4

Training Problem

In Chapter 2, we introduce a training problem in a deterministic, discrete-timed, and finite reinforcement learning system. In this chapter, we intend to numerically solve the previous problem and compare the results we attained by applying the linear program approach (3.14) formulated in Chapter 3.

4.1 Training Problem

Just as the problem stated in Chapter 2, suppose we have a deterministic and finite reinforcement learning problem. Let the state space be $X = \{0, 1, 2, 3, 4, 5, 6\}$ and at each state we have the actions $U = \{-1, 0, 1\}$ and the model is

$$f(x, u) = x + u. \tag{4.1}$$

The running cost is given by

$$c(x,u) = \begin{cases} 0 & x = 0 \text{ or } x = 6 \\ 1 & \text{else.} \end{cases} \tag{4.2}$$

Let the distribution of initial states, ρ, be defined by

$$\rho(x) = \begin{cases} \frac{1}{5}, & x \in \{1,2,3,4,5\} \\ 0 & \text{else.} \end{cases} \tag{4.3}$$

4.1.1 Numerical Solution

Intuitively, no matter which action the agent decides to choose, the key to solve this problem is to find the converging state. In other words, we need to define when the cost of action vanishes at a specific action. Observing the cost function, we find that at state $x = 0$ and $x = 6$, the cost of taking any action $u \in U$ will be 0. Also, if the agent takes action $c = 0$ at state $x = 0$ and $x = 6$, the next state x' remains the same based on the model $f(x,u)$.

Thus we find the equilibrium state-action pair $(x_1^e, u_1^e) = (0,0)$ and $(x_2^e, u_2^e) = (6,0)$ such that

$$f(x_1^e, u_2^e) = 0$$

$$f(x_2^e, u_2^e) = 6,$$

Based on the definition of $\Phi(x)$, finding $\Phi(x)$ are equivalent to finding minimum cost the agent receives for it to move from the initial state x_0 to the equilibrium state x^e.

Suppose the initial state $x_0 = 2$. There are two choices of x^e for the agent to move toward. As x_0 is closer to $x_1^e = 0$, the agent will receive less cost if it decide to move toward x_1^e. After deciding the destination, there are also many choices of actions for the agent to take. It can either move further from the destination at some time step and then move toward the destination at other time step. But as the cost of action is always greater or equal to zero, with more actions, the agent will receive more cost along the way to the equilibrium state.

Thus, with the desire to gain least cost, the agent's best choice is to move directly toward $x_1^e = 0$. By taking action $u_0 = -1$, moving to $x_1 = 1$, taking action $u_1 = -1$, the agent finally arrives at $x_2 = x_1^e = 0$. With each of the action, the agent receive a cost of 1, then the minimum total cost for the agent starting at $x_0 = 2$ is 2. Therefore we have manually calculated the $\Phi(2) = 2$.

Suppose the initial state $x_0 = 5$, which is closer to $x_2^e = 6$. Thus, the agent will receive less cost if it decide to move toward x_2^e. Similarly, with the desire to gain least cost, the agent's best choice is to move directly toward $x_2^e = 6$. By taking action $u_0 = 1$, the agent finally arrives at $x_1 = x_2^e = 6$. With action u_0, the agent receive a cost $c(5, 1) = 1$, then the minimum total cost for the agent starting at $x_0 = 5$ is 1. Therefore we have manually calculated the $\Phi(5) = 1$.

The calculation of other cases of initial states are similar to the previous examples, which provides us a result table shown below.

$\Phi(x)$							
State(x)	0	1	2	3	4	5	6
$\Phi(x)$	0	1	2	3	2	1	0

Also, based on the definition of $Q(x, u)$, finding $Q(x, u)$ are equivalent to finding minimum cost the agent receives for it to move from the initial state x_0 with an initial action u_0 to the equilibrium state x^e. Note that $Q(0, -1)$ and $Q(6, 1)$ do not exist as the subsequent state x' is out of bound.

With the $\Phi(x)$ in mind, we can also manually calculate the $Q(x, u)$. The only difference between calculating $Q(x, u)$ and $\Phi(x)$ is that the initial action u_0 is given.

For example, suppose we want to calculate $Q(1, 1)$. With initial state $x_0 = 1$ and initial action $u_0 = 1$, $x_1 = f(1, 1) = 2$. As x_1 is closer to x_e^1, the agent will receive the least cost if it moves directly toward x_e^1. Thus by taking actions $x_1 = -1$, $x_2 = -1$, the agent finally lands on $x_3 = x_e^1 = 0$, with total cost $G = 3$. Therefore, $\min Q(1, 1) = 3$.

The calculation of other $\min Q(x, u)$ are similar to the previous examples, and we have the following summary $\min Q(x, u)$ table.

$\min Q(x,u)$							
State(x)	$x=0$	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$	$x=6$
Action($u=1$)	0	3	4	3	2	1	N/A
Action($u=0$)	0	2	3	4	3	2	1
Action($u=-1$)	N/A	1	2	3	4	3	2

4.1.2 Linear Programming Approach

In order to formulate the problem as linear programming, we follow (3.14), which we derived based on [1].

We first start by finding the equilibrium x^e. Since the running cost is 0 at $x = 0$ or $x = 6$, based on the model $f(x, u) = x + u$, when the action is 0 or 6, it reaches the optimal state. Thus following (3.14d) we explicitly write out the equilibrium conditions as

$$\Phi(0) = 0, \quad \Phi(6) = 0. \tag{4.4}$$

The objective function $\mathbb{E}_{x\sim\mu}[\Phi(x)]$ can be written as

$$\sum_{p\sim\mu}(p_0 \times \Phi(x_0) + \cdots + p_6 \times \Phi(x_6)). \tag{4.5}$$

For simplicity, we set each $p = \frac{1}{7}$ assuming that the probability of selection the initial 7 states are the same.

Also as we discussed in section 4.1.1, $Q(0, -1)$ and $Q(6, 1)$ do not exist, thus the relating constraints also do not exist.

Following (3.14b) (3.14c) and we can explicitly write out the equality and inequality constraints in a matrix form in MATLAB.

In MATLAB, we specified linear inequalities as

$$Ax \leq b,$$

where `A` is a k-by-n inequality matrix,`k` is the number of inequalities, `n` is the number of variables (size of x), and `b` is a inequality vector of length k.

And we specified linear equalities as

$$A_{eq}x = b_{eq},$$

where A_{eq} is an m-by-n equality matrix, m is the number of equalities n is the number of variables (size of x), and b_{eq} is a equality vector of length m.

Then we can use MATLAB to create variables for indexing and solve the linear programming problem with the built-in MATLAB function `linprog` following Algorithm 1.

4.1.3 Results

Solving the previous training problem, we have attained the results for $\Phi(x), Q(x, u)$ in the following tables.

Algorithm 1 Linear Programming Approach for Q-learning

```
  Input Unknown variables: 7 $\Phi(x)$ and 19 $Q(x,u)$
  Output Exact value of each $\Phi(x)$ and $Q(x)$
 1: Combine 26 variables into one vector variables
 2: for $v = 1, \ldots, 26$ do
 3:   eval([variables\{v\},' = ', num2str(v),';'])
      {create variables for indexing}
 4: end for
 5: Specify Linear Equality Constraints $A_{eq}$ and $b_{eq}$
 6: Write Linear Inequality Constraints $A$ and $b$
 7: Write the objective
 8: options = optimoptions('linprog','Algorithm','dual-simplex')
 9: [x fval] = linprog(-f,A,b,Aeq,beq,[],[],options)
    {Call the solver}
10: for $d = 1, \cdots 26$ do
11:   fprintf('%12.2f \t%s\n',x(d),variables{d})
      {Print the output}
12: end for
13: return $Q$ and $\Phi$
```

$\Phi(x)$							
State(x)	0	1	2	3	4	5	6
$\Phi(x)$	0	1	2	3	2	1	0

$\min Q(x,u)$							
State(x)	$x=0$	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$	$x=6$
Action($u=1$)	0	3	4	3	2	1	N/A
Action($u=0$)	0	2	3	4	3	2	1
Action($u=-1$)	N/A	1	2	3	4	3	2

Based on the solution we gained using linear program approach (3.14), we attained the same solution with numerical method and the LP-based approach.

Chapter 5

Discussion

From the previous chapter, we can see that the numerical results and analytical results are the same, which adds to the credibility of the linear programming approach.

Although the approach proposed in (3.14) provides insights on how to use linear programming in Q-learning, the algorithm has several limitations that deserve further discussion and research on.

Large or Infinite State Space X and Action Space U

Although based on our result, the linear programming approach are very successful, our training problem has a finite and very small discreet state space and action space. What if we want to train the agent to play checkers game, can we still use the linear programming approach to numerically solve for the optimal action sequence? With more states and actions, it is very difficult to capture the agent's action-reward pairs using the linear programming approach. We believe that the linear program will become infeasible as the number of objective functions and constraints increases, and future research can be focused on finding the maximum finite state space and action for the linear programming approach to be still feasible. Also, in more realistic scenario, reinforcement learning systems has infinite action and states sequence, and thus the future costs become uncertain. In [1], Mehta and Meyn have some heuristic algorithm to address this limitation and more investigation should be included in the future.

Discount Factor

In the infinite-time system, the total costs keep growing, leading to an unbounded problem that are very hard to find the maximum or minimum value. To mathematically formulate this scenario, we can introduce discount factors when calculating the total costs and scale down each cost to reach a convergence.

Thus, let $\gamma \in [0, 1]$ be the discount factor, the total cost G_t becomes

$$G_t = \gamma^0 c_{t+1} + \gamma^1 c_{t+2} + \gamma^2 c_{t+3} + \cdots + \gamma^{T-t-1} c_T. \tag{5.1}$$

As $T \to \infty$,

$$G_t = \sum_{k=0}^{\infty} \gamma^k c_{t+k+1}, \tag{5.2}$$

and the Bellman equation for value function and action-value function will thus change accordingly.

By introducing a discount factor bounded to be smaller than 1, we are able to use the infinite geometric series summation formula to generalize the formulation. When $\gamma \in [0, 1)$ we give greater weight to sooner costs and less weight to further costs in future. When $\gamma = 1$, we treat all costs equally, and this is the same case we discussed in (3.14).

Stochastic Policy ψ

As stated in the beginning of this thesis, we only focus on deterministic policies as there always exists at least one deterministic policy that is optimal. Also, under a deterministic policy, at every state, the agent has clear-defined actions to take and thus determine the outcomes. But in more practical cases, the agent may follow stochastic policies. Under stochastic policies, instead of having a clear-defined action sequence at every state, the agent has a probability distribution for actions to take. Thus, instead of being certain of taking an action, the agent has a probability of taking another action. Thus, in a stochastic environment, there is an uncertainty about the action's rewards or costs. When the agent repeats doing the same action in a given state, the new state and received reward may not be the same each time. In this circumstance, can we apply the linear programming approach in stochastic system and extract the optimal value function and optimal Q-function?

Although empirical experiments are not conducted because of the limited time of honors thesis, we believe that the linear programming approach is not applicable. As we discussed in section 3.1, under a stochastic policy, the Bellman equation for value function and state-value function all includes the probability of taking action u at state x and landing in state x'. As such probability is uncertain, we could not solve MDP by applying the Bellman equation directly.

Even though with the help of other estimation methods to calculate value function and action-value function, the constraints grow even faster than the deterministic policies with the same amount of action and state space as we need to consider each action's probability and its action effect. Thus, the constraints in (3.14) will become intractable very fast under a stochastic policy.

Chapter 6

Conclusion

The linear programming approach for Q-learning we proposed is very successful in dealing with small action and state space under deterministic policies. With the advantages like small time-cost and memory-cost, linear programming can be used in solving deterministic and finite MDP problems. Our approach also connects the optimal value function and Q-function. By calculating either value, we are able to attain the optimal policy without numerous observations on the agent's choices of decisions. However, our approach has limitations in dealing with RL problems, whose state and action space are large or infinite, because the constraints in the linear program will grow very fast and thus become intractable. Further research on testing the limitations of the action and state space under deterministic and stochastic process will be included in future.

Bibliography

[1] P. G. Mehta and S. P. Meyn. Convex Q-Learning, Part 1: Deterministic Optimal Control. *arXiv:2008.03559*, Aug. 2020. 4, 16, 21, 26, 31

[2] R. S. Sutton and A. Barto. *Reinforcement learning.* The MIT Press. The MIT Press, 2nd edition edition, 2018. 1, 2, 14, 16, 17

[3] H. R. Tizhoosh. Opposition-based reinforcement learning. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 10(3), 2006. 2

[4] M. van Otterlo and M. Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. 6

[5] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, 1989. 3