Markov Decision Processes: A gentle tutorial

Oualid Missaoui[*]

August 2, 2023

Abstract

Markov Decision Processes (MDPs) form the cornerstone of reinforcement learning (RL) and serve as a fundamental modeling tool for making sequential decisions. In this note, we present a comprehensive definition of MDPs and provide a detailed derivation of the Bellman equations, along with the optimality results. Our approach aims to ensure a thorough understanding by avoiding the omission of any steps in the mathematical proofs. The primary goal is to facilitate reading classic textbooks on (approximate) dynamic programming, optimal control, and reinforcement learning, where proofs and derivations can sometimes obscure crucial details, making them less accessible to readers from diverse scientific and engineering backgrounds.

Index terms— Markov Decision Processes, reinforcement learning, dynamic programming, Bellman equations

[*]The views and opinions expressed in this tutorial are those of the authors and do not necessarily reflect the official policy or position of any other agency, organization, employer or company.

Contents

1 Introduction

Markov Decision Processes (MDPs) were introduced by the American applied mathematician Richard Ernest Bellman in 1957 as a powerful mathematical framework for solving discrete stochastic versions of optimal control problems. Bellman's earlier work in 1953 laid the foundation for dynamic programming, which plays a vital role in solving MDPs.

MDPs serve as a mathematical tool for modeling decision-making problems in discrete, stochastic, and sequential environments. The core idea is to represent a decision maker, or agent, operating in an environment where state transitions occur randomly in response to the agent's actions. The agent receives rewards based on its actions in a given state.

The primary goal of solving an MDP is to find an optimal policy that prescribes the best course of actions to maximize the expected reward or return. Dynamic programming is a powerful technique used to identify such optimal action policies.

Markov Decision Processes and Reinforcement Learning have found widespread applications in various fields due to their ability to model decision-making under uncertainty and find optimal solutions to complex problems. In robotics, MDPs and RL are utilized to design autonomous agents capable of learning to navigate unknown environments, grasp objects, and perform intricate tasks. RL has proven invaluable in the field of finance, where it is employed to optimize investment strategies, portfolio management, and algorithmic trading. Additionally, MDPs are extensively used in healthcare to devise personalized treatment plans, manage resource allocation in hospitals, and optimize drug dosages. In the context of artificial intelligence, RL has enabled the development of game-playing agents that can outperform human players in board games like Go and Chess. Furthermore, MDPs have found applications in energy management, transportation, and industrial control systems, allowing for energy-efficient operations, traffic optimization, and process control. The adaptability and versatility of MDPs and RL continue to drive innovation across numerous domains, making them indispensable tools for tackling real-world decision-making challenges.

In this tutorial, we predominantly follow the notation of Sutton [1], a widely respected source in the field of reinforcement learning. The subsequent sections will cover various aspects of MDPs, providing a comprehensive understanding of this important topic.

We begin by defining MDPs and outlining their underlying assumptions, which include the Markov property—where the future state and reward depend only on the current state and action, making MDPs memoryless. Then, we delve into different value functions, such as the state-value function and action-value function, and explore their interrelationships, paving the way for grasping the concept of the Bellman equations.

The Bellman equations play a key role in solving MDPs, enabling us to formulate the prediction and control problems. By understanding these essential equations, we can uncover the requirements for finding the optimal policy and value functions.

The Bellman operator, a fundamental concept in dynamic programming, will be introduced. We will explore its role in providing optimality guarantees to resolve both the prediction and control problems. Additionally, we will outline the policy evaluation procedure, which helps us estimate the value function under a given policy.

To solve the control problems in MDPs, we present synchronous and asynchronous policy and value iteration algorithms. These algorithms are essential tools that iteratively improve the policy and value functions until convergence, providing a systematic way to find the optimal solution.

To illustrate the concepts presented in this tutorial, we have provided an illustrative Python code [1] that implements policy and value iteration algorithms to solve the Frozen-lake problem. This classic problem from the world of reinforcement learning serves as a practical example to see how MDPs and their algorithms can be applied in practice.

Whether you are a student, researcher, or practitioner interested in reinforcement learning, optimal control, or decision-making, this tutorial will serve as an accessible and comprehensive guide to understanding the fundamentals of Markov Decision Processes and how to apply them to real-world problems.

2 Markov Decision processes

A Markov Decision Process (MDP) is a mathematical framework used to model sequential decision-making problems in stochastic environments. It consists of a set of states, actions, transition probabilities, and reward functions. At each time step, an agent, or decision-maker, observes the current state of the system, selects an action from a set of available actions, and transitions to a new state based on the chosen action's stochastic transition probabilities. The agent receives a numerical reward as a consequence of its action, and the goal is to find an optimal policy that prescribes the best action to take in each state to maximize the cumulative expected rewards over time.

2.1 Definition

Formally, a Markov decision process (MDP) is defined by the tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, r, \gamma)$$

where

- $\mathcal{S}$: is the state space
- $\mathcal{A}$: is the action space
- $\mathcal{R}$: is the reward space
- γ: is the discount factor (rate)

with $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ all finite sets for a finite MDP, and p is the probability distribution

$$p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$$

governing the dynamic of the MDP and satisfying

$$p(s', r \mid s, a) = Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

and

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

where

- S_t: a random variable characterizing the state of the MDP at time t

[1] The code for this tutorial is available on GitHub: https://github.com/omroot/MDP.

- A_t: a random variable characterizing the action taken by the MDP at time t
- R_t: a random variable characterizing the reward perceived at time t from the action and state at time $t-1$

With the information provided above, we can establish the following meaningful measures that simplify the handling of Markov Decision Processes (MDPs):

- state-transition probabilities

$$\begin{aligned} p(s' \mid s, a) &= Pr(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) && (2.1) \\ &= \sum_{r \in \mathcal{R}} p(s', r \mid s, a) && (2.2) \end{aligned}$$

- expected reward for state-action pairs

$$\begin{aligned} r(s, a) &= \mathbb{E}\big[R_t \mid S_{t-1} = s, A_{t-1} = a\big] && (2.3) \\ &= \sum_{r \in \mathcal{R}} r Pr(R_t = r \mid S_{t-1} = s, A_{t-1} = a) && (2.4) \\ &= \sum_{r \in \mathcal{R}} r \sum_{s'} Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a) && (2.5) \\ &= \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) && (2.6) \end{aligned}$$

- expected reward for state-action-next-state triples

$$\begin{aligned} r(s, a, s') &= \mathbb{E}\big[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'\big] && (2.7) \\ &= \sum_{r \in \mathcal{R}} r Pr(R_t = r \mid S_{t-1} = s, A_{t-1} = a, S_t = s') && (2.8) \\ &= \sum_{r \in \mathcal{R}} r \frac{Pr(R_t = r, S_t = s' \mid S_{t-1} = s, A_{t-1} = a)}{Pr(S_t = s' \mid S_{t-1} = s, A_{t-1} = a)} && (2.9) \\ &= \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)} && (2.10) \end{aligned}$$

2.2 Assumptions

The Markovian assumptions in MDPs are fundamental to their modeling approach. They revolve around the Markov property, which states that the future state of the system solely depends on the current state and the action taken, regardless of the past history. This assumption ensures that the state transitions in an MDP are memoryless and independent of the agent's previous trajectory, simplifying the problem's representation and making it computationally tractable. The Markovian property allows MDPs to be used to model a wide range of real-world decision-making problems, where the current state provides all the necessary information to make optimal decisions.

Formally, the Markovian assumptions governing the MDP dynamic are:

- State dynamic: the state S_{t+1}, only depends on the action A_t taken at state S_t

$$Pr(S_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, \cdots, S_1, A_1) = Pr(S_{t+1} \mid S_t, A_t)$$

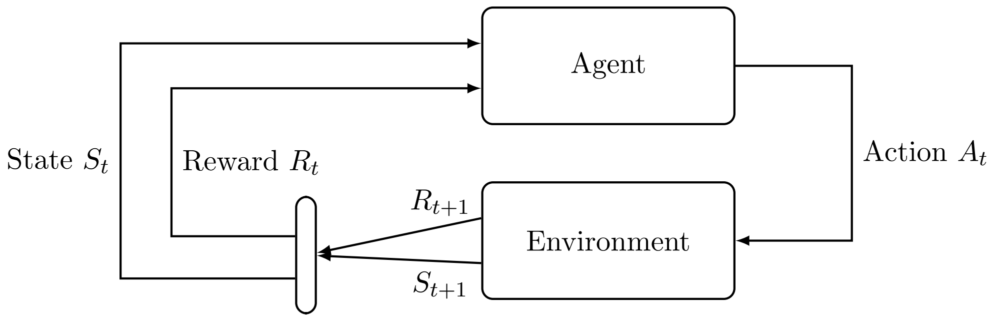

Figure 1: MDP Illustration

- Reward dynamic: the reward received at time $t+1$ only depends on the action taken at state S_t

$$Pr(R_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, \cdots, S_1, A_1) = Pr(R_{t+1} \mid S_t, A_t)$$

2.3 Reward function

In Markov Decision Processes (MDPs), the reward function plays a critical role in guiding the agent's decision-making process. The reward function assigns a numerical value to each state-action pair, representing the immediate desirability of taking a specific action in a given state. The agent's objective is to maximize the cumulative rewards obtained over time, leading to the selection of actions that result in favorable outcomes. The reward function serves as a key component in MDPs, enabling the agent to learn and adapt its policies to achieve its long-term objectives effectively.

We will consider two common choices

- Finite Horizon

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- Infinite Horizon discounted value

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where $0 \leq \gamma \leq 1$, called the discount rate.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} && (2.11) \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}) && (2.12) \\ &= R_{t+1} + \gamma G_{t+1} && (2.13) \end{aligned}$$

Alternatively, G_t can be expressed as:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

describing both the finite horizon ($\gamma = 1$) and the discounted horizon ($T = \infty$).

2.4 Policy functions

A policy is a mapping from states to actions, specifying the action the agent should take in each state to achieve its goals. The policy function guides the agent's behavior and determines its actions in response to the environment's stochastic transitions. There are different types of policies, including deterministic policies that select a single action for each state and stochastic policies that assign probabilities to each action, allowing for exploration and uncertainty handling. The ultimate goal in MDPs is to find an optimal policy that maximizes the agent's expected cumulative rewards over time, enabling the agent to make effective decisions in the face of uncertainty and achieve its objectives efficiently.

Formally, a policy is a mapping

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$$

that, for every state $s \in \mathcal{S}$ assigns for each action $a \in \mathcal{A}$ the probability $\pi(a \mid s)$ of taking that action a in state s.
For deterministic policies, we sometimes use the notation $a_t = \pi(s_t)$ to represent the action taken by the policy.
The goal of an RL agent is to find a behaviour policy that maximizes the expected return G_t.
A policy is stationary if it is independent of time.

2.5 Value functions

The value function in an MDP is a fundamental concept that quantifies the expected return an agent can achieve starting from a particular state and following a given policy. It represents the long-term desirability of being in a state and is crucial for decision-making. Specifically, the value function provides a way to rank states based on their expected cumulative rewards, guiding the agent towards more favorable states and actions.

There are two types of value functions in MDPs: the state value function and the state-action value function. The state value function quantifies the expected return starting from a particular state s and following the policy's action selections from that state onward. It measures how desirable a state is in the long run under a given policy. On the other hand, the state-action value function quantifies the expected return when the agent is in state s, takes action a, and then follows a specific policy thereafter. It captures the desirability of taking action a in state s and is used to make action selections based on maximizing expected rewards.

2.5.1 State-value function

Once at a given state, its quality can be quantified by the expected future reward generated by the policy, regardless of the action taken at this state

$$\begin{aligned} v_t^{\pi}(s) &= \mathbb{E}_{\pi}[G_t \mid S_t = s] && (2.14) \\ &= \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\Big] && (2.15) \end{aligned}$$

2.5.2 action-value function

Once at a given state s, and taking a given action a, the value of the state-action pair is the expected future reward generated by the policy conditioned on action a taken at the present state s

$$\begin{aligned} q_t^\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] &(2.16)\\ &= \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \mid S_t = s, A_t = a\Big] &(2.17)\end{aligned}$$

2.5.3 Relation between $v_t^\pi(s)$ and $q_t^\pi(s,a)$

The relationship between both value functions is given by the following theorem.

Theorem 1. *$\forall s \in \mathcal{S}$, the state-value function is given by*

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) q_\pi(s,a)$$

Proof. Using the law of total expectations:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s]\\ &= \mathbb{E}[\mathbb{E}_\pi[G_t \mid A_t, S_t = s]]\\ &= \sum_{a\in\mathcal{A}} \mathbb{E}_\pi[G_t \mid A_t = a, S_t = s] P(A_t = a \mid S_t = s)\\ &= \sum_{a\in\mathcal{A}} P(A_t = a \mid S_t = s) q_t^\pi(s,a)\end{aligned}$$

□

Moreover, if we have a deterministic policy π, then for every state s, π will take an action $a = \pi(s)$ with probability equal to one. Therefore,

$$v^\pi(s) = q^\pi(s, \pi(s))$$

Theorem 2. *$\forall s \in \mathcal{S}$, the action-value function is given by*

$$q_t^\pi(s,a) = \sum_{s'}\sum_{r} p(s', r \mid s, a)[r + \gamma v_{t+1}^\pi]$$

Equivalently, it can be written as

$$q_t^\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_{t+1}^\pi \mid S_t = s, A_t = a]$$

Proof.

$$\begin{aligned}
q_t^\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\
&= \mathbb{E}[\mathbb{E}_\pi[G_t \mid S_{t+1}, S_t = s, A_t = a]] \\
&= \sum_{s'} \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a, S_{t+1} = s']\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\
&= \sum_{s'} \mathbb{E}_\pi[G_t \mid S_{t+1} = s']\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\
&= \sum_{s'} \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_{t+1} = s']\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\
&= \sum_{s'} \mathbb{E}_\pi[R_{t+1} \mid S_{t+1} = s'])\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\
&\quad +\gamma \sum_{s'} \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\
&= \sum_{s'}\sum_{r} r\mathbb{P}(R_{t+1} = r \mid S_{t+1} = s'))\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\
&\quad +\gamma \sum_{s'} v_{t+1}^\pi(s')\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) \\
&= \sum_{s'}\sum_{r} r\, \underbrace{\mathbb{P}(R_{t+1} = r \mid S_{t+1} = s', S_t = s, A_t = a))\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)}_{\mathbb{P}(R_{t+1}=r, S_{t+1}=s' \mid S_t = s, A_t = a)} \\
&\quad +\gamma \sum_{s'}\sum_{r} v_{t+1}^\pi(s')\mathbb{P}(R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a) \\
&= \sum_{s'}\sum_{r} [r + \gamma v_{t+1}^\pi]\mathbb{P}(R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a)
\end{aligned}$$

□

2.6 Bellman equations

In the preceding section, we introduced the state and action value functions and established a relationship between them. In the following sections, we will derive additional recursive relationships that link the state-value and action-value functions. These relationships, known as Bellman equations, enable us to efficiently compute the value functions from initial values. In the upcoming two sections, we will present and prove these Bellman equations.

2.6.1 For the state-value function

Theorem 3. *The state-value function captures the value of a state s in the context of a policy π which can be computed recursively as*

$$v_t^\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_{t+1}^\pi \mid S_t = s]$$

or equivalently

$$v_t^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{t+1}^\pi(s')]$$

Proof. By definition, the state-value function is given by

$$\begin{aligned} v_t^\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] && (2.18) \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && (2.19) \\ &= \mathbb{E}_\pi[R_{t+1}] + \mathbb{E}_\pi[\gamma G_{t+1} \mid S_t = s] && (2.20) \\ & && (2.21) \end{aligned}$$

We have

$$v_{t+1}^\pi(s) = \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s]$$

In addition , we know from conditional expectations properties that

$$\mathbb{E}[\mathbb{E}(X \mid Y, Z) \mid Y] = \mathbb{E}(X \mid Y)$$

which if we apply it on $X = G_{t+1}, Y = S_{t+1}, Z = S_t$ we get

$$\mathbb{E}[\mathbb{E}(G_{t+1} \mid S_{t+1}, S_t) \mid S_t] = \mathbb{E}(G_{t+1} \mid S_t)$$

By the Markovian assumptions of the MDP

$$\mathbb{E}(G_{t+1} \mid S_{t+1}, S_t) = \mathbb{E}(G_{t+1} \mid S_{t+1}) = v_{t+1}^\pi$$

where v_{t+1}^π is the random variable such that

$$v_{t+1}^\pi(s') = \mathbb{E}(G_{t+1} \mid S_{t+1} = s')$$

Therefore

$$\mathbb{E}(G_{t+1} \mid S_t) = \mathbb{E}[v_{t+1}^\pi \mid S_t]$$

Plugging into $v_t^\pi(s)$, we get

$$\begin{aligned} v_t^\pi(s) &= \mathbb{E}_\pi[R_{t+1}] + \mathbb{E}_\pi[\gamma G_{t+1} \mid S_t = s] && (2.22) \\ &= \mathbb{E}_\pi[R_{t+1}] + \gamma \mathbb{E}_\pi[v_{t+1}^\pi \mid S_t = s] && (2.23) \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_{t+1}^\pi \mid S_t = s] && (2.24) \end{aligned}$$

□

2.6.2 For the action-value function

Theorem 4. *The action-value function captures the value of an action a taken by a policy π which can be computed recursively as*

$$q_t^\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_{t+1}^\pi \mid S_t = s, A_t = a]$$

or equivalently

$$q_t^\pi(s,a) = \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \sum_{a'} q_{t+1}^\pi \pi(a' \mid s')\Big]$$

Proof.

$$\begin{aligned}
q_t^\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] && (2.25)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && (2.26)\\
&= \mathbb{E}_\pi[R_{t+1} \mid S_t = s, A_t = a] + \mathbb{E}_\pi[\gamma G_{t+1} \mid S_t = s, A_t = a] && (2.27)\\
& && (2.28)
\end{aligned}$$

We have

$$\begin{aligned}
\mathbb{E}_\pi[G_{t+1} \mid S_t = s, A_t = a] &= \sum_{s'} \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s', S_t = s, A_t = a]\\
&\quad \times \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)\\
&= \sum_{s'} \sum_{a'} \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a', S_t = s, A_t = a]\\
&\quad \times \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)\mathbb{P}(A_{t+1} = a' \mid S_{t+1} = s')\\
&= \sum_{s'} \sum_{a'} q_{t+1}^\pi(s', a') p(s' \mid s, a) p(a' \mid s')\\
&= \sum_{s'} p(s' \mid s, a) \sum_{a'} q_{t+1}^\pi \pi(a' \mid s')\\
&= \sum_{s', r} p(s', r \mid s, a) \sum_{a'} q_{t+1}^\pi \pi(a' \mid s')
\end{aligned}$$

In addition , we have

$$\mathbb{E}_\pi[R_{t+1} \mid S_t = s, A_t = a] = \sum_{s', r} r p(s', r \mid s, a)$$

It follows that

$$q_t^\pi(s,a) = \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma \sum_{a'} q_{t+1}^\pi \pi(a' \mid s') \Big]$$

□

2.7 Solving and MDP

Solving a Markov Decision Process (MDP) involves finding an optimal policy that maximizes the expected cumulative reward in a given environment. This process typically requires determining the value function associated with the MDP, allowing us to evaluate states, actions, and state-action pairs. The optimal policy then dictates the best action to take in each state.

Various algorithms can be employed to solve an MDP, including dynamic programming, Monte Carlo methods, and reinforcement learning techniques. These algorithms iteratively estimate the optimal value function or policy by evaluating and improving the current estimate based on observed rewards and state transitions.

The ultimate objective of solving an MDP is to obtain a policy that guides the agent's decision-making process, enabling it to make optimal choices in uncertain environments to maximize the expected long-term reward.

In the following section, we will focus on solving the MDP using a dynamic programming approach.

3 Optimality results and guarantees

In this section, we will formalize the definition of an optimal value function and an optimal policy. We aim to prove the existence of an optimal policy within the context of a finite MDP. The proof relies on the concept of the Bellman operator, which we will define and demonstrate its properties. Building upon the previously introduced Bellman equations, this section will derive the Bellman optimality equations, enabling the computation of optimal value functions and consequently identifying optimal policies.

3.1 Optimal value function

The optimal state-value function $v^*(s)$ is the maximum value function over all policies

$$v^*(s) = \max_\pi v^\pi(s)$$

The optimal action-value function $q^*(s,a)$ is the maximum action-value function over all policies

$$q^*(s,a) = \max_\pi q^\pi(s,a)$$

The optimal value function specifies the best possible performance in the MDP. An MDP is *solved* when we know how to compute the optimal value function.

3.2 Optimal policy

We can define a partial ordering over policies such that

$$\pi \geq \pi' \equiv v^\pi(s) \geq v^{\pi'}(s) \quad \forall s$$

Definition 1. An policy π^* is said to be optimal if and only if , $\forall s$,

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

Theorem 5. *An optimal policy π^* also satisfies:*

$$q^{\pi^*}(s,a) = \max_\pi q^\pi(s,a) = q^*(s,a)$$

Proof. Since we have for any policy π

$$q_t^\pi(s,a) = \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_{t+1}^\pi]$$

Then,

$$\begin{aligned} q_t^{\pi^*}(s,a) &= \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_{t+1}^{\pi^*}] && (3.1)\\ &\geq \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_{t+1}^{\pi}] && (3.2)\\ &\geq q_t^\pi(s,a) && (3.3) \end{aligned}$$

Since this is valid for all π, then

$$q_t^{\pi^*}(s,a) \geq \max_\pi q^\pi(s,a)$$

which proves the theorem. □

3.3 Relationship between optimal state-value and action-value functions

In this section, we will establish the relationship between the optimal state-value function and the optimal action-value function. To achieve this, we require the following theorem.

Theorem 6. $\forall\ s \in \mathcal{S}$, *the optimal state-value and action value functions satisfy:*

$$v^*(s) = \max_{a\in\mathcal{A}} q^*(s,a)$$

Proof. Let's first show that

$$v^*(s) \leq \max_{a\in\mathcal{A}} q^*(s,a)$$

Let $s_0 \in \mathcal{S}$ and define $a_* \in \underset{a\in\mathcal{A}(s_0)}{\operatorname{argmax}}\, q^*(s_0,a)$. Then for any policy π, we have

$$\begin{aligned}
v_\pi(s_0) &= \sum_{a\in\mathcal{A}(s_0)} \pi(a \mid s_0) q^\pi(s_0,a) \\
&\leq \sum_{a\in\mathcal{A}(s_0)} \pi(a \mid s_0) q^*(s_0,a) \\
&\leq \sum_{a\in\mathcal{A}(s_0)} \pi(a \mid s_0) q^*(s_0,a_*) \\
&= q^*(s_0,a_*) \sum_{a\in\mathcal{A}(s_0)} \pi(a \mid s_0) \\
&= q^*(s_0,a_*) \\
&= \max_{a\in\mathcal{A}(s_0)} q^*(s_0,a)
\end{aligned}$$

Since this is true for all policies π, then it must be that

$$v^*(s_0) = \max_\pi v^\pi(s_0) \leq \max_{a\in\mathcal{A}(s_0)} q^*(s_0,a)$$

To show that

$$v^*(s) \geq \max_{a\in\mathcal{A}} q^*(s,a)$$

it would be enough to find a policy $\tilde{\pi}$ that satisfies

$$v^{\tilde{\pi}}(s) \geq \max_{a\in\mathcal{A}(s)} q^*(s,a)$$

Then by the optimality of v^* , we can achieve

$$v^*(s) \geq \max_{a\in\mathcal{A}} q^*(s,a)$$

Let's define $\tilde{\pi}$ to be the policy that for any states s returns an action a that is equal to $\operatorname{argmax}_a q^*(s,a)$. This policy $\tilde{\pi}$ is clearly deterministic and therefore

$$\begin{aligned}
v^{\tilde{\pi}}(s) &= q^{\tilde{\pi}}(s,\tilde{\pi}(s)) && (3.4)\\
&= \max_a q^*(s,a) && (3.5)
\end{aligned}$$

which concludes the proof. □

If an optimal policy π^* exists, then $\forall s \in \mathcal{S}, a \in \mathcal{A}$

$$v_{\pi^*}(s) = \max_{\pi} v^{\pi}(s) = v^*(s)$$

and

$$q^{\pi^*}(s,a) = \max_{\pi} q^{\pi}(s,a) = q^*(s,a)$$

Therefore, by the previous theorem,

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} q^{\pi^*}(s,a)$$

Since we have

$$q^{\pi^*}(s,a) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v_{\pi^*}(s)(S_{t+1}) \mid S_t = s, A_t = a]$$

Then

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v_{\pi^*}(s)(S_{t+1}) \mid S_t = s, A_t = a]$$

Having gained insights into the connection between value functions of a potential optimal policy, our current focus is on demonstrating the existence of such a policy.

3.4 Bellman optimality operator

Let V a set of values over states, that we call also a value function, defined as the vector

$$V = \{v(s) : s \in \mathcal{S}\} \in \mathbb{R}^{|\mathcal{S}|}$$

Then, we define the Bellman optimality operator $\mathbb{T}$ as the mapping

$$\mathbb{T} \colon \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$$

where

$$\begin{aligned} \mathbb{T}[V](s) &= \max_{a \in \mathcal{A}(s)} \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi^*}(s)(S_{t+1}) \mid S_t = s, A_t = a] && (3.6) \\ &= \max_{a} \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')] && (3.7) \end{aligned}$$

which is equivalent to

$$\mathbb{T}_{\pi}[V] = V_{\pi}$$

For an optimal policy π^* , the above writes as:

$$\mathbb{T}_{\pi^*}[V] = V_{\pi^*}$$

This means that the value function of the optimal policy is a fixed point of the Bellman optimality operator $\mathbb{T}$.

Theorem 7. *Let V_1 and V_2 be two value functions for two policies π_1 and π_2.*
If $V_1 \leq V_2$, then

$$\mathbb{T}[V_1] \leq \mathbb{T}[V_2]$$

We say that $\mathbb{T}$ is monotonic.

Proof. Let's consider two value functions V_1 and V_2, where $V_1 \leq V_2$ for all states.

Using the Bellman operator $\mathbb{T}$, the updated value functions can be written as:

$$\mathbb{T}[V_1](s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_1(s')]$$

$$\mathbb{T}[V_2](s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_2(s')]$$

Now, we need to show that $\mathbb{T}[V_1](s) \leq \mathbb{T}[V_1](s)$ for all states s . Since $V_1(s) \leq V_2(s) \quad \forall s,\ \gamma < 0$ then ,

$$\sum_{s',r} p(s', r \mid s, a)[r + \gamma V_1(s')] \leq \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_2(s')]$$

Taking the maximum over all actions a on both sides proves the monotonicity of the Bellman operator.

□

Theorem 8. *On the metric space $(\mathbb{R}^{|\mathcal{S}|}, \|.\|_\infty)$, where $\|.\|_\infty$ is so called infinity or supremum norm. The Bellman optimality operator $\mathbb{T}\colon \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ as defined above is a contracting mapping, i.e. there exists $0 \leq k < 1$ such that*

$$\|\mathbb{T}[V] - \mathbb{T}[V']\|_\infty \leq k\|V - V'\|_\infty$$

$\forall \quad V, V' \in \mathbb{R}^{|\mathcal{S}|}$

Proof.

$$\begin{aligned}
\|\mathbb{T}[V] - \mathbb{T}[V']\|_\infty &\leq \max_{s\in\mathcal{S}} \mid \mathbb{T}[V] - \mathbb{T}[V'] \mid \\
&= \max_{s\in\mathcal{S}} \mid \max_{a\in\mathcal{A}(s)} \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v_\pi(s)(S_{t+1}) \mid s, a] \\
&- \max_{a\in\mathcal{A}(s)} \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v'_\pi(s)(S_{t+1}) \mid s, a] \mid \\
&\leq \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}(s)} \mid \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v_\pi(s)(S_{t+1}) \mid s, a] \\
&- \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v'_\pi(s)(S_{t+1}) \mid s, a] \mid \\
&= \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}(s)} \mid \gamma \mathbb{E}_{\pi^*}[v_\pi(s)(S_{t+1}) - v'_\pi(s)(S_{t+1}) \mid s, a] \mid \\
&\leq \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}(s)} \gamma \mathbb{E}_{\pi^*}[\mid v_\pi(s)(S_{t+1}) - v'_\pi(s)(S_{t+1}) \mid\mid s, a] \\
&\leq \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}(s)} \gamma \mathbb{E}_{\pi^*}[\|V - V'\|_\infty \mid s, a] \\
&= \gamma\|V - V'\|_\infty
\end{aligned}$$

□

3.5 Existence of an optimal policy

Theorem 9. *For any Markov decision process,*

- *there exists an optimal policy π^* that is better or equal to all other policies,*

$$\pi^* \geq \pi, \quad \forall \pi$$

  There can be more than one such optimal policy.

- *All optimal policies achieve the optimal value function, $v^{\pi^*}(s) = v^*(s)$*
- *All optimal policies achieve the optimal action-value function, $q^{\pi^*}(s,a) = q^*(s,a)$*

Proof. We know from the above that if an optimal policy exists, then it should be a fixed point of the Bellman Optimality operator $\mathbb{T}$. In addition, we know that the Bellman optimality operator is a contracting map, and therefore , by the Banach fixed-point theorem (see Appendix) , has a fixed point which proves that an optimal policy exists.
By definition, this optimal policy realizes the optimal state-value optimal function.

Now that we know that an MDP $\mathcal{M}$ has an optimal policy π^* and this policy realizes both optimal value functions v^* and q^*, we would like to identify this policy. If we assume we have access to (know) q^*, we can construct the optimal policy by maximizing over $q^*(s,a)$

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{if } a = \text{argmax}_{a \in \mathcal{A}} q^*(s,a) \\ 0 & \text{otherwsie} \end{cases}$$

□

3.6 Bellman Optimality Equations

3.6.1 state-value function

Theorem 10. *Given an MDP $\mathcal{M}$, the optimal state value function obey*

$$v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) \mid S_t = s, A_t = a]$$

or equivalently

$$v^*(s) = \max_a \left[r(s,a) + \gamma \sum_{s' p(s'|a,s)} v^*(s') \right]$$

Proof. This has been shown in the above using

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} q^{\pi^*}(s,a)$$

and the definition of q^{π^*} □

3.6.2 action-value function

Theorem 11. *Given an MDP $\mathcal{M}$, the optimal action value function obey*

$$q^*(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') \mid S_t = s, A_t = a \Big]$$

or equivalently

$$q^*(s,a) = r(s,a) + \gamma \sum_{s' p(s'|a,s)} \max_{a' \in \mathcal{A}} q^*(s',a')$$

Proof. We have already shown that

$$q^{\pi^*}(s,a) = \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v_{\pi^*}(s)(S_{t+1}) \mid S_t = s, A_t = a]$$

and using thte fact that

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} q^{\pi^*}(s,a)$$

we get the desired result. □

The problem of calculating the value function for a given policy $v^\pi(s)$ is known as the prediction problem.

The problem of calculating the optimal value function v^* , and hence the optimal policy π^* , is known as the control problem.

4 Solving the prediction problem

Solving the prediction problem involves computing state-value or action-value functions for an arbitrary policy π. In this section, we will formalize a procedure (an algorithm) that leverages Bellman equations to iteratively calculate the value function of a policy.

We will begin by expressing the Bellman equations in a matrix format.

4.1 Bellman equations in matrix form

Let's assume that:

- $v_i = v(s_i)$
- $r_i^\pi = \mathbb{E}_\pi[R_{t+1} \mid S_t = s_i, A_t \sim \pi(S_t)]$
- $P_{ij}^\pi = p(s_j \mid s_i) = \sum_a \pi(a \mid s_i) p(s_j \mid s_i, a)$

Then the Bellman state-value equation

$$v_t^\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_{t+1}^\pi \mid S_t = s]$$

we can be expressed in matrix form as:

$$\boldsymbol{v} = \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}$$

This is a linear equation that be solved directly

$$\begin{aligned} (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi \boldsymbol{v})\boldsymbol{v} &= \boldsymbol{r}^\pi & (4.1) \\ \boldsymbol{v} &= (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi \boldsymbol{v})^{-1} \boldsymbol{r}^\pi & (4.2) \end{aligned}$$

While using a matrix representation for Bellman equations in finite Markov Decision Processes (MDPs) can be computationally efficient in certain cases, it also comes with its challenges. Some of the main challenges of the matrix representation in finite MDPs are as follows:

- Memory requirements: As the size of the state and action spaces increases, the matrices used to represent the Bellman equations can become quite large. This can lead to significant memory requirements, making it impractical or even infeasible to store and manipulate these matrices in memory.
- Curse of dimensionality: As the number of states and actions grows, the matrix size increases exponentially. This leads to the "curse of dimensionality," where computation and storage requirements escalate rapidly with the problem's dimensionality, making it challenging to handle large or high-dimensional MDPs.

4.2 Policy Evaluation

In order to evaluate a policy π, i.e., compute $v^{\pi}(s)$ for all $s \in \mathcal{S}$, we could use the Bellman equation:

$$v_t^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{t+1}^{\pi}(s')]$$

and leverage its recursive nature to iteratively compute the state value of a given policy provided that the model (environment) is defined

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')]$$

The iterative policy evaluation converges only in the limit.
Policy evaluation can be implemented in two ways:

- state-sweep value updates (as in algorithm 1): new values use old state values, which requires maintaining two data structures for old and new values
- In-place value updates (as in algorithm 2): each new value immediately overwrite the old one. Only one data structure (e.g. an array) is required to update the state values.

Both algorithms converge to the exact value v^{π}. However, usually, the in-place approach converges faster as it updates the state values sooner.

It's important to note that policy evaluation assumes a known MDP with the transition probabilities and reward functions provided. In practice, model-free methods like Monte Carlo or Temporal Difference learning can be used for policy evaluation in cases where the MDP model is not available.

5 Solving the control problem

Solving the control problem is achieved by solving the Bellman optimality equations. Since the equations are non-linear, we cannot use linear algebra and direct matrix solutions. Alternatively, we can leverage the recursive nature of Bellman optimality equations, to iteratively compute and identify the optimal value function and therefore the optimal policy.

In the following, we describe two main algorithms that aim to find the optimal policy: policy iteration and value iteration.

Algorithm 1 Policy Evaluation - State sweep

```
Require: A policy $\pi$ to be evaluated and an MDP $\mathcal{M}$
  $\theta > 0$ a small positive number
  $\Delta \leftarrow 0$
  Initialize $V$
  $\tilde{V} \leftarrow V$
  t=0
  while $\Delta > 0$ do
      for $s \in \mathcal{S}$ do
          $\tilde{V}_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_t(s')]$
          $\Delta \leftarrow \max(\Delta, \mid \tilde{V}_{t+1}(s) - V_t(s) \mid)$
      t = t+1
      $V \leftarrow \tilde{V}$
      Until $\Delta < \theta$
```

Algorithm 2 Policy Evaluation - In place

```
Require: A policy $\pi$ to be evaluated and an MDP $\mathcal{M}$
  $\theta > 0$ a small positive number
  $\Delta \leftarrow 0$
  Initialize $V$
  t=0
  while $\Delta > 0$ do
      for $s \in \mathcal{S}$ do
          $v = V_t(s)$
          $V_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_t(s')]$
          $\Delta \leftarrow \max(\Delta, \mid v - V_{t+1}(s) \mid)$
          t=t+1
      Until $\Delta < \theta$
```

5.1 Policy iteration

Policy iteration is a two-step iterative algorithm that alternates between

- policy evaluation (which we have already solved in the prediction problem) and
- policy improvement which we will present next, and consists of improving upon a present policy

until it converges to the optimal policy.

In other words, policy iteration is an iterative approach that alternates between policy evaluation and policy improvement. During policy evaluation, it estimates the value function of a given policy by solving the Bellman equations iteratively until convergence. This step computes how good each state is under the current policy. In policy improvement, the algorithm greedily updates the policy to select actions that lead to better state values. The process continues until the policy converges to an optimal one that maximizes the expected return for each state.

5.1.1 Policy Improvement

Policy improvement is a core step in the policy iteration algorithm used to solve MDPs. It involves updating the agent's current policy to create a new policy that achieves higher expected rewards.

In policy improvement, the agent evaluates the expected return (i.e., cumulative rewards) of different actions in each state based on its current policy. This evaluation is typically done using action-value functions , which estimate the expected rewards of taking specific actions in specific states.

Once the action-value functions are computed, the agent then constructs a new policy that selects actions with the highest action-value in each state. We will show that this new policy is guaranteed to be equal to or better than the previous one in terms of expected rewards. The process of using action-value functions to construct the improved policy is known as the policy improvement step.

Policy improvement is inspired in a way by the fact that given an optimal state-value function v^*, the optimal policy can be constructed by choosing the greedy action:

$$\begin{aligned}
\pi^*(s) &= \underset{a\in\mathcal{A}}{\operatorname{argmax}}\, q^*(s,a) && (5.1)\\
&= \underset{a\in\mathcal{A}}{\operatorname{argmax}}\, \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v_{\pi^*}(s)(S_{t+1}) \mid S_t = s, A_t = a] && (5.2)
\end{aligned}$$

The question becomes: if you construct a greedy policy based on an arbitrary state-value function, is this policy a strict improvement over π unless π if it is not already optimal? The following theorem provides a positive answer to this question.

Theorem 12. *If a policy π has no improvable states, then it is optimal, otherwise, if there exists a policy π' with*

$$q^\pi(s,\pi') = \mathbb{E}_{a\sim\pi'(a|s)}[q^\pi(s,a)]$$

such that

$$q^\pi(s,\pi') \geq v^\pi(s) \quad \forall \quad s \in \mathcal{S}$$

then it holds that

$$v^{\pi'}(s) \geq v^{\pi}(s) \quad \forall \quad s \in \mathcal{S}$$

This means that π' is at least as good as π.

Proof. If π cannot be improved (has no improvable states), then $\forall \quad s$, $\forall \quad \pi'$

$$\begin{aligned}
v^{\pi}(s) &\geq q^{\pi}(s, \pi') && (5.3)\\
&= \mathbb{E}_{a\sim\pi'(a|s)}[r(s,a) + \gamma v^{\pi}(s')] && (5.4)\\
&= \mathbb{E}_{a,a'\sim\pi'(a|s)}[r(s,a) + \gamma r(s',a') + \gamma^2 v^{\pi}(s'')] && (5.5)\\
&\geq \cdots && (5.6)\\
&\geq \mathbb{E}_{a,a'\sim\pi'(a|s)}[r(s,a) + \gamma r(s',a') + \gamma^2 r(s'',a'') + \cdots] && (5.7)\\
&= v^{\pi'}(s) && (5.8)
\end{aligned}$$

and therefore π is optimal.

In the other hand, if π can be improved by π', then by expanding q^{π}, we can get that $\forall s \in \mathcal{S}$:

$$\begin{aligned}
v^{\pi}(s) &\leq q^{\pi}(s, \pi') && (5.9)\\
&= \mathbb{E}_{a\sim\pi'(a|s)}[r(s,a) + \gamma v^{\pi}(s')] && (5.10)\\
&= \mathbb{E}_{a,a'\sim\pi'(a|s)}[r(s,a) + \gamma r(s',a') + \gamma^2 v^{\pi}(s'')] && (5.11)\\
&\leq \cdots && (5.12)\\
&\leq \mathbb{E}_{a,a'\sim\pi'(a|s)}[r(s,a) + \gamma r(s',a') + \gamma^2 r(s'',a'') + \cdots] && (5.13)\\
&= v^{\pi'}(s) && (5.14)
\end{aligned}$$

□

5.1.2 Policy iteration

Upon evaluation v^{π} of a policy π , it can be improved to produce a better policy π' which can be evaluated with $v^{\pi'}$ and improved again. A sequence of increasingly improving policies and value functions can be produced:

$$\pi_0 \xrightarrow{E} v^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} v^*$$

where $\xrightarrow{E}$ denotes a policy evaluation and $\xrightarrow{I}$ denotes a policy improvement.

As presented in algorithm 3, policy iteration is a two-step iterative algorithm that alternates between policy evaluation and policy improvement until it converges to the optimal policy.

- Policy Evaluation: In this step, the algorithm evaluates the value function for the current policy. It iteratively updates the value estimates for each state using the Bellman expectation equation until the value function converges.

- Policy Improvement: After policy evaluation, the algorithm improves the current policy by greedily selecting the best action in each state based on the updated value function. This step ensures that the policy becomes more optimal with respect to the current value function.

Figures 2 and 3 illustrate the policy iteration two steps mechanism.

In addition, since policy iteration performs a policy evaluation step, this step can be performed in-place as shown in 4.

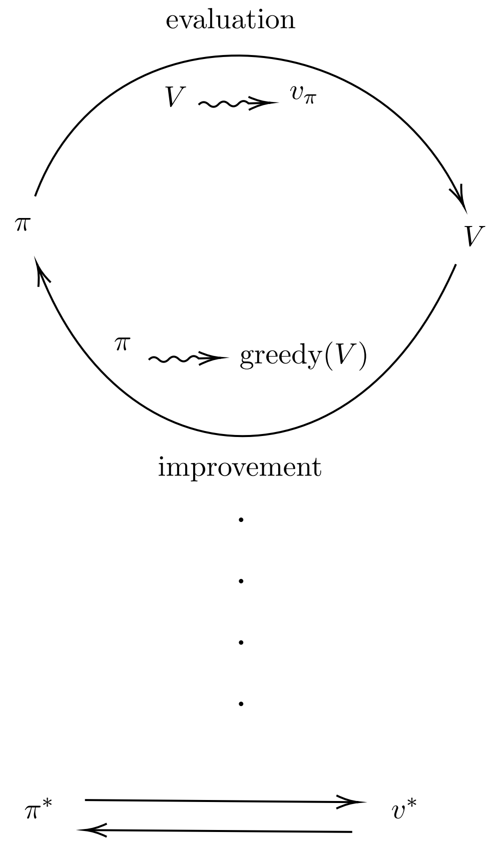
evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \text{greedy}(V)$
improvement
π^*
v^*

Figure 2: Policy iteration

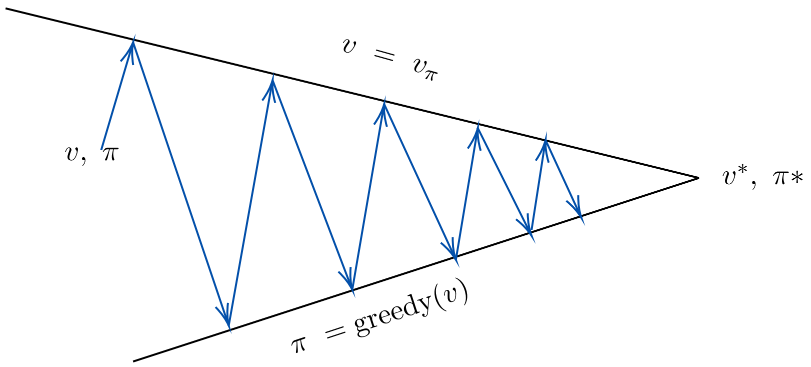

Figure 3: Policy iteration illustration

Algorithm 3 Policy Iteration - state sweep

Require: A policy π to be evaluated and an MDP $\mathcal{M}$

```
1. Initialization
$\theta > 0$ a small positive number
$\Delta \leftarrow 0$
$\tilde{V} \leftarrow V$
t=0

2. Policy Evaluation
while $\Delta > 0$ do
    for $s \in \mathcal{S}$ do
        $\tilde{V}_{t+1}(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_t(s')]$
        $\Delta \leftarrow \max(\Delta, \mid \tilde{V}_{t+1}(s) - V_t(s) \mid)$
    t = t+1
    $V \leftarrow \tilde{V}$
    Until $\Delta < \theta$

3. Policy Improvement
policy-stable $\leftarrow$ true
for $s \in \mathcal{S}$ do
    $\tilde{a} \leftarrow \pi(s)$
    $\pi(s) = \underset{a}{\text{argmax}} \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
    if $\tilde{a} \neq \pi(s)$, then policy-stable $\leftarrow$ false
if policy-stable then stop and return $\leftarrow$ false, then return $V$ and $\pi$; otherwise go to 2.
```

Algorithm 4 Policy Iteration - in place

```
Require: A policy $\pi$ to be evaluated and an MDP $\mathcal{M}$

  1. Initialization
  $\theta > 0$ a small positive number
  $\Delta \leftarrow 0$
  t=0

  2. Policy Evaluation
  while $\Delta > 0$ do
      for $s \in \mathcal{S}$ do
          $V_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_t(s')]$
          $\Delta \leftarrow \max(\Delta, \mid V_{t+1}(s) - V_t(s) \mid)$
          t = t+1
      Until $\Delta < \theta$

  3. Policy Improvement
  $policy\text{-}stable \leftarrow true$
  for $s \in \mathcal{S}$ do
      $\tilde{a} \leftarrow \pi(s)$
      $\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
      if $\tilde{a} \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
  if $policy\text{-}stable$ then stop and return $\leftarrow false$, then return $V$ and $\pi$; otherwise go to 2.
```

5.2 Value iteration

In policy iteration, policy evaluation is executed until convergence in each step. Then, the policy is improved/updated and the whole process repeats.
However, based on the state-value Bellman optimal equation

$$v^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) \mid S_t = s, A_t = a]$$

we could reduce the evaluation stage of policy iteration down to a single sweep of the states and combine policy evaluation and policy improvement in one single step as shown in algorithm 5. In fact, value iteration is simpler than policy iteration. It is a one-step iterative algorithm that directly updates the value function in each iteration without explicitly maintaining a policy.

It starts with an arbitrary value function and repeatedly applies the Bellman optimality equation to improve the value estimates. The algorithm converges to the optimal value function after a finite number of iterations. Once the optimal value function is obtained, an optimal policy can be easily derived by selecting actions that lead to the maximum value in each state.

While policy iteration guarantees convergence to the optimal policy in a finite number of steps, value iteration tends to converge faster as it directly optimizes the value function. Both methods have their strengths and are widely used in various RL applications. However, the choice between policy iteration and value iteration depends on factors such as the problem complexity, computational resources, and convergence speed required for a particular application.

Algorithm 5 Value Iteration - State sweep

```
Require: A policy $\pi$ to be evaluated and an MDP $\mathcal{M}$
  $\theta > 0$ a small positive number
  $\Delta \leftarrow 0$
  Initialize $V$
  $\tilde{V} \leftarrow V$
  $t = 0$
  while $\Delta > 0$ do
      for $s \in \mathcal{S}$ do
          $\tilde{V}_{t+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_t(s')]$
          $\Delta \leftarrow \max(\Delta, \mid \tilde{V}_{t+1}(s) - V_t(s) \mid)$
      $V_{t+1} \leftarrow \tilde{V}_{t+1}$
      $t \leftarrow t + 1$
      Until $\Delta < \theta$
  Output a deterministic policy $\pi \sim \pi^*$, such that:
  for $s \in \mathcal{S}$ do
      $\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
```

5.3 Asynchronous value iteration

In both policy evaluation, and optimization (finding the optimal policy), we have tried to refine and enhance the dynamic programming approach by :

- suggesting in-place version of policy evaluation instead of full state-sweep

- suggesting value iteration instead of policy iteration

In both cases, the tendency is to avoid a full state sweep required in the standard dynamic programming methods that can be computationally expensive (if the state space is very large) and may slow down convergence and locks the algorithm in long sweeps before making any progress. In fact, in updating one state at a time, interim results for other states are incorporated.

The in-place approach can be further applied to the value iteration as presented in algorithm 6. However, convergence is not guaranteed unless all states continue to be selected.

The asynchronous value iteration approach is motivated by the following theorem.

Theorem 13. *For an infinite sequence of states*

$$s^{(0)}, s^{(1)}, s^{2}, \cdots$$

such that every state $s \in \mathcal{S}$ occurs infinitely often.
Let the operators $\mathbb{T}_{s^{(k)}}$ defined as

$$V_{k+1}(s) = \mathbb{T}_{s^{(k)}}[V](s) = \begin{cases} \mathbb{T}[V](s) & \textit{if } s = s^{(k)} \\ V_k(s) & \textit{otherwise} \end{cases}$$

then the asynchronous *value iteration that initializes V and applies in sequence:*

$$\mathbb{T}_{s^{(0)}}, \mathbb{T}_{s^{(1)}}, \mathbb{T}_{s^{(2)}}, \cdots$$

converges to v^*

Proof. Let l_1 be a sequence such that all states have appeared at least once in: $s^{(0)}, s^{(1)}, s^{2}, \cdots, s^{l_1}$
Let l_2 be a sequence such that all states have appeared at least once in: $s^{(l_1+1)}, s^{(l_1+2)}, s^{l_1+3}, \cdots, s^{l_2}$
Similarly, we can define $l_3, l_4, \cdots$
To show that the asynchronous value iteration converges to v^*, it is enough to show that $\forall\ i$, we have that the combined operator $\mathbb{T}_{s^{(l_{i+1})}} \circ \cdots \circ \mathbb{T}_{s^{(l_i)}}$ is a contraction, i.e., for any V_1 , V_2, we have that

$$\| \mathbb{T}_{s^{(l_{i+1})}} \circ \cdots \circ \mathbb{T}_{s^{(l_i)}}[V_1] - \mathbb{T}_{s^{(l_{i+1})}} \circ \cdots \circ \mathbb{T}_{s^{(l_i)}}[V_2] \|_\infty \leq \gamma \| V_1 - V_2 \|_\infty$$

Since all states appear in the sequence l_i, then we can see that the operator $\mathbb{T}_{s^{(l_{i+1})}} \circ \cdots \circ \mathbb{T}_{s^{(l_i)}}$ will always applies the Bellman operator $\mathbb{T}$. Since $\mathbb{T}$ is contracting, then $\mathbb{T}_{s^{(l_{i+1})}} \circ \cdots \circ \mathbb{T}_{s^{(l_i)}}$ is contracting as well, which guarantees the convergence of the asynchronous value iteration scheme. □

So, asynchronous value iteration does not necessarily require every state to appear in every iteration for it to converge. In fact, one of the main advantages of asynchronous value iteration over synchronous value iteration is that it allows updating the values of states asynchronously, meaning that not all states are updated simultaneously in each iteration.

In asynchronous value iteration, the values of states are updated based on their successor states and the immediate rewards obtained by taking actions. The algorithm selects states to update based on a certain policy or strategy, and not all states need to appear in every iteration.

However, for asynchronous value iteration to guarantee convergence, it must satisfy the condition of *policy evaluation.* This means that every state should have a non-zero probability of being selected for update during the policy evaluation step. In other words, all states' values need to be updated infinitely often during the course of the algorithm's iterations.

The policy used to select states for update should ensure sufficient exploration of the state space to avoid getting stuck in local optima or missing critical states. Proper exploration is essential to ensure that the algorithm can converge to the optimal value function.

In summary, asynchronous value iteration does not require every state to appear in every iteration, but it does require all states' values to be updated infinitely often, which can be achieved through proper policy evaluation and exploration strategies. By satisfying this condition, the algorithm will converge to the optimal value function for the given Markov Decision Process (MDP).

In asynchronous value iteration, ensuring that all states' values are updated infinitely often is essential for convergence. There are several strategies to achieve this goal, and researchers have proposed different methods to select states for update. Here are some common strategies along with references to relevant literature:

- Random Asynchronous Updates: In this strategy [10], states are randomly selected for update during each iteration. The randomness ensures that all states have a non-zero probability of being selected at any point, leading to sufficient exploration. This approach is simple to implement and can be effective, especially in situations with a large state space.

- Prioritized Sweeping: Prioritized sweeping [11] is a strategy that focuses on updating states that have the most impact on the overall value function. States that are expected to have a large Bellman error (difference between the current estimate and the updated estimate) are prioritized for update. This approach can improve the efficiency of the algorithm by updating critical states first.

- Trajectory Sampling: In this strategy [12], the agent collects trajectories or experiences while interacting with the environment. States encountered during these trajectories are then used for value updates. This approach leverages real experiences to drive the updates and can be particularly useful in scenarios with complex or unknown state spaces.

- Stochastic Updates with Exploration Bonuses: In this approach [13], states are chosen for update based on a combination of the Bellman error and an exploration bonus term. The exploration bonus encourages the selection of less visited states, promoting exploration and preventing premature convergence.

These strategies are just a few examples of the many possible approaches to selecting states for update in asynchronous value iteration. Each strategy has its strengths and weaknesses depending on the specific problem domain and state space characteristics.

Apart from policy iteration and value iteration, other notable methods for solving MDPs and RL problems include Q-learning, SARSA, Monte Carlo methods, and function approximation techniques like Deep Q Networks (DQNs) and Policy Gradient methods. These approaches cater to different scenarios and have unique advantages and limitations. Researchers and practitioners often select the most suitable algorithm based on the specific characteristics of the problem and the desired trade-offs between computational complexity and performance.

Algorithm 6 Value Iteration - In place

```
Require: A policy $\pi$ to be evaluated and an MDP $\mathcal{M}$
  $\theta > 0$ a small positive number
  $\Delta \leftarrow 0$
  Initialize $V$
  $t = 0$
  while $\Delta > 0$ do
      for $s \in \mathcal{S}$ do
          $V_{t+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_t(s')]$
          $\Delta \leftarrow \max(\Delta, \mid V_{t+1}(s) - V_t(s) \mid)$
          $t \leftarrow t + 1$
      Until $\Delta < \theta$
  Output a deterministic policy $\pi \sim \pi^*$, such that:
  for $s \in \mathcal{S}$ do
      $\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
```

6 Illustration

In this section, we use the Frozen Lake environment to illustrate the Markov Decision Processes (MDPs) and concepts that we have presented so far.
The Frozen-Lake is in fact a classic example that consists of a 4×4 grid where each cell is either:

- a starting cell (S),
- a frozen cell (F) indicating a safe position,
- a hole (H) which must be avoided,
- or the goal (G) which must be reached.

Any agent starts at S and needs to find its way to G via F avoiding H. The agent's actions result in deterministic transitions and work as intended. The reward is 1 if you reach G and 0 otherwise.
If the agent falls into a hole or reaches the goal the game restarts.

The states are the positions on the grid. The actions available to the agent are:

- Move up
- Move down
- Move left
- Move right

So we have a total of 16 states and 4 actions which are illustrated in figure 4.

The aim now is for the agent to navigate this lake that contains holes and reach the goal (G) without falling in a hole (H) .

The Frozen Lake environment comes in two flavors:

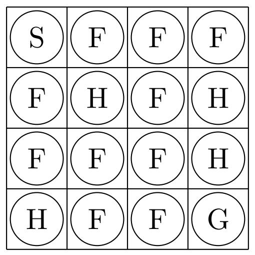

Figure 4: Illustration of 16 states frozen lake environment

- non-slippery: when the agent decides to move in any direction, it is not guaranteed that she will end up in the desired state since that could
- slippery: when the agent decides to move in any direction, it is not guaranteed that she will end up in the desired state since that could

The implication of this on the transition probabilities:

- Non-Slippery Frozen Lake: In the non-slippery version, the transition probabilities are deterministic, meaning that the agent's chosen action will lead to the desired next state with a probability of 1. In other words, if the agent chooses a specific action in a particular state, it will always end up in the intended next state.
- Slippery Frozen Lake: In the slippery version, the transition probabilities are stochastic, meaning that the agent's chosen action may not always lead to the intended next state. There is some probability of ”slipping” and ending up in an unintended neighboring state. Typically, the slipping probability is higher when the environment is slippery. For example, in the original OpenAI Gym FrozenLake environment, the slipping probability is 0.33 for each non-goal action.

We implemented state sweep and in-place policy evaluation algorithms and evaluated 1000 arbitrarily generated policies using both methods. We found that both algorithms converged to the same state value function, and the in-place algorithm required around 22% fewer iterations than the state sweep algorithm.

We have also implemented policy improvement, policy iteration and value iteration algorithms (both state-sweep and in-place for policy and value iterations).

This notebook illustrates its application to the frozen lake environment.

Both policy iterations and value iterations converge to the optimal policy.

7 Conclusion

In this note, we present a comprehensive derivation of the Markov Decision Process (MDP) Bellman equations, accompanied by rigorous proofs of the related optimality results. Each proof step is thoroughly clarified for better understanding, making this material accessible to readers with various levels of expertise.

Moreover, we provide detailed explanations of the policy iteration and value iteration algorithms, essential tools for finding optimal policies and value functions in MDPs. By understanding these algorithms, readers can gain practical insights into the process of decision-making under uncertainty and learn how to design efficient and effective strategies for complex tasks.

Additionally, to enhance the practicality of our exposition, we include an illustrative example on a benchmark problem that demonstrates the application of the concepts discussed. This example allows readers to see MDPs in action, showcasing how an agent navigates through a dynamic environment, making decisions that maximize cumulative rewards.

To further facilitate exploration and implementation, we have made the accompanying code available in a public GitHub repository. By providing access to the code, we aim to encourage hands-on experimentation and foster a deeper understanding of MDPs' practical implementation in real-world scenarios.

We hope that this comprehensive note serves as a valuable resource for students, researchers, and practitioners interested in understanding and utilizing Markov Decision Processes as a powerful framework for modeling and solving complex decision-making problems under uncertainty.

Author Details

Oualid Missaoui is an SVP, Quantitative Researcher and Data Scientist at Jefferies LLC, New York, NY. You can contact the author via email at `oualid.missaoui@gmail.com`.

References

[1] Richard S. Sutton and Andrew G. Barto (2018) Reinforcement Learning. An Introduction. The MIT Press. Second Edition.

[2] Csaba Szepesvari (2010) Algorithms for Reinforcement Learning. Morgan and Claypool Publishers. First Edition.

[3] Martin L. Puterman (2005) Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience. First Edition.

[4] Dimitri Bertsekas (2019) Reinforcement Learning and Optimal Control. Athena Scientific. First Edition.

[5] Dimitri Bertsekas, John N. Tsitsiklis (1996) Neuro-Dynamic Programming. Athena Scientific. First Edition.

[6] Laura Graesser, Wah Loon Keng (2020) Foundations of Deep Reinforcement Learning. Theory and Practice in Python. Addison-Wesley . First Edition.

[7] Warren Powell (2022) Reinforcement Learning and stochastic optimization . Wiley. First Edition.

[8] Ashwin Rao and Tikhon Jelvis (2023). Foundations of Reinforcement Learning with Applications in Finance. CRC Press. First Edition.

[9] Volodymyr Mnih et al. (2016) Asynchronous Methods for Deep Reinforcement Learning. Journal of Machine Learning Research.

[10] Watkins, C. J. C. H., and Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292.

[11] Moore, A. W., and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning, 13(1), 103-130.

[12] Maei, H. R., and Sutton, R. S. (2010). GQ (λ) and Off-Policy TD (λ). In Proceedings of the 27th International Conference on Machine Learning (ICML-10), 719-726.

[13] Osband, I., and Van Roy, B. (2013). More effective feature construction and evaluation via reinforcement learning bias. In Proceedings of the 30th International Conference on Machine Learning (ICML-13), 109-117.

A Preliminaries

Theorem 14. *Let X, Y, Z be random variables. Assuming all the following expectations exist, we have*

- $\mathbb{E}[\mathbb{E}[X \mid Y]] = \mathbb{E}[X]$
- $\mathbb{E}[\mathbb{E}[X \mid Y, Z] \mid Y] = \mathbb{E}[X \mid Y]$

Proof.

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[X \mid Y]] &= \sum_y \mathbb{E}[X \mid Y = y] P(y) && \text{(A.1)}\\
&= \sum_y \sum_x x P(x|y) P(y) && \text{(A.2)}\\
&= \sum_x \sum_y x P(x, y) && \text{(A.3)}\\
&= \sum_x x \sum_y P(x, y) && \text{(A.4)}\\
&= \sum_x x P(x) && \text{(A.5)}\\
&= \mathbb{E}[X] && \text{(A.6)}\\
& && \text{(A.7)}
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[X \mid Y, Z] \mid Y = y] &= \sum_z \mathbb{E}[X \mid Y = y, Z = z] P(Z = z | Y = y) && \text{(A.8)}\\
&= \sum_z \sum_x x P(X = x | Y = y, Z = z) P(Z = z, Y = y) \frac{1}{P(Y = y)} && \text{(A.9)}\\
&= \sum_z \sum_x x P(X = x, Y = y, Z = z) \frac{1}{P(Y = y)} && \text{(A.10)}\\
&= \sum_x x \frac{1}{P(Y = y)} \sum_z P(X = x, Y = y, Z = z) && \text{(A.11)}\\
&= \sum_x x \frac{1}{P(Y = y)} P(X = x, Y = y) && \text{(A.12)}\\
&= \sum_x x P(X = x \mid Y = y) && \text{(A.13)}\\
&= \mathbb{E}[X \mid Y = y] && \text{(A.14)}\\
& && \text{(A.15)}
\end{aligned}$$

□

B Metric space

Definition 2. A metric on a set $\mathcal{X}$ is a function called the distance $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of non-negative real numbers. For all $x, y, z \in \mathcal{X}$, this function is required to satisfy the following conditions:

- $d(x, y) \geq 0$ (non-negativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Definition 3. A metric space $(\mathcal{X}, d)$ is a set $\mathcal{X}$ together with a metric d on the set.

Definition 4. A sequence of elements x_n is a Cauchy sequence if and only if for every $\epsilon > 0$, there is some $N \geq 1$ such that

$$d(x_n, x_m) > \epsilon \quad \forall \quad m, n \geq N$$

Definition 5. A metric space $\mathcal{X}$ is complete if and only if every Cauchy sequence in $\mathcal{X}$ converges in $\mathcal{X}$.

C Banach fixed-point theorem

Definition 6. Let (M, d) be a complete metric space. A mapping $T \colon M \to M$ is a contraction mapping if there exists $0 \leq k < 1$ such for any x and y in M, we have

$$d(T(x), T(y)) \leq kd(x, y)$$

Theorem 15. *Let (M, d) be a non-empty complete metric space with a contraction mapping $T \colon M \to M$. Then T admits a unique fixed-point x^* in M (i.e., $T(x^*) = x^*$). Furthermore, x^* can be found as follows: start with an arbitrary element x_0 in M and define a sequence $\{x_n\}$ by $x_n = T(x_{n-1})$*

Proof. Let $x_0 \in M$ be arbitrary and define a sequence $(x_n)_{n\in\mathbb{N}}$ by setting $x_n = T(x_{n-1})$.
First, let's show by induction that

$$d(x_{n+1}, x_n) \leq k^n d(x_1, x_0)$$

In fact, for $n = 0$, it is clear that

$$d(x_{0+1}, x_0) \leq k^0 d(x_1, x_0)$$

Now, assume that the inequality holds for n

$$d(x_{n+1}, x_n) \leq k^n d(x_1, x_0)$$

and let's prove it for $n + 1$

$$d(x_{n+2}, x_{n+1}) \leq k^{n+1} d(x_1, x_0)$$

Since T is a contracting mapping, then

$$\begin{aligned} d(T(x_{n+1}), T(x_n)) &\leq kd(x_{n+1}, x_n) && \text{(C.1)} \\ d(x_{n+2}, x_{n+1}) &\leq k \times k^n d(x_1, x_0) && \text{(C.2)} \\ &\leq k^{n+1} d(x_1, x_0) && \text{(C.3)} \\ & && \text{(C.4)} \end{aligned}$$

Second, if we show that $(x_n)_{n\in\mathbb{N}}$ is a Cauchy sequence, then by the completness of M, it converges to x^* in M.

Let m, $n \in \mathbb{N}$ such that $m > n$

$$\begin{aligned}
d(x_m, x_n) &\leq d(x_m, x_{m-1}) + d(x_{m-1}, x_{m-2}) + \cdots + d(x_{n+1}, x_n) && \text{(C.5)}\\
&\leq k^{m-1}d(x_1, x_0) + k^{m-2}d(x_1, x_0) + \cdots + k^n d(x_1, x_0) && \text{(C.6)}\\
&= k^n d(x_1, x_0)\sum_{l=0}^{m-n-1} k^l && \text{(C.7)}\\
&\leq k^n d(x_1, x_0)\sum_{l=0}^{\infty} k^l && \text{(C.8)}\\
&= k^n d(x_1, x_0)\frac{1}{1-k} && \text{(C.9)}
\end{aligned}$$

For an arbitrary $\epsilon > 0$, we can find N such that
$k^n d(x_1, x_0)\frac{1}{1-k} \leq \epsilon$ which means

$$n \leq \frac{1}{\log k}[\log \epsilon(1-k) - \log(d(x_1, x_0))]$$

This proves that the sequence $(x_n)_{n\in\mathbb{N}}$ is a Cauchy sequence.

In addition, using the continuity of T (since it is a contraction map) to justify bringing a limit inside T, we have

$$\begin{aligned}
x^* &= \lim_{n\to\infty} x_n && \text{(C.10)}\\
&= \lim_{n\to\infty} T(x_{n-1}) && \text{(C.11)}\\
&= T(\lim_{n\to\infty} x_{n-1}) && \text{(C.12)}\\
&= T(x^*) && \text{(C.13)}
\end{aligned}$$

Therefore, x^* must be a fixed point of T. □

Lemma 16. *Given a finite set X, for any two real value function $f\colon X \to \mathbb{R}$ and $g\colon X \to \mathbb{R}$, we have*

$$\mid \max_{x\in X} f(x) - \max_{x\in X} g(x) \mid \leq \max_{x\in X} \mid f(x) - g(x) \mid$$

Proof. We have

$$\begin{aligned}
f(x) - g(x) &\leq \mid f(x) - g(x) \mid && \text{(C.14)}\\
f(x) &\leq \mid f(x) - g(x) \mid + g(x) && \text{(C.15)}\\
\max_{x\in X} f(x) &\leq \max_{x\in X}[\mid f(x) - g(x) \mid + g(x)] && \text{(C.16)}\\
&\leq \max_{x\in X}[\mid f(x) - g(x) \mid] + \max_{x\in X}[g(x)] && \text{(C.17)}\\
\max_{x\in X} f(x) - \max_{x\in X}[g(x)] &\leq \max_{x\in X}[\mid f(x) - g(x) \mid] && \text{(C.18)}
\end{aligned}$$

Similarly, we can show that

$$\max_{x\in X} g(x) - \max_{x\in X}[f(x)] \leq \max_{x\in X}[\mid g(x) - f(x) \mid]$$

Therefore

$$\mid \max_{x\in X} f(x) - \max_{x\in X} g(x) \mid \leq \max_{x\in X} \mid f(x) - g(x) \mid$$

$\square$