Hamburg University of Technology

Bachelor's Thesis

Niklas Dieckow

Methods of Machine Learning and their Application to the Repair Limit Replacement Problem

September 16, 2021

Supervised by:
Prof. Karl-Heinz Zimmermann
M.Sc. Merve Nur Cakir

Hamburg University of Technology (TUHH)
Technische Universität Hamburg
Institute of Embedded Systems
21073 Hamburg

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum	Unterschrift

Contents

1 Introduction

Over the course of our lives as humans, we learn a variety of skills and behaviors that allow us to navigate everyday life with more ease. This learning process is largely unconscious, and usually motivated by a desire to improve certain outcomes. Every such learning experience seems to follow a general protocol:

1. Set up an objective,
2. take action toward that objective,
3. observe how well you did,
4. figure out what to do differently in order to do better,
5. repeat steps 2-4 until you consistently do well (or give up).

This is already a basic learning algorithm, and those that are used in practice to solve machine learning problems possess the same high-level structure – although a computer will usually not give up, unless you tell it to.

Despite being verbal and not very precise, each of these steps highlights a component that is necessary for the learning process and hence for the construction of a proper learning algorithm. Step 1 requires an *objective*, some goal to work toward. For us humans, this is typically a skill, for example, riding a bike. For computers, we will see that the objective is usually numerical in nature, such as locating the maximum or minimum of a function.

Step 2 highlights that there needs to be a set of actions to perform. In the human analogy, these are hard to define clearly. After all, does "trying to ride the bike" already count as an action? Or do we need to be more precise, and define the actions to be "mount the bike", "push the right pedal down", "push the left pedal down", and so forth? A computer algorithm cannot deal with such ambiguities and needs a well-defined set of actions.

Step 3 indicates the necessity of some way to measure how well one performed the task at hand. In the human example, these are external observations, such as "fell from the bike" or "received praise from parent". Since computers like to work with numerical values, *reward* or *cost* values are used.

Step 4 indicates that there is some *strategy* the learner is following – something that dictates which action to perform under which circumstances. Depending on the results of step 3, this strategy may need to be altered, so that better feedback is received in the next iteration of step 3. Finally, step 5 requires some notion of "good enough" – a terminating condition.

The field of *reinforcement learning* is concerned with finding an optimal action strategy for an agent who is placed in a (possibly unknown) environment. Optimal, in this case, refers to either maximizing reward or minimizing cost. The five steps outlined above are closely related to this pursuit, together with the highlighted necessary components. We will find the latter to be incorporated into the mathematical model introduced in the next chapter.

1.1 Origins of Reinforcement Learning

Reinforcement learning originated with the combination of two independent and seemingly unrelated fields – animal learning, and optimal control. The first field concerns itself with the study of how animals learn. One of the earliest discoveries in this field is what the psychologist Edward Thorndike in 1911 called the "Law of Effect". It describes the observation that an animal's response which is followed by a rewarding stimulus is more likely to recur, while a response followed by a discomforting stimulus is less likely to recur. Nowadays, this concept of learning by reward-and-punishment is common knowledge and most often utilized to train animals.

The second field, optimal control, emerged in the 1950s and concerns itself with finding an optimal controller for a dynamical system such that some objective function is minimized. One approach to the optimal control problem, developed by Richard Bellman, consists of setting up a functional equation, nowadays known as the *Bellman equation*, and solving it using dynamic programming. This approach is of great use in the theory of reinforcement learning.

A much more in-depth coverage of the history of reinforcement learning can be found in Chapter 1 of [8], which the brief history in this section is also based on.

1.2 Structure of this Thesis

In Chapter 2, we will deal with the theory behind reinforcement learning. After covering some concepts from probability theory, the notion of a finite *Markov decision process* will be introduced, which serves as a flexible mathematical model of the environment of a reinforcement learning problem. Subsequently, the notion of an *optimal policy*, which can be seen as the solution to an reinforcement learning problem, is introduced and existence as well as uniqueness of optimal policies are discussed. This discussion naturally transitions into the presentation of an algorithm that computes an optimal policy. In the last section of the chapter, we exit the realm of mathematical idealism and consider reinforcement learning problems with unknown or partially known rewards and transition probabilities. Such problems require interaction with the environment. The algorithm we will familiarize ourselves with in this section, called *Q-Learning*, will in fact have the same basic structure as the five-step "algorithm" from above.

In Chapter 3, the *repair limit replacement problem* is introduced. We will use the tools covered in the previous chapter to model a simple variant of this problem, which aims to find an optimal repair-replace-strategy, given that repair limits are already known. Finally, the two algorithms introduced in the previous chapter will be applied to an example instance of this problem, and the results will be compared.

The thesis concludes with Chapter 4, in which the topics of the thesis are summarized and possible future work is discussed.

2 Theory

The aim of this chapter is to give the reader a solid introduction into the theory behind Markov decision processes and reinforcement learning. At first, some preliminaries are discussed, namely stochastic processes, the Markov property, as well as Markov chains. The reader is assumed to be familiar with basic notions from probability theory, particularly the notion of a random variable. Throughout, let Ω be a suitable sample space.

2.1 Preliminaries

Definition 2.1 (Stochastic process)**.** A *stochastic process* is a collection of random variables $(X_t)_{t\in T}$, where T is an arbitrary index set.

If T is at most countable, we speak of a *discrete-time* process, whereas we speak of a *continuous-time* process, if T is uncountable. Commonly, T is a subset of $\mathbb{R}$, and in the discrete-time case, we usually choose $T = \mathbb{N}_0$. The set of values that X_t can take on, for any $t \in T$, is termed *state space* and denoted by S.

Example 2.2.

a) Suppose $X_t \in \{0, 1\}$ for all $t \in T$ and $\mathbb{P}(X_t = 1) = p$ for some $p \in [0, 1]$. Then $(X_t)_{t\in T}$ is a particular type of stochastic process, known as a Bernoulli process.

b) Suppose $T = \mathbb{N}_0$, $X_t \in \{0, 1\}$ and

$$\mathbb{P}(X_{t+1} = 1 \mid X_t = 0) = 0.5, \qquad \mathbb{P}(X_{t+1} = 1 \mid X_t = 1) = 1.$$

Then, $(X_t)_{t\in T}$ is a stochastic process.

c) $T = \{0, 1, 2\}$, $X_t \in \{\mathbf{R}, \mathbf{G}, \mathbf{B}\}$. We consider an urn containing a red, green and blue marble, each. At every time step, a marble is removed from the urn, and not put back. Define X_t to be the color of the marble removed at time t. Then, $(X_t)_{t\in T} = (X_0, X_1, X_2)$ is also a stochastic process.

❖

Definition 2.3 (Markov property)**.** A stochastic process $(X_t)_{t\in T}$ with $T = \mathbb{N}_0$ is said to possess the *Markov property*, if it fulfills

$$\mathbb{P}(X_{t+1} = x \mid X_1 = x_1, \dots, X_t = x_t) = \mathbb{P}(X_{t+1} = x \mid X_t = x_t)$$

for all $t \in T$, $x, x_1, \dots, x_t \in S$. A stochastic process with this property is also called a *Markov process.*

In other words, the transition probabilities of a Markov process only depend on the current state, while information from previous states is ignored. Example b) from above is a Markov process, while a) is one if $T = \mathbb{N}_0$. Example c), on the other hand, does not possess the Markov property: Suppose we would like to know the probability that the marble removed at $t = 2$ is red, given that the other two marbles were removed at the previous two time steps. It is easy to see that

$$\mathbb{P}(X_2 = \mathbf{R} \mid X_1 = \mathbf{G}, X_0 = \mathbf{B}) = 1,$$

as the red marble is the only one left. But if we drop the information about $t = 0$, we have

$$\mathbb{P}(X_2 = \mathbf{R} \mid X_1 = \mathbf{G}) = \frac{\mathbb{P}(\{X_2 = \mathbf{R}\} \cap \{X_1 = \mathbf{G}\})}{\mathbb{P}(X_1 = \mathbf{G})} = \frac{1}{2},$$

which can be quickly confirmed by looking at the six possible permutations of $\{\mathbf{R}, \mathbf{G}, \mathbf{B}\}$. Hence, the process from Example c) does not satisfy the Markov property.

Definition 2.4 (Markov chain)**.** A Markov process $(X_t)_{t \in T}$ with an at most countable state space is called a *Markov chain.*

From the above examples, b) is a Markov chain and a) is one, if $T = \mathbb{N}_0$. Because c) is not a Markov process, it cannot be a Markov chain.

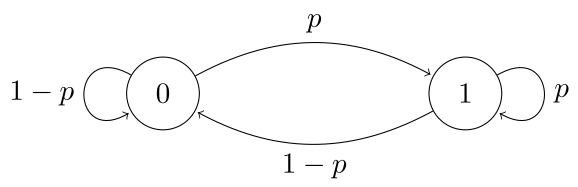
Definition 2.5 (Transition probability matrix)**.** If the transition probabilites $p_{ij} := \mathbb{P}(X_{t+1} = j \mid X_t = i)$ are independent of the chosen time $t \in T$, a Markov chain with state space $S = \{1, \ldots, n\}$, $n \in \mathbb{N}$, can be described by its *transition probability matrix* P, defined as

$$P = (p_{ij}) = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{pmatrix}.$$

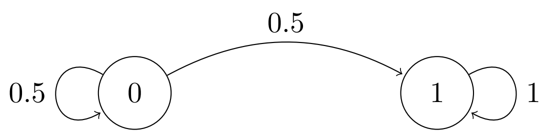
This goes hand-in-hand with the definition of the transition graph of a Markov chain.

Definition 2.6 (Transition graph)**.** Let $P = (p_{ij})$ be the transition probability matrix of a Markov chain. Then, one can visualize a Markov chain by drawing its transition graph (V, E) with $V = S$, $E = \{(i, j) \mid p_{ij} > 0\}$ and edge labels p_{ij}.

Example 2.7. The transition graph of example a) looks like this:



The transition graph of example b) looks as follows:



❖

In the following section, we will introduce an extension to Markov chains, which allows for an agent to perform actions that can influence the transition probabilities.

2.2 Markov Decision Processes

We observe a system in a discrete-time universe, which, at any time step $t \in T \subset \mathbb{N}_0$, is in exactly one state s out of many contained in a finite state set $\mathcal{S} = \{s^{(1)}, \dots, s^{(N)}\}$. The system is under the influence of a decision maker who, at each time step, performs an action a out of a set of actions $\mathcal{A}$, thereby potentially altering the state of the system.

The behavior of the system in state s upon an action a performed at time step t is described by the conditional probability distribution $p(s' \mid s, a)$, in which $s' \in \mathcal{S}$ is the state of the system at time $t + 1$. Upon performing an action, the decision-maker receives an immediate reward $\rho_a(s, s') \in \mathbb{R}$. More precisely, there exists a reward function $\rho\colon \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and we write $\rho_a(s, s')$ instead of $\rho(s, s', a)$.

Definition 2.8 (Markov decision process)**.** We call the quadruple $(\mathcal{S}, \mathcal{A}, p, \rho)$ a *Markov decision process*, or Markov decision problem – referred to as MDP from now on.

If the set of time steps, T, is finite, we call the corresponding problem a *finite horizon* MDP. Similary, if T is countably infinite, we call the corresponding problem an *infinite horizon* MDP. For the sake of notational simplicity, we will assume every MDP to be infinite horizon, i.e. $T = \mathbb{N}_0$, but it should always be possible to carry the formulas and results over to finite horizon problems without running into difficulties.

As the notation suggests, we will assume the set of states, actions, the collection of probability distributions, and the reward function to be independent of the time t. In the literature in which this is not inherently assumed, such processes are commonly called *stationary*.

Remark. Even further generalizations can be made. For example, one can define an action set $\mathcal{A}_s$ for each state $s \in \mathcal{S}$, making it possible to allow certain actions in specific states only. We will make use of this generalization, and it will be identifiable by the subscript.

Example 2.9 (Two-state system (cf. [6]))**.** We consider a two-state system modeled by an MDP with state set $\mathcal{S} = \{s^{(1)}, s^{(2)}\}$ and action sets $\mathcal{A}_1 = \{a_1, a_2\}$ and $\mathcal{A}_2 = \{a_3\}$.

The actions a_1 and a_2 can be performed while the system is in state $s^{(1)}$, whereas action a_3 is possible if and only if the system is in state $s^{(2)}$. Action a_1 has a 0.5 chance of causing the system to switch states, and a 0.5 chance of not altering the state, in both cases giving a reward value of 5. Action a_2 always causes the system to switch to state $s^{(2)}$ and gives a reward of 10. If the system is in state 2, the only allowed action, a_3, cannot change the state of the system, and gives a reward of -1. In other words, the system is stuck in a negative reward loop, once it reaches state 2.

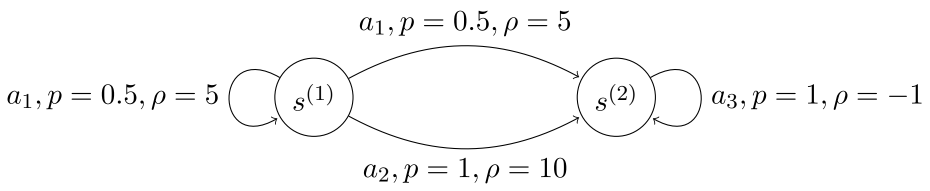


Figure 2.1: The two-state automaton corresponding to the MDP in Example 2.9.

❖

2.3 Policies and Value Functions

We would like to solve an optimization problem over an MDP, which consists of maximizing the total reward obtained over time. Note that this is not as simple as choosing the action with the highest expected reward in each state. Consider the last example: Such a heuristic would result in choosing a_2 in state 1, as it gives the highest reward. But the system would then immediately end up in state 2, which is the negative reward loop. In other words, such an approach is too short-sighted. Instead, future rewards need to be taken into account as well. Naively, one could look to maximize the sum of rewards

$$\sum_{t \in T} \rho_{a_t}(s_t, s_{t+1}),$$

which would work out if T is finite. However, if we consider the two-state example over an infinite time horizon, this sum would diverge toward $-\infty$ in all cases. By introducing a discount factor, this problem can be eliminated.

Definition 2.10 (Total discounted reward)**.** The *total discounted reward* of a sequence $(s_0, a_0, s_1, a_1, s_2, \dots)$ alternating between states and actions is given by

$$\sum_{t=0}^{\infty} \gamma^t \rho_{a_t}(s_t, s_{t+1}), \tag{2.1}$$

where $0 < \gamma \leq 1$ is the *discount factor*.

A lower discount factor means that γ^t decays at a faster rate as $t \to \infty$, putting a stronger emphasis on rewards that are obtained in the early stages of the process. This motivates the decision maker to act sooner rather than later. Furthermore, it guarantees the convergence of certain algorithms, as we will see in the next section. If $\gamma < 1$, the MDP is called *discounted*.

Definition 2.11 (Policy)**.** A *policy* for the decision maker of an MDP is an N-tuple

$$\pi = (\pi(s^{(1)}), \dots, \pi(s^{(N)})) \in \mathcal{A}_{s^{(1)}} \times \cdots \times \mathcal{A}_{s^{(N)}} =: \Pi.$$

In case the actions are independent of the state, a policy can also be seen as a mapping $\pi \colon \mathcal{S} \to \mathcal{A}$ from the set of states into the set of actions.

A policy tells the decision-maker what action to perform in which state. Once a policy has been specified, the transition probabilities $p(s' \mid s, \pi(s)) = p(s' \mid s)$ depend solely on the previous state s, reducing the MDP to a Markov chain. In this sense, MDPs can be seen as a generalization of Markov chains.

Next, we would like to determine the discounted total reward not of a single state-action-sequence, but of an entire policy. Setting $a_t = \pi(s_t)$ in Equation (2.1) gives us *some* total discounted reward with respect to the policy π. This reward is not yet well-defined, as there are many possible sequences of the form $(s_0, \pi(s_0), s_1, \pi(s_1), s_2, \dots)$, each of which occur with different probability. Picking a starting state s and taking the expectation over the set of these alternating sequences starting in state s yields

$$V_\pi(s) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \rho_{\pi(s_t)}(s_t, s_{t+1}) \mid s_0 = s\right], \tag{2.2}$$

which we call the *expected total discounted reward* of π. A different term is given in the following definition.

Definition 2.12 (State-value function)**.** Let π be a policy for an MDP. The function $V_\pi \colon \mathcal{S} \to \mathbb{R}$, as defined in Equation (2.2) is called *state-value function* for π.

For each state $s \in S$, this function tells us the expected total reward that is received, when the process starts in state s and the agent follows policy π.

Example 2.13. Reconsider the two-state automaton from Example 2.9. There are two possible policies, namely $\pi_1 = (a_1, a_3)$ and $\pi_2 = (a_2, a_3)$. We would like to compute the expected total discounted rewards of these two policies, in order to determine which one is the better choice. We will always start in state $s^{(1)}$.

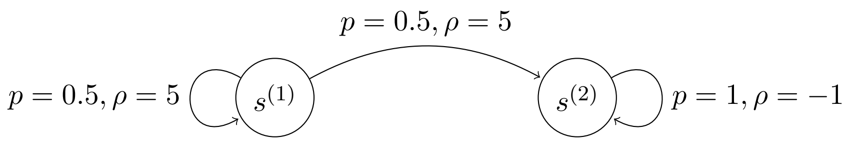
For policy π_1, let $X \colon \Omega \to \mathbb{N}_0$ be a geometrically distributed random variable with parameter $1/2$, that is,

$$\mathbb{P}(X = k) = \left(\frac{1}{2}\right)^{k+1}, \quad k \in \mathbb{N}_0.$$

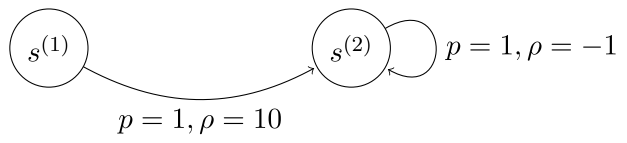
It describes how many “rounds” the system remains in state 1, before transitioning to state 2. The expected total discounted reward of π_1 is then given by

$$\begin{aligned} V_{\pi_1}(s^{(1)}) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \rho_{\pi_1(s_t)}(s_t, s_{t+1}) \mid s_0 = s^{(1)}\right] \\ &= \mathbb{E}_X\left[\sum_{t=0}^{X} \gamma^t \cdot 5 + \sum_{t=X+1}^{\infty} \gamma^t \cdot (-1)\right] \\ &= \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^{k+1} \left(\frac{5 - 6\gamma^{k+1}}{1-\gamma}\right) = \frac{10 - 11\gamma}{(1-\gamma)(2-\gamma)}, \end{aligned}$$

which is a rational function in $\gamma \in (0, 1)$ that obtains its maximum at $\gamma = \frac{10-\sqrt{12}}{11} \approx 0.594$ with approximate function value 6.072.



(a) Under policy π_1.



(b) Under policy π_2.

Figure 2.2: Two-state MDP under policies π_1 and π_2, illustrated as Markov chains (with rewards).

For policy π_2, there is only one possible sequence, namely $(s^{(1)}, a_2, s^{(2)}, a_3, s^{(2)}, a_3, \dots)$, making the expectation trivial to compute. Note that Equations (2.1) and (2.2) are equivalent in this case.

$$\begin{aligned} V_{\pi_2}(s^{(1)}) &= \mathbb{E}\left[\gamma^0 \cdot 10 - \sum_{t=1}^{\infty} \gamma^t \mid s_0 = s^{(1)}\right] \\ &= 10 - \left(\frac{1}{1-\gamma} - 1\right) = \frac{10 - 11\gamma}{1-\gamma}. \end{aligned}$$

Both functions take on the same value if $(10 - 11\gamma)(2 - \gamma) = 10 - 11\gamma$, which is the case if $\gamma = \frac{10}{11}$ or $\gamma = 1$, but the latter is not in our range. Hence, policy π_1 is preferred for $\gamma > \frac{10}{11}$, while policy π_2 is the better choice if $\gamma < \frac{10}{11}$. ❖

Definition 2.14 (Optimal policy)**.** A policy $\pi^\star \in \Pi$ is called *optimal*, if it fulfills $V_{\pi^\star}(s) \geq V_\pi(s)$ for all states $s \in \mathcal{S}$ and all policies $\pi \in \Pi$.

In the above example, we already determined the optimal policy, given γ, even though we did not explicitly compute and compare $V_{\pi_i}(s^{(2)})$, $i = 1, 2$. This is due to the fact that the policies do not differ once $s^{(2)}$ is reached, implying that the value functions are identical in this case.

2.4 Computing Optimal Policies

We concluded the previous section by giving a definition of an optimal policy $\pi^\star$ for an MDP. In this section, we are interested in the computation of such an optimal policy. We will approach this by first developing the theory of discounted MDPs, which is necessary if we want to know whether an optimal policy exists in the first place, and whether it is unique. This will naturally lead us to an algorithm for finding an optimal policy – known as the *value iteration* – whose convergence will be proven as well.

2.4.1 The Bellman Equation

By Definition 2.14, it is our goal to maximize the value of V_π, which was defined in Equation (2.2). The expression given in said equation is not yet suitable for a numerical computation, mainly due to the expectation, which is taken with respect to all possible state-action-sequences. The following derivation of a more suitable expression is based on the one found in Chapter 3 of [8], as well as [5].

Let the underlying sample space Ω be given by the set of all state-sequences of the form $(s_0, s_1, s_2, \dots)$. For $t \in \mathbb{N}_0$, define the two random variables

$$R_t \coloneqq \rho_{\pi(s_t)}(s_t, s_{t+1}), \qquad G_t \coloneqq \sum_{k=0}^{\infty} \gamma^k R_{k+t}.$$

They fulfill the recursive relation

$$G_t = R_t + \gamma \sum_{k=0}^{\infty} \gamma^k R_{k+(t+1)} = R_t + \gamma G_{t+1}. \tag{2.3}$$

Using this relation together with the linearity of expectation, we find

$$\begin{aligned} \mathbb{E}[G_t \mid s_t = s] &= \mathbb{E}[R_t + \gamma G_{t+1} \mid s_t = s] \\ &= \mathbb{E}[R_t \mid s_t = s] + \gamma \mathbb{E}[G_{t+1} \mid s_t = s]. \end{aligned}$$

Writing out the first expectation yields

$$\mathbb{E}[R_t \mid s_t = s] = \sum_{s' \in \mathcal{S}} p(s' \mid s) \rho_{\pi(s)}(s, s').$$

For the second expectation, let $\mathcal{G}$ denote the set of possible values that G_{t+1} can take on. Then,

$$\begin{aligned} \mathbb{E}[G_{t+1} \mid s_t = s] &= \sum_{g \in \mathcal{G}} p(g \mid s) g = \sum_{g \in \mathcal{G}} \sum_{s' \in \mathcal{S}} p(s', g \mid s) g \\ &= \sum_{g \in \mathcal{G}} \sum_{s' \in \mathcal{S}} p(g \mid s', s) p(s' \mid s) g \\ &= \sum_{g \in \mathcal{G}} \sum_{s' \in \mathcal{S}} p(g \mid s') p(s' \mid s) g \\ &= \sum_{s' \in \mathcal{S}} p(s' \mid s) \sum_{g \in \mathcal{G}} p(g \mid s') g \\ &= \sum_{s' \in \mathcal{S}} p(s' \mid s) \mathbb{E}[G_{t+1} \mid s_{t+1} = s']. \end{aligned}$$

Note that the Markov property was used in line 3. Finally, before putting everything back together, realize that $V_\pi(s) = \mathbb{E}[G_t \mid s_t = s]$ for any $t \in \mathbb{N}_0$. Combining all of our previous calculations, we then find that

$$V_\pi(s) = \sum_{s' \in \mathcal{S}} p(s' \mid s)(\rho_{\pi(s)}(s, s') + \gamma V_\pi(s')).$$

This expression is much more suitable for computation.

Instead of considering a fixed policy, but choosing for each state the action that maximizes the right-hand side, we arrive at the *Bellman equation*

$$v(s) = \max_{a \in \mathcal{A}_s} \left\{ \sum_{s' \in \mathcal{S}} p(s' \mid s, a)(\rho_a(s, s') + \gamma v(s')) \right\}, \tag{2.4}$$

named after Richard E. Bellman. It relates the optimal value of the system in state s to the optimal value of the system in all other states s', allowing us to find the optimal value function using *dynamic programming.* Dynamic programming is an optimization technique based on dividing a problem into smaller, easily solvable sub-problems. If the solutions to these sub-problems can be used to construct a solution to the original problem of larger size, then the overall problem is said to have *optimal substructure.* In our case, the sub-problem consists of maximizing the expected current reward and the expected discounted value of the next state, ignoring any further states. However, there is still a problem. If the computation of $v(s)$ requires the values $v(s')$ of all other states $s' \in \mathcal{S}$, where do we even start? To solve this problem, we bring the equation into a different form.

By defining $\rho_a(s)$ as the expected reward for performing action a in state s, (2.4) can also be expressed as

$$v(s) = \max_{a \in \mathcal{A}_s} \left\{ \rho_a(s) + \sum_{s' \in \mathcal{S}} \gamma p(s' \mid s, a) v(s') \right\}. \tag{2.5}$$

Let $\pi \in \Pi$ be fixed. If we view each of the values $v(s)$ and $\rho_{\pi(s)}(s)$ as components of n-dimensional real vectors v and ρ_π, respectively, and consider the transition probability matrix $P_\pi \in \mathbb{R}^{N \times N}$, given by $(P_\pi)_{ij} = p(s^{(j)} \mid s^{(i)}, \pi(s^{(i)}))$, then in matrix-vector notation, the Bellman equation can be stated as

$$v = \max_{\pi \in \Pi} \{\rho_\pi + \gamma P_\pi v\}. \tag{2.6}$$

The right-hand side of (2.6) only depends on v, hence we can use it to define a new operator L, allowing us to state the Bellman equation as the fixed-point equation $Lv = v$.

Definition 2.15 (Bellman operator)**.** The operator $L\colon \mathbb{R}^N \to \mathbb{R}^N$ defined by

$$v \mapsto Lv = \max_{\pi \in \Pi} \{\rho_\pi + \gamma P_\pi v\}$$

is called the *Bellman operator.*

From here on out, we will assume the vector space $\mathbb{R}^N$ to be equipped with the maximum-norm. That means, we set $\|\cdot\| := \|\cdot\|_\infty$, where the latter is defined by $\|v\|_\infty := \max_{i=1,\dots,N} |v_i|$.

Definition 2.16 (Contraction)**.** An operator $T\colon \mathbb{R}^N \to \mathbb{R}^N$ is called *contraction mapping* or simply *contraction*, if there exists a constant $0 \leq c < 1$ such that

$$\|Tv - Tu\| \leq c\|v - u\|$$

for all $u, v \in \mathbb{R}^N$. The value c is called *contraction constant* of T.

By definition, a contraction is a mapping that reduces the distance between any two points in the space. Repeatedly applying the contraction will therefore inevitably bring any two points closer to each other, until they eventually – as the number of applications approaches infinity – coalesce into a single point. The Banach fixed-point theorem, stated below, makes this precise.

Theorem 2.17 (Banach fixed-point theorem)**.** *Let V be a Banach space (complete normed space) and $T\colon V \to V$ a contraction mapping with contraction constant $0 \leq c < 1$. Then the following statements hold:*

i) There exists a unique point $v^\star \in V$ such that $Tv^\star = v^\star$,

ii) The fixed-point iteration $v^{n+1} = Tv^n$ converges to $v^\star$ for every starting vector $v^0 \in V$.

Proof. Consider the sequence $(v^n)_{n\in\mathbb{N}}$, defined by

$$v^{n+1} = Tv^n = T^{n+1}v^0,$$

for an arbitrary starting vector $v^0 \in V$. We want to show that (v^n) is a Cauchy sequence. First, there holds

$$\|v^{n+1} - v^n\| = \|Tv^n - Tv^{n-1}\| \leq c\|v^n - v^{n-1}\| \leq c^n\|v^1 - v^0\|. \tag{2.7}$$

Using the triangle inequality multiple times,

$$\|v^{n+m} - v^n\| \leq \|v^{n+m} - v^{n+m-1}\| + \ldots + \|v^{n+1} - v^n\|. \tag{2.8}$$

By putting both of the above equations together, we get for all $m, n \in \mathbb{N}$,

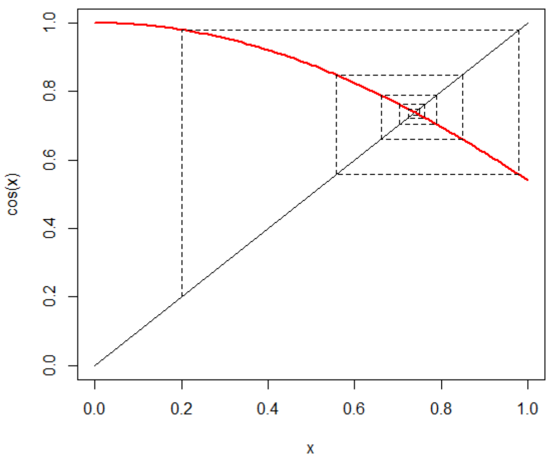
$$\|v^{n+m} - v^n\| \leq c^n(1 + \cdots + c^{m-1})\|v^1 - v^0\| \leq \frac{c^n}{1-c}\|v^1 - v^0\|,$$

which, because $c < 1$, can be made arbitrary small by choosing n large enough. Thus, (v^n) is a Cauchy sequence and by the completeness of V has a limit $v^\star \in V$.

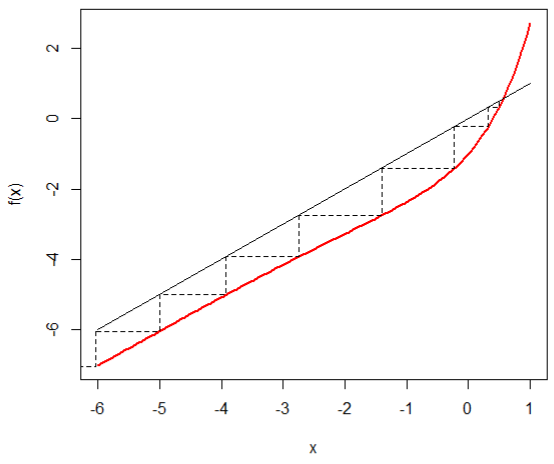
Now suppose there is another point $u^\star \in V$ fulfilling $Tu^\star = u^\star$. Then,

$$\|v^\star - u^\star\| = \|Tv^\star - Tu^\star\| \leq c\|v^\star - u^\star\|$$

implies that $\|v^\star - u^\star\| = 0$ and hence $v^\star = u^\star$. □



(a) $f(x) = \cos(x)$, $I = [0, 1]$, $x_{\text{start}} = 0.2$



(b) $f(x) = xe^x - 1 + x$, $I = [-6, 1]$, $x_{\text{start}} = 0.5$

Figure 2.3: Ten steps of the fixed-point iteration applied to $f(x)$ on the interval I, with starting point x_{start}.

Figure 2.3 illustrates the behavior of a function $\mathbb{R} \to \mathbb{R}$ that is being repeatedly applied to its previous output – a *fixed-point iteration.* Part a) shows the behavior for a contraction (with contraction constant $c = \frac{1}{\sqrt{2}} \approx 0.707$). The iteration converges nicely, which can be seen by the spiraling of the dotted lines.

Figure b) shows the behavior for a function that is not a contraction. While there is a fixed-point at $x \approx 0.5$, the iteration does not converge to it.

Next, we will show that the Bellman operator of a discounted MDP defines a contraction.

Lemma 2.18. *For a discounted MDP, the corresponding Bellman operator L is a contraction mapping with contraction constant γ.*

Proof. Given $\pi \in \Pi$, define the operator $L_\pi \colon \mathbb{R}^N \to \mathbb{R}^N$ by

$$L_\pi v = \rho_\pi + \gamma P_\pi v.$$

For all $v, u \in \mathbb{R}^N$, this operator satisfies

$$\|L_\pi v - L_\pi u\| = \|\gamma P_\pi (v - u)\| \leq \gamma \|P_\pi\| \|v - u\| = \gamma \|v - u\|, \tag{2.9}$$

where the last equality follows from the fact that the probability matrices P_π are row-stochastic, implying that the sum of each matrix row, and thus the operator ∞-norm, is equal to 1.

Let $v, u \in \mathbb{R}^N$, choose $s \in \mathcal{S}$ such that $|(Lv)(s) - (Lu)(s)|$ is maximal, and suppose without loss of generality that $(Lv)(s) \geq (Lu)(s)$. Then,

$$\begin{aligned}\|Lv - Lu\| &= |(Lv)(s) - (Lu)(s)| = (Lv)(s) - (Lu)(s) = \max_{\pi\in\Pi}(L_\pi v)(s) - \max_{\pi\in\Pi}(L_\pi u)(s)\\ &= (L_{\pi^\star} v)(s) - \underbrace{\max_{\pi\in\Pi}(L_\pi u)(s)}_{\geq (L_{\pi^\star} u)(s)} \leq (L_{\pi^\star} v)(s) - (L_{\pi^\star} u)(s) \leq \|L_{\pi^\star} v - L_{\pi^\star} u\|,\end{aligned}$$

and the statement follows with (2.9). □

This result gives a clear answer to at least one of the questions posed at the beginning of this section. Provided that the MDP is discounted, there exists an optimal policy. But is it unique? By the Banach fixed point theorem, we know that the fixed-point is unique. That is, there exists only one $v^\star$ such that $\max_{\pi\in\Pi}\{\rho_p + \gamma P_\pi v^\star\}$ is maximal. However, it might be possible that there are two policies tied for the maximum, hence we cannot claim that there is a unique optimal policy.

2.4.2 The Value Iteration

Based on the observations made and results proven above, a natural choice for an algorithm to find such an optimal policy is the fixed-point iteration with the Bellman operator. The following error estimate provides us with a suitable stopping criterion.

Proposition 2.19. *Let T, $(v^n)_{n\in\mathbb{N}}$ and c be as above. Then,*

$$\|v^\star - v^n\| \leq \frac{c}{1-c}\|v^n - v^{n-1}\|.$$

Proof. Combining the first part of (2.7) with (2.8) leads to

$$\|v^{n+m} - v^n\| \leq c(1 + \ldots + c^{m-1})\|v^n - v^{n-1}\| = \frac{c(1-c^m)}{1-c}\|v^n - v^{n-1}\|.$$

Letting $m \to \infty$ yields the desired statement. □

Thus, if we desire that the iteration error $\|v^\star - v^n\|$ lies below a certain threshold $\varepsilon > 0$, then, by Proposition 2.19 it is sufficient to require the last two iterates to fulfill the inequality $\|v^n - v^{n-1}\| < \frac{\varepsilon(1-c)}{c}$.

Algorithm 1 Value Iteration

```
Input: MDP $(\mathcal{S}, \mathcal{A}, p, \rho)$, $\gamma \in (0,1)$, start vector $v^0$, $\varepsilon > 0$
 1: $n \leftarrow 0$
 2: while $n = 0$ or $\|v^{n+1} - v^n\| \geq \frac{\varepsilon(1-\gamma)}{\gamma}$ do
 3:     for $s \in \mathcal{S}$ do
 4:         $v^{n+1}(s) \leftarrow \max_{a \in \mathcal{A}_s} \left\{ \sum_{s' \in \mathcal{S}} p(s' \mid s, a)(\rho_a(s, s') + \gamma \cdot v^n(s')) \right\}$
 5:     end for
 6:     $n \leftarrow n + 1$
 7: end while
 8: for $s \in \mathcal{S}$ do
 9:     $\pi^\varepsilon(s) \leftarrow \text{argmax}_{a \in \mathcal{A}_s} \left\{ \sum_{s' \in \mathcal{S}} p(s' \mid s, a)(\rho_a(s, s') + \gamma \cdot v^n(s')) \right\}$
10: end for
```

Algorithm 1 describes the value iteration in pseudocode. It operates in two steps. In step 1 (lines 2-7), the value function is maximized iteratively by computing the Bellman operator's fixed-point. In step 2 (lines 8-10), a maximal policy is found by considering, for each state, all possible actions and choosing the one which maximizes the value function.

Example 2.20. An application of the value iteration to the two-state system requires a probability matrix for each action,

$$P(,,1) = \begin{pmatrix} 0.5 & 0.5 \\ 0 & 0 \end{pmatrix}, \qquad P(,,2) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad P(,,3) = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix},$$

as well as an average reward vector for each action,

$$\rho(,1) = \begin{pmatrix} 5 \\ 0 \end{pmatrix}, \qquad \rho(,2) = \begin{pmatrix} 10 \\ 0 \end{pmatrix} \qquad \rho(,3) = \begin{pmatrix} 0 \\ -1 \end{pmatrix}.$$

The probability matrices are not row-stochastic, because not every action is allowed in every state. As a result, the algorithm will also consider invalid actions, which give reward 0, and prefer them to actions with negative reward. To mitigate this, there are two options:

i) Fill the probability matrices up in such a way that invalid actions never cause a state change, and set the reward to a large negative number,

ii) Introduce an "allowed" matrix, in which an entry (i, j) is set to 1 if and only if action j is allowed in state i. In our example, it would look like this:

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

These are some numerical results with $\varepsilon = 0.01$, $v^0 = (0,0)$ and $\gamma \in \{0.1, 0.5, 0.99\}$:

γ	π^ε	V_{π^ε}
0.1	(a_2, a_3)	$(9.888889, -1.111111)$
0.5	(a_2, a_3)	$(9, -2)$
0.99	(a_1, a_3)	$(-88.11449, -99.99568)$

Note that $0.99 > \frac{10}{11}$, and hence the optimal policy switches over to (a_1, a_3). ❖

2.5 Reinforcement Learning

Reinforcement learning refers to a class of machine learning problems and algorithms, which center around an agent performing actions in an environment, with the goal of maximizing some cumulative reward. A reinforcement learning problem can be stated in the form of an MDP, in which case the MDP serves as an idealized model of the environment – generally, very little may be known about the environment of a reinforcement learning problem.

Example 2.21 (k-armed bandit (cf. [8], Ch. 2))**.** At each time step $t \in T$, an agent can perform one out of k different actions, each of which yield an immediate numerical reward that follows some stationary (independent of the time t) probability distribution. No information about the rewards is available beforehand. The goal is to maximize the expected total reward over some time period. ❖

This example corresponds to a one-state MDP with k actions, but with unknown rewards. It is therefore not possible to solve it analytically or by using value iteration. Interaction with the environment is required in order to obtain information about the rewards. Of particular interest is the mean of the reward of an action a. We call this the *action value*, denoted by $Q_\star(a)$. Once the action values are known, an optimal action can be easily chosen by selecting the action with the largest action value.

2.5.1 Exploration vs. Exploitation

In the beginning, $Q_\star$ is unknown. The agent is forced to *explore* the environment by choosing an action a uniformly at random. Observing the reward gives him an estimate $Q(a)$ of the action value. If he performs the same action later on, he can update $Q(a)$ using this new information (for example, by maintaining a *moving average*). By the law of large numbers, the more he explores, the more accurate his estimate of $Q_\star(a)$ becomes.

Now suppose that the agent has gathered enough information to have a good estimate of $Q_\star$. At the next time step, he can pick the action with the highest (estimated) action value. This is called *exploitation*.

In practice, it makes sense to keep exploring, even if good estimates are already available. One option is to define a fixed value $0 < \varepsilon < 1$, such that exploitation occurs with probability $1 - \varepsilon$, and exploration with a probability of ε. This is known as an *ε-greedy strategy.* A more extensive option is to let $\varepsilon = \varepsilon(t)$ be dependent on the number of the current iteration t. For example, $\varepsilon(t) = \frac{1}{\log(t+2)}$, meaning that exploration is chosen less and less likely, as the number of iterations increases.

2.5.2 Q-Learning

In the k-armed bandit problem, we had only one state. If we want to generalize this approach to finitely many states, we have to extend the action-value function to allow for state-action pairs as inputs. We call $Q_\pi \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the *action-value function* for policy π. Then, $Q_\pi(s, a)$ is the expected total reward for performing action a in state s and continuing with policy π. Expressed mathematically,

$$Q_\pi(s, a) = \rho_a(s) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, \pi(s)) V_\pi(s').$$

Using this, we can define the optimal action-value function as

$$Q_\star(s, a) := \max_{\pi \in \Pi} Q_\pi(s, a),$$

which is the expected total reward for performing action a in state s and continuing with an optimal policy. In consequence, $\pi^\star(s) := \arg\max_a Q_\star(s, a)$ is an optimal policy. It relates to the optimal state-value function $V_\star = V_{\pi^\star}$ by

$$V_\star(s) = \max_{a \in \mathcal{A}_s} Q_\star(s, a).$$

Before we begin formulating an algorithm for the computation of $Q_\star$, we need one more piece of terminology, which we have not yet encountered.

Definition 2.22 (Episode)**.** Given an MDP and a set $E \subset \mathcal{S}$ of terminating states, we call a sequence $(s_0, a_0, s_1, \dots, a_{m-1}, s_m)$ an *episode*, if $s_m \in E$.

Our examples so far had no terminating states. A good example would be a game of chess: A game ends when one side's king is in checkmate, when one of the players resigns, or when there is a draw (a stalemate, for example). These make up the terminal states. Consequently, an episode is one game of chess.

Our goal is to construct an algorithm that learns a function Q which approximates $Q_\star$ based on interactions with the environment. Assuming that Q is initialized with arbitrary values, this means for every step t of the algorithm,

- select an action s_t to perform (explore or exploit),
- observe the reward r_t and the new state s_{t+1},
- update $Q(s_t, a_t)$ accordingly.

For the update step, an exponentially weighted moving average is commonly used.

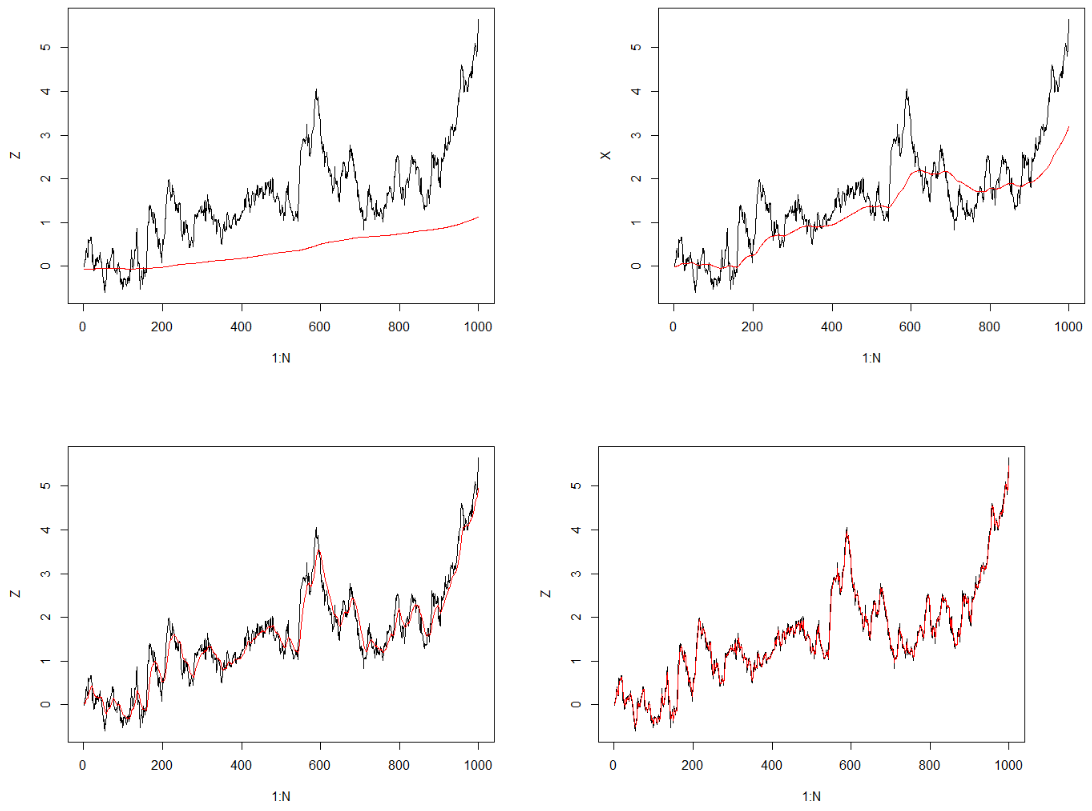


Figure 2.4: EWMA of an example process, $\alpha \in \{0.001, 0.01, 0.1, 0.5\}$. The black graph shows the original process, while the red graph shows the corresponding EWMA process.

Definition 2.23 (EWMA)**.** Given a stochastic process $(X_t)_{t \in T}$, an *exponentially weighted moving average* (EWMA) is a stochastic process $(S_t)_{t \in T}$, such that

$$S_t = \begin{cases} X_0, & t = 0 \\ \alpha X_t + (1-\alpha) S_{t-1}, & t > 1 \end{cases},$$

where $0 < \alpha \leq 1$ is a suitably chosen factor, which we here call the *learning rate*.

The effect of the learning rate α is visible in Figure 2.4. If α is very low, the EWMA adapts slowly and carefully. For larger values of α, it follows the original process more closely, only noticeably deviating from the original process, when the latter increases or decreases rapidly.

In our model, the sample X_t corresponds to the sum of the current reward r_t and the upcoming (discounted) rewards, when following an optimal policy. The EWMA value S_t

corresponds to $Q(s_t, a_t)$. The update step will then look as follows,

$$\begin{aligned} Q^{\text{new}}(s_t, a_t) &= \alpha_t \left(r_t + \gamma \max_{a \in \mathcal{A}_{s_{t+1}}} Q(s_{t+1}, a) \right) + (1 - \alpha_t) Q(s_t, a_t) \\ &= Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_{a \in \mathcal{A}_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t) \right). \end{aligned}$$

Here, we allow the learning rate to depend on t.

Combining the above-mentioned steps, we arrive at the Q-learning algorithm, which was first introduced by Christopher Watkins in his 1989 Ph.D. thesis [9].

Algorithm 2 Q-learning

```
Input: state and action sets $\mathcal{S}, \mathcal{A}$, $\gamma \in (0,1)$, $\alpha \in (0,1]$, small $\varepsilon > 0$, $n \in \mathbb{N}$
 1: Initialize $Q$ with arbitrary values
 2: for each episode do
 3:     choose some starting state $s \in \mathcal{S}$
 4:     while $s$ is not a terminating state do
 5:         pick action $a$ based on $\varepsilon$-greedy strategy
 6:         observe reward $r$ and next state $s'$
 7:         $Q(s,a) \leftarrow Q(s,a) + \alpha \left( r + \gamma \max_{a' \in \mathcal{A}_{s'}} Q(s', a') - Q(s,a) \right)$
 8:         $s \leftarrow s'$
 9:     end while
10: end for
```

Q-learning is a *model-free* algorithm, which means that it does not require explicit knowledge of the transition probabilities and the rewards. Furthermore, it is an *off-policy* algorithm. That means, even though it is trying to find a certain policy (the optimal one), this is not the same as the one it is using: due to the ε-greedy strategy, it chooses a random action with probability ε, ignoring the policy in those cases.

Without proof, the following convergence theorem for Q-learning is stated. A proof can be found in [10].

Theorem 2.24. *Let states $\mathcal{S}$ and actions $\mathcal{A}$ be given. Define $t^i(s,a)$ as the index $t \in T$ for which action a is tried in state s for the ith time. Then, given bounded rewards $|r_t| \le R < \infty$, learning rates $0 \le \alpha_t < 1$, and*

$$\sum_{i=1}^{\infty} \alpha_{t^i(s,a)} = \infty, \quad \sum_{i=1}^{\infty} (\alpha_{t^i(s,a)})^2 < \infty, \quad \forall s \in \mathcal{S}, a \in \mathcal{A},$$

$Q_t(s,a) \to Q_\star(s,a)$ as $t \to \infty$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$ with probability 1.

A noteworthy consequence of this theorem is the fact that the learning rate needs to be adaptive, since a constant learning rate $\alpha_t = \alpha$ cannot fulfill both conditions at the same time.

Example 2.25. We apply the Q-learning algorithm to the two-state MDP. As it does not have terminating states, we simulate episodes by randomizing the state on each 100th iteration. In total, we let the algorithm run for $n = 10000$ iterations, resulting in 100 episodes. Also, $\varepsilon_t = \frac{1}{\log(t+2)}$ and $\alpha_t = \frac{1}{\sqrt{t+2}}$. Then,

γ	π^ε	V_{π^ε}
0.1	(a_2, a_3)	$(9.888889, -1.111111)$
0.5	(a_2, a_3)	$(9, -2)$
0.99	(a_2, a_3)	$(-60.70343, -78.25509)$

For $\gamma \in \{0.1, 0.5\}$, the results coincide with the ones produced by value iteration. However, for $\gamma = 0.99$, the results do not coincide with the exact results we had computed. This should not come as a surprise, since $\gamma = 0.99$ is very close to the MDP being undiscounted.

By increasing the number of iterations to $n = 50000$, the result looks better, but still not perfect. In particular, the results fluctuate, indicating a lack of convergence. This table contains several examples of just $\gamma = 0.99$:

$\sharp$	π^ε	V_{π^ε}
1	(a_1, a_3)	$(-85.02718 - 97.11568)$
2	(a_1, a_3)	$(-84.26298, -97.32420)$
3	(a_1, a_3)	$(-84.69676, -97.18149)$

with corresponding Q-tables

	a_1	a_2	a_3
$s^{(1)}$	-85.02718	-85.0686	-150.73003
$s^{(2)}$	-182.45319	-182.4210	-97.11568

	a_1	a_2	a_3
$s^{(1)}$	-84.26298	-84.97545	-141.3318
$s^{(2)}$	-182.67051	-182.60698	-97.3242

and

	a_1	a_2	a_3
$s^{(1)}$	-84.69676	-84.86077	-145.83266
$s^{(2)}$	-182.38388	-182.41331	-97.18149

In figure 2.5, a heatmap is shown, indicating the number of times action a (column) has been tried in state s (row), and hence the number of times that $Q(s, a)$ has been updated. Darker shades indicate a high number of tries, while lighter shades indicate the opposite. The combination $(s^{(1)}, a_3)$ received the least amount of attention by the algorithm. This reflects in the fact that the corresponding Q-value has the strongest variation. The reason for this can be explained: Action 3 is not actually allowed in state 1, which, in this case, was implemented by setting the corresponding reward to -100. Along with an exploration probability ε close to zero, this option is chosen less often. If the action is not permitted anyway, this is not a problem, but it is good to keep an eye on this. ❖

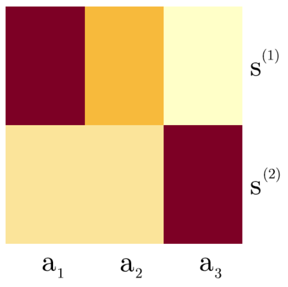


Figure 2.5: Heatmap indicating how often each action (column) was performed in each state (row) throughout all iterations of the Q-learning algorithm.

3 The Repair Limit Replacement Method

Let us consider an item – this could be a machine or a tool of some other sort – that becomes outworn as it is being used over time, and will eventually break or malfunction. Upon this happening, the item is inspected and the repair cost estimated. If this cost exceeds a certain amount, known as the *repair limit*, the item is replaced instead of repaired.

This strategy is known as the *repair limit replacement method.* In his 1969 paper of the same name [3], Hastings formulates the task of finding such repair limits as a Markov decision process. The aim of this thesis is to find an optimal policy $\pi \in \Pi = \{\text{repair}, \text{replace}\}^N$, given that the repair limits are already known.

In the following, we will state the problem in the form of an MDP, solve it using value iteration and Q-learning, and compare the results produced by the two methods.

3.1 Problem Formulation

Let the state s refer to the age of the tool in years. First, we make some simplifying assumptions. We assume the failure rate (number of failures per time interval) to be an age-dependent constant $k = k_s$, and the repair costs to follow an age-dependent distribution $f(x) = f_s(x)$. We can then define the probability that the repair cost will exceed a chosen repair limit $L = L_s$, and hence the item be replaced, as

$$P(L) := 1 - \int_0^L f(x)\,\mathrm{d}x. \tag{3.1}$$

Common choices of probability distributions fall inside of the Gamma family. One of the distributions in this family is the *Erlang distribution*,

$$f(x) = \frac{b(bx)^{c-1}}{(c-1)!}\exp(-bx), \quad x, b > 0,\ c \in \mathbb{N}. \tag{3.2}$$

By demanding that $c = 1$, the expression simplifies to

$$f(x) = b\exp(-bx), \quad b > 0, \tag{3.3}$$

called the *negative exponential distribution.* Using this particular distribution, where the parameter $b = b_s > 0$ is given by the reciprocal of the mean repair cost in year s, Equation (3.1) becomes

$$P(L) = \exp(-bL).$$

The probability that an item experiences no malfunction with repair costs exceeding L over the course of an entire year is given by

$$S(L) := 1 - \int_0^1 p(t)\,\mathrm{d}t,$$

where $p(t)$ is the probability density of malfunctions. It is beyond the scope of this thesis to derive an expression for $p(t)$. In the appendix of his paper, Hastings derives $p(t) = Pk\exp(-Pkt)$, where $P = P(L)$, such that

$$S(L) = \exp(-Pk).$$

This is also called the *probability of survival.*

We have now derived all necessary quantities in order to formulate the MDP corresponding to this problem. In order, we specify states, actions, transition probabilities and rewards.

States The states are given by the time intervals $\mathcal{S} = \{1, 2, \ldots, N\}$, which we think of as the year that the tool is currently in. Here, N denotes the maximum age of a tool upon reaching which it needs to be replaced regardless of its functionality.

Actions The actions in this scenario are "replace" and "repair". The first one always replaces (even if there is no malfunction), whereas the second one repairs, should a malfunction occur.

Probabilities Concerning the probabilities, if the chosen action is to replace, the probability matrix is simply given by

$$P(,,\text{replace}) = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \ldots & 0 \end{pmatrix},$$

because a new item is in year 1, regardless of the previous item's age.

If the chosen action is to repair, there are two possible cases for $s < N$:

1. There are no malfunctions in this year, so the item proceeds to the next state. This happens with the probability of survival $S_s := S(L_s)$.
2. There is a malfunction at some point in the year. This happens with probability $1 - S_s$. As a result, the item remains in its current state, because there might be further malfunctions.

In the final state, the item is always replaced. Accordingly, the probability matrix is given by

$$P(,,\text{repair}) = \begin{pmatrix} 1-S_1 & S_1 & & & \\ & 1-S_2 & S_2 & & \\ & & \ddots & \ddots & \\ & & & 1-S_{N-1} & S_{N-1} \\ 1 & & & & \end{pmatrix}.$$

In Figure 3.1, this behavior is illustrated as an automaton. Note that this does not describe the MDP as a whole, but only if the chosen action is to repair.

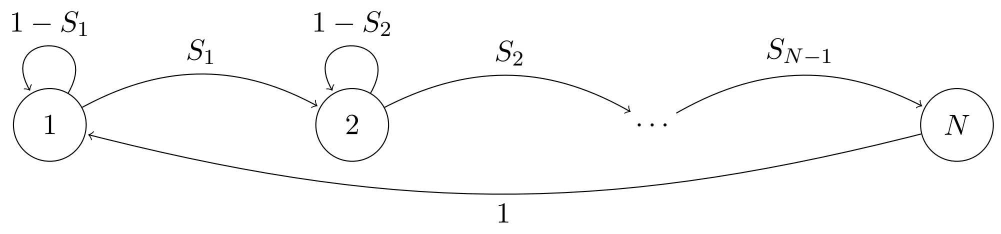


Figure 3.1: State automaton illustrating the behavior when the “repair” action is chosen.

Rewards Finally, we are interested in minimizing cost, which can be seen as negative reward. In state $N-1$, the item is always being replaced, the costs of which are equal to the acquisition costs c_A of a new item. In all other states, the repair costs average at k_s/b_s, which is the product of the failure rate in state s with the mean repair costs in state s. Hence, we have

$$\rho_{\text{replace}}(s) \coloneqq -c_A \quad \forall s \in \mathcal{S}$$

and

$$\rho_{\text{repair}}(s) \coloneqq \begin{cases} -\frac{k_s}{b_s}, & \text{if } s < N-1 \\ -c_A, & \text{if } s = N-1 \end{cases}.$$

In the following sections, the two algorithms introduced in the previous chapter, value iteration and Q-learning, will be applied to an example problem of the above type. We consider a span of three years, i.e., the item is replaced after its second year. There are, on average, two malfunctions in the first year (state 0), and three in the second (state 1). The mean repair cost in the first year will be 100, and 150 in the second year. Finally, the repair limits are 300 in the first year, and 100 in the second. In summary:

- States $\mathcal{S} = \{1, 2, 3\}$,
- failure rates $k_1 = 2$ and $k_2 = 3$,
- mean repair costs $1/b_1 = 100$ and $1/b_2 = 150$,
- repair limits $L_1 = 300$ and $L_2 = 100$.

The (rounded) probability matrices look as follows:

$$P(,,\text{replace}) = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad P(,,\text{repair}) = \begin{pmatrix} 0.095 & 0.905 & 0 \\ 0 & 0.786 & 0.214 \\ 1 & 0 & 0 \end{pmatrix},$$

and the reward vectors are

$$\rho_{\text{replace}} = \begin{pmatrix} -400 \\ -400 \\ -400 \end{pmatrix}, \qquad \rho_{\text{repair}} = \begin{pmatrix} -200 \\ -450 \\ -400 \end{pmatrix}.$$

3.2 Application of Value Iteration

The value iteration was applied to these inputs for three different discount values $\gamma \in \{0.1, 0.5, 0.9\}$. Before the presentation of the results, some parts of the code implementation will be briefly highlighted. The implementation was done using the statistical computing language R.

3.2.1 Implementation

The function header reads `function(prob,rho,gamma,eps,start)`, where the first argument, `prob`, is a multidimensional array with dimensions $|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|$, i.e., a probability matrix for each action. The second argument, `rho`, is a matrix with dimensions $|\mathcal{S}| \times |\mathcal{A}|$, interpreted as a reward vector for each action. The discount value is set by the third argument, `gamma`, and `eps` corresponds to ε in Algorithm 1, used as a stopping criterion. Finally, `start` is the initial value vector, meaning v^0.

The implementation itself is very straightforward, essentially following the steps detailed in Algorithm 1, but making use of linear algebra operations. For instance, line 4 is implemented as follows:

```
  for (a in 1:acts) {
      Q[,a] <- rho[,a] + gamma * prob[,,a] %*% V_prev
  }
  V_next <- apply(Q,1,max)
```

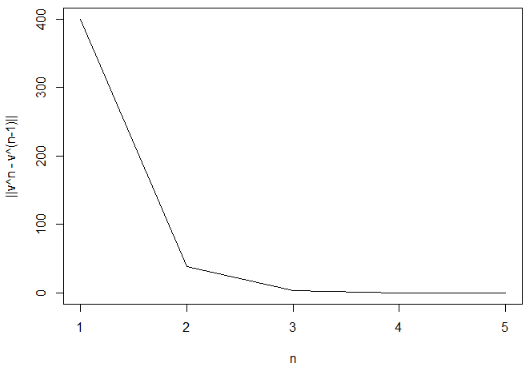
In this case, `Q[,a]` corresponds to $v^{n+1}(s)$, assuming that `a` is the next chosen action. The operator `\%*\%` denotes matrix-vector multiplication. After computing these vectors for all actions, the maximum vector `V_next` is computed by making use of R's `apply(x,margin,fun)` function, which applies the function `fun` to the array/matrix `x` along the axis indicated by `margin`. A value of `1` indicates row-wise application, while a value of two `2` indicates column-wise application.

3.2.2 Application

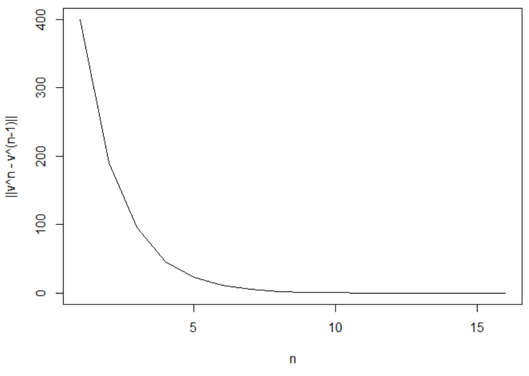
The following table contains the results obtained from applying the value iteration to the problem with different discount values.

γ	π^ε	V_{π^ε}	iterations
0.1	$(1,2,1)$	$(-240.6647, -424.0642, -424.0642)$	5
0.5	$(1,2,1)$	$(-524.6251, -662.3077, -662.3077)$	16
0.9	$(1,2,1)$	$(-2897.880, -3008.091, -3008.091)$	120

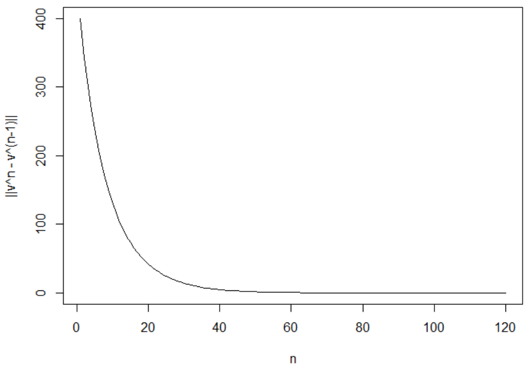
For all three values of γ, the resulting policy is to repair in state 1, and replace in state 2. While the action in state 3 is to repair again, we can ignore this, as we enforce replacement in the last state. Both actions have the same transition probabilities and rewards in this state, hence the result does not matter.



(a) $\gamma = 0.1$.



(b) $\gamma = 0.5$.



(c) $\gamma = 0.9$.

Figure 3.2: Value Iteration: Plots of the error curve $\|v^n - v^{n-1}\|$ for $\gamma \in \{0.1, 0.5, 0.9\}$.

Further note that the values of states 2 and 3 always coincide. This can be explained by the fact that the machine is replaced in both states, i.e., the reward is $-c_A$, and the follow-up state is $s^{(1)}$.

Next, we will look at each discount-strategy a bit more closely.

Short-Sighted Decision Maker With a discount value of $\gamma = 0.1$, the decision maker is very short-sighted: The reward obtained in the current state is deemed 10 times more important than the reward gained in the following state. The algorithm converges after 5 iterations, with an error curve shown in Figure 3.2a.

Balanced Decision Maker With a discount value of $\gamma = 0.5$, the decision maker puts more emphasis on future rewards than the short-sighted decision maker. The current reward is deemed twice as important as the follow-up reward. The value iteration converges after 16 iterations, with an error curve shown in Figure 3.2b.

Far-Sighted Decision Maker This last decision maker is far-sighted, using a discount value of $\gamma = 0.9$. Put into perspective, this decision maker deems the state that lies 21 iterations in the future about as important as the short-sighted decision-maker deems the immediate follow-up state. The value iteration takes 120 iterations to converge, with an error curve shown in Figure 3.2c.

3.3 Application of Q-Learning

Again, the algorithm is applied to the inputs with different discount values $\gamma \in \{0.1, 0.5, 0.9\}$. And as before, a part of the code implementation will be explained.

3.3.1 Implementation

The header of the function reads `function(prob,rho,gamma,iter=50000)`, with the first three arguments being the same as in the implementation of the value iteration. The last argument, `iter`, specifies the number of iterations that should be performed, with the default value set to `50000`.

As in the Q-learning example from the previous chapter, the episode loop from line 2 in Algorithm 2 is simulated by randomizing the state on every 100th iteration. In effect, this creates 5000 episodes, when using the default value for `iter`.

Line 5 is realized by sampling a random value in $[0, 1]$, and evaluating whether it lies below $\varepsilon_t = \frac{1}{\log(t+2)}$. If so, the decision-maker will explore, and if not, it will exploit. In code, this part looks as follows:

```
    rnd <- runif(1)
    eps <- 1 / log(i+2)
    if (rnd < eps) { # Explore
        a <- sample(acts,1)
    } else { # Exploit
        a <- which.max(Q[s,])
    }
```

In R, `which.max` corresponds to arg max, returning the array index of the maximal element, or the first occurrence thereof.

3.3.2 Application

The outputs of one application of Q-learning are documented in the following table.

γ	π^ε	V_{π^ε}
0.1	$(1, 2, 2)$	$(-240.6664, -424.0732, -375.5875)$
0.5	$(1, 2, 1)$	$(-526.5513, -662.3687, -612.3953)$
0.9	$(1, 2, 1)$	$(-2895.623, -3006.552, -2854.552)$

Note that, because Q-learning is not a deterministic algorithm, the results will not always look exactly like this, but fluctuate a bit.

There are no iteration numbers to compare, but we can look at the matrix indicating the frequencies of different state-action combinations, i.e., how often they were explored. Technically, there is a different matrix for each of the three discount values, but there is no significant difference between those matrices. Hence, we only take a look at the $\gamma = 0.1$ case:

```
      [,1]  [,2]
[1,] 24894  1365
[2,]  1179 22148
[3,]   203   211
```

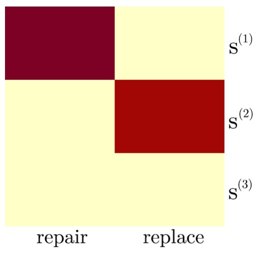


Figure 3.3: Heatmap corresponding to the left-hand matrix.

The combinations $(s^{(1)}, a_1)$ and $(s^{(2)}, a_2)$ are visited most often, which can be explained due to the high exploitation probability as the iteration number grows larger. State 3 is visited rarely in general. Again, the high exploitation probability will trigger the "replace" action in state 2 most of the time, causing the system to go back to state 1.

3.4 Comparison

For the sake of comparison, consider the following table, which presents the computed state-value functions of both algorithms next to each other.

γ	Value Iteration	Q-Learning
0.1	$(-240.6647, -424.0642, -424.0642)$	$(-240.6664, -424.0732, -375.5875)$
0.5	$(-524.6251, -662.3077, -662.3077)$	$(-526.5513, -662.3687, -612.3953)$
0.9	$(-2897.880, -3008.091, -3008.091)$	$(-2895.623, -3006.552, -2854.552)$

In the first two states, the values of both algorithms are fairly similar. At most, there is an absolute difference of 2, which is not too bad, considering the overall magnitude of the values. However in the last state, the values differ much more. This can be explained using Figure 3.3. As we already observed, state 3 is rarely visited at all, regardless of the action chosen. Hence, the corresponding row in the Q-table is updated less often, leading to a worse estimate of $V(s^{(3)})$. A consequence of this is the fact that, for $\gamma = 0.1$, the policy computed by Q-learning is $(1, 2, 2)$, which differs from the $(1, 2, 1)$ policy computed via value iteration. But because the tool always gets replaced in the last state, this is not a problem in this case.

Both methods computed a correct solution – an optimal policy. In this sense, no method performed better or worse. But if we consider the optimal value vector as part of the solution as well, value iteration is the clear winner: From Chapter 2, we know that its solution coincides with the analytic one, provided that ε is sufficiently small.

Given any reinforcement learning problem, if the parameters of the environment – that is, the rewards and transition probabilities – are fully known beforehand, value iteration will always provide better results.

Q-learning cannot compete with this, as it relies on the combination of exploration and exploitation to learn the parameters of the environment. Even when they are given explicitly, they are only being used to simulate the environment. The example problem chosen was playing to the strengths of the value iteration, whilst not giving Q-learning an opportunity to present its strengths, which consist of exploring and exploiting an unknown environment.

4 Conclusion

In this work, both the theoretical and practical aspects of reinforcement learning problems have been considered, as well as two methods aimed at solving them. To conclude this thesis, the results will be summarized and possible future work discussed.

4.1 Summary

The two algorithms explored in this thesis – value iteration and Q-learning – are radically different in the way they approach the problem of finding an optimal policy. The value iteration is the result of using mathematical analysis to describe the problem, and solving it using an appropriate analytic tool – in this case, the fixed-point iteration. As a consequence, the computed solution is highly accurate and converges after a certain number of iterations, which can even be estimated beforehand. A downside to this approach, however, is the requirement for complete information about the environment. Transition probabilities and rewards need to be fully known in order for this method to be applied successfully. As such, the value iteration is not an appropriate tool for the task of developing an optimal behavioral strategy in an unknown environment.

Q-Learning, on the other hand, excels at the latter task. The algorithm learns the parameters of the environment by maintaining a Q-table, which is updated with each new experience, consisting of a reward as well as a change in state. The class of problems this algorithm can be applied to is hence much broader, compared to the value iteration. On the downside, Q-learning requires many more iterations to converge, usually on the order of tens of thousands. And even then, it might not always converge satisfactorily. We have been able to observe, however, that the result, even if it did not converge, was close enough to the analytic result (or the result obtained by value iteration) and hence usable.

When applied to the repair limit replacement problem in Chapter 3, these differences became clear. We concluded that value iteration is a better choice, with the reason being that all information about the environment – consisting of transition probabilities and rewards – is known, and hence a much more accurate solution could be obtained. Still, Q-learning also produced usable results. In particular, the resulting policies of both methods were identical.

4.2 Future Work

The topic of reinforcement learning and/or its application the repair limit replacement problem can be further explored in several directions, three of which will be briefly described.

First of all, one could take a closer look at the reasons as to why Q-learning did not fully converge in our examples, and find possible ways to ensure or at least improve convergence.

A second option is to consider a generalized version of the repair limit replacement problem in which certain parameters of the problem are either unknown or randomized. The task would be to examine the performance of Q-learning on this problem.

Another option is to examine the combination of both algorithms. That is, given a reinforcement learning problem with partially unknown parameters, first apply several iterations of Q-learning for parameter estimation and follow this up by applying the value iteration. It would be interesting to study how well this approach performs compared to Q-learning on its own, both in terms of accuracy as well as computation time.

References

[1] Thomas H. Cormen et al. *Introduction to Algorithms*. Third Edition. The MIT Press, 2009.

[2] Walter Gander, Martin J. Gander, and Felix Kwok. *Scientific Computing: An Introduction using Maple and MATLAB*. Springer, 2014.

[3] N. A. J. Hastings. “The Repair Limit Replacement Method”. In: *Journal of the Operational Research Society 20, 337-349* (1969).

[4] J. A. E. E. van Nunen. “A Set of Successive Approximation Methods for Discounted Markovian Decision Problems”. In: *Mathematical Methods of Operations Research* (1974).

[5] Finncent Price. *Deriving Bellman’s Equation in Reinforcement Learning*. Cross Validated. URL: `https://stats.stackexchange.com/q/413974` (visited on 09/11/2021).

[6] Martin L. Puterman. “Markov Decision Processes”. In: *Handbooks in OR & MS, Vol. 2*. 1990. Chap. 8, pp. 331–433.

[7] RDocumentation. *R base package version 3.6.2*. URL: `https://www.rdocumentation.org/packages/base/versions/3.6.2` (visited on 09/13/2021).

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second Edition. The MIT Press, 2018.

[9] Christopher J.C.H. Watkins. “Learning from Delayed Rewards”. Ph.D. thesis. King’s College London, 1989.

[10] Christopher J.C.H. Watkins and Peter Dayan. “Q-Learning”. In: *Machine Learning, 8, 279-292* (1992).