Mathematics of Reinforcement Learning and Handling of the Curse of Dimensionality

Nouralden Mohammed (nouralden@aims.ac.za)
African Institute for Mathematical Sciences (AIMS)

Supervised by: Professor Montaz Ali
Witwatersrand University, Johannesburg, South Africa

23 May 2019

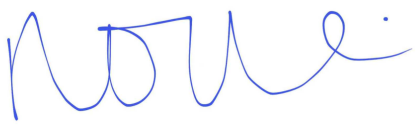
Submitted in partial fulfillment of a structured masters degree at AIMS South Africa

Abstract

Reinforcement learning has a rich history in the study of trial-and-error learning. However, as the problem domain grows in size, so does the cost of solving the problem—the curse of dimensionality. One way to address this issue is through hierarchical architecture. Although hierarchical methods help in reducing the curse of dimensionality, the serial execution makes the running time longer. In this study, we exploited a class of stochastic problems called Linearly-solvable Markov Decision Processes (LMDPs). This class of problems guarantees the compositionality and concurrent execution of actions. The LMDP allows a learning agent to execute multiple actions simultaneously in order to learn new tasks. In this study, we build our hierarchical architecture based on the LMDP which has proven to be advantageous in terms of computational complexity.

Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.



Nouralden Mohammed Jadalla Mohammed, 23 May 2019

Contents

1. Introduction

This work looks at specific tasks of Machine Learning (ML). ML is the ability of the machine to learn by its own without being programmed explicitly. It can be categorized as supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning (RL).

From a ML point of view, the RL problem is described as an interaction between a decision maker (agent) and some environment with the aim of achieving a desired goal. A RL agent learns through interactions with the environment which refers to the learning process. This is because when the agent starts it may not possess the information of the environment, so this training process is achieved by trial-and-error until it gets an optimal behaviour to follow. In each iteration, the agent will make transition from one state to another state by performing an action and receiving a signal from the environment called a reward.

In RL, the problem is broken into small sub-problems and they are solved iteratively to make the agent maximize the rewards it receives from the environment due to transitioning from state to another state given an action. The problems of this form are modelled using the Bellman optimality equation (Bellman, 1954).

The Bellman optimality equation has a global optimal solution using iterative methods like value iteration, policy iteration and Q-learning, just to name a few. Practically, the number of states grows dramatically as the size of the domain increases known as the curse of dimensionality (Saxe et al., 2017). For example, an agent is supposed to select between two sets of states it may learn in product space instead of manipulating each set individually and combining the resulting outcomes. In addition, the actions which the agent may perform also suffer from the curse of dimensionality. Consider a robot learning to travel between N rooms individually. The robot needs to learn N choose 2 policies[1] to navigate between each pair of rooms and N choose 3 policies between a triple of them and so on. When the number of states or actions increases exponentially, which lead to a curse of dimensionality, the RL problem become a large scale problem. One powerful tool to address the problem is through the Hierarchical Reinforcement Learning (HRL).

The hierarchical approaches decompose the original RL problem into hierarchical sub-problems such that the higher layers (parents) contain subtask states (children). Executing the higher layer results in calling the layer below which in turn calling the layer below it and so on. The hierarchy will help the agent to learn a new task providing the learned task, which saves the time needed to learn the new task. Also, the agent will work in subspace of the domain of the problem instead of working on the whole domain.

An important concept of HRL is macro-action or option. The macro-action can be thought as a sequence of actions chosen from the primitive actions[2] of the problem (Randlov, 1999). Most of the hierarchical frameworks rely on the serial execution of actions. An agent equipped with a set of macro-actions is able to plan against a sequence of primitive actions rather than against the primitive actions themselves.

Although the features of HRL methods are powerful in addressing the curse of dimensionality, the serial execution of HRL does not benefit from the powerful of the computer and the execution time may be inefficient. Todorov (2007) introduced one class of stochastic control problems in which the actions and cost functions (rewards) are defined in a way that makes the Bellman equation linear. This class of problem is known as Linearly-Solvable Markov Decision Process (LMDP). By making the

[1] A policy is an oracle that inform the agent as to which action to take in a given state in order to achieve a goal.

[2] Primitive actions are the leaves of action hierarchy which terminate after one time-step or invoke a higher level action that might execute many times before terminating.

Bellman equation linear, LMDP allows one to construct the optimal control policy efficiently by a linear combination of the previous optimal policies (Todorov, 2009a). The optimal policy for LMDP is learned with less cost using off-policy learning called Z-learning, which operates on the set of states instead of the Cartesian product of states and actions. Although LMDP helps in reducing the curse of dimensionality but it is still challenging when the number of states grows exponentially.

In this essay, we are going to combine the HRL and LMDP to benefit from the characteristics of each method by creating a hierarchy of LMDP. The HRL will benefit from the LMDP by composing the optimal policies of the new tasks from the previous learned tasks without significant effort. Also, the class of LMDP problems change the discrete actions to be transition probabilities defined over next states, and those transitions can be influenced by many subtasks which operates at the same time. This property relaxes the assumption of the serial execution of the HRL methods. Moreover, the LMDP benefits from the HRL by addressing the curse of dimensionality.

The rest of the essay is organized as follows. Chapter 2 reviews Markov chains and conditions for existence and uniqueness of stationary distribution. It also reviews dynamic programming, the Markov Decision Process (MDP) and hierarchical reinforcement learning. The formulation of LMDP is presented in Chapter 3. Chapter 4 discusses the hierarchical LMDP which is the main core of this essay. Chapter 5 presents and discusses empirical results. In Chapter 6 we conclude.

2. Background

In this chapter, we present the background needed to formulate our RL problem. This chapter is divided into four sections. In Section 2.1 we introduce Markov chaisn and discuss the existence and uniqueness of the stationary distribution. In Section 2.2 we present the notion of dynamic programming followed by the Markov decision process in Section 2.3. Finally, we give a brief introduction to hierarchical reinforcement learning in Section 2.4.

2.1 Markov Chains

Markov chains play an important role in many fields including statistics, financial mathematics, graph theory and computer science, to mention but a few.

The stochastic process is denoted by $\{X(t)\ ;\ t \in T\}$, that is, a random variable function of time. The values of X belong to the set of state space denoted as S. The state space could be continuous or discrete, finite or infinite. Time T can also be continuous or discrete. When T is discretized it consists of infinite time points, i.e. $T = \{1, 2, \ldots\}$. Whenever we mention that the state S is discrete, we mean it is finite unless otherwise stated. When the time and state space are discrete, then the process is a called the Markov chain (MC).

Markov chains are discrete-time stochastic processes that describes the events being transitioned from state to state with a certain probability. In the discrete time, the stochastic process becomes a sequence of random variables X_n with the probability distribution λ^n, i.e. $X_n \sim \lambda^n$ (Hoff, 2016).

A MC satisfies the Markov property, which states that the probability of the future actions are independent of the past actions and depend only on the current state (Serfozo, 2009). In other words, the knowledge of the previous state is all that is needed to determine the probability of the current state. This can be expressed mathematically as follows

$$\Pr\left[X_{n+1} = j | X_1, X_2, ..., X_n = i\right] = \Pr\left[X_{n+1} = j | X_n = i\right]. \tag{2.1.1}$$

A stochastic process is called homogeneous if the transition probability depends on the length of the time interval but not on the starting time (Ullah and Wolkenhauer, 2010). Therefore, the time homogeneous MC is defined as

$$\Pr\left[X_{n+t+1} = j | X_{n+t} = i\right] = \Pr\left[X_{n+1} = j | X_n = i\right] = p_{ij}, \quad \forall t, n.$$

The matrix element p_{ij} is known as one-step transition probability from state i to state j. The corresponding one-step transition stochastic matrix is denoted by $P = (p_{ij})$. The two step transition means moving from step i to step j will be done in two steps, which can be expressed as

$$p_{ij}^{(2)} = \Pr\left[X_{n+2} = j | X_n = i\right] = \sum_{k \in S} p_{ik} p_{kj},$$

with corresponding transition matrix $P^{(2)} = \left(p_{ij}^{(2)}\right)$. Similarly, the m step transition probability

$$p_{ij}^{(m)} = \Pr\left[X_{n+m} = j | X_n = i\right],$$

with transition matrix $P^{(m)} = \left(p_{ij}^{(m)}\right)$. In transitioning from state i to state j in m steps, the process can first transition from i to r in $m-k$ steps and from r to j in k steps, $0 < k < m$, that is to say

$$p_{ij}^{(m)} = \sum_{r \in S} p_{ir}^{(m-k)} p_{rj}^{(k)}.$$

The initial distribution $\lambda^{(1)} \in \mathbb{R}^{|S|}$ (a column vector) is given by $\lambda^{(1)}(i) = \Pr[X_1 = i], \quad i \in S$.

The marginal probability is a projection of the join distribution onto one of its random variables. For instance, the marginal distribution of $\Pr[X_{n+1} = j]$ is given by

$$\Pr[X_{n+1} = j] = \sum_{i \in S} \Pr[X_{n+1} = j, X_n = i], \tag{2.1.2}$$

but from the conditional probability we have

$$\begin{aligned}\Pr[X_{n+1} = j] &= \sum_{i \in S} \Pr[X_{n+1} = j | X_n = i] \Pr[X_n = i] \\ &= \sum_{i \in S} p_{ij} \Pr[X_n = i].\end{aligned}$$

It follows that

$$\begin{aligned}\Pr[X_{n+1} = j] &= \sum_{i \in S} p_{ij} \Pr[X_n = i] \\ &= \sum_{i \in S} p_{ij} \sum_{k \in S} p_{ki} \Pr[X_{n-1} = k] \\ &= \sum_{k \in S} \left[\sum_{i \in S} p_{ki} p_{ij}\right] \Pr[X_{n-1} = k] \\ &= \sum_{k \in S} p_{kj}^{(2)} \Pr[X_{n-1} = k],\end{aligned}$$

and after n-step going backward we have the following

$$\Pr[X_{n+1} = j] = \sum_{i \in S} p_{ij}^{(n)} \Pr[X_1 = i].$$

We can deduce from the above that

$$\begin{aligned}&{\lambda^{(2)}}^T = {\lambda^{(1)}}^T P; \quad {\lambda^{(3)}}^T = {\lambda^{(2)}}^T P = {\lambda^{(1)}}^T PP = {\lambda^{(1)}}^T P^2 \\ &{\lambda^{(4)}}^T = {\lambda^{(3)}}^T P = {\lambda^{(2)}}^T PP = {\lambda^{(1)}}^T PPP = {\lambda^{(1)}}^T P^3.\end{aligned}$$

Hence

$${\lambda^{(n+1)}}^T = {\lambda^{(1)}}^T P^n, \qquad P^n = \left(p_{ij}^{(n)}\right).$$

It therefore follows that $P^{(n)} = P^n$.

Thus the probability of the chain after n steps is obtained from the one step transition matrix P and the distribution of the initial probabilities $\lambda^{(1)}$.

If the $\lim_{n\longrightarrow\infty} \lambda^n$ exists as $n \longrightarrow \infty$, then there is a stationary solution (or steady state solution)

$$\lim_{n\longrightarrow\infty} \lambda^n = \pi,$$

which satisfies

$$\begin{aligned} 0 \leq \pi_j \leq 1, \quad \forall j \in S \\ \sum_{j\in S} \pi_j = 1 \\ \pi^T P = \pi^T. \end{aligned}$$

The question of interest is whether or not a distribution π always exists, and is it unique? Below we introduce the necessary and sufficient conditions to guarantee the existence and uniqueness of the stationary distribution. Let us start with some definitions.

2.1.1 Definition. Reducibility: An irreducible MC is a chain in which every state can be accessed from any state with non-zero probability in finite amount of time. For instance, a state j is accessible from a state i if there exists an integer k such that

$$p_{ij}^{(k)} = \Pr\left[X_{n+k} = j | X_n = i\right] > 0, \quad \forall i, j.$$

We can deduce from the above definition that is any state accessible from itself (Hoff, 2016).

2.1.2 Definition. Periodicity: A state i is a periodic if the process starts from it and returns in a finite amount of time with positive probability. If state i has a period of k, then it can be expressed as

$$k = \gcd\{n > 1 : \Pr\left[X_n = i | X_1 = i\right] > 0\},$$

assuming that the set is not empty. Here $\gcd$ means the greatest common divisor. In case $k = 1$, the state is called aperiodic and returning back to the same state occurs at irregular times. If we start at state i and never return back to the same state, then the state i is called **transient**. A MC is aperiodic if all the states are aperiodic (Hoff, 2016).

2.1.3 Definition. Reversibility: A MC is reversible if there exists a stationary distribution π such that

$$\pi_i \Pr\left[X_{n+1} = j | X_n = i\right] = \pi_j \Pr\left[X_{n+1} = i | X_n = j\right], \quad \forall\, n \in T, \quad i, j \in S.$$

This condition is called the detailed balance (Hoff, 2016). For example, one can say Witwatersrand University is in a detailed balance, if the number of accepted students per unit time equals the number of graduated students per unit time.

So if a MC is reversible then it follows that

$$\begin{aligned} \sum_{j\in S} \pi_j \Pr\left[X_{n+1} = i | X_n = j\right] &= \sum_{j\in S} \pi_i \Pr\left[X_{n+1} = j | X_n = i\right] \\ &= \pi_i \sum_{j\in S} \Pr\left[X_{n+1} = j | X_n = i\right] = \pi_i\,, \end{aligned}$$

or in matrix notation as

$$\pi^T = \pi^T P. \tag{2.1.3}$$

It is clear from (2.1.3) that Reversibility implies stationarity.

2.1.4 Definition. Mean recurrence time & positive recurrence: Mean recurrence is the expected return time to state i and it is denoted by M_i. Formally,

$$M_i = E[T_i] = \sum_{n=1}^{\infty} n \cdot p_{ii}^{(n)}.$$

A state i is called a **positive recurrence**, if the expected amount of time of returning back to the state i once it leaves is a finite time, that is M_i is finite.

2.1.5 Definition. Ergodicity: When the state is aperiodic and positive recurrent, then it called an ergodic state. If all the states of MC are irreducible and ergodic, then the chain is called an ergodic MC (Hoff, 2016).

Now we have the tools to identify the necessary and sufficient conditions for a MC to have a stationary distribution. A MC has a unique stationary distribution if it is ergodic, while the reversibility is a sufficient condition for stationarity. Under those circumstances, the stationary distribution is completely characterized (Hoff, 2016).

If we set the initial distribution of X_1 to be π, then a Markov chain will have a unique stationary distribution π if it satisfies the reversibility condition (2.1.3) and ergodicity. If we set $\lambda^{(n)}$ to be the distribution of X_n (that is, $\lambda_j^{(n)} = P[X_n = j]$), then

$$\begin{aligned}
\lambda^{(n)^T} &= \pi^T P^n \\
&= \left(\pi^T P\right) P^{n-1} = \pi^T P^{n-1} \\
&= \left(\pi^T P\right) P^{n-2} = \pi^T P^{n-2} \\
&\vdots \\
&= \pi^T P \\
&= \pi^T.
\end{aligned}$$

In this case, no matter what the starting distribution is, the MC will converge to a unique stationary distribution π.

In the following section we will introduce the notion of dynamic programming and the Bellman optimality equation which they use to model RL problems.

2.2 Dynamic Programming

Dynamical programming (DP) is a mathematical method directed to solve the optimization problems that arises from multi-stage decision processes Bellman (1954). In DP, the problem is broken down into smaller sub-problems; solutions of each of the subproblem are stored to use in future iterations or stages. This characteristic makes DP special, which guarantees the efficiency and correctness which is not provided on most of the methods used to solve problems.

Sub-problems are smaller versions of the original problem. A good formulation of the sub-problems leads to obtain the original solution. Hence sub-problems overlap, and the core idea of DP is to avoid repeated work by memorizing solutions of sub-problems. The dynamic part of DP is that, we only has

to apply one function recursively to the problem, and this function will return optimal values of the sub-problem as well as the full problem.

A problem can be solved using DP if:

- It has an overlapping sub-problems;
- The optimal solution of the original problem is attainable from the optimal solutions of its sub-problems.

Thus, DP solves any problem in multi-stages recursively. At each stage there are decision variables, called actions, that have to be decided and these actions alter the problem from state to state (possibly with some probability) and receive a reward for this decision. Solving the problem in one step defines the characteristics of the next step. Finally, a recursive optimization procedure is needed to solve a problem using DP. A recursive procedure must satisfy the well known Bellman optimality condition.

The **Bellman optimality condition** is defined as: "An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" (Bellman, 1954).

Consider the multi-stage decision process where the reward of a particular stage is given by r: $r_{t+1} = f_t(a_t, s_t)$ where action a_t taken at stage t in state s_t. The new state $s_{t+1} = F_t(a_t, s_t)$. Since the DP approach consists of a recursive optimization procedure, we can define the following optimal value function $v_t(s_t)$ as the maximum return of cumulative rewards up to t stage (or over the t stages remaining):

$$v_t(s_t) = \max \{f_1(a_1, s_1) + \cdots + f_{t-1}(a_{t-1}, s_{t-1}) + f_t(a_t, s_t)\}$$

subject to $s_{i+1} = F_i(a_i, s_i)$, $i = 1, \ldots, t$.

The above equation can be rewritten as:

$$v_t(s_t) = \max\{f_1(a_1, s_1) + \cdots + f_{t-1}(a_{t-1}, s_{t-1})\} + \max f_t(a_t, s_t).$$

It follows that

$$v_t(s_t) = \max_{a_t} \{v_{t-1}(s_{t-1}) + f_t(a_t, s_t)\}. \tag{2.2.1}$$

In DP problems, we use the idea of the value function to search for good policies. The value functions must satisfy the Bellman equation (2.2.1).

Since, now we know the MC and Bellman optimality equation, we are in the stage to introduce Markov decision process and reinforcement leaning problems which are presented in the following section.

2.3 Markov Decision Process

Markov decision process (MDP) is a mathematical method for reinforcement learning (RL). An MDP consists of an environment, an agent, a goal, actions, and rewards. The agent act as decision maker which interacts with the environment. These interactions occurs sequentially over time. At each time

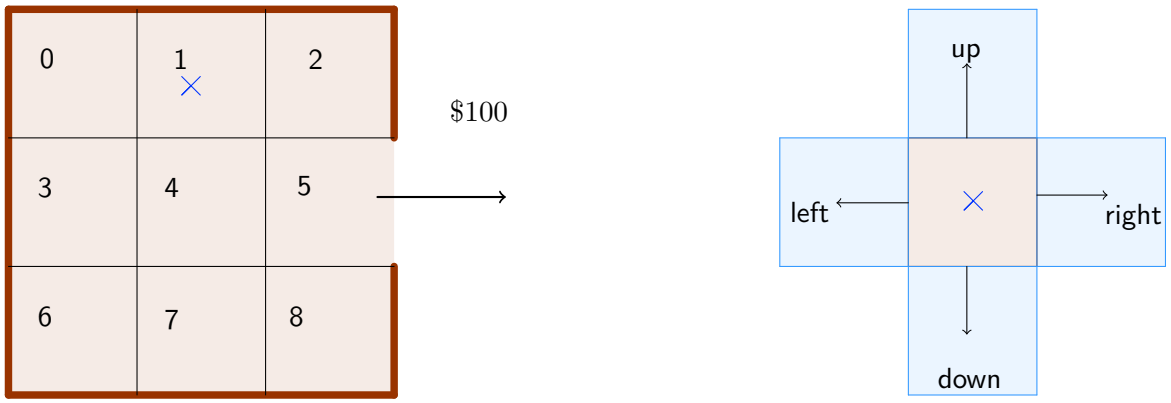


Figure 2.1: A 3×3 room problem. The agent is required to find the gate though a series of interactions with the environment without any prior knowledge where the gate is located using the notion of reward. The figure on right hand side describes the possible actions that the agent may take.

step, the agent selects an action to make a transition from state to state. Accordingly, the agent will transition to a new state and will be given a reward as consequence of the action.

An MDP (Jonsson and Gómez, 2016) is defined as $M = \langle S, A, P, R \rangle$, where S define the set of states; A a set of actions that the agent can choose; $P : S \times A \times S \mapsto [0, 1]$ defines the probability of transitioning from state to another under given action satisfying the stochastic property $\sum_{s'} P(s'|s, a) = 1$; and $R : S \times A \mapsto \mathbb{R}$ represent the reward signal from the environment for performing an action in a particular state.

The formulation of an MDP is based on the Markov property, in which the state of stochastic process at time t_k it depends only on the previous state t_{k-1} and independent from the all previous states. That is, the transition probability to a new state is depend only on the current state.

Consider the following Room problem in Figure 2.1 which has 9 discrete states, $S = \{0, 1, \cdots 8\}$, and 5 discrete actions, A=$\{$up, down, right, down, stay$\}$. The cross sign here represents the agent. The agent start at random position and chooses an action. Every time the agent make transition to new state or stay still losses one dollar. When the agent selects the action "right" at state 5, i.e. the agent finds the gate, it will receive a $\$100$ as reward. The agent's goal is to maximize the future rewards by selecting a good actions relative to its state.

The process of selecting an action on a given state to make a transition to a new state happens sequentially over time creating a trajectory of states, actions, and rewards. Throughout the process, the agent's goal is to maximize its cumulative rewards that it receives from taking actions on states by learning the optimal policy π that map between the states and the actions, i.e. $\pi : S \mapsto A$.

At each time step $t = 1, 2, \ldots,$ the agent receives some representation of the environment's state $s_t \in S$.

Based on this state, the agent selects an action $a_t \in A$. This gives us the state-action pair (s_t, a_t).

The process of receiving a reward r_t at time t because of performing an action a_t on state s_t can be interpreted as an arbitrary function that maps state-action pairs to rewards, that is to say

$$\mathcal{R}(s_t, s_t) = r_{t+1}.$$

So, the trajectory of representing the sequential process of transition from a state to another by applying an action and receiving a reward can be represented as

$$s_1, a_1, r_2, s_2, a_2, r_3, \cdots .$$

In our context, the sets S and R are finite, and since the transition of agent from state to state follow a well defined probability distribution, then we can treat the parameters s_t , r_t and a_t at time t as random variables with well specified probability depending on the previous state and action at time $t-1$.

For instance, consider $s' \in S$ and $r \in R$. At time t, there is some probability that $s_t = s'$ and $r_t = r$. The probability of transitioning from state s to state s' by taking the action a and receiving a reward r is stated as

$$p\left(s', r \mid s, a\right) = \Pr\left[s_t = s', r_t = r \mid s_{t-1} = s, a_{t-1} = a\right],$$

where $A(s)$ is set of admissible actions at state s. This follows the definition of a Markov Process in Section 2.1.

We stated that the objective of learning of an agent in an MDP is to maximize its cumulative rewards over time. We need a formal way to represent this cumulative rewards. Given a sequence of the rewards $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$, we introduce the notion of the expected return of the rewards at times t, denoted by G_t, as

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_{t_f},$$

where t_f is the final time step.

Maximizing the expected return G_t is what is driving the agent to make a good decisions that it wishes.

Episodic and continuing tasks

In the definition of the expected return, we introduced the final time step t_f where the final state is executed and we receive the final reward. When we have the notion of the final time step, the interaction between the agent and the environment can be broken up into a subsequences called episodes.

Each episode terminates at final time step t_f. After that the environment will be reset into a new starting state (may be the same previous starting state). Usually the tasks with episodes are called episodic tasks.

On the other hand, when the interaction between the agent and the environment happens continuously without time limit, the tasks are called continuing tasks. In this case, the definition of the expected return will be problematic, because the final time step now is unbounded, i.e. $t_f = \infty$, and therefore the expected return may be infinite also. Because of this, we need to redefine the expected return to include the continuing tasks.

Discounted return

In the case of continuing tasks we need to refine the definition of the expected return to be more general. In this case, the agent's goal will be is to maximize its discounted return of rewards.

To define the discounted return, we introduce the discount factor $\gamma \in (0, 1)$. When the agent chooses an action a_t, then the expected discounted return will be (Sutton and Barto, 2018)

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \end{aligned} \tag{2.3.1}$$

The discount factor determines how much the future rewards are valuable. When $\gamma \ll 1$, the agent will be concerning more by maximizing immediate rewards. While as γ approaches 1, the agent becomes far-sighted and the discounted return takes the future rewards into account more strongly.

Now, we are presenting an important relationship in reinforcement learning that shows how returns at sequential time steps are related to each other

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+3} + \cdots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+3} + \cdots \right) \\ &= r_{t+1} + \gamma G_{t+1}. \end{aligned}$$

Despite the fact that the discounted return at time t is a sum of infinite terms, the return is finite provided that the reward is non-zero and constant, and that $\gamma < 1$.

Now let us return back to our agent, with all possible actions that the agent might take in any state in the environment, there is a couple of questions of interest.

Firstly, we would like to know how the agent will be able to select any given action in any given state. This is where the notion of policies comes into play.

Secondly, after we are able to know the probability of selecting an action in a given state, we need to know the quality of choosing an action in a given state for the agent. In terms of rewards, selecting an action from a given state will results in either decreasing or increasing the agent's rewards, and by knowing this beforehand it will help the agent to decide which action to choose in a given state. This is where value functions become useful, is used to organize and structure the search for good policies (Sutton and Barto, 2018).

A policy $\pi : S \mapsto A$, as stated in Sutton and Barto (2018), is a function that maps a given state to probabilities of selecting each possible action from that state. Like π described in Section 2.1, $\pi(s)$ is a discrete probability distribution over $A(s)$, i.e. over admissible actions set.

A value function $V^\pi : S \mapsto \mathbb{R}$, is defined for each state as the expected discounted cumulative reward for all future time steps under a given policy π. This defines the concept of "how good" a state is as (Earle, 2019):

$$\begin{aligned} V^\pi(s) &= E_\pi \left[G_t \mid s_t = s \right] \\ &= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right], \end{aligned} \tag{2.3.2}$$

where E_π is the expected value under the policy π. Hence the optimal policy π^* can be defined as

$$\pi^* = \arg\max_\pi E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid \pi, s_t = s \right]. \tag{2.3.3}$$

The optimal policy π^* is not necessarily unique, but all optimal policies share an optimal value function V^* (Sutton and Barto, 2018). The value function defined above is called the **state-value function**.

Another useful quantity is the **action-value function** for a policy π, commonly known as Q-value and denoted as Q^π. The objective of Q-value is to tells us how good it is for the agent to take any given action from a given state while following policy π. To put it differently, it gives us the value of an action under the policy π.

Formally, the value of action a in state s under policy π is the expected return from starting from state s at time t, taking action a, and following policy π thereafter. Mathematically, we define $Q^\pi(s,a)$ as

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\left[G_t \mid s_t = s, a_t = a\right] \\ &= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right]. \end{aligned} \tag{2.3.4}$$

The model-fee learning in RL is known as Q-learning which is based on Q-value.

Both the value function and the action-value functions satisfy the well-known the Bellman Optimality condition; given below for value functions:

$$V(s) = \max_{a \in A}\left\{R(s,a) + \sum_{s' \in S} \Pr(s'|s,a)V(s')\right\}. \tag{2.3.5}$$

Since there is a probability of moving from state s to any other state s' under action a, the total probability (excepted reward) $\sum_{s' \in S} \Pr(s'|s,a)V(s')$ is a function of action a.

Hence, the optimal value function at time t is given by

$$V_t^*(s) = \max_{a \in A}\left\{R(s,a) + \sum_{s' \in S} \Pr(s'|s,\pi(s))V_{t+1}^*(s')\right\}. \tag{2.3.6}$$

The value function can be solved by a variety of different methods such as value iteration and policy iteration (for more details see Sutton and Barto (2018)). Unfortunately, those remain plagued by the curse of dimensionality. It is a well-known fact that the number of parameters that an agent must learn increases exponentially as the size of the states, $|S|$, and action space, $|A|$, grow. An online algorithm, called Q-learning, for MDPs was suggested by Watkins (1989) that makes a local updates on the value function online instead of finding a global solution. Given an agent transitioning from state s_t to state s_{t+1} when taking the action a_t and receiving reward r_t, Q-learning makes the following updates to find the optimal action-state value function:

$$Q(s_t,a_t) = (1-\alpha)\,Q(s_t,a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1},a)\right), \tag{2.3.7}$$

where $\alpha, \gamma \in (0,1)$ denoting the learning rate and the discounting factor respectively.

2.4 Hierarchical Reinforcement Learning

The value function and action-value function based RL are inefficient when the number of states or actions increases exponentially making the computational complexity worse. One of the approaches of tackling the curse of dimensionality is Hierarchical Reinforcement Learning (HRL).

HRL approaches aimed at tackling the problem of curse of dimensionality by breaking the RL problem into a hierarchical sub-problems where each sub-problem is a RL problem. For example consider an agent in a particular environment that needs to select between 12 objects. Each object is characterized by its colour and shape, where the colour can be red, green or blue, and the shape can be circle, triangle, square and rectangle. Each colour and shape has specific reward where the reward of the object is given by R(object)= R(colour) + R(shape). The agent aims to find best combination of the characteristics of the object. In classical approach, we solve this problem by finding all the combinatorial, i.e. the product space of the problem which is $3 \times 4 = 12$. In a HRL we decompose the problem into two sub-problems. The first one is to find the best colour which has the highest reward, and the second one is to find the shape that has the highest reward. After solving these sub-problems separately, we pick the optimal (colour, shape) and we combine them and the total cost of solving them separately will be $3 + 4 = 7$ which is much better than working in product space.

Additionally, a HRL can be seen as a way of transferring knowledge. Going back to our example of the object, suppose that the agent is tasked to find the object with (blue, circle) and now asked the agent to find a new characteristics, say (green, circle). If the agent was found the solution using the combinatorial approach, then we need to solve the problem from scratch to find the object with the property (green, circle). In contrast, in HRL the structure is immediately transferable which the agent can used it to find the best colour and best shape without the need to solve the original problem again.

The use of HRL helps to reduce the curse of dimensionality by using the existing solutions to construct a new solution instead of solving the problem from scratch. Some HRL methods are described as follow:

2.4.1 Options. Options framework is a hierarchical approach in which the action space is extended to include a new temporal actions called options. The options aims to help the agent to reach the goal gradually through levels of hierarchy.

Each option is described, as in (Earle, 2019), by a tuple $\langle \mathcal{I}, \pi, \mathcal{T} \rangle$; where $\mathcal{I} \subset S$ is called the initialization set which defines the set where the option can be executed. $\pi : S \mapsto A$ is the option policy which defines the probability over actions for each state of the option; and $\mathcal{T} : S \mapsto [0, 1]$ is a termination map of the option which defines the probability of terminating the option in each state.

At time t in state s_t, the selection of the next action a_t is governed by the distribution $\pi(s_t, \cdot)$ while the termination is governed by $\mathcal{T}(s_t)$. Subsequently, the environment will make transition to a new state s_{t+1} where the option either select a new action a_{t+1} according to $\pi(s_{t+1}, \cdot)$, or terminate with probability $\mathcal{T}(s_{t+1})$ and so on.

Generally speaking, the primitive actions can be seen as one-step option. Every primitive action a corresponds to an option with the initialization set $\mathcal{I} = \{s : a \in A(s)\}$, where the option selects a everywhere $\pi(s) = a\,, \forall s \in \mathcal{I}$ and terminates after one time step $\mathcal{T}(s) = 1\,, \forall s \in S$. Thus, all the agent's choice can be considered from options, where some of the options terminate after one time-step (primitive action) and others need more time to finish.

By extending the action space to include options, the problem is no longer an MDP. These type of problems are modelled as Semi-Markov Decision Process (SMDP).

Semi-Markov Decision Process (SMDP). In the MDP, the time it takes to complete an action is one unit. On the other hand, the amount of time to complete an action on the class of SMDP is not fixed, since it take more than one-time step. Accordingly, the Bellman equation (2.3.5) is adjusted to

be in the form

$$\begin{aligned} V(s) &= \max_{a \in A} E_\tau \left\{ R(s,a,\tau) + \sum_{s'} \Pr(s', \tau | s, a) V(s') \right\} \\ &= \max_{a \in A} \sum_{\tau} \sum_{s'} \Pr(s', \tau | s, a) \left\{ R(s,a,\tau) + V(s') \right\}, \end{aligned} \tag{2.4.1}$$

where τ is the associated time step with action a; $R(s,a,\tau)$ is the expected reward when the action a is applied in the state s in τ time steps and $\Pr(s', \tau | s, a)$ is the probability of transition from state s to state s' when applying the action a in τ steps (Jonsson and Gómez, 2016).

SMDP can be reduced to MDP in the following steps. By defining the new reward function as $\bar{R}(s,a) = \sum_\tau \sum_{s'} \Pr(s', \tau | s, a) R(s,a,\tau)$, and the transition probability as $\bar{\Pr}(s'|s,a) = \sum_\tau \Pr(s', \tau | s, a)$, the SMDP is reduced to the form

$$V(s) = \max_{a \in A} \left\{ \bar{R}(s,a) + \sum_{s'} \bar{\Pr}(s'|s,a) V(s') \right\}, \tag{2.4.2}$$

which can be solved as an MDP.

2.4.2 MAXQ Decomposition. The MAXQ decomposition is a hierarchical reinforcement learning procedure that aims to learn a policy $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ in a hierarchical fashion by decomposing an MDP $M = \langle S, A, P, R \rangle$ into a finite set of subtasks $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ representing acyclic graph with root task M_1. Each subtask $M_i, 1 \leq i \leq n$, consists of a tuple $\left\langle T_i, A_i, \hat{R}_i \right\rangle$, where $T_i \subset S$ is a set of termination states of the subtask M_i; $A_i \subset \mathcal{M}$ is set of actions, and $\hat{R}_i : T_i \mapsto \mathbb{R}$ is pseudo reward function.

A subtask M_i is a primitive if it is a leaf and has no subtasks, that is to say $A_i = \emptyset$, and it correspond to an action $a \in A$ in the original MDP and terminates after one time step. On the contrary, a non-primitive subtasks can only be applied in the non-terminal states $S \setminus T_i$. Terminating in state $s \in T_i$ results in a pseudo reward $\hat{R}_i(s)$.

Since MAXQ has a graphical structure, the node M_j is connected to the node M_i only if M_j is an action of the subtask M_i, that is $M_j \in A_i$.

In order to find an optimal policy $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, where each policy $\pi_i : S \mapsto A_i$ indicates how each M_i chooses actions in order to be in good states. Each subtask defines its own value function $V_i(s)$ for each $s \in S$ and terminates when M_i chooses state $s \in T_i$. A reward of applying the action $M_j \in A_i$ on the state s is estimated by the value function of the subtask M_j, i.e $\mathcal{R}(s, M_j) = V_j(s)$. Therefore the Bellman optimality equation of the subtask M_i decomposes as

$$V_i(s) = \max_{M_j \in A_i} \left\{ V_j(s) + \sum_{s'} P(s'|s, M_j) V(s') \right\},$$

where $P(s'|s, M_j)$ is the probability of transitioning from the state s to the state s' when we applying the action M_j. When M_j is primitive subtask corresponding to the action a in original MPD, the reward of applying the action a in state s will be $\mathcal{R}_i(s,a) = \mathcal{R}(s,a)$. The pseudo-reward $\hat{R}_i$ has no contribution to the value function, rather is used to learn an optimal policy π_i.

To give a brief illustration about MAXQ decomposition, let us consider the following Taxi problem as it described in (Dietterich, 1998). Figure 2.3 shows a 5-by-5 grid-world map with four locations marked as R, B, G, and Y and a taxi (the cross sign). There is rider at one of these locations (randomly chosen)

yearn to be travelled to one of the four locations (also random). Then the taxi's job is to pick up the rider and delivers him/her to the right destination.

The action space consist of 6 primitive actions, the four directions (Up, Down, Right, Left), besides the Pickup and Putdown actions. The taxi receives 20 points as reward for delivering the rider to the right location, and losses 1 point at each time step. There is also -10 point penalty for illegal Pickup and Putdown actions.

Fot this problem, we have created a straightforward hierarchical architecture represented by two main subtasks: Get the rider and Put the rider. Each of these subtasks depends on the subtask Navigate which navigating the taxi to one of the four location and then performing a Pickup or Putdown action. We have compared the Q-learning with MAXQ decomposition. The results shows, as in Figure 2.4, that MAXQ outperforming Q-learning in throughput and convergence time.

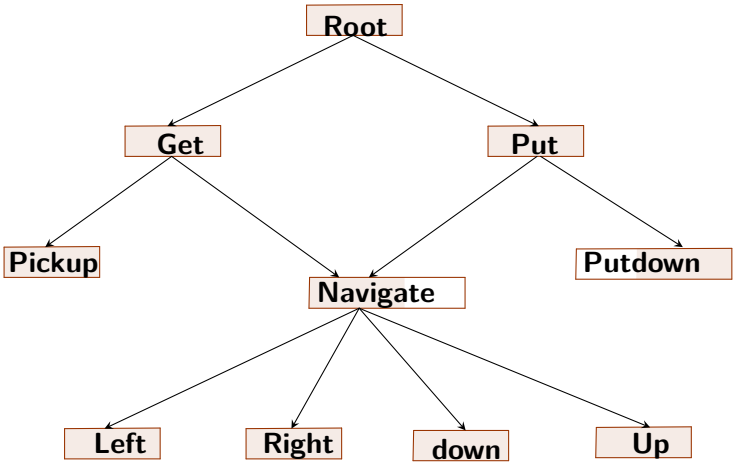


Figure 2.2: The hierarchy of the Taxi problem

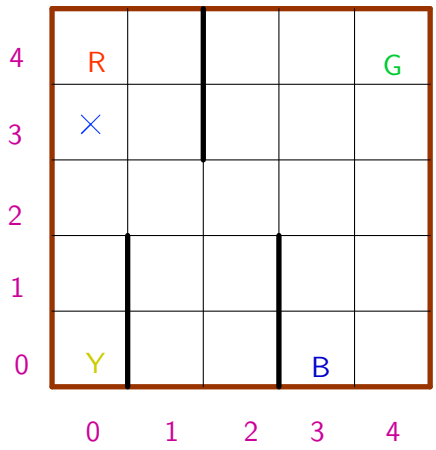


Figure 2.3: The Taxi Domain

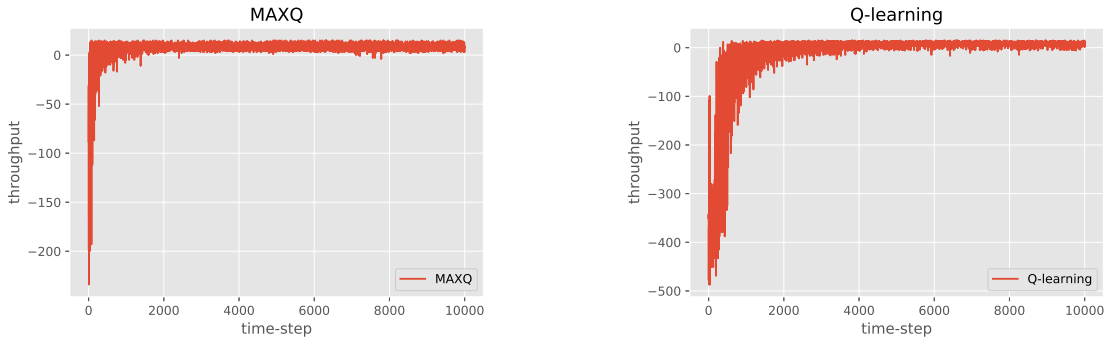


Figure 2.4: Comparision between Q-learning and MAXQ decomposition

All the methods we discussed so far try to solve the curse of dimensionality by constructing a hierarchical structure. But none of them support a notion of decomposing a new policy from the previous optimal policies. Instead of using the previous learned policies to find the new policy, the new policy must be solved by using the Bellman optimality equation from beginning. In Chapter 4 we will show how benefits from the previously learned policy is used in constructing a new policy without incurring new significant cost.

3. Linearly Solvable-Markov Decision Processes

Richard Bellman (Bellman, 1954) made a revolution in the theory of dynamic programming. The Bellman equation (2.3.6) can be solved iteratively to find an optimal solution, but the computational size grows exponentially when the number of states increases. In addition, the actions are executed sequentially and only one action at a given point of time is admissible, which we can not build up an action from several subtasks they run in parallel.

Furthermore, the Bellman equation (2.3.6) is non-linear, which means that the optimal solutions of the value functions can not be decomposed in a simple way. For instance, given any two MDPs $M_1 = \langle S, A, P, R_1 \rangle$ and $M_2 = \langle S, A, P, R_2 \rangle$, they share the same state space, action space, and the probability distributions and differs on their state-rewards. The MDP $M_{1+2} = \langle S, A, P, R_1 + R_2 \rangle$ can not be obtained from any combination between M_1 and M_2. In fact, we need to solve Equation (2.3.6) again to find the optimal solution V_{1+2}.

Our goal is to solve those problems by making the Bellman equation linear, and by doing so, we can solve multiple problems simultaneously through a hierarchical framework.

This chapter organized as follows. In Section 3.1 we are going to construct a class of stochastic problems called Linearly-solvable Markov Decision Processes (LMDPs). In Section 3.2 we extending the class of LMDPs to include transition dependent reward.

3.1 The Formulation of LMDPs

In this section, we aim to present the offline method of LMDP which we considered to solve our problem. The method will be described in three different ways as follows. We will first show the proof of LMDP, followed by the mathematical description of the method and finally the pseudo-code (Algorithm 1) derived from the method.

In line with our usual notations, S is a finite set of states, $A(s_i)$ a set of possible actions that the agent can takes while in state $s_i \in S$, $\mathcal{R}(s_i, a)$ is a cost of being in state s_i and choosing action $a \in A(s_i)$, and $P(a)$ is stochastic matrix whose elements $p_{ij}(a)$ is the transition probability from state s_i to state s_j while taking the action a.

The Bellman equation, depending on the cumulative cost function, takes different forms as given in (Todorov, 2009b)

$$\begin{aligned}
&\text{First exit total cost} && V(s_i) = \max_{a \in A(s_i)} \left\{ \mathcal{R}(s_i, a) + \sum_{s'} \Pr(s_j \mid s_i, a) V(s_j) \right\}, \\
&\text{Infinite horizon discounted cost} && V(s_i) = \max_{a \in A(s_i)} \left\{ \mathcal{R}(s_i, a) + \gamma \sum_{s'} \Pr(s_j \mid s_i, a) V(s_j) \right\}, \\
&\text{Finite horizon total cost} && V_t(s_i) = \max_{a \in A(s_i)} \left\{ \mathcal{R}(s_i, a) + \sum_{s'} \Pr(s_j \mid s_i, a) V_{t+1}(s_j) \right\},
\end{aligned} \tag{3.1.1}$$

where s_j in $V(\cdot)$ and $V_t(\cdot)$ is next state due to the action a. In the discounted cost formulation, the constant $0 < \gamma < 1$ is a discount factor. The only formula that is time dependent is the finite horizon total cost, while the others are time invariant.

In this class of new MDPs, we define the action $\boldsymbol{a} \in \mathbb{R}^{|S|}$ to be a real valued function with dimensionality $|S|$. Given the uncontrolled transition probability matrix P, we define the controlled transition probabilities as follows:

$$\pi_{ij}(\boldsymbol{a}) = p_{ij} \exp(a_j), \tag{3.1.2}$$

where $\pi(0) = P$. If $p_{ij} = 0$, implies $\pi_{ij}(\boldsymbol{a}) = 0$. Then by imposing the condition that $\pi(\boldsymbol{a})$ must have row-sums equal 1 in order to obtain stochastic matrix, that is to say

$$\sum_j p_{ij} \exp(a_j) = 1. \tag{3.1.3}$$

The action (control) vector $\boldsymbol{a}$ has direct interact with the transition probabilities. The uncontrolled matrix P is a random walk. Thus, in order to measure the difference between the controlled and uncontrolled distribution we use KL divergence, which panellizing the controls that are significant different from P.

Now, from the definition of KL divergence, we define the cost $\mathcal{R}(s_i, \boldsymbol{a})$ for selecting the action $\boldsymbol{a}$ in state s_i as:

$$\begin{aligned} \mathcal{R}(s_i, \boldsymbol{a}) &= R(s_i) - \mathsf{KL}(\boldsymbol{\pi_i(a)} || \boldsymbol{p_i}) \\ &= R(s_i) - \sum_{j:p_{ij}\neq 0} \pi_{ij}(\boldsymbol{a}) \log \frac{\pi_{ij}(\boldsymbol{a})}{p_{ij}}, \end{aligned} \tag{3.1.4}$$

where $R(s_i)$ is the reward associated with state s_i, and $\boldsymbol{\pi}_i$ and $\mathbf{p}_i$ are the i^{th} rows of π and P respectively. Substituting (3.1.2) into (3.1.4), the cost function becomes

$$\mathcal{R}(s_i, \boldsymbol{a}) = R(s_i) - \sum_j \pi_{ij}(\boldsymbol{a}) a_j. \tag{3.1.5}$$

Then, by substituting (3.1.5) into Equation (2.3.5), the Bellman equation becomes

$$\begin{aligned} V(s_i) &= \max_{a_j \in A(s_i)} \left\{ R(s_i) - \sum_j \pi_{ij}(\boldsymbol{a}) a_j + \sum_j \pi_{ij}(\boldsymbol{a}) V(s_j) \right\} \\ &= R(s_i) + \max_{a_j \in A(s_i)} \left[\sum_j p_{ij} \exp(a_j) \left(V(s_j) - a_j \right) \right]. \end{aligned} \tag{3.1.6}$$

The maximization problem in Equation (3.1.6) subject to the condition (3.1.3), is obtained using Lagrange multipliers as follows. For each i, define the Lagrangian

$$\mathcal{L}(i, \boldsymbol{a}) = \sum_j p_{ij} \exp(a_j) \left(V(s_j) - a_j \right) - \lambda_i \sum_j \left(p_{ij} \exp(a_j) - 1 \right), \tag{3.1.7}$$

where the constant $R(s_i)$ has been ignored.

The necessary condition for extremum with respect to a_j is

$$0 = \frac{\partial \mathcal{L}}{\partial a_j} = p_{ij} \exp(a_j)(V(s_j) - a_j - \lambda_i - 1), \tag{3.1.8}$$

where when $p_{ij} \neq 0$ the only solution is

$$a_j^*(i) = V(s_j) - \lambda_i - 1. \tag{3.1.9}$$

The sufficient condition for the maximization is guaranteed by taking the second derivative of Lagrangian given the value of a_j obtained from Equation (3.1.9) as

$$\begin{aligned} \frac{\partial^2 \mathcal{L}}{\partial^2 a_j}\Big|_{a_j=a_j^*(i)} &= p_{ij}\exp(a_j)(V(s_j) - a_j - \lambda_i - 2)\Big|_{a_j=a_j^*(i)} \\ &= -p_{ij}\exp(a_j) < 0\,. \end{aligned} \tag{3.1.10}$$

Therefore the value of a_j which was obtained in Equation (3.1.9) is optimal. To find the Lagrange multiplier λ_i, we will apply the constraint (3.1.3) on the optimal control Equation (3.1.9), resulting in

$$\lambda_i = \log\left(\sum_j p_{ij}\exp(V(s_j)\right) - 1. \tag{3.1.11}$$

Therefore the optimal control law is

$$a_j^*(i) = V(s_j) - \log\left(\sum_k p_{ik}\exp(V(s_k)\right). \tag{3.1.12}$$

Therefore, the optimally-controlled transition probabilities will becomes

$$\pi_{ij}(\boldsymbol{a^*(i)}) = \frac{p_{ij}\exp(V(s_j))}{\sum_k p_{ik}\exp(V(s_k))}. \tag{3.1.13}$$

Substituting the optimal control (3.1.13) in the Bellman equation (3.1.6) and dropping the $\max$ operator

$$\begin{aligned} V(s_i) &= R(s_i) + \sum_j \pi_{ij}(\boldsymbol{a^*(i)})\left(V(s_j) - a_j^*(i)\right) \\ &= R(s_i) + \log\left(\sum_k p_{ik}\exp(V(s_k)\right)\sum_j \pi_{ij}(\boldsymbol{a^*(i)}) \\ &= R(s_i) + \log\left(\sum_k p_{ik}\exp(V(s_k)\right), \end{aligned}$$

and by exponentiating both sides yields

$$\exp(V(s_i)) = \exp(R(s_i))\left(\sum_j p_{ij}\exp(V(s_j)\right). \tag{3.1.14}$$

Introducing the desirability function

$$z(i) = \exp(V(s_i)), \tag{3.1.15}$$

then Equation (3.1.14) is written as

$$z(i) = \exp(R(i))\left(\sum_j p_{ij} z(j)\right). \tag{3.1.16}$$

Writing the vector $\boldsymbol{z}$ with the elements $z(i)$, and the diagonal matrix Q with the diagonal elements $\exp(R(i))$, the Equation (3.1.16) becomes

$$\mathbf{z} = QP\mathbf{z}. \tag{3.1.17}$$

The Equation (3.1.17) is an eigenvalue problem that can be solved iteratively using the power method as

$$\mathbf{z_{k+1}} = QP\mathbf{z_k}, \qquad \mathbf{z_0} = \mathbf{1}. \tag{3.1.18}$$

Let $\mathcal{N}$ and $\mathcal{T}$ be the sets of non-terminal and terminal states respectively. We can write Equation (3.1.17) as

$$\begin{bmatrix} \mathbf{z}_\mathcal{N} \\ \mathbf{z}_\mathcal{T} \end{bmatrix} = \begin{bmatrix} Q_\mathcal{N} & O \\ O & Q_\mathcal{T} \end{bmatrix} \begin{bmatrix} P_{\mathcal{N}\mathcal{N}} & P_{\mathcal{N}\mathcal{T}} \\ O & I \end{bmatrix} \begin{bmatrix} \mathbf{z}_\mathcal{N} \\ \mathbf{z}_\mathcal{T} \end{bmatrix}.$$

The reason that we have the O and the identity matrices in the probability matrix is because of the absorbing nature of the terminal states in which they can not transition back to the non-terminal states. Moreover, since the value function of the terminal states is equal the reward of those states, that is to say $V_\mathcal{T}(s) = R(s)\,, \forall s \in \mathcal{T}$, it follows that $\mathbf{z}_\mathcal{T} = \exp(\{R(s)|\ s \in \mathcal{T}\}) = \mathbf{q}_\mathcal{T}$. Accordingly, we have

$$\begin{bmatrix} \mathbf{z}_\mathcal{N} \\ \mathbf{z}_\mathcal{T} \end{bmatrix} = \begin{bmatrix} Q_\mathcal{N} P_{\mathcal{N}\mathcal{N}} \mathbf{z}_\mathcal{N} + Q_\mathcal{N} P_{\mathcal{N}\mathcal{T}} \mathbf{z}_\mathcal{T} \\ Q_\mathcal{T} \mathbf{z}_\mathcal{T} \end{bmatrix}. \tag{3.1.19}$$

Then, by comparing element-wise of the matrix in Equation (3.1.19), we have

$$\begin{aligned} \mathbf{z}_\mathcal{N} &= Q_\mathcal{N} P_{\mathcal{N}\mathcal{N}} \mathbf{z}_\mathcal{N} + Q_\mathcal{N} P_{\mathcal{N}\mathcal{T}} \mathbf{z}_\mathcal{T} \\ \Big(I - Q_\mathcal{N} P_{\mathcal{N}\mathcal{N}}\Big) \mathbf{z}_\mathcal{N} &= Q_\mathcal{N} P_{\mathcal{N}\mathcal{T}} \mathbf{z}_\mathcal{T}, \end{aligned}$$

which will be solved directly as

$$\mathbf{z}_\mathcal{N} = \Big(I - Q_\mathcal{N} P_{\mathcal{N}\mathcal{N}}\Big)^{-1} Q_\mathcal{N} P_{\mathcal{N}\mathcal{T}} \mathbf{z}_\mathcal{T}, \tag{3.1.20}$$

or via Z-learning method (Todorov, 2007),

$$\mathbf{z}_\mathcal{N} \longleftarrow Q_\mathcal{N} P_{\mathcal{N}\mathcal{N}} \mathbf{z}_\mathcal{N} + Q_\mathcal{N} P_{\mathcal{N}\mathcal{T}} \mathbf{z}_\mathcal{T}. \tag{3.1.21}$$

LMDP, as in Equation (3.1.13), replaces the discrete state space by a continues concurrent actions expressed as transition probabilities over the next states, which can be influenced by many subtasks running at the same time. Also, LMDP makes it possible to decompose the optimal solution into simple way. Consider the two LMDPs $L_1 = \langle S, P, \mathbf{q}_\mathcal{N}, \mathbf{q}^1_\mathcal{T} \rangle$ and $L_2 = \langle S, P, \mathbf{q}_\mathcal{N}, \mathbf{q}^2_\mathcal{T} \rangle$, which they share the same state space, a probability distribution and the non-terminal exponential rewards and differs only in their terminal exponential rewards. They can be solved separately and producing the desirability functions $\mathbf{z}^1$ and $\mathbf{z}^2$. It follows from Equation (3.1.19) is that the desirability function of the LMDP $L_{1+2} = \langle S, P, \mathbf{q}_\mathcal{N}, c_1\mathbf{q}^1_\mathcal{T} + c_2\mathbf{q}^2_\mathcal{T} \rangle$, with the linear combination of the terminal exponential rewards, is given by the linear combination $\mathbf{z}^{1+2} = c_1\mathbf{z}^1 + c_2\mathbf{z}^2$. This property represents the base for our hierarchical scheme.

Algorithm 1 LMDP

procedure SOLVE($\mathbf{q}_{\mathcal{N}}, \mathbf{q}_{\mathcal{T}}, P_{\mathcal{N}\mathcal{N}}, P_{\mathcal{N}\mathcal{T}},$ tol) ▷ Offline Z-learning
  $\mathbf{z}_{\mathcal{T}} \leftarrow \mathbf{q}_{\mathcal{T}}$
  while True **do** ▷ Z-iteration loop
    $\mathbf{z}_{\mathcal{N}} \leftarrow Q_{\mathcal{N}} P_{\mathcal{N}\mathcal{N}} \mathbf{z}_{\mathcal{N}} + Q_{\mathcal{N}} P_{\mathcal{N}\mathcal{T}} \mathbf{z}_{\mathcal{T}}$ ▷ Update $\mathbf{z}_{\mathcal{N}}$
    if norm$(\mathbf{z}_{\mathcal{N}} - \mathbf{z}_{\mathcal{N}}^1) \leq$ tol **then** ▷ Checking the convergence
      break ▷ Convergence is obtained
    end if
    $\mathbf{z}_{\mathcal{N}}^1 \leftarrow \mathbf{z}_{\mathcal{N}}$ ▷ Continue iterating
  end while
  $\mathbf{z} \leftarrow \begin{bmatrix} \mathbf{z}_{\mathcal{N}} \\ \mathbf{z}_{\mathcal{T}} \end{bmatrix}$ ▷ Construct the full $\mathbf{z}$ vector
  return $\mathbf{z}$ ▷ The optimal desirability function
end procedure
procedure OPTIMAL_POLICY$(\mathbf{z}, P)$ ▷ Offline optimal policy
  for $i \leftarrow 1$ to n **do**
    $\hat{\mathcal{N}}_Z(i) \leftarrow \sum_{i=1}^{n} P_{i,j} z_j$ ▷ The normalized factor
    $\boldsymbol{\pi}_{\cdot,i} \leftarrow (P_{\cdot,i} \times \mathbf{z}) / \hat{\mathcal{N}}_Z(i)$ ▷ The optimal concurrent policy
  end for
  return π ▷ The optimal policy function
end procedure

We can obtain similar results for all other problem formulations. For the infinite horizon discounted problems, the Equation (3.1.16) becomes

$$\mathbf{z} = QP\mathbf{z}^{\gamma}, \tag{3.1.22}$$

where $0 < \gamma < 1$ is discount factor. While for the finite horizon problems the Equation (3.1.16) becomes

$$\mathbf{z}(t) = QP(t)\mathbf{z}(t+1). \tag{3.1.23}$$

What we discussed so far is the LMDP when the reward depends on the current state, but in some applications, the reward depends on the current state and the upcoming state as well. In the next section, we will introduce the idea of the LMDPs with transition-dependent rewards.

3.2 LMDPs with Transition Dependent Reward

In this section, we will expand the problem of LMDP to transition dependent reward while the expected reward function depends on the current state as well as the next state, in other words $R : S \times S \longrightarrow \mathbb{R}$ is defined on the Cartesian products of states. It follows that the cost $\mathcal{R}(s_i, a)$ for applying the control a in the state s_i is

$$\begin{aligned} R(s_i, \boldsymbol{a}) &= E_{s_j}\left[R(s_i, s_j)\right] - KL(\boldsymbol{\pi_i(a)} || \boldsymbol{p_i}) \\ &= \sum_{j: p_{ij} \neq 0} \pi_{ij}(\boldsymbol{a}) \left[R(s_i, s_j) - \log \frac{\pi_{ij}(\boldsymbol{a})}{p_{ij}} \right] \\ &= \sum_{j} \pi_{ij}(\boldsymbol{a}) \left[R(s_i, s_j) - a_j \right]. \end{aligned} \tag{3.2.1}$$

Substituting (3.2.1) into the Bellman optimally equation (2.3.5)

$$\begin{aligned} V(s_i) &= \max_{a_j \in A(s_i)} \sum_j \pi_{ij}(\mathbf{a}) \left[R(s_i, s_j) + V(s_j) - a_j \right] \\ &= \max_{a_j \in A(s_i)} \sum_j p_{ij} \exp(a_j) \left[R(s_i, s_j) + V(s_j) - a_j \right]. \end{aligned}$$

Solving the Lagrangian

$$\mathcal{L}(i, \boldsymbol{a}) = \sum_j p_{ij} \exp(a_j) \left(R(s_i, s_j) + V(s_j) - a_j \right) - \lambda_i \sum_j \left(p_{ij} \exp(a_j) - 1 \right),$$

results in the optimal control law

$$a_j^*(i) = R(s_i, s_j) + V(s_j) - \log \left(\sum_k p_{ik} \exp(R(s_i, s_k)) \exp(V(s_k)) \right), \tag{3.2.2}$$

and thereafter the optimally-controlled transition probabilities

$$\pi_{ij}(\boldsymbol{a}^*(\boldsymbol{i})) = \frac{p_{ij} \exp(R(s_i, s_j)) \exp(V(s_j))}{\sum_k p_{ik} \exp(R(s_i, s_k)) \exp(V(s_k))}. \tag{3.2.3}$$

Then the optimal value function in the transition-dependent reward problem is given by

$$\begin{aligned} V(s_i) &= \log \sum_j p_{ij} \exp(R(s_i, s_j)) \exp(V(s_j)) \\ &= \log \sum_j \Lambda(s_i, s_j) \exp(V(s_j)), \end{aligned} \tag{3.2.4}$$

where $\Lambda(s_i, s_j) = p_{ij} \exp(R(s_i, s_j))$. Introducing the desirability function $Z(i) = \exp(V(s_i))$ and exponentiating the both sides, the Equation (3.2.4) will be written in the form

$$Z(i) = \log \sum_j \Lambda(s_i, s_j) Z(j),$$

and in the matrix form as

$$\mathbf{z} = \Lambda \mathbf{z}. \tag{3.2.5}$$

Z-learning

A Z-learning is an online stochastic approximation of the optimal value function. Given the sample (i_k, j_k, r_k), where i_k is the current state, j_k is the next state, r_k is the reward incurred at state i_k, and k is sample number. Equation (3.1.16) can be written in the form

$$z(i_k) = \exp(r_k) E_P \big[z(j_k) \big]. \tag{3.2.6}$$

Z-learning maintains a stochastic estimation $\hat{z}$ of the optimal policy z by the following iteration

$$\hat{z}(i_k) \longleftarrow (1 - \alpha) \hat{z}(i_k) + \alpha \exp(r_k) \hat{z}(j_k), \tag{3.2.7}$$

where $0 < \alpha < 1$, is a learning rate. Sometimes instead of choosing α to be fixed, we can choose it as sequences that increases appropriately as k increases. It can be seen from Equation (3.1.15) that this approximation to the value function $V(i)$ is simply $\log(\hat{z}(i))$.

Convergence analysis

The vector $\mathbf{z}$ in Equation (3.2.5) is an eigenvector of Λ with eigenvalue 1. Moreover $z(s) > 0\,, \forall s \in S$ and $z(s) = 1\,, \forall s \in \mathcal{T}$, where $\mathcal{T}$ is the set of terminal states. The question that one might ask is whether the vector $\mathbf{z}$ exists, and if it is unique? The answer to both of these questions is yes because the Bellman equation has a unique solution and $\mathbf{v}$ is the Bellman's solution since $\mathbf{z} = e^{\mathbf{v}}$.

In Equation (3.2.5), the matrix Λ has the elements $\Lambda(s_i, s_j) = p_{ij} \exp(R(s_i, s_j))$, which may results in scaling up or down some rows of P, and it might not be true that the spectral radius of Λ is equal 1. In order to obtain an eigenvector $\mathbf{z}$ with eigenvalue 1 we need to rescale the matrix Λ by dividing it by its spectral radius, which gives a normalized matrix called Λ_1 given as

$$\Lambda_1 = \frac{1}{\rho(\Lambda)}\Lambda,$$

where $\rho(\Lambda)$ is the spectral radius of Λ. The eigenvalues of Λ_1 can be ordered as

$$1 = |\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

Then we transform Equation (3.2.5) into the form

$$\mathbf{z} = \Lambda_1 \mathbf{z}, \tag{3.2.8}$$

which can be solved iteratively as

$$\mathbf{z_{k+1}} = \Lambda_1 \mathbf{z_k}, \qquad \mathbf{z_0} = \mathbf{1}. \tag{3.2.9}$$

Now, we are in the step of analysing the rate of convergence. Let $m = |\mathcal{T}|$ and $n = |\mathcal{N}|$. The states can be permuted so that we can write Λ_1 in the canonical form

$$\Lambda_1 = \begin{bmatrix} A_1 & A_2 \\ \mathbf{O} & \mathbf{I} \end{bmatrix}, \tag{3.2.10}$$

where $A_1 \in \mathbb{R}^{n \times n}$ and $A_2 \in \mathbb{R}^{n \times m}$.

By iterating Equation (3.2.10) k-times, it follows

$$\begin{aligned} \Lambda_1^k &= \begin{bmatrix} A_1^k & (I + A_1 + A_1^2 + \cdots + A_1^{k-1})A_2 \\ \mathbf{O} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} A_1^k & (1 - A_1)^{-1}(1 - A_1^k)A_2 \\ \mathbf{O} & \mathbf{I} \end{bmatrix}. \end{aligned}$$

The matrix Λ_1 has m terminal states with an eigenvalue 1, and all the other eigenvalues are less than 1, implies that $\rho(A_1) < 1$. If λ is eigenvalue of A_1 with eigenvector x then

$$\begin{aligned} A_1^k \mathbf{x} &= \lambda^k \mathbf{x} \\ \lim_{k \longrightarrow \infty} A_1^k \mathbf{x} &= \lim_{k \longrightarrow \infty} \lambda^k \mathbf{x} \longrightarrow 0 \\ \lim_{k \longrightarrow \infty} A_1^k &\longrightarrow O. \end{aligned}$$

Hence the iteration in Equation (3.2.9) converges to the terminal states exponentially by the rate $\rho(A_1)$, and faster convergence is obtained if there are large transition probabilities from non-terminal states to the terminal states or small state costs $R(s_i, s_j)$. Noting that the convergence is independent of problem size because $\rho(A_1)$ is not affected by the size of A_1.

4. Hierarchical LMDPs

In this chapter, we present hierarchical LMDPs (HLMDPs). The chapter will be divided into three sections. In Section 4.1 we describe the Multi-task LMDPs followed by parallel hierarchical LMDPs in Section 4.2. Finally, we exhibit the computational complexity of HLMDPs in Section 4.3.

4.1 Multi-task LMDPs

In this section, we will use the idea of compositionality of LMDP to build up a multiple-tasks learning framework and we present the pseudo-code of the method in Algorithm 2.

Saxe et al. (2017) define a set of LMDPs, corresponding to a different tasks, as $L_k = \left\langle S, P, \mathbf{q}_{\mathcal{N}}, \mathbf{q}_{\mathcal{T}}^k \right\rangle, k = 1, \ldots, N_k$. Each of these subtasks operates in the same state space, and under the same probability distribution, and share the same non-terminal rewards and differs in their terminal rewards structure $\mathbf{q}_{\mathcal{T}}^k = \exp(\mathbf{r}_{\mathcal{T}}^k)$. We denote the multiple-tasks LMDP (MLMDP) as $\mathcal{M} = \left\langle S, P, \mathbf{q}_{\mathcal{N}}, \mathcal{Q}_{\mathcal{T}} \right\rangle$, where $\mathcal{Q}_{\mathcal{T}} = [\mathbf{q}_{\mathcal{T}}^1, \mathbf{q}_{\mathcal{T}}^2, \ldots, \mathbf{q}_{\mathcal{T}}^{N_k}] \in \mathbb{R}^{N_t \times N_k}$ encodes a library of exponential rewards of the terminal states of the MLMDP. The set of states is decomposed into non-terminal and terminal states $S = S_{\mathcal{N}} \cup S_{\mathcal{T}}$ and the transitions probability matrix $P \in \mathbb{R}^{|S| \times |S|}$ can be decomposed into non-terminal and terminal transitions as follows:

$$P = \left[\begin{array}{c|c} P_{\mathcal{N}\mathcal{N}} & P_{\mathcal{N}\mathcal{T}} \\ \hline O & I \end{array}\right],$$

where the probability of transitioning from the terminal states to the non-terminal states is zeros, i.e. $P_{\mathcal{T}\mathcal{N}} = O \in \mathbb{R}^{N_t \times N}$, and the probability of terminal-to-terminal is $P_{\mathcal{T}\mathcal{T}} = I \in \mathbb{R}^{N_t \times N_t}$ this due to absorbing nature of the terminal states.

Solving each single LMDP's task gives the desirability function $\mathbf{z}_{\mathcal{N}}^k$; $k = 1, \ldots, N_k$. Hence, we can create the desirability matrix $Z_{\mathcal{N}} = [\mathbf{z}_{\mathcal{N}}^1, \mathbf{z}_{\mathcal{N}}^2, \ldots, \mathbf{z}_{\mathcal{N}}^k] \in \mathbb{R}^{N \times N_k}$. Notice that $Z_{\mathcal{T}} = \mathcal{Q}_{\mathcal{T}} \in \mathbb{R}^{N_t \times N_t}$ which results from creating a basis set of tasks where each task has the reward $-\infty$ at all the terminal states except for one, which has the reward 0. Thus the task basis matrix $\mathcal{Q}_{\mathcal{T}}$ is an identity matrix. Notice that this implies that tasks differ only in their reward structure (which state gets reward and how much). Here $N, N_t, N_k \in \mathbb{Z}^+$ denotes the number of non-terminal, terminal and subtask states respectively.

Then, given a new task in the form of a specific reward structure, $\mathbf{q}_{\mathcal{T}} = \exp(\mathbf{r}_{\mathcal{T}})$, for all the terminal states. If the vector $\mathbf{q}_{\mathcal{T}}$ lies in the span of the $\mathcal{Q}_{\mathcal{T}}$'s columns, it can be written as a linear combination of the previous terminal rewards

$$\begin{aligned} \mathbf{q}_{\mathcal{T}} &= c_1 \mathbf{q}_{\mathcal{T}}^1 + c_2 \mathbf{q}_{\mathcal{T}}^2 + \cdots + c_{N_k} \mathbf{q}_{\mathcal{T}}^{N_k} \\ &= \mathcal{Q}_{\mathcal{T}} \mathbf{w}, \end{aligned} \tag{4.1.1}$$

where $\mathbf{w} \in \mathbb{R}^{N_k}$ is a vector blending weights. The approximated solution $\hat{\mathbf{w}}$ to $\mathbf{w}$ will be founded using the least square method if $\mathbf{w}$ is not in the span of $\mathcal{Q}_{\mathcal{T}}$ as

$$\hat{\mathbf{w}} = \min_{\mathbf{w}} ||\mathcal{Q}_{\mathcal{T}} \mathbf{w} - \mathbf{q}_{\mathcal{T}}||, \quad \text{subject to } \mathcal{Q}_{\mathcal{T}} \mathbf{w} \geq 0, \tag{4.1.2}$$

which minimizes the distance between $\mathcal{Q}_{\mathcal{T}}\mathbf{w}$ and the rewards vector $\mathbf{q}_{\mathcal{T}}$. Actually, the constraint $\mathcal{Q}_{\mathcal{T}}\mathbf{w} \geq 0$ is because of the relationship $\mathbf{q}_{\mathcal{T}} = \exp(\mathbf{r}_{\mathcal{T}}) \geq 0$. In particular, the Equation (4.1.2) is solved using Moore-Penrose inverse as

$$\hat{\mathbf{w}} = \mathcal{Q}_{\mathcal{T}}^{+}\mathbf{q}_{\mathcal{T}},$$

where the $+$ sign indicate the Moore-Penrose generalized inverse.

Accordingly, because of the fascinating properties of LMDP, this means that the desirability function for the optimal policy is

$$\mathbf{z}_{\mathcal{N}} = Z_{\mathcal{N}}\mathbf{w}.$$

After finding $\mathbf{z}_{\mathcal{N}}$ we constitute the vector $\mathbf{z}$, and from that we directly derive the actual optimal policy π through Equation (3.1.13). Therefore, the agent perform optimality on new tasks which are never seen before, given that they lie within the span of the previous learned tasks.

There are important features that LMDP deserves. Firstly, the optimal policy $\mathbf{z}_{\mathcal{N}}$ is obtained from a linear combination of the previous optimal policies they are running in parallel. This attitude is uncommon in the MDP problems in which the actions run sequentially and are chosen mutually exclusively. The LMDP makes this behaviour possible by transforming the action from the discrete form to a continues concurrent form in which the actions are built from the probability over the next states as shown in Equation (3.1.13).

Secondly, the compositionality of the LMDP makes the new policies to be rescaled by element-wise multiplication. This allows to adjust the relative importance of the states when the rewards are different.

Algorithm 2 MLMDP

procedure PRESOLVE($\mathcal{Q}_{\mathcal{T}}$) ▷ Pre-solve an initialized MLMDP for all basis tasks
  Inherent from LMDP $\mathbf{q}_{\mathcal{N}}, P_{\mathcal{N}\mathcal{N}}, P_{\mathcal{N}\mathcal{T}}$, tol
  for $i \leftarrow 1$ to N_k **do** ▷ Iterate over the subtasks
    $\mathbf{q}_{\mathcal{T}} \leftarrow \mathcal{Q}_{\mathcal{T},i}$
    $\mathbf{z} \leftarrow$ LMDP$(\mathbf{q}_{\mathcal{N}}, \mathbf{q}_{\mathcal{T}}, P_{\mathcal{N}\mathcal{N}}, P_{\mathcal{N}\mathcal{T}}$, tol$)$ ▷ Obtain the optimal desirability function
    $Z_{\mathcal{N}} \leftarrow$ append$(\mathbf{z}_{\mathcal{N}})$
  end for
  return $Z_{\mathcal{N}}$ ▷ The optimal desirability matrix
end procedure
procedure SOLVEMLMDP($\mathbf{r}_{\mathcal{T}}$)
  $\mathbf{q}_{\mathcal{T}} \leftarrow \exp(\mathbf{r}_{\mathcal{T}})$
  $\mathbf{w} \leftarrow \min_{\mathbf{w}} ||\mathcal{Q}_{\mathcal{T}}\mathbf{w} - \mathbf{q}_{\mathcal{T}}||$ ▷ The blend weigh vector
  $\mathbf{z}_{\mathcal{T}} \leftarrow Z_{\mathcal{N}}\mathbf{w}$ ▷ Construct the optimal Z-value
  $\pi \leftarrow$ LMDP Optimal policy$(\mathbf{z})$
  return π ▷ The optimal policy function
end procedure

4.2 Parallel Hierarchical LMDPs

In this section, we are going to build up a hierarchical MLMDP by describing an augmented LMDP, and therefore we will be able to construct a stack of MLMDPs and finally the pseudo-code of online Hierarchical LMDP (HLMDP) will be given in Algorithm 3.

Following our notation, the set of states S is composed of the disjoint subsets denoting the N non-terminal states $S_{\mathcal{N}}$, and the N_t terminal states $S_{\mathcal{T}}$. We begin with the base MLMDP $\mathcal{M}^1 = \left\langle S^1, P^1, \mathbf{q}^1_{\mathcal{N}}, \mathcal{Q}^1_{\mathcal{T}} \right\rangle$, which represent the lower level of the hierarchy. At level l, the MLMDP defined as $\mathcal{M}^l = \left\langle S^l, P^l, \mathbf{q}^l_{\mathcal{N}}, \mathcal{Q}^l_{\mathcal{T}} \right\rangle$. At this level, the set of states is extended to include a new N_s states called subtask states and denoted by $S^l_{\mathcal{S}}$, resulting in the augmented set of states $\hat{S}^l = S^l \cup S^l_{\mathcal{S}}$. Those subtasks are not part of the original problem of MDP, but used in the hierarchy to help the policy to know which state in the lower level the agent must visit in order to reach the goal.

Accordingly, the agent will operates on the augmented MLMDP $\hat{\mathcal{M}}^l = \left\langle \hat{S}^l, \hat{P}^l, \mathbf{q}^l_{\mathcal{N}}, \hat{\mathcal{Q}}^l_{\mathcal{T}} \right\rangle$, where $\hat{S}^l = S^l_{\mathcal{N}} \cup S^l_{\mathcal{T}} \cup S^l_{\mathcal{S}}$, $\hat{P}^l = [P^l_{\mathcal{NN}}, P^l_{\mathcal{NT}}, P^l_{\mathcal{NS}}]$, and the reward structure $\hat{\mathcal{Q}}^l_{\mathcal{T}} = [\mathcal{Q}^l_{\mathcal{T}}\ , \mathcal{Q}^l_{\mathcal{S}}]$ over the $\hat{N} = N + N_t + N_s$ states. Transitions from the non-terminal states to the new subtasks states are given by the $N \times N_s$ transition matrix $P^l_{\mathcal{NS}}$. Usually the matrix $P^l_{\mathcal{NS}}$ is hand crafted.

In the hierarchy, the base layer is an augmented LMDP, while the higher layers are MLMDPs defined over the N_s subtask states of the layer below instead of the whole state space. Firstly, the non-terminal states of the higher layer correspond to the subtask states of the lower layer in which each non-terminal state has a corresponding terminal state. The transition dynamics between the subtask states, $P^{l+1}_{\mathcal{SS}}$, of the higher layer are defined by the transition dynamics $\hat{P}^l_{\mathcal{NN}}$ of the lower layer, such that the subtask states that are closer to the lower layer have a high probability in the higher layer and are given by

$$P^{l+1}_{\mathcal{SS}} = \hat{P}^{l^T}_{\mathcal{NS}} \left(I - \hat{P}^l_{\mathcal{NN}} \right)^{-1} \hat{P}^l_{\mathcal{NS}}. \tag{4.2.1}$$

Likewise, the transition dynamics from the subtask states to the terminal states are given by

$$P^{l+1}_{\mathcal{ST}} = \hat{P}^{l^T}_{\mathcal{NT}} \left(I - \hat{P}^l_{\mathcal{NN}} \right)^{-1} \hat{P}^l_{\mathcal{NS}}. \tag{4.2.2}$$

Like regular MLMDP's, the higher layer basis set of tasks $\mathcal{Q}_{\mathcal{T}}$ is the identity matrix, that is each task has small negative rewards, namely $-\infty$, at all terminal states except for one, which has reward 0.

Then, given a new task in the form of a specific reward structure $\mathbf{r}^l_{\mathcal{T}}$ which is the rewards for all terminal states in the lower layer, we are finding a step-by-step solution via the hierarchy. First, we need to solve the lowest layer, noticing that right now we do need the entire solution, so we are do not really need the hierarchy. However, it becomes valuable if we do Z-iteration with only a few steps instead. By finding an optimal blend of basis tasks defined by the weights $\mathbf{w}^l$ and computing the blend desirability function $\mathbf{z}^l_{\mathcal{N}} = Z_{\mathcal{N}}\mathbf{w}^l$, we are able to derive the optimal policy π^l directly.

Initially, subtask states are only given small rewards to encourage exploration. Then we start following the lower-level policy defined by π^l. Once it enters a subtask state, it goes to the higher level of the hierarchy. The first thing it does then is to come up with a corresponding higher-layer task, that is a set of rewards $\mathbf{r}^{l+1}_{\mathcal{T}}$ for the terminal states of the higher level.

So on the higher layer, we will compute the exponential of the terminal states rewards $\mathbf{q}^{l+1}_{\mathcal{T}} = \exp(\mathbf{r}^{l+1}_{\mathcal{T}})$. Forthwith, we are able to find the blending weight vector $\mathbf{w}^{l+1} = {\mathcal{Q}_{\mathcal{T}}}^{+}\mathbf{q}^{l+1}_{\mathcal{T}}$ to compute the desirability function $\mathbf{z}^{l+1}_{\mathcal{T}} = Z_{\mathcal{N}}\mathbf{w}^{l+1}$ then finding the optimal policy π^{l+1}. These actions tell us which non-terminal states on the higher layer are preferred when starting at a specific state; which subtask states on the lower layer are preferred from wherever we currently are (i.e. when we entered the higher layer). We use this to re-compute the rewards $\mathbf{r}^l_{\mathcal{S}}$ for the subtask states on the lower layer using the formula

$$\mathbf{r}^l_{\mathcal{S}} \propto \pi^{l+1}_{\mathcal{N}}(\cdot) - p^{l+1}_{\mathcal{N}}(\cdot).$$

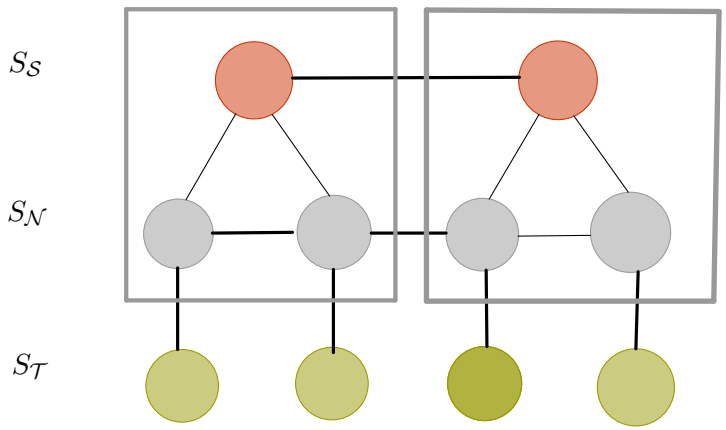


Figure 4.1: A single layer of hierarchy

In turn we re-compute $\mathbf{q}^l_{\mathcal{T}}$, $\mathbf{w}^l$, $\mathbf{z}^l_{\mathcal{N}}$, and π^l on the lower layer. Then we go back to the lower layer and continue sampling.

Hence, since the highest layer in the hierarchy is an LMDP $\mathcal{M}^{l+1}$ and other layers are augmented MLMDP $\hat{\mathcal{M}}^l$. This process can be repeated to create a deep hierarchy of MLMDPs $\{\hat{\mathcal{M}}^1, \hat{\mathcal{M}}^2, \ldots, \hat{\mathcal{M}}^l, \mathcal{M}^{l+1}\}$, where all except the highest is an LMDP. Figure 4.1 shows a graphical illustration of hierarchical framework with a single layer of hierarchy.

4.3 Computational Complexity of HLMDP

In this section, we intend to compare the computational complexity of Q-learning, LMDP and HLMDP.

In Q-learning, the computational complexity is $O(NM)$, where N is the number of iterations and M is the number of steps per iteration. The numbers N and M increase proportionally with the number of states. Furthermore, the convergence is obtained after a large number of iterations.

For LMDP, we have proved in Section 3.2 that the convergence rate is the highest eigenvalue of the non-terminal transition probability, which is less than 1 and is independent of the problem size.

Suppose we have a RL problem with $|S| = n$ states. Using LMDP, the computational complexity to find the optimal desirability function $\mathbf{z}$, is $O(n^3)$ via Equation (3.1.20).

In a hierarchical architecture, suppose that the hierarchy is constructed by setting subtask every $s = \log n$ states. And we keep doing this recursively to construct l layers of hierarchy. Hence, we have n states on the base layer, $\frac{n}{s}$ subtask on the first layer, $\frac{n}{s^2}$ on the second layer, and $\frac{n}{s^l}$ subtask on the l-th layer. The recursion terminates when $\frac{n}{s^l} \approx 1$, then the depth of the hierarchy will be $l = \log_s n$.

At each layer, every policy is required to learn to transition between the neighbour subtasks which are not more than s states apart. Therefore, Z-iteration terminates in $O(s^3)$ iteration. At level $i = 1, 2, \ldots, l$ of the hierarchy, there $\frac{n}{s^i}$ subtasks, and $\frac{n}{s^{i-1}}$ terminal reward tasks to learn. Therefore, the computational

complexity of the whole hierarchy is given by

$$\begin{aligned} C(n) &= \sum_{i=1}^{l} s^3\Big(\frac{n}{s^i} + \frac{n}{s^{i-1}}\Big) \\ &= n(s+1)\sum_{i=1}^{l}\frac{1}{s^{i-3}} \\ &= n(s+1)\Big(\frac{1-\frac{1}{s^{l-2}}}{1-\frac{1}{s}}\Big) \\ &\approx n(s+1) = n(\log n + 1) \in O(n\log n). \end{aligned}$$

Hence, the hierarchy is advantageous when implementing multiple tasks, due to the reuse of prior policies.

Algorithm 3 HMLMDP

procedure ONLINE HLMDP(M) ▷ Stack of hierarchies
  while M is not empty **do** ▷ Pre-solve all MLMDP's of the hierarchy
    $Z_{\mathcal{N}} \leftarrow$Presolve$(Q_{\mathcal{T}})$
  end while
  current state $\leftarrow$ find the agent location
  $\pi_{\mathcal{T}} \leftarrow$ solveMLMDP$(\mathbf{r}_{\mathcal{T}})$ ▷ Obtain the optimal policy of the terminal state
  while Not at goal **do** ▷ Sampling
    next state $\sim \pi(\cdot,$ currnet state$)$ ▷ Draw the next state
    if new state $\in S_{\mathcal{N}}$ **then**
      agent moves to the new state
      current state $\leftarrow$ new state
    else if new state $\in S_{\mathcal{S}}$ **then**
      $\mathbf{z}_{\mathcal{N}} \leftarrow Z_{\mathcal{N}}\mathbf{w}$ ▷ Set $\mathbf{z}_{\mathcal{N}}$ as blend of basis functions
      reward of next level $\leftarrow \log(\mathbf{z}_{\mathcal{N}_{\mathcal{S}}})$ ▷ Calculate the reward of the next level
      SolveMLMDP(reward of next level) ▷ Solve next level MLMDP
      $\mathbf{r}_{\mathcal{S}} \propto a_{\mathcal{N}}(\cdot,$ state next level$) - P_{\mathcal{N}}(\cdot,$ state next level$)$ ▷ Recalculate reward structure
      $\mathbf{w} \leftarrow$ solveMLMDP$(\mathbf{r}_{\mathcal{T}})$ ▷ Recompute the optimal policy based on the new reward structure
    else
      the new state $\in S_{\mathcal{T}}$
      break ▷ Terminate
    end if
  end while
end procedure

5. Experiments

In this section, we present numerical results. First we give a full description of the problem and we numerically implement the RL methods, namely Q-learning, LMDP and HLMDP.

To evaluate the performance of our RL scheme, we applied it on a 10-by-19 grid-world using Matlab software. The grid-world consists of an agent, a goal, holes, obstacles, and free states where the agent can move. The agent can move in the four directions, {Up, Down, Right, Left}, or remain still.

In this case, our environment consists of:

- The agent, and is encoded by the symbol X;
- The goal, given by the symbol \$;
- Sub-goals, given by the symbols S;
- Holes given by the red cells, and
- Obstacles denoted by the black cells.

The agent's objective is to reach the goal using an optimal path. Therefore, the agent will receive a reward of -1 every time when it moves to new free state or remain still, and a reward of $-\infty$ if it falls in a hole, while it receives a reward of $+10$ in case it reaches the goal. For hierarchical LMDP we added supplementary subtasks goals with 0 reward, denoted by S, to guide the agent to reach the goal.The subtasks symbols have no effects on LMDP and Q-learning, and becomes worth only in hierarchical LMDP.

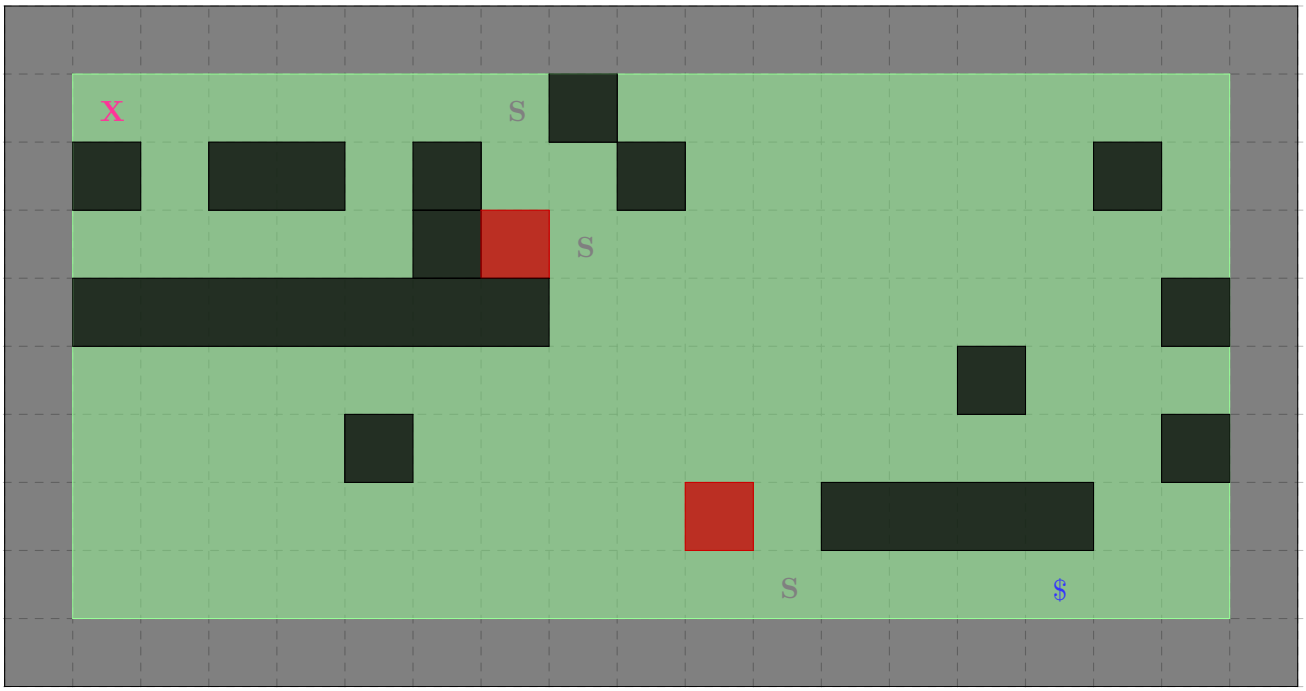


Figure 5.1: A 10-by-19 grid world representing the environment. The agent is given by the red symbol X and the goal is given by \$. The green cells represents free states, the red cells represents the holes, and the obstacles are given by the black one.

We tested LMDP, Q-learning and hierarchical LMDP methods on any randomly created grid-world with specified probabilities for free states, holes, obstacles, an agent, and a goal and they works perfectly. But for sake of comparison we chose a particular grid world as can be seen in Figure 5.1.

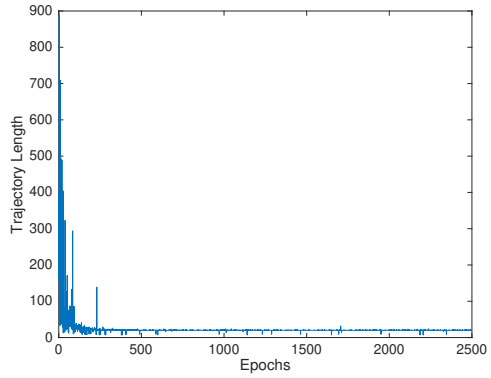


Figure 5.2: Learning rates of Q-learning

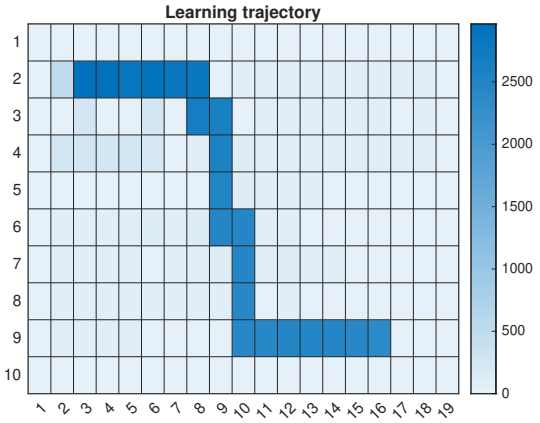


Figure 5.3: Q-Learning trajectory

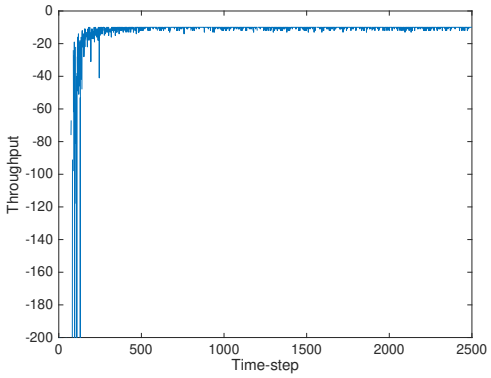


Figure 5.4: Q-learning agent' rewards over time-step.

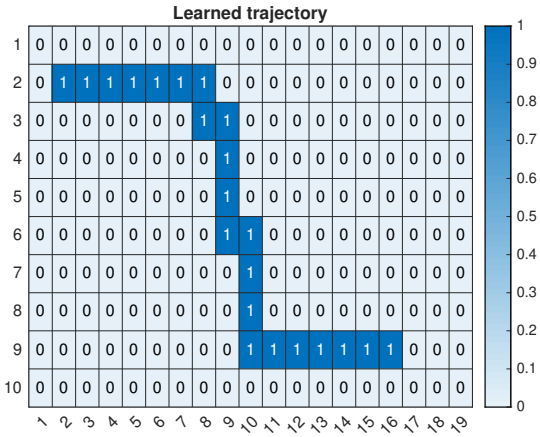


Figure 5.5: Optimal path using Q-learning

We used an online ϵ-greedy Q-learning. Firstly, we start by creating a $|S| \times |A|$ Q-value table initialized to zeros. Accordingly, the agent chooses its action based on the maximum Q-value for the given state. But since Q-value table was initialized to zeros, the agent will not be able to identify which state is good in the early stages. As alternative, in ϵ-greedy approach, the agent starts by exploration the environment (learning phase) firstly before it will be able to exploit it by defining the parameter ϵ.

Initially, the parameter ϵ will be initialized to 1, that is to say with 100% probability the agent will start by exploring the environment. Every step we generate a random variable $r \in (0, 1)$, in which our decision between exploration and exploitation depend on its value. If the random variable r greater than ϵ, the agent chooses its action via exploitation, otherwise will explore. As time goes by, the value of ϵ will decay with some factor, which implies that the agent will become more greedy by exploiting the environment.

Training of an agent requires a finite number of iterations to converge which we do not know, but assuming it exists. To address that issue, we set a large number of rounds, let say 2500, over which we trained our agent. Experiments showed that the convergence is reached at around iteration 300 in most cases as the trajectory shown in Figures 5.2 & 5.3 & 5.4. After the convergence is reached, the agent

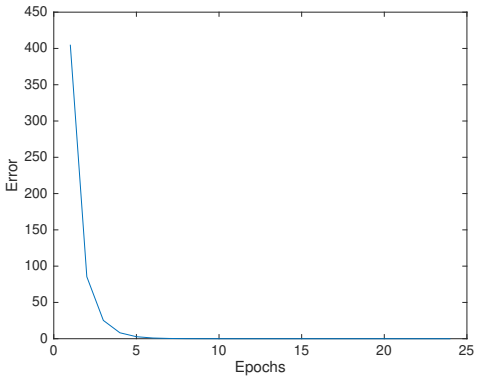


Figure 5.6: LMDP convergence

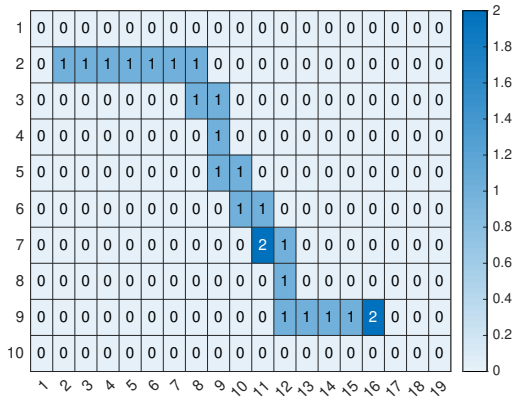


Figure 5.7: LMDP optimal path

becomes able to know the optimal path deterministically by selecting the best action at a given state as depicted in Figure 5.5. After running Q-learning 100 times, we found that, on average, the running time is 1.3540 seconds.

Using the same example, we found that the LMDP method, as described in Section 3.1, converges within 24 iterations. In this case, the optimal policy is given by the stochastic matrix π in Equation 3.1.13. Due to that, the agent, as can be seen in Figure 5.7, is moving not deterministically and may spent more than one time-step staying in the same state.

The LMDP outperforms the Q-learning approach in the running time using the same number of runs. We found that, on average, the run time spent by LMDP was 0.0313 seconds.

In HLMDP, we have created two layers of hierarchy, in which the base layer is just augmented LMDP and the higher layer is MLMDP. The higher layer is constituted of the subtask states that are used to help the agent to reach the goal. At each subtask the agent is obliged to reach the nearest subtask state. Once the agent is reached the subtask it start to look for the next subtask until reaches the goal and terminates. One average, we have found the run time of HLMDP is approximately 0.0310 seconds.

The reason that the run time of LMDP and HLMDP are almost the same may be due to inefficient implementation or the way we have chosen the transition probabilities.

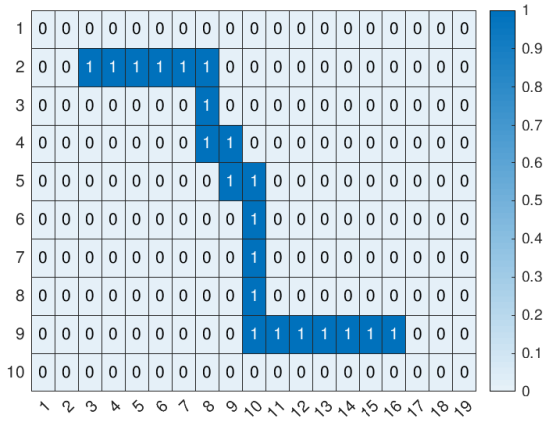


Figure 5.8: HLMDP optimal path

6. Conclusion and Future Work

This project was about understanding some useful mathematics of reinforcement learning methods. In this essay, we have presented the mathematics of reinforcement learning by transforming the class of Markov Decision Processes (MDPs) to Linearly-solvable MDPs (LMDPs). A critical analysis for convergence of LMDPs was also undertaken. We then leveraged the compositionality and concurrent execution of LMDP to create a hierarchical framework in which we can execute multiple tasks in a parallel fashion. The main take away of this study was the outperformance of HLMDPs over Q-learning method. In Q-learning at each task, the agent learns a new policy but without considering its previous policies.The use of previous policies can improve the quality of a new policy. In HLMDPs rather than learning a fresh policy at a new task, the new policy is composed of previous policies with negligible computational cost.

Experiments indicated that both LMDP and HLMDP speed up the learning process in a 10-by-19 grid-world environment as compared to Q-learning. HLMDP shows a better performance in the running time since it divides the whole problem into smaller sub-tasks running in parallel.

Although, HLMDP has a better performance in terms of computational complexity, the optimal policy needs extra space to store the transition probabilities. In LMDP and HLMDP, the concurrent actions which were represented by the optimal policy π, consumed $O(|S|^2)$ units of memory which is higher compared to the space used by Q-learning, which required a $O(|S| \times |A|)$, where $|S|$ and $|A|$ represent the number of states and actions respectively. In a future study, the storage drawback of HLMDP can be refined to avoid using extra space. In addition, one can address the hierarchical architecture methods in the dynamic environments when the number of states and actions are not fixed.

Acknowledgements

First and foremost, I would like to thank Allah for helping me to complete this project. I would like to express my gratitude to my benevolent and sincere supervisor Professor Montaz Ali for the tremendous effort he spent on the project. Professor Ali kindly devoted his time and insightful guidance to help me to complete my project. Faithfully-earned credit goes to my parents and my family for their support and prayers. Special thanks goes to the African Institute for Mathematical Sciences (AIMS) for providing the facilities needed to conduct my research and creating an academic atmosphere that always remained motivating. And finally I would like thank enough AIMS's director Professor Barry Green and the academic director Professor Jeff Sanders, the English and communication lecturer Noluvuyo, the tutors especially Jordan, Lebeko, Elbasher, Taboka and Kenneth, the IT manger Jan, the facility manger Igssan, AIMS's staff and my colleagues for their cooperation and helpful advice.

References

Atkinson, K. E. *An introduction to numerical analysis*. John Wiley & Sons, 2008.

Bellman, R. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 1954.

Dietterich, T. G. The MAXQ Method for Hierarchical Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, volume 98, pages 118–126. Citeseer, 1998.

Earle, A. C. *Spectral Reinforcement Learning*. Phd, University of the Witwatersrand, 2019.

Earle, A. C., Saxe, A. M., and Rosman, B. Hierarchical subtask discovery with non-negative matrix factorization. *arXiv preprint arXiv:1708.00463*, 2017.

Hoff, M. R. *A review and application of hidden Markov models and double chain Markov models*. PhD thesis, 2016.

Jonsson, A. and Gómez, V. Hierarchical linearly-solvable Markov decision problems. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

Randlov, J. Learning macro-actions in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1045–1051, 1999.

Saxe, A. M., Earle, A. C., and Rosman, B. Hierarchy through composition with multitask LMDPs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3017–3026. JMLR. org, 2017.

Serfozo, R. *Basics of applied stochastic processes*. Springer Science & Business Media, 2009.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Todorov, E. Linearly-solvable Markov decision problems. In *Advances in neural information processing systems*, pages 1369–1376, 2007.

Todorov, E. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, pages 1856–1864, 2009a.

Todorov, E. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009b.

Ullah, M. and Wolkenhauer, O. Stochastic approaches in systems biology. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 2(4):385–397, 2010.

Watkins, C. J. C. H. *Learning from delayed rewards*. King's College, Cambridge, 1989.