Imperial College London

Department of Mathematics

Reinforcement learning for problem resolution in a telecommunications setting

Yuet Ying Chau

CID: 01564104

Supervised by Dr Ciara Pike-Burke and Dr Lucy Gullon

1 July 2022

Submitted in partial fulfilment of the requirements for the MSc in Statistics of Imperial College London

The work contained in this thesis is my own work unless otherwise stated.

Signed: Yuet Ying Chau Date: 1 July 2022

Contents

Abstract

Reinforcement learning is a machine learning technique that determines an adaptive sequence of decisions which achieves the goal of a given problem. A usual goal of a reinforcement learning problem is to minimise the long-term cost when the execution of each action is associated with a cost. Examples of applications of reinforcement learning are autonomous driving and gaming, such as AlphaGo. The thesis first describes a stochastic multi-armed bandits model, which may be viewed as a simplified reinforcement learning model, then elaborates limitations of the model compared to reinforcement learning. Besides, we introduce how reinforcement learning can be used in a telecommunications scenario, where the aim is to seek the optimal sequence of partial resolutions to be taken given a detected fault. We aim to formulate the scenario mathematically as a Markov decision process and demonstrate how the objective can be achieved using reinforcement learning. The performance of a reinforcement learning model mainly depends on the strategy of choosing actions. An algorithm is said to be model-based if it attempts to build a model to represent the problem of interest and it is said to be model-free otherwise. A variety of action selection algorithms, for instance, value iteration, posterior sampling, and Q-value sampling are discussed. The thesis also provides insights into how dynamic programming speeds up the computation for determining the optimal sequence of actions. Cumulative expected regret is a loss function that measures the goodness of the selected action relative to the optimal choice. Experiments on four Markov decision process models, each with different settings of cost distributions and transition functions, are conducted to analyse the performance of the algorithms, where the performance is measured by the computation time and convergence of the cumulative absolute expected regret. The models are designed with cost distributions that either depend on the present action and states of transitions or just the action itself, which imitate different levels of complexity of the telecommunications scenario. It is found that Bayesian reinforcement learning approaches often have satisfactory performance, while model-free algorithms are more efficient than model-based methods.

Acknowledgements

I would like to express my gratitude to my supportive supervisor Dr Pike-Burke and co-supervisor Dr Gullon for their guidance in the compilation of this project.

1 Introduction

Reinforcement learning is a machine learning method that seeks an optimal sequence of decisions, which minimises long-term cost, in a particular problem. Unlike many machine learning techniques that train models with a significant amount of data, a reinforcement learning model studies the problem, which is often referred to as the virtual learning environment. Reinforcement learning may be used when there is insufficient data to build supervised or unsupervised learning models, or when the problem can only be solved by trial and error through interacting with the virtual learning environment. A typical reinforcement learning problem is structured such that the problem is encountered repeatedly, which provides opportunities to learn about the best sequence of actions. The agent in a reinforcement learning algorithm is a character who interacts with the environment, selects an action, and receives cost from the environment. Moreover, learning about the environment and finding the optimal sequence of decisions ought to occur in parallel.

A reinforcement learning algorithm mainly consists of active exploration of the environment and exploitation of the agent's knowledge. It is defined in (Sutton and Barto, 2018) that exploration is a search for the best action by trial and error, the learning agent exploits when it chooses an action based on its experience and knowledge of how good the selected action is. The authors in (Sutton and Barto, 2018) suggest that it is challenging to balance the trade-off between exploration and exploitation of the environment. The agent is required to discover actions that bring the lowest cost, which can be achieved by exploring unfamiliar actions until a stable estimate of the expected cost associated with each action is obtained. To reduce the total cost, the agent needs to repeatedly execute actions that are considered to bring the lowest costs, based on the agent's knowledge of each explored action. As a result, a reinforcement learning algorithm should be implemented such that the agent can sufficiently explore the environment, while being allowed to make full use of information gained from exploration.

For a stochastic multi-armed bandits problem, we aim to find the best action in an action space that gives the minimum long-term cost. Each arm represents an action that is linked to a fixed but unknown cost distribution. While interacting with the environment, the agent selects an action by pulling the corresponding arm and observes the cost sampled from the cost distribution from the environment. The learning goal for an agent is to determine the best arm to pull to optimise total cost in a certain number of interactions, as stated in (Slivkins, 2019). A difference between multi-armed bandits and reinforcement learning is that the environment in reinforcement learning includes a state space, where states may be viewed as descriptions of the environment. Therefore, a reinforcement learning problem may be seen as an extension of a multi-armed bandits problem. The execution of an action in a reinforcement learning model induces a cost,

as well as a transition in states. Once an action is executed, a change in the environment may be induced and it may lead to a transition in states. The transition between states can be deterministic or stochastic, we shall focus on stochastic transition in this thesis. A reinforcement learning model is preferable to a multi-armed bandits model when it is of interest to also learn about the transition dynamics of the environment, as the bandits model does not imitate the status of the environment. On the other hand, a multi-armed bandits model assumes every arm can be pulled at all times, so the model may not be suitable for problems where some actions cannot be executed in some states.

A Markov decision process captures the transition function, cost distributions, action and state spaces. A Markov decision process can be solved by finding the optimal policy, which is a mapping from each state to the action that minimises the expected total cost when it is executed at the mapped state. The optimal policy can be determined by solving the Bellman Equations according to (Alagoz et al., 2010). Reinforcement learning is an algorithm that finds the optimal policy of a Markov decision process. The reinforcement learning agent has full information about the state and action spaces, while it should learn the transition probabilities and cost distributions.

Reinforcement Learning in a Telecommunications Scenario

This thesis presents an application of reinforcement learning for problem resolution, where we aim to solve the problem raised by the telecommunications company BT. We attempt to simulate the choice of resolution to a fault. The BT system detects faults that happen across the country automatically and humans often need to decide on the action to resolve the fault. It is also assumed that all faults can eventually be resolved, so the objective is to find the sequence of partial resolutions that contributes to the lowest expected total cost. We attempt to implement a reinforcement learning algorithm such that the agent makes all choices of partial resolution while minimising the long-term cost. We shall refer the scenario to as the BT scenario or the telecommunications scenario in the rest of the thesis.

To frame the scenario above mathematically, each state in the state space represents a status of fault; an action in the action space is a partial resolution. The cost distribution associated with an action quantifies how much taking an action costs, in terms of labour cost or cost of time. As one may expect, it may cost more to send two engineers to fix a landline problem when one engineer is capable of doing so. Moreover, the transition function indicates the probability of moving from one status of fault to another. A transition in states happens when there is a change in the status of fault. For instance, the status of fault after sending out an engineer may change from faulty to resolved or require an engineer with specialist skills. Note that the sizes of state space and action space in the scenario above are finite, while there are usually less than five actions in the action space. Possible actions are sending engineers to fix the faulty equipment, instructing staff at the local office to tackle the issue and waiting until more information is gained. There is a natural stopping point in the scenario where the status of the fault becomes fixed.

The assumptions for the scenario are outlined below,

1. There exists a terminal state, which we may refer to as the goal state, such that an interaction between the agent and the environment stops once the goal state is visited.
2. The states are discrete.
3. Each action is linked to a cost distribution, which may vary in transitions of states and depend on the current or successive state.
4. The transition between states is stochastic rather than deterministic.
5. Some actions may contribute to a great instant cost but a lower expected total cost.
6. There exists at least one path of finite length from any initial state to the goal state.
7. The learning agent may not be able to travel from one state to another.

In the first assumption, the terminal state may be reached when the fault is resolved and it is assumed that all faults can ultimately be fixed. Moreover, the terminal cost is set to zero here. We may view each status of fault to be an individual case, so discrete states shall be used to simulate the scenario. Assumptions 4 and 6 ensure that the scenario can be translated into a Markov decision process with a stochastic transition function, and more specifically a stochastic shortest path problem, which guarantees it can be solved with reinforcement learning. Since the learning agent will start at a preset state and make a sequence of actions until it reaches the goal state, interactions with the learning environment are often divided into episodes. In an episode, the agent decides on an action to act at the current state in each round, so the length of an episode equals the number of rounds. Therefore, an episode of interactions records the states visited from the initial state to the goal state and the actions executed by the agent. When an episode ends, the new initial state is randomly sampled from the set of non-terminal states. According to (Jafarnia-Jahromi et al., 2021), the length of an episode is dependent on the agent's choices of actions and is allowed to be unbounded if the goal state is never visited. It follows from finite state space and action space that there are finitely many possible policies, where a policy is a mapping from each non-goal state to an action to act in the input state. A realistic interpretation of assumption 3 is given that a landline and router fault is detected, the cost of scheduling an engineer may be greater if the resulting status of fault remains unchanged or gets worse. To illustrate the fifth assumption, taking the action to wait at any status of fault may contribute to a greater immediate cost, although gaining additional information may help the agent to optimise its choice of action in the future and result in lower long-term costs. The last assumption addresses the fact that given any initial status of fault, there may not be an action that can resolve all faults at once.

Structure of Thesis

The structure of this thesis is as follows. Formal definitions of a stochastic multi-armed bandits model and details about Markov decision processes are established in the Background chapter. The performances of several reinforcement learning algorithms, including Q-learning from (Liu et al., 2020), posterior sampling based on (Agrawal and Jia, 2017), and value iteration from (Kaelbling et al., 1996) will be evaluated in the Results chapter, while the pseudo-codes and descriptions of the algorithms are highlighted in the Methods chapter. The comparisons between algorithms will be based on the average computation time and the rate of convergence of cumulative absolute expected regret, which measures the deviation between the optimal policy and the policy suggested by the agent. On the other hand, we shall also study the impact of the size of the Markov decision process, which mainly depends on the size of state space in the scenario, on the performance of each algorithm. Designs of the Markov decision processes for the experiments can also be found in the Designs chapter. It is believed that the cumulative expected regret of a Markov decision process that has a fixed cost distribution associated with each action will converge at a smaller value than a process where the cost distributions vary in state transitions for most reinforcement learning algorithms. This belief is motivated by the fact that fewer explorations are required to learn the cost distributions of the actions if they are fixed and independent of the transition, especially in a large state-action environment, so the agent can spend more time exploiting its knowledge to obtain the least total cost. We shall also examine the validity of the belief by running several experiments, where the results are analysed in the Results chapter.

2 Background to Reinforcement Learning

In this chapter, we shall introduce the definitions of a multi-armed bandits model and a Markov decision process. Note that we denote $\mathcal{A}$ to be the action space with finite size. It is fair to assume that the action space is of size less than five in the telecommunications scenario presented in the Introduction chapter, where each element in the action space is a possible partial resolution. Readers may recall that the learning agent is an entity that interacts with the virtual learning environment by selecting actions and collecting observed costs from the environment.

2.1 Multi-armed Bandits

In a multi-armed bandits setting, we shall denote each action in the action space to be an arm, while the agent should learn the cost distribution associated with each arm and determine the best arm to pull to minimise the long-term sum of costs. Unless otherwise specified, the definitions and algorithms included in this section follow from (Slivkins, 2019).

Definition 2.1.1. A Stochastic multi-armed bandits model has $|\mathcal{A}|$ possible actions to select from, where the costs of each action are independent and identically distributed. The number of rounds of interactions T and $|\mathcal{A}|$ are fixed and known, while the agent picks an action and observes the cost in each round. The assumptions of the model are as follows,

- In each round, the agent only observes the cost for the chosen action.
- The cost distribution of each action is fixed but unknown to the agent.
- When an action is selected, the cost is sampled independently from the corresponding cost distribution.

There are several strategies to select an action in each round, and we shall focus on uniform exploration and Epsilon-greedy policy from (Slivkins, 2019), which are given in Algorithm 1 and 2. Uniform exploration is a special case of Epsilon-greedy, where the agent pulls each arm $N \leq \frac{T}{|\mathcal{A}|}$ times then estimates the mean cost of each arm using the sample mean of N observed collected costs. For the remaining rounds of interaction, the agent keeps selecting the arm associated with the lowest estimated mean cost. The hyperparameter $\epsilon \in [0, 1]$ in Epsilon-greedy policy controls the ratio of exploitation against exploration of the environment. In practice, it may be ideal to increase ϵ as

more explorations have been done, to avoid waste in computation from gaining information that is already acquired. We can see that both algorithms involve exploring the environment and exploiting actions that contribute to the lowest costs in history, while Epsilon-greedy is more flexible in adjusting the proportion of explorations.

Algorithm 1 Uniform exploration (Slivkins, 2019)

Input: Number of rounds T, number of exploration for each arm N.

Set $t = 1$.
while $t < T$ **do**
    if Each arm has been pulled N times **then**
        Pull the arm a_t that has the lowest estimated mean cost.
    else
        Pull an arm a_t that has not been selected for N times.
    end if
    Sample c_t from the cost distribution associated with the action a_t independently and collect the cost c_t.
    Update the estimated mean cost of arm a_t with the collected cost.
    $t \leftarrow t + 1$.
end while

Algorithm 2 Epsilon-greedy policy (Slivkins, 2019)

Input: Number of rounds T, hyperparameter ϵ.

Set $t = 1$.
while $t < T$ **do**
    Sample u from a Bernoulli distribution with probability of success ϵ.
    Randomly select an action $a_t \in \mathcal{A}$ if $u = 0$. Otherwise, pull the arm a_t with the lowest estimated mean cost.
    Sample c_t from the cost distribution associated with the action a_t independently and collect the cost c_t.
    Update the estimated mean cost of arm a_t with the collected cost.
    $t \leftarrow t + 1$.
end while

Thompson sampling is a Bayesian approach to solving Multi-armed bandits problems proposed in (Thompson, 1933), which assigns prior distributions to parameters of the cost distributions. In particular, it enables us to express a degree of uncertainty about the mean of each cost distribution, rather than returning an estimate of the mean cost for each arm. By updating the prior, the agent can combine past information, that is gained from previous interactions, about each parameter in the cost distributions. Thompson sampling is often used to solve Bernoulli bandits, where the observed cost is either 1 or 0, we shall generalise Thompson sampling to any cost distribution, as shown in Algorithm 3. At the start of each round of interaction, the average cost associated

with each action is drawn from a prior distribution p, where p is usually a conjugate prior. Then the agent shall select the action with the lowest sampled average cost and use the collected cost to update the prior. The agent will select an action based on samples drawn from the posterior distribution in the following round.

Algorithm 3 Thompson sampling (Russo and Van Roy, 2014)

Input: Number of rounds T, prior distribution $p(a)$ to parameters of the cost distribution for each arm a.

1: Set $t = 1$.
2: **while** $t < T$ **do**
3: Sample the mean cost associated with each arm from the corresponding prior distribution, $\hat{c}(a) \sim p(a)$.
4: Pull the arm a_t with the lowest sampled mean cost.
5: Sample c_t from the cost distribution associated with the action a_t independently and observe the collected cost c_t.
6: Update the prior distribution for each arm using the observed cost.
7: $t \leftarrow t + 1$.
8: **end while**

In most multi-armed bandits learning algorithms, the agent benefits from the feedback mechanism, where observed costs are used to update the empirical cost distributions and impact the next round of action selection. To decide the optimal arm, one possible approach is to estimate the average cost for each arm $\mu(a) = \mathbb{E}(C_a)$, where C_a is the cost distribution for each action $a \in \mathcal{A}$, by computing the sample mean of cost collected from interactions with the environment. An action $a^* \in \mathcal{A}$ is optimal if $\mu(a^*) = \min_{a \in \mathcal{A}} \mu(a)$, while there could be multiple optimal actions. The performance of an action selection strategy can be measured by cumulative expected regret, which is defined in Definition 2.1.2. Cumulative expected regret is a loss function that quantifies how much the action chosen by the agent deviates from the optimal action.

Definition 2.1.2. (Slivkins, 2019) Given the optimal action has true mean cost μ^* and let a_t be the action chosen by the agent at round $t = 1, 2, ..., T$, the cumulative expected regret at round T is

$$\mathbb{E}[R(T)] = \mathbb{E}[T\mu^* - \sum_{t=1}^{T} \mu(a_t)]$$

In practice, we may approximate the cumulative expected regret at round T with

$$\mathbb{E}[\hat{R}(T)] = T\mu^* - \sum_{t=1}^{T} \widehat{\mu(a_t)},$$

where $\widehat{\mu(a_t)}$ is the sample mean of cost associated to the arm a_t chosen at round t.

However, there are a few limitations to using multi-armed bandits for the purpose of problem resolution. In the telecommunications scenario, the resulting status of fault is non-deterministic upon taking an action while a multi-armed bandits learning environment does not imitate the status of fault in general. Therefore, the agent is not able to also study the transition probabilities of getting from one status of fault to another, which is a learning goal of interest. Apart from the difference in learning goals, a multi-armed bandits model assumes that every arm can be pulled at all times, while it is possible in the scenario that some actions cannot be executed for some status of fault.

2.2 Markov Decision Processes

A Markov decision process is a mathematical structure that describes a virtual learning environment. It includes the state space, action space, transition function, and cost distributions associated with the actions. Note that a Stochastic Shortest Path model is also a Markov decision process. It follows that the BT scenario can be expressed as a Markov decision process, and more specifically a Stochastic Shortest Path problem. The readers may interpret a state as a status of fault, an action in the action space to be a partial resolution, the transition function indicates the probability of moving from a status of fault to another, and a cost distribution describes how the costs of an action are distributed. In general, a reinforcement learning problem can be framed as a Markov decision process, which guarantees the agent can directly observe its current state. Unlike multi-armed bandits, the learning agent is required to learn the transition probabilities, as well as the optimal action for each non-terminal state such that the long-run total cost is minimised. The definitions in this section are from (Otterlo and Wiering, 2012), unless otherwise stated. To illustrate how reinforcement learning can be applied to the scenario, high-level interpretations of a Markov decision process are also given. In the rest of the chapter, we shall denote s_t and a_t to be the state arrived and the action chosen at round t.

Definition 2.2.1. (Otterlo and Wiering, 2012) A Markov decision process can be described as a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, C)$, where $\mathcal{S}$ and $\mathcal{A}$ are finite state space and finite action space respectively. We denote the transition function to be $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$. For each $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, there exists a random variable $c_{s,a,s'} \in C$ linked to the cost of executing action a at state s and results in moving to state s'.

In general, a Markov decision process can be represented visually as a stochastic graph that captures all possible decisions made by the agent and the associated transition probabilities at each state in the state space. As mentioned previously, the BT scenario can be framed as a Stochastic Shortest Path problem. According to (Jafarnia-Jahromi et al., 2021), a Stochastic Shortest Path problem can be modelled as a Markov decision process. The number of rounds in an episode equals the length of an episode of interaction between the agent and the environment, which can be infinite, and it is determined by the actions chosen by the agent. In an episode, the initial state is pre-defined at the start and the interaction terminates only if the agent visits the goal state. We shall define a

Stochastic Shortest Path problem and a Stochastic graph in Definition 2.2.2 and 2.2.3 respectively.

Definition 2.2.2. (Jafarnia-Jahromi et al., 2021) A Stochastic Shortest Path model is represented by a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, C, s_{init}, g)$. $\mathcal{S}, \mathcal{A}$ are the state space and action space, while s_{init} is the pre-defined initial state and $g \notin \mathcal{S}$ is the goal state (also known as the terminal state). Let $\mathcal{S}^+ = \mathcal{S} \cup \{g\}$, the transition function is denoted by $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}^+ \mapsto [0, 1]$. For each $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}^+$, there exists a random variable $c_{s,a,s'} \in C$ linked to the cost of executing action a at state s and results in moving to state s'.

Definition 2.2.3. (Jamali, 2006) A stochastic graph G is denoted to be a triple $G = (V, E, F)$, where $V = \{1, 2, ..., n\}$ is a set of vertices, E is the adjacency matrix of dimension $n \times n$ that indicates which pairs of vertices are connected by an edge, and F is a $n \times n$ matrix which describes the statistics of costs for each edge. The cost $C_{i,j}$ of edge (i, j) is a random variable with probability density function $f_{i,j}$ from the matrix F. The probability distribution $f_{i,j}$ is supposed to be unknown to the learning agent.

The readers may assume that simulations performed in this thesis are based on Stochastic Shortest Path models. We shall also explain the elements in the tuple of a Stochastic Shortest Path model in detail below and their representations in the telecommunications scenario.

State Space $\mathcal{S}$

We shall assume that there are finitely many discrete states in the state space, where each state represents a status of the virtual learning environment. In the telecommunications scenario, a status of fault can be viewed as a state and hence, the size of $\mathcal{S}$ can be interpreted as the number of the status of fault. For simplicity, we may represent a state as an integer and $\mathcal{S} \subset \mathbb{N}$. We have one goal state $g \in \mathcal{S}^+$ as defined in Definition 2.2.2, which can be achieved by resolving all faults. Note that all faults are assumed to be resolvable.

Action Space $\mathcal{A}$

For a multi-armed bandits model, we assume that all actions in the action space can be applied at any round. In the reinforcement learning context, however, we can restrict the available actions to be applied at each state. We use the notation $\mathcal{A}(s)$ to define the set of actions that can be chosen when the agent is at state $s \in \mathcal{S}$. Compared to the bandits setting, adding such a restriction to the actions may be more consistent with the telecommunications setting, for instance, a landline engineer should not be sent when a broadband connection issue is encountered.

Transition Function $\mathcal{P}$

It is assumed that the set of vertices and adjacency matrix of the stochastic graph, defined in Definition 2.2.3, of the Markov decision process of interest are known to the

agent in advance. In other words, the agent has a priori information about the state space, action space, and $\mathcal{A}(s)$ for all non-goal states s. This assumption provides the learning agent with some prior knowledge in identifying if a state can lead to another by applying a particular action. The transition probabilities, however, are to be learnt by the agent. The probability of arriving at state s' from state s by applying action a is denoted to be $\mathcal{P}(s, a, s')$. The following conditions are required to be satisfied,

- $0 \leq \mathcal{P}(s, a, s') \leq 1$ for all $a \in \mathcal{A}, s \in \mathcal{S}, s' \in \mathcal{S}^+$ by the definition of probability.
- For each $s \in \mathcal{S}$,
$$\sum_{s' \in \mathcal{S}^+} \mathcal{P}(s, a, s') = \begin{cases} 1 & a \in \mathcal{A}(s) \\ 0 & a \notin \mathcal{A}(s) \end{cases}$$
- A Markov decision process has the Markov property, so the successive state only depends on the current state and present action. Let s_t and a_t be the state and the chosen action at round t respectively, then
$$\mathbb{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_1, a_1) = \mathbb{P}(s_{t+1}|s_t, a_t) = \mathcal{P}(s_t, a_t, s_{t+1}).$$

The empirical transition probabilities can be estimated from previous episodes, which is the proportion of arriving at state s' by applying action a at state s,

$$\widehat{\mathcal{P}(s, a, s')} = \frac{\sum \mathbb{1}_{\{s,a,s'\}}}{\sum \mathbb{1}_{\{s,a\}}}. \tag{2.1}$$

Cost Distributions

The definition of a cost distribution implies that the cost associated with an action may depend on the current or the successive state. A demonstration of the scenario is that sending two engineers may cost more if the fault is not resolved afterwards, as more time and human resources are required in the long run to fix the fault. Note that we assume the cost distribution $C(s, a, s')$ is non-positive for all $s \in \mathcal{S}$, $s' \in \mathcal{S}^+$ and $a \in \mathcal{A}$ and the cost of reaching the goal state is zero for the rest of the thesis. We define the expected total cost of an episode for a Stochastic Shortest Path model as

$$\mathbb{E}\left[\sum_{t=1}^{\tau} C(s_t, a_t, s_{t+1})\right] = \sum_{t=1}^{\tau} \sum_{s' \in \mathcal{A}(s_t)} \mathcal{P}(s_t, a_t, s') C(s_t, a_t, s'),$$

where τ is the length of an episode. An episode of interactions with the environment ends only when the agent arrives at the goal state for any Stochastic Shortest Path model. Nonetheless, we shall fix the maximum length of an episode T as a correction in case the agent never reaches the goal state due to randomness in our experiments. Therefore, an episode ends either when the goal state is visited or when T rounds of interactions have been done here, it then follows that $\tau \leq T$. If the agent does not reach the terminal

state at the end of an episode, a significant cost will be added to the expected total cost as a penalty. Note that the sum of costs is always bounded since T is finite.

A reinforcement learning agent learns the cost distributions, as well as the transition probability for each transition between states by interacting with the environment. It is worth noting that the successive state given the current state and chosen action is not deterministic in a stochastic Markov decision process model, as well as in a Stochastic Shortest Path model. The information gained from an episode includes the sequence of visited states, selected action for each transition, and the observed cost at each round. Instead of finding the optimal arm in the multi-armed bandits problem, the main learning goal for reinforcement learning is to find the best action to act at each non-goal state such that the lowest expected total cost is achieved, and we refer to it as the optimal policy defined in Definition 2.2.4. Note that this thesis focuses on deterministic policy and readers can read about stochastic policy in (Fard and Pineau, 2011).

Definition 2.2.4. (Barreto et al., 2020; Tarbouriech et al., 2021) A policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ is a mapping from a state to an action in the action space. A policy π is optimal if it minimises the action-value function, which is also known as the Q-function or Q value

$$Q^{\pi}(s,a) = \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} C(s_t, \pi(s_t), s_{t+1}) | s_1 = s, \pi(s_1) = a\right], \tag{2.2}$$

for every state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$.

Most reinforcement learning algorithms can be classified as model-based or model-free. In model-based learning algorithms, the agent is responsible for learning the cost distributions, as well as the transition dynamics according to (Pal and Leon, 2020). Therefore, the algorithm involves computations of empirical transition probabilities as shown in Equation 2.1. A benefit of learning the transition dynamics in model-based algorithms is that the agent is able to predict the result of executing different actions before making a decision in the environment. By (Pal and Leon, 2020), model-free algorithms optimise the policy without modelling the environment explicitly. Unlike model-based algorithms, model-free algorithms do not attempt to model the learning environment and the agent does not learn about the transition probabilities. The model-free algorithms discussed in this thesis rely on the Q-function for each state-action pair, which is stated in Equation 2.2, as we shall see in the Methods chapter. These algorithms usually return a table of Q values for each action-state pair at the end of the interactions, while the optimal policy can be deduced from the table as the values are expected to converge.

The Bellman Equations is a system of non-linear equations whose solution is the optimal policy of a Markov decision process. It is discussed in (Bertsekas and Yu, 2013) and (Atanasov, 2020) that an optimal policy for a Stochastic Shortest Path model exists and is unique if the following assumptions hold.

- The terminal cost, which is the cost of achieving the terminal state, is zero.

- There exists at least one proper policy, where the terminal state is visited with probability 1 starting from any initial state. Besides, each improper policy leads to an infinite sum of costs starting from at least one initial state.

Notice that the first assumption is satisfied as we assumed the goal state is cost-free. All faults in the telecommunications scenario are resolvable, implying there exists at least one sequence of actions that leads to the goal state with probability 1. The value function defined in Definition 2.2.5 measures the goodness of reaching a state. In Bellman Equations stated in Definition 2.2.6, the value function of a state is written as the expectation of a sum of instant observed cost and the value function of the successive state. Lemma 2.2.7 states that the optimal policy minimises the value function, which is non-positive here, for each state. The optimal policy can be computed by the value iteration algorithm, which solves Bellman Equations and finds the optimal value function through backward recursions as illustrated in Lemma 2.2.8. The transition probability and the expected cost of each (current state, action, next state) pair can be estimated from previous episodes and model-based dynamic programming algorithms compute the optimal policy by using these estimates.

Definition 2.2.5. (Sennott, 1996; Tarbouriech et al., 2021) Given a policy π, the value function of a non-terminal state $s \in \mathcal{S}$ is

$$V^{\pi}(s) = \lim_{T \to \infty} \mathbb{E}\left[\sum_{t=1}^{T} C(s_t, \pi(s_t), s_{t+1}) | s_1 = s\right].$$

The value function of the goal state g equals zero.

Definition 2.2.6. (Busoniu et al., 2017) For all non-terminal states $s \in \mathcal{S}$ and a given policy π, the Bellman Equations for the value function are as follows.

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{s' \sim \mathcal{P}(s,\pi(s),\cdot)}\left[C(s, \pi(s), s') + V^{\pi}(s')\right] \\ &= \sum_{s' \in \mathcal{A}(s)} \mathcal{P}(s, \pi(s), s')[C(s, \pi(s), s') + V^{\pi}(s')] \end{aligned} \tag{2.3}$$

$$\begin{aligned} V^{*}(s) &= \min_{a \in \mathcal{A}(s)} \mathbb{E}_{s' \sim \mathcal{P}(s,a,\cdot)}\left[C(s, a, s') + V^{*}(s')\right] \\ &= \sum_{s' \in \mathcal{A}(s)} \mathcal{P}(s, a, s')[C(s, a, s') + V^{*}(s')] \end{aligned} \tag{2.4}$$

Lemma 2.2.7. (Busoniu et al., 2017) A policy π is optimal if for all non-goal states $s \in \mathcal{S}$,

$$\pi(s) \in \arg\min_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{A}(s)} \mathcal{P}(s, a, s')[C(s, a, s') + V(s')]$$

Lemma 2.2.8. (Busoniu et al., 2017) Let $\mathcal{S}_t \subseteq \mathcal{S}$ be a set of states that are achievable

at round t and define

$$V_k(s) = \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=k}^{T} C_t(s_t, a_t, s_{t+1})|s_k = s\right].$$

For any given start state $s \in \mathcal{S}$, the value function can be updated as follows, with $V_1^*(s) = V^*(s)$ given in Equation 2.4,

$$V_t^*(s) = \min_{a\in\mathcal{A}(s_t)} \sum_{s'\in\mathcal{S}_{t+1}} \mathcal{P}(s_t, a, s')\left[C(s_t, a, s') + V_{t+1}(s')\right] \qquad t = T-1, T-2, ..., 1.$$

Similar to multi-armed bandits problems, the performance of a reinforcement learning algorithm can be measured by cumulative expected regret, which is the difference in value functions between the optimal policy and the policy chosen by the agent. It quantifies how much the action chosen by the algorithm, given the current state, deviates from the optimal choice of action. The algorithm calculates the cumulative expected regret at the end of each episode. Ideally, the cumulative expected regret should converge as the number of episodes grows, since the agent ought to optimise its choice of actions as it interacts with the environment more often.

Definition 2.2.9. (Zhang and Ji, 2019) Suppose π^* and π are an optimal policy and the policy returned by the learning agent after T rounds of interactions respectively. Suppose the initial state is s, the cumulative expected regret is given as

$$\mathbb{E}[R_T] = \sum_{t=1}^{T} \mathbb{E}[C(s_t, \pi^*(s_t), s_{t+1})|s_1 = s] - \mathbb{E}[C(s_t, \pi(s_t), s_{t+1})].$$

While performing the experiments in this thesis, we approximate $\mathbb{E}[C(s_t, \pi(s_t), s_{t+1})]$ in the definition above with sample mean of the collected costs $\hat{\mu}(s_t, \pi(s_t), s_{t+1})$ and compute the estimated cumulative absolute expected regret

$$\mathbb{E}[\hat{R_T}] = \sum_{t=1}^{T} \left|\mathbb{E}[C(s_t, \pi^*(s_t), s_{t+1})|s_1 = s] - \hat{\mu}(s_t, \pi(s_t), s_{t+1})\right|. \tag{2.5}$$

We have briefly introduced a stochastic multi-armed bandits model and the limitations of its application to the BT scenario. Also, we presented the definitions of a Markov decision process and a Stochastic Shortest Path model, as well as how the scenario can be framed mathematically as a Stochastic Shortest Path model, which guarantees the optimal sequence of actions can be found by using reinforcement learning. Cumulative expected regret, which is the loss function that measures the performance of a learning algorithm, shall be used throughout the rest of the thesis. We shall discuss a few model-based and model-free reinforcement learning algorithms in detail in the following Methods chapter. Several concepts mentioned in this chapter such as the Bellman Equations, value function, and Q-function will be revisited in the Methods chapter, as they are the fundamental tools in constructing learning algorithms.

3 Methods and Reinforcement Learning Algorithms

In this chapter, we shall introduce several reinforcement learning algorithms, for instance, Q-learning and posterior sampling. Model-based and model-free algorithms are mentioned in the Background chapter, and we aim to highlight the differences in their implementations in this chapter. Moreover, the pseudo-codes of direct policy search, value iteration, and policy iteration by dynamic programming will be discussed here.

3.1 Dynamic Programming Algorithms

We have introduced the value iteration algorithm that determines the optimal policy for a Markov decision process in the Background chapter. It may be computationally expensive to search for an optimal policy in general, as the task consists of exploring all possible combinations of states and actions and calculating the expected total sum of each combination. Dynamic programming, however, reduces the complexity of the problem by first dividing the problem into sub-problems, whose solutions are then stored and reused, rather than computing the solution of the same sub-problem repeatedly, according to (Cormen et al., 2022). A key element in dynamic programming is that the problem has an optimal structure, which means the optimal solution to the problem contains the optimal solution to any sub-problems, as stated in (Cormen et al., 2022). As mentioned in the Background chapter, the successive state depends on the current state but not states in the past due to the Markov property, so the optimal sequence of actions for a trajectory of states includes the optimal sequence of actions for any sub-trajectory. It follows that dynamic programming allows us to compute all possible policies to be applied between an initial state and a terminal state by searching the policies between the terminal state and the intermediate states.

We shall introduce three model-based algorithms, which are direct policy search, value iteration, and policy iteration, that incorporate dynamic programming in their implementations. These action selection strategies require knowledge about the transition function and the cost distributions, which are both unknown to the agent, in order to model the environment and determine the optimal policy. Hence, the agent is responsible to estimate the empirical transition probabilities and cost distributions using the response from the environment during their interactions. The algorithms for direct policy search and value iteration (VI) to determine the optimal policy are given in Algorithm 4 and 5 respectively. Note that direct policy search requires a significant amount of computations and is unpopular compared to the value iteration algorithm. Considering there are limited resources for building a direct policy search algorithm, we shall develop

our own here. In value iteration, the update rule of the value function can be viewed as choosing the action that minimises the expected total cost by looking a step ahead. For a large Markov decision process, it may take more iterations to achieve convergence of the value functions. To investigate the performances of the two algorithms, we designed two reinforcement learning models that incorporate the Epsilon-greedy algorithm given in Algorithm 2, such that either a random action is selected for exploration or the action returned by value iteration is chosen for one model, and the other model chooses the action output from direct policy search for exploitation. Sufficient exploration of the environment is important for the learning agent in model-based algorithms as the agent has no prior knowledge about the transition function and cost distributions at the beginning of interactions, and it needs this information to decide the best action to act. As a result, the exploration phase provides opportunities for the agent to model the environment and find the optimal policy.

Algorithm 4 Direct policy search

Input: A set of non-goal states, maximum number of time steps to look ahead T, the Markov decision process model.

If the true Markov decision process is unknown, compute the empirical transition probabilities for each transition and each action by Equation 2.1 and estimate the cost distribution for each action and transition by the sample mean, using historical episodes.
for each non-goal state s_0 **do**
    Initialise a list that contains the goal state g. This list records a reversed path of states that can be visited from the initial state s_0 to the goal state.
    Create a dictionary D indexed by each action in the action space.
    while the maximum length of all lists $< T$ **do**
        for states that can transit to the last state in the list **do**
            if the last state of the list is not s_0 **then**
                Append the state to the list.
            end if
        end for
    end while
    for all reversed paths computed above **do**
        if the last state in the list is s_0 **then**
            Determine all sequences of action that may give rise to the path.
            For each possible path, calculate and increment the expected total cost to the entry indexed by the action taken in state s_0 in D.
        else
            Discard the list.
        end if
    end for
end for

Output: The optimal action to act in each non-goal state.

Algorithm 5 Value iteration (Kaelbling et al., 1996)

Input: A Markov decision process model, maximum number of iterations n, δ.

Initialise a list V of length equals the number of states with every entry being zero. V stores the value function for each state.
Create a list π of length equals the number of states. The list stores the optimal policy.
Set $D \leftarrow \delta + 1$.
while $D > \delta$ and number of iteration is less than n **do**
    Set $D \leftarrow 0$.
    for each state $s \in \mathcal{S}$ **do**
        Compute $V^*(s)$ using Equation 2.4.
        Calculate the absolute difference between the old and updated value $\Delta \leftarrow |V(s) - V^*(s)|$.
        $V(s) \leftarrow V^\pi(s)$.
        Replace the policy for state s with the action that gives rise to $V^*(s)$.
        if $\Delta > D$ **then**
            $D \leftarrow \Delta$.
        end if
    end for
end while

Output: A list of value functions for each state and the optimal policy.

An alternative approach to searching for the optimal policy is suggested in (Pashenkova et al., 1996), which is called policy iteration (PI). This approach recursively computes the value function for each state given the current policy until convergence, then updates the current policy based on the updated value functions. Recall from the Background chapter that a policy is proper if the probability of reaching the goal state is one for all initial states. It is described in (Bertsekas and Tsitsiklis, 1991) that policy iteration and value iteration are able to determine the optimal proper policy for Stochastic Shortest Path problems. We shall verify the claim and compare the efficiency between the algorithms in the Results chapter. The policy iteration algorithm is provided in Algorithm 6.

Algorithm 6 Policy iteration (Pashenkova et al., 1996)

Input: A Markov decision process model, maximum number of iterations n, δ.

Initialise a list V of length equals the number of states with every entry being zero. V stores the value function for each state.
Initialise a policy by randomly assigning an action to each non-goal state.
`policy_stable` $\leftarrow$ `False`.
while `policy_stable` = `False` **do**
    `policy_stable` $\leftarrow$ `True`.
    Set $\Delta \leftarrow 0$.
    while $\Delta > \delta$ **do**
        for each state $s \in \mathcal{S}$ **do**
            Compute $V^{\pi}(s)$ using Equation 2.2.5 and policy π.
            Calculate the absolute difference between the old and updated value $\Delta \leftarrow |V(s) - V^{\pi}(s)|$.
            $V(s) \leftarrow V^{\pi}(s)$.
            if $\Delta > D$ **then**
                $D \leftarrow \Delta$.
            end if
        end for
    end while
    for each state $s \in \mathcal{S}$ **do**
        Find new optimal action for state s, $\pi'(s)$, that minimises $V^{\pi'}(s)$.
        if $\pi'(s)$ is different from $\pi(s)$ **then**
            `policy_stable` $\leftarrow$ `False`.
        end if
    end for
end while

Output: A list of value functions for each state and the optimal policy.

3.2 Q-learning

Q-learning is briefly presented in the Background chapter as a model-free learning algorithm. The algorithm does not attempt to model the environment, unlike value iteration and direct policy search, while it dedicates to calculating the expected sum of cost given any action taken at any initial state. We shall introduce the Greedy policy, which is a special case of Epsilon-greedy policy when the hyperparameter ϵ equals one. The greedy policy aims to always pick the action that is most likely to give the lowest long-term cost based on the agent's knowledge. Note that the Q-learning model designed in this thesis adapts the Epsilon-greedy exploration policy, introduced in Algorithm 2, and updates Q values given in Equation 2.2 with the Greedy policy. The update rule of Q value is

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[C(s_t, a_t, s_{t+1}) + \max_{a \in \mathcal{A}(s_{t+1})} Q_t(s_{t+1}, a) - Q_t(s_t, a_t)], \quad (3.1)$$

where s_t, a_t are the state and action selected at round t in an episode. Note that α is a hyperparameter that represents the learning rate. The learning rate controls the step size at which the algorithm updates the Q values and it is chosen such that the mean of expected total cost across all episodes is minimised. The optimal policy π can be approximated from the table of Q values returned by the algorithm, where $\pi(s)$ is the action that corresponds to the largest Q value for each $s \in \mathcal{S}$.

Nonetheless, it is shown in (Hasselt, 2010) and (Jang et al., 2019) that Q-learning may overestimate the Q values for stochastic Markov decision processes and the algorithm may perform poorly in a large state-action environment, due to a lack of exploration of each state-action pair. The author of (Hasselt, 2010) suggests that overestimation of Q values may result from using the maximum operator and observed collected costs to update Q values. Each observed cost is sampled from a distribution that may have a large variance and the noise in the sample may induce a positive bias when the agent selects the maximum Q value in the update step. The issues mentioned above motivate the application of double Q-learning. However, we shall be aware that double Q-learning may lead to underestimation of the Q values, as raised in (Ren et al., 2021). We shall demonstrate the pseudo-codes of generating an episode of interaction for both algorithms below. The Q table returned by the algorithm at the end of an episode will be used for selecting actions in the next episode, while each entry in the table shall be set to zero initially.

Algorithm 7 Q-learning (Liu et al., 2020)

Input: Exploitation probability ϵ, learning rate α, maximum number of round T.

1: Initialise a Q table or use the Q table returned from the previous episode. The Q table is of dimension $|\mathcal{S}| \times |\mathcal{A}|$.
2: Randomly select an initial state s and set $t = 0$.
3: **while** $t < T$ and s is not the goal state **do**
4: Draw a Bernoulli sample u with a probability of success equals ϵ. Randomly choose an action $a_t \in \mathcal{A}(s)$ if $u = 0$, select $a_t \in \underset{a \in \mathcal{A}(s)}{\arg\max} Q_t(s, a)$ otherwise.
5: Observe the successive state s' by acting a_t at state s and the collected cost $c(s, a_t, s')$.
6: Update the Q-value $Q_{t+1}(s, a_t)$ using Equation 3.1.
7: Increment t by one and set $s \leftarrow s'$
8: **end while**

Output: A Q-table.

Algorithm 8 Double Q-learning (Hasselt, 2010)

Input: Exploitation probability ϵ, learning rate α, maximum number of round T.

Initialise two Q tables Q^A, Q^B, each of dimension $|\mathcal{S}| \times |\mathcal{A}|$, or use Q^A, Q^B returned from the previous episode.
Randomly select an initial state s and set $t = 0$.
while $t < T$ and s is not the terminal state **do**
    Draw a Bernoulli sample u with a probability of success equals ϵ. Randomly choose an action $a_t \in \mathcal{A}(s)$ if $u = 0$, select $a_t \in \arg\max_{a \in \mathcal{A}(s)}[Q_t^A(s,a) + Q_t^B(s,a)]$ otherwise.
    Observe the successive state s' by executing a_t at state s and the collected cost $c(s, a_t, s')$.
    Draw a Bernoulli sample v with a probability of success 0.5.
    if $v = 0$ **then**
        Let $a^* = \underset{a \in \mathcal{A}(s)}{\arg\max} Q_t^A(s', a)$
        $Q_{t+1}^A(s, a_t) \leftarrow Q_t^A(s, a_t) + \alpha(c(s, a_t, s') + Q_t^B(s', a^*) - Q_t^A(s, a_t))$
    else
        Let $b^* = \underset{a \in \mathcal{A}(s)}{\arg\max} Q_t^B(s', a)$
        $Q_{t+1}^B(s, a_t) \leftarrow Q_t^B(s, a_t) + \alpha(c(s, a_t, s') + Q_t^A(s', b^*) - Q_t^B(s, a_t))$
    end if
    Increment t by one and set $s \leftarrow s'$
end while

Output: Q-tables Q^A, Q^B.

3.3 Bayesian Reinforcement Learning

We discussed Thompson sampling which determines the best arm in Multi-armed bandits problems with a Bayesian approach in the Background chapter. We are interested in adopting this approach in a reinforcement learning setting. In this section, the readers will be presented with posterior sampling reinforcement learning algorithm introduced in (Agrawal and Jia, 2017; Russo et al., 2018) and Bayesian Q-learning from (Dearden et al., 1998). Furthermore, we shall establish the appropriate choices of the prior distributions for each learning algorithm. It is suggested in (Russo et al., 2018) that the prior distributions for both the transition function and cost distributions should be updated at the end of each episode, as the cumulative expected regret converges with significantly fewer episodes than updating at the end of each round in an episode. Therefore, we shall analyse and investigate the performance of each algorithm when the prior distributions are updated per episode.

3.3.1 Posterior Sampling

For Bayesian reinforcement learning, it is required to assign prior distributions to the transition function and the cost distributions, which are both unknown to the agent. It is discussed in (Russo et al., 2018) that the Dirichlet distribution comes as a handy choice when the states are discrete, as it is a conjugate prior of the multinomial distribution. The distribution is defined in Definition 3.3.1.

Definition 3.3.1. (Lin, 2016) Denote $y^k = (y_1, ..., y_k)$ to be a vector of length k with non-negative $y_i, i = 1, ..., k$ and $\sum_{i=1}^{k} y_i = 1$. Let $\alpha = (\alpha_1, ..., \alpha_k)$ with $\alpha_i > 0$ for all i. The probability density function for a Dirichlet distribution $Dir(\alpha_1, \alpha_2, ..., \alpha_k)$ is

$$f(y^k) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} y_i^{\alpha_i - 1}.$$

During the interactions between the agent and environment for posterior sampling, a Dirichlet prior distribution is assigned for each state-action pair, while a $Dir(1, ..., 1)$ prior should be assigned initially. In a sample $(y_1, ..., y_k)$ from the Dirichlet distribution, with k equals the size of state space $\mathcal{S}$, the entry y_i can be interpreted as the probability of reaching state i from the current state. The parameter $\alpha_i, i = 1, ..., k$, in the transition prior distribution for state s and action a is the pseudo-count of the agent's arrivals to state i from state s by executing a during interactions.

Since the cost distributions are not known to the agent in general, the choice of the prior distribution is crucial. If the prior cost distribution for an action is narrowed and shifted with small variance and low mean, then the agent may avoid selecting such action because the prior distribution implies it may induce a great cost. As a result, the action may less likely to be explored and the agent may fail to determine the optimal policy if such action is the best action to execute in some states. It is preferable to use an uninformative prior or prior with a large variance if we have insufficient knowledge about the cost distributions, which guarantees none of the action will lack being explored. Although it is assumed that the cost distributions are non-positive with known variances and are continuous in the BT scenario, it is more straightforward to use a normal prior. It will be illustrated in the Results chapter that models with normal cost prior distributions perform well. For each state-action pair, a standard Normal distribution should initially be assigned as the prior cost distribution for each action. The posterior distribution for a normal prior is given in Lemma 3.3.2.

Lemma 3.3.2. (Murphy, 2007) Given a Normal distribution $N(\mu, \sigma^2)$ with known variance and a Normal prior distribution $N(\mu_0, \sigma_0^2)$. Upon observing n observations $x_1, ..., x_n$, the posterior mean μ_n and posterior variance σ_n^2 are

$$\sigma_n^2 = \frac{\sigma^2 \sigma_0^2}{n\sigma_0^2 + \sigma^2} \tag{3.2}$$

$$\mu_n = \sigma_n^2 \left(\frac{\mu_0}{\sigma_0^2} + \frac{n\bar{x}}{\sigma^2} \right). \tag{3.3}$$

The posterior sampling algorithm for generating an episode of interaction is presented in Algorithm 9. It is worth noting that steps 8 and 9 in the algorithm update the prior transition function and prior cost distributions using the observed states and collected costs, which are the responses from the environment. The updated prior distributions will be utilised to sample the empirical transition function and estimated cost distributions in the following episode of interactions, to optimise the policy determined by the agent.

Algorithm 9 Posterior sampling (Agrawal and Jia, 2017)

Input: Maximum number of round in an episode T, prior transition distributions $Dir(\alpha_{(s,a)})$ for each state-action pair and Normal prior cost distributions $N(\mu_{(s,a,s')}, \sigma^2_{(s,a,s'))}$ for all $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}^+$.

1: Randomly select an initial state s and set $t = 0$.
2: Sample the transition function and the cost distribution for each action from the prior distributions.
3: **while** $t < T$ and s is not the goal state **do**
4: Apply the Greedy policy to select an action a_t that has the smallest mean in the sampled cost distribution. Alternative, apply the Epsilon-greedy algorithm with the transition function and cost distributions samples to select an action a_t.
5: Observe and store the cost collected by executing a_t at state s and the successive state s'.
6: Increment t by one and $s \leftarrow s'$.
7: **end while**
8: Update the prior transition distribution for each state-action pair (s, a) by updating the pseudo-counts $\alpha_{(s,a)}$ with the observed transitions.
9: Compute the posterior mean and posterior variance for each prior cost distribution with Equation 3.3 and 3.2 using the costs collected from the environment.

Output: An episode of interaction, posterior distribution of transition function, and posterior cost distributions.

3.3.2 Bayesian Q-learning

Bayesian Q-learning aims to study the optimal Q value $Q^*(s, a)$, which is the expected total cost given action a is executed at the initial state s, through sampling Q values. Since Q-learning is model-free, it is not necessary to assign a prior distribution to the transition function. This section will be based on the results analysed in (Dearden et al., 1998), unless stated otherwise.

Let $Q_{s,a}$ be a random variable that represents the Q value for state s and action a, which is the expected total cost when action a is executed at state s. It follows that the optimal Q value $Q^*(s, a) = \mathbb{E}[Q_{s,a}] = \mu_{s,a}$. Assumptions of Bayesian Q-learning are listed below. Notice that the goal of the algorithm is to learn $\mu_{s,a}$ for all non-goal states and actions.

- $Q_{s,a}$ follows a Normal distribution with mean $\mu_{s,a}$ and precision $\tau_{s,a}$. The precision

of a Normal distribution is the reciprocal of the variance.

- The prior distribution of $\mu_{s,a}$ and $\tau_{s,a}$ is a Normal-Gamma distribution $NG(\mu_0, \lambda, \alpha, \beta)$, such that
$$p(\mu, \tau) \propto \tau^{\alpha+\frac{1}{2}} \exp\left(-\frac{1}{2}\lambda\tau(\mu-\mu_0)^2 + \beta\tau\right).$$
- For $s \neq s'$ and $a \neq a'$, the prior distributions of $(\mu_{s,a}, \tau_{s,a})$ and $(\mu_{s',a'}, \tau_{s',a'})$ are independent.
- For $s \neq s'$ and $a \neq a'$, the posterior distributions of $(\mu_{s,a}, \tau_{s,a})$ and $(\mu_{s',a'}, \tau_{s',a'})$ are independent in any episode of interaction.

In (Dearden et al., 1998), the paper suggests the motivation behind Q-value sampling is that an action is chosen given that it is optimal in our current belief. It also follows from (Dearden et al., 1998) that the probability of choosing the action a is

$$\begin{aligned}\mathbb{P}(a = \arg\max_{a'} \mu_{s,a'}) &= \mathbb{P}(\mu_{s,a} > \mu_{s,a'} \quad \forall a \neq a') \\ &= \int_{-\infty}^{\infty} \mathbb{P}(\mu_{s,a} = q_a) \prod_{a' \neq a} \mathbb{P}(\mu_{s,a'} < q_a) dq_a\end{aligned}$$

Next, we shall state two standard results about a Normal-Gamma distribution and the computation of a posterior distribution. It follows from Lemma 3.3.3 that a Normal-Gamma distribution is also a conjugate prior. At the start of the learning algorithm, the prior distribution for each pair of state and action should be $NG(0, 1, 1, 1)$ and samples of Q values can be drawn using the result stated in Lemma 3.3.4. The Q-value sampling algorithm for returning an episode of interaction is provided in Algorithm 10. The costs observed throughout an episode will be used to update the prior distributions and the Q values are sampled from the updated prior in the following episode.

Lemma 3.3.3. (Dearden et al., 1998) Suppose we have a prior distribution $p(\mu, \tau) \sim NG(\mu_0, \lambda, \alpha, \beta)$ over the unknown parameters μ, τ of a Normal distribution and n independent samples $r_1, ..., r_n$. Then the posterior distribution $p(\mu, \tau | r_1, ..., r_n)$ is a Normal-Gamma distribution with parameters $(\mu_0', \lambda', \alpha', \beta')$, where

$$M_1 = \frac{1}{n}\sum_{i=1}^{n} r_i, \quad M_2 = \frac{1}{n}\sum_{i=1}^{n} r_i^2$$

$$\mu_0' = \frac{\lambda\mu_0 + nM_1}{\lambda + n}, \tag{3.4}$$

$$\lambda' = \lambda + n, \tag{3.5}$$

$$\alpha' = \alpha + \frac{1}{2}n, \tag{3.6}$$

$$\beta' = \beta + \frac{1}{2}n(M_2 - M_1^2) + \frac{n\lambda(M_1 - \mu_0)^2}{2(\lambda + n)}. \tag{3.7}$$

Lemma 3.3.4. (Dearden et al., 1998) The marginal density function $p(\mu)$ and the cumulative density function $F(x)$ of μ given a Normal-Gamma distribution with parameters $(\mu_0, \lambda, \alpha, \beta)$ are

$$p(\mu) = \sqrt{\frac{\lambda}{2\pi}}\beta^{\alpha}\frac{\Gamma\left(\alpha+\frac{1}{2}\right)}{\Gamma(\alpha)}\left(\beta+\frac{1}{2}\lambda(\mu-\mu_0)^2\right)^{(\alpha+\frac{1}{2})}$$

$$F(x) = \mathbb{P}(\mu < x) = T((x-\mu_0)\left(\frac{\lambda\alpha}{\beta}\right)^{\frac{1}{2}} : 2\alpha). \tag{3.8}$$

Note that the cumulative density function of a t-distribution with v degrees of freedom is denoted as $T(x; v)$. A Q value can be generated by first drawing a sample u from a t-distribution with 2α degrees of freedom, then compute

$$u\left(\frac{\beta}{\lambda\alpha}\right)^{\frac{1}{2}} + \mu_0.$$

Algorithm 10 Q-value sampling (Dearden et al., 1998)

Input: Maximum number of round T in an episode and prior distributions of Q value for each state-action pair.

1: Randomly select an initial state s and set $t = 0$.
2: Sample a Q value $Q(s, a)$ from the prior distribution for each state-action pair (s, a) using Lemma 3.3.4.
3: **while** $t < T$ and s is not the goal state **do**
4: Apply the Greedy algorithm to select $a \in \arg\max_{a' \in \mathcal{A}(s)} Q(s, a')$.
5: Observe and store the cost collected by executing a at state s and the successive state s'.
6: Increment t by one and $s \leftarrow s'$.
7: **end while**
8: Update the prior distribution for each state-action pair using the costs collected from the environment and Lemma 3.3.3.

Output: An episode of interaction and the posterior distributions of Q value $Q_{s,a}$ for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

In this chapter, we described several model-based and model-free reinforcement learning algorithms and discussed the options of prior distributions for the Bayesian reinforcement learning models. We shall present the Stochastic Shortest Path models that simulate the BT scenario and the choice of learning algorithms for experiments in the Designs chapter. The performance of the action selection strategies introduced in this chapter shall be discussed in the Results chapter.

4 Designs of Models for Experiments

The Markov decision process models, which simulate the telecommunications scenario, for the experiments will be presented in this chapter. It is mentioned in the Introduction chapter that a state and an action represent a status of fault and a partial resolution respectively. Since there are usually fewer than five choices of partial resolution in the BT scenario, the models are designed with an action space of size less than five. We consider a model with 7 states and 3 actions, and a model with 14 states and 4 actions. For each model, we are interested in examining the stability of each action selection strategy demonstrated in the Methods chapter when the cost distributions are fixed and when they vary in transitions. The performances of each algorithm, in terms of computation time and rate of convergence of the cumulative absolute expected regret, will be illustrated in the Results chapter.

To evaluate the performance of a reinforcement learning algorithm, we conduct 20 simulations where each simulation consist of 5000 episodes and compute the cumulative absolute expected regret at the end of each episode. The true optimal policy is calculated using the true transition function and cost distributions of each model with direct policy search, value iteration, and policy iteration, and we shall examine if these algorithms return the same policy. The estimated mean cost associated with each action, which is used to approximate the cumulative absolute expected regret, is updated at the end of each episode. A Stochastic Shortest Path model does not impose a constraint on the length of an episode of interactions and an episode ends if and only if the learning agent visits the goal state, but we shall set an upper bound for the length here as a correction in case the agent never reaches the goal state. Given that we assume all faults can eventually be resolved and the state space in both models is not significantly large, the maximum number of rounds in an episode is set to be 100. The models are designed such that there is a low probability of the agent getting stuck in some states during the early phase of exploration. Once the agent gained some knowledge about the model from the beginning of exploration, it should be able to choose actions sensibly and determine a sequence of actions that minimises the expected total cost. It follows that the learning agent is unlikely to suggest a sequence of actions of length more than 100, as such sequence tends to contribute to a great long-term cost. Therefore, 100 rounds should provide sufficient time for the agent to interact with the environment in an episode. The choices of hyperparameters for the algorithms, including ϵ in Epsilon-greedy, learning rate α in Q-learning and double Q-learning will be explained for each model. We then proceed by taking the average of the cumulative absolute expected regret for each episode across 20 simulations and plotting the cumulative absolute expected regret with a 99% confidence interval against the index of the episode. A 99% confidence interval of cumulative absolute expected regret for the t-th episode can be computed as

follows,

$$\left(\mathbb{E}[\hat{R}_T] + z_{0.005}\sigma, \mathbb{E}[\hat{R}_T] + z_{0.995}\sigma\right),$$

where $\mathbb{E}[\hat{R}_T]$ is given in Equation 2.5 in the Background chapter, z_α is the α quantile of a standard normal distribution and σ is the standard error of the cumulative absolute expected regrets of 20 simulations. The running time of each learning algorithm will be recorded and the comparison shall be based on the average computation time for a simulation. We set the variance for each action's cost distribution to be one in all models.

Model 1

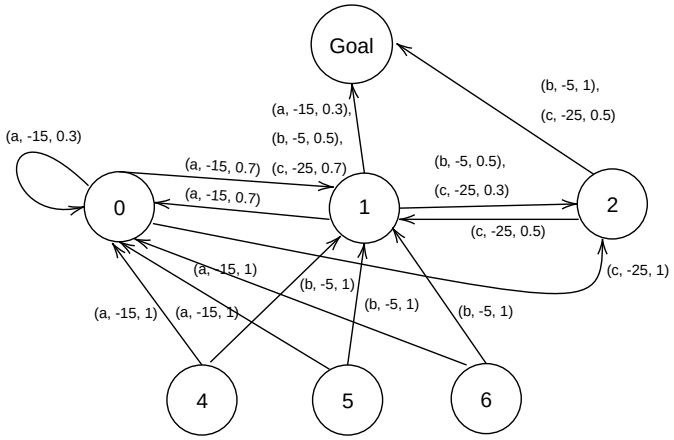


Figure 4.1: Stochastic graph for model 1: Markov decision process with 7 states and fixed cost distributions. Each tuple (x_1, x_2, x_3) in the diagram represents (action, mean cost associated with the action, transition probability from a state to the state pointed to by an arrow).

The first model, whose stochastic graph is shown in Figure 4.1, has 7 states and 3 actions in the action space $\{a, b, c\}$, where the costs for the actions are normally distributed with mean -15, -5, and -25 respectively and their variances are 1. The initial state for this model is chosen among states 4, 5, and 6. This model has the simplest design, as the cost setting is identical to that of a multi-armed bandits problem. We apply all three dynamic programming algorithms to compute the optimal policy and verify if they return identical policies. The maximum number of time step T that the learning agent in direct policy search needs to look ahead is set to be 15. This experiment aims to compare the converged value of cumulative absolute expected regret and the running time of the learning algorithms. We shall be running experiments on all algorithms introduced earlier in the Methods chapter. With this model, we can observe which algorithms perform poorly even with simple settings of the environment, then discard using them for other more complex models.

Next, we shall mention our choice of hyperparameters in different reinforcement learning algorithms for the experiment. Note that separate simulations will be generated with increasing ϵ and constant ϵ for all dynamic programming algorithms which incorporate the Epsilon-greedy policy. Recall from the Background and Methods chapters that the benefit of using an increasing ϵ in Epsilon-greedy policy is to reduce waste in computation from doing explorations when the agent has sufficient knowledge about the environment to decide the optimal policy. The hyperparameter ϵ in Epsilon-greedy grows from 0.5 to 1 at a constant rate in the first 3000 episodes and stays 1 for the remaining episodes. On the other hand, the learning rate α in Q-learning and Double Q-learning controls the step size of updating Q values and is chosen by first simulating 5000 episodes of interactions using Epsilon-greedy policy with increasing ϵ for each $\alpha \in \{0.1, 0.11, ..., 1\}$, then select the value of α that minimises the mean total cost across the episodes. Recall that policy iteration requires exploration of the environment without any exploitation at the beginning, such that the value functions can converge. Therefore, we let the agent randomly select actions in the first 500 episodes, then apply Epsilon-greedy policy with ϵ increasing from 0.5 to 1 steadily in the following 2500 episodes and set $\epsilon = 1$ for the last 2000 episodes. For both value iteration and policy iteration, we let the maximum number of iterations to be 1000 and the bound of convergence δ to be 0.001. The rate of increase in ϵ and procedures of selecting the hyperparameters are the same for experiments on other models unless stated otherwise. A list of algorithms being experimented with model 1 is outlined below.

- Direct policy search incorporates Epsilon-greedy policy with increasing ϵ and $\epsilon = 0.9$.
- Value iteration incorporates Epsilon-greedy policy with increasing ϵ and $\epsilon = 0.9$.
- Policy iteration incorporates Epsilon-greedy policy.
- Q-learning with learning rate $\alpha = 0.9$.
- Double Q-learning with learning rate $\alpha = 0.54$.
- Posterior sampling with direct policy search incorporates Epsilon-greedy policy with increasing ϵ and Greedy policy.
- Posterior sampling using value iteration with Epsilon-greedy policy with increasing ϵ and Greedy policy.
- Posterior sampling using policy iteration with Epsilon-greedy policy with increasing ϵ and Greedy policy.
- Q-value sampling.

Model 2

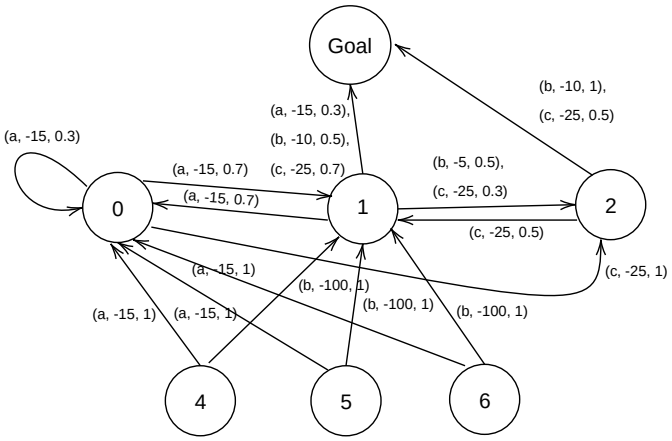


Figure 4.2: Stochastic graph for model 2: Markov decision process with 7 states and cost distributions dependent on the current and successive states. Each tuple (x_1, x_2, x_3) in the diagram represents (action, mean cost associated with the action, transition probability from a state to the state pointed to by an arrow).

The second model shares the same transition function as model 1, while the cost distribution for action b varies with transitions, as illustrated in Figure 4.2. The initial state of an episode is randomly selected between states 4, 5, and 6. Observe that the mean cost of executing action b is -100 at state 5 and is -10 when achieving the goal state from state 1. Also, executing action b in state 1 induces a mean cost of -5 if the agent achieves the state 2 next, so the cost distribution depends on the current and successive states. The objective of this experiment is to test the stability of the algorithms when the cost distributions vary in transition. Since the optimal policy returned by direct policy search is different from the true optimal policy, it is decided not to run experiments on the algorithm due to its instability. It is discovered in the previous experiment on model 1 that combining dynamic programming and Epsilon-greedy policy with constant ϵ gives unsatisfactory results, so we shall not run any experiments on them again. We will be running experiments for model 2 on the following algorithms.

- Value iteration incorporates Epsilon-greedy policy with increasing ϵ.
- Policy iteration incorporates Epsilon-greedy policy with increasing ϵ.
- Q-learning with learning rate $\alpha = 0.30$.
- Double Q-learning with learning rate $\alpha = 0.33$.
- Posterior sampling using value iteration and Greedy policy.

- Posterior sampling using policy iteration and Greedy policy.
- Q-value sampling.

Model 3

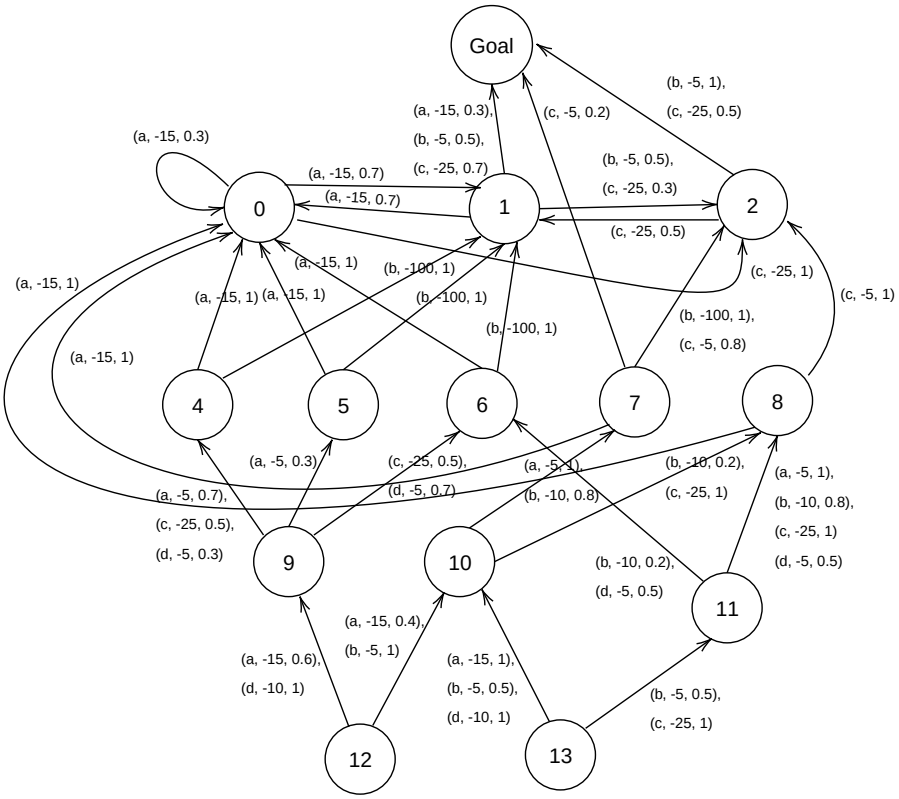


Figure 4.3: Stochastic graph for model 3: Markov decision process with 14 states and varying cost distributions. Each tuple (x_1, x_2, x_3) in the diagram represents (action, mean cost associated with the action, transition probability from a state to the state pointed to by an arrow).

The third model consists of 14 states and 4 actions $\{a, b, c, d\}$ with varying cost distributions, and its stochastic graph is given in Figure 4.3. At the start of each episode, the initial state is chosen randomly between states 12 and 13. Note that the cost distribution for each action only depends on the current state, while it also depends on the successive state in model 2. We notice from the experiments on model 2 that the cumulative absolute expected regret for the Q-value sampling algorithm does not converge, so we shall investigate if the algorithm is sensitive to cost distributions that depend on the successive state. Experiments on model 3 will be run for the same set of algorithms in

model 2, while the learning rates α for Q-learning and double Q-learning are chosen to be 0.23 and 0.32 respectively.

Model 4

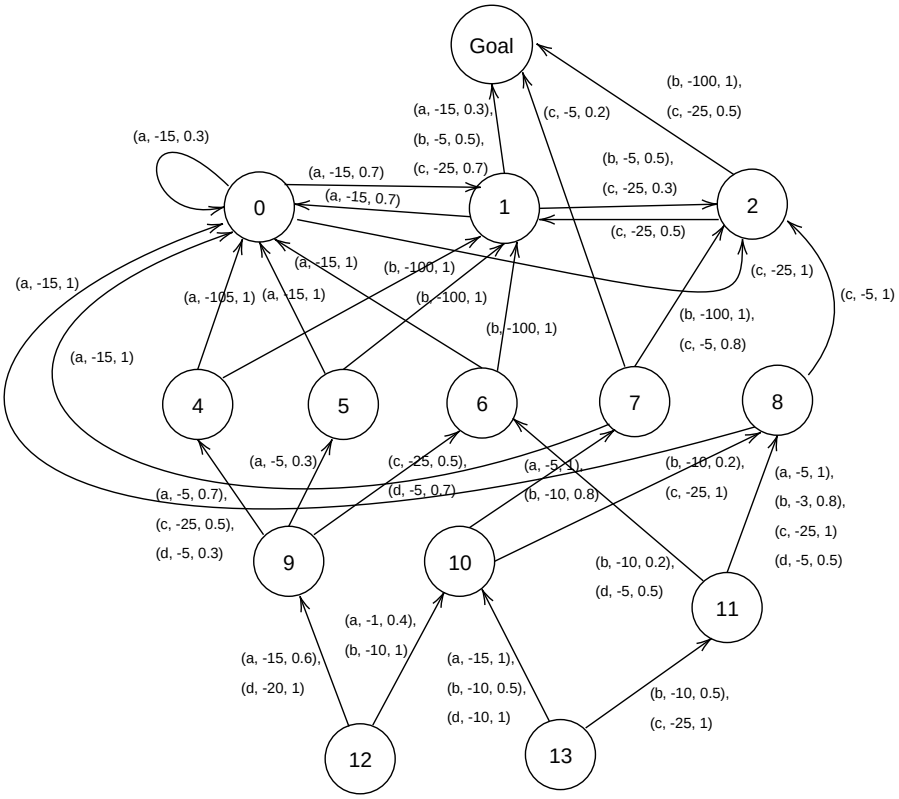


Figure 4.4: Stochastic graph for model 4: Markov decision process with 14 states and the cost distributions depend on both current and successive states. Each tuple (x_1, x_2, x_3) in the diagram represents (action, mean cost associated with the action, transition probability from a state to the state pointed to by an arrow).

It is found in earlier experiments that double Q-learning suffers from an underestimation of Q values for model 3, where the model has a large state space. On the other hand, the cumulative absolute expected regret of Q-value sampling does not converge in model 2, where the cost distributions depend on the successive state. Therefore, we shall design a model that has a large state-action environment with cost distributions dependent on the successive state and investigate if similar issues are encountered again with double Q-learning and Q-value sampling. The stochastic graph for the fourth model is shown in Figure 4.4. The transition function, state space, action space, and selection of the initial state for models 3 and 4 are identical, while the cost distributions in model 4 depend on

both current and successive states. For instance, the mean costs of executing action a in state 12 are -15 if the agent reaches state 9 afterwards and -1 if the agent next arrives in state 10. We shall perform an experiment on the same set of learning algorithms as in model 2. Note that the learning rates α for Q-learning and double Q-learning are selected to be 0.97 and 0.14 respectively.

To summarise, the models shown in this chapter imitate the telecommunications scenario and simulate the situations when the cost distribution for each action is fixed, when it depends on the current state, and when it depends on both the current and successive states. We also describe the selection of hyperparameters and present the choices of learning algorithms for the experiments. The results of the experiments performed for each model introduced in this chapter will be discussed in the following Results chapter.

5 Results of Experiments and Discussions

The results of experiments on the reinforcement learning algorithms for each model described in the Designs chapter are analysed and presented in this chapter. The readers can find plots of the average cumulative absolute expected regret and the average computation time of producing a simulation of 5000 episodes for each action selection strategy below, where the running times are rounded to four significant figures. For clarity in the plots, we shall be referring to each algorithm in a short form indicated below.

Algorithms	Short form
Direct policy search	Direct
Value iteration	VI
Policy iteration	PI
Q-learning	QL
Double Q-learning	Double QL
Posterior sampling with Greedy policy incorporates direct policy search	PS Direct
Posterior sampling with Greedy policy incorporates value iteration	PS VI
Posterior sampling with Greedy policy incorporates policy iteration	PS PI
Q-value sampling	QS

Table 5.1: Short forms of the experimented algorithms.

Model 1

Experiments on model 1 are conducted for all combinations of action selection strategies, as outlined in the Designs chapter. The optimal policies returned by all three dynamic programming algorithms align with each other and are given in Table 5.2.

state s	0	1	2	4	5	6
$\pi(s)$	a	b	b	b	b	b

Table 5.2: Optimal policy π for model 1.

A plot of average cumulative absolute expected regret for all algorithms is displayed

in Figure 5.1. We can observe from the plot that the converged value of cumulative absolute expected regret for policy iteration is significantly larger than that for other algorithms. Recall that the agent is required to explore the environment without any exploitation at the start of policy iteration, which may execute actions with higher costs more often and hence, contribute to receiving higher costs in an episode. On the other hand, Double Q-learning, Q-learning, Epsilon-greedy with increasing ϵ that combines direct policy search and value iteration achieve similar converged values. It is worth mentioning the convergence for direct policy search and Q-learning are almost identical, although Q-learning gives a narrower confidence interval. Despite the limit of cumulative absolute expected regret for double Q-learning being slightly higher than Q-learning, it may result from the randomness in exploring the environment. The confidence interval of value iteration is notably wider among the four algorithms, showing it contributes to greater variance. Since the values of convergence of cumulative absolute expected regret for posterior sampling with value iteration and policy iteration, as well as Q-value sampling, are much lower than other algorithms, we shall make a separate plot in Figure 5.2 for comparison.

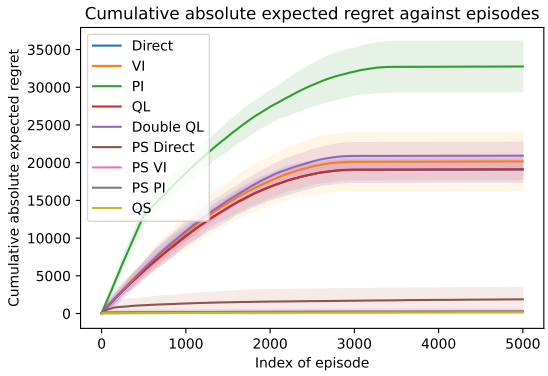


Figure 5.1: The plot of average cumulative absolute expected regret of the experimented algorithms against the index of the episode for model 1. The shaded region corresponds to a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

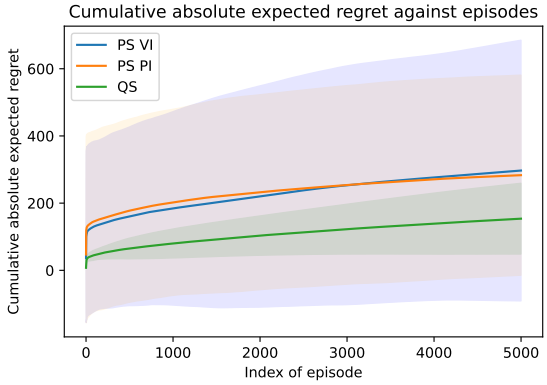


Figure 5.2: The plot of average cumulative absolute expected regret of posterior sampling with Greedy policy that incorporates value iteration and policy iteration respectively, and Q-value sampling for model 1. The shaded area is a 99% confidence interval of the algorithm represented by the same colour.

It is shown that Q-value sampling performs the best in this experiment since it has the narrowest confidence interval, and hence the smallest variance of cumulative absolute

expected regret, and converges quickly. Therefore, we may deduce that Q-value sampling is able to find the optimal policy of model 1 in the early stage of interaction, such that the agent has more opportunities to exploit its knowledge about the environment in later episodes to minimise the long-term cost. Posterior sampling with value iteration and policy iteration perform equally good, while it is noticed that value iteration gives rise to a wider confidence interval. A table of the average running time for the algorithms, apart from posterior sampling, is outlined in Table 5.6. Note that the average computation times for posterior sampling will be discussed later on, as we shall compare its performance when different choices of ϵ are used. We can see model-free algorithms that rely on a Q-table, including Q-learning, double Q-learning, and Q-value sampling, account for the top three shortest execution times. Note that the time used for choosing the optimal learning rate α does not count towards the computation time for Q-learning and double Q-learning here. The short running times may be explained by the nature of a model-free algorithm that it does not attempt to model the environment. Overall, we may deduce that Q-value sampling performs the best to reduce the expected total sum of cost during its interaction with the environment in model 1.

Direct policy search	VI	PI	Q-learning	Double Q-learning	Q-value sampling
20.00	43.85	73.62	5.475	6.316	11.95

Table 5.3: Average running time (in seconds) of generating 5000 episodes by algorithms apart from posterior sampling for model 1.

Next, we shall compare the performances of the Epsilon-greedy policy with direct policy search and value iteration when ϵ is increasing and when it is a constant. The plots of average cumulative absolute expected regret for each algorithm are shown in Figure 5.3 and 5.4. It appears that the cumulative absolute expected regret does not converge when ϵ is set to be 0.9 for both algorithms, which may result from the additional costs collected in an episode as the agent cannot just perform exploitation alone. It is worth noting that the y-axis of the plots have different scales, and direct policy search with increasing ϵ converges at a slightly smaller value than that of value iteration. We also spot that the confidence interval for value iteration is wider than direct policy search, implying value iteration tends to have a great variance in cumulative absolute expected regret.

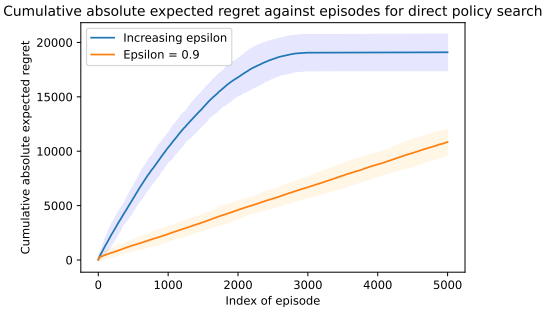


Figure 5.3: The plot of average cumulative absolute expected regret of direct policy search with Epsilon-greedy policy for increasing ϵ and $\epsilon = 0.9$ for model 1.

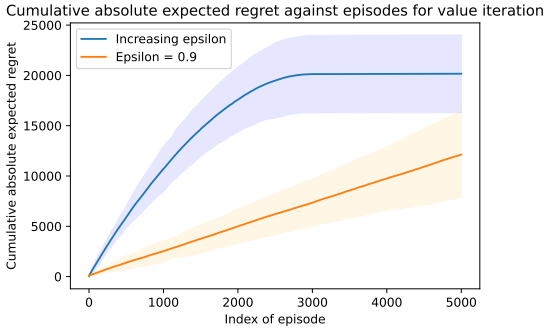
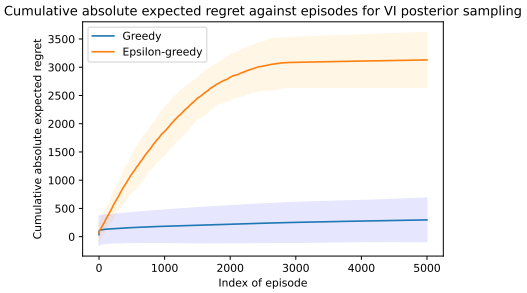


Figure 5.4: The plot of average cumulative absolute expected regret of value iteration with Epsilon-greedy policy for increasing ϵ and $\epsilon = 0.9$ for model 1.

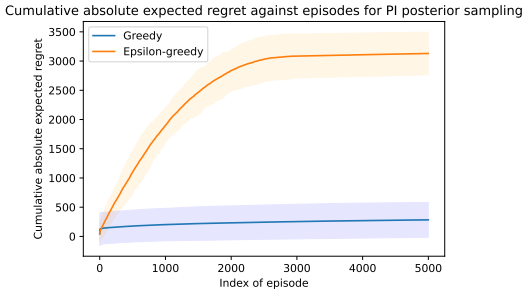
The impact of applying Epsilon-greedy and Greedy policy, with posterior sampling and the three dynamic programming algorithms, on the convergence in cumulative absolute expected regret shall also be evaluated, where the plots are illustrated in Figure 5.5a, 5.5b, and 5.5c. Since a large difference in the converged values is observed between the algorithms and the plots are more comprehensible with nonequivalent y-axes, the range of y values in Figure 5.5a is [0, 20000], while the range in the other two figures is [0, 3500]. It appears that convergence happens earlier when Greedy policy is used for all three algorithms, and the converged values are remarkably greater when the Epsilon-greedy policy is applied. Posterior sampling aims to update the prior distributions while exploiting its current knowledge about the environment, and the algorithm can employ the updating step to explore the model. As a result, combining posterior sampling and Epsilon-greedy gives the agent additional time for exploration, which may not be necessarily helpful and may contribute to a greater cumulative absolute expected regret.

The average computation times of generating a simulation of 5000 episodes for the posterior sampling algorithms are given in Table 5.4. Direct policy search outperforms the other two algorithms regardless of which policy is chosen, in terms of computation time, but we shall be aware that the cumulative absolute expected regret for this algorithm converges at a larger value. The convergences for value iteration and policy iteration are alike, while policy iteration takes approximately twice the time in running one simulation on average but has a narrower confidence interval. Despite the cumulative absolute expected regret converging at a relatively small value for posterior sampling, it is significantly inefficient compared to other examined learning algorithms. While other experimented algorithms need at most a minute to produce a simulation, posterior sampling with value iteration and policy iteration require at least 4 minutes to do so.

(a) Posterior sampling with direct policy search.



(b) Posterior sampling with value iteration.



(c) Posterior sampling with policy iteration.

Figure 5.5: The plots of average cumulative absolute expected regret of posterior sampling with Epsilon-greedy or Greedy policy.

Policy	Direct policy search	Value iteration	Policy iteration
Epsilon-greedy	57.72	248.6	579.4
Greedy	70.34	268.7	538.7

Table 5.4: Average running time (in seconds) of producing 5000 episodes by posterior sampling with different combinations of action selection methods for model 1.

Model 2

We decide to run experiments on algorithms that have satisfactory results on model 1, where the list of algorithms can be found in the Designs chapter. Since the cost distribution for action b depends on the transition of states, the optimal policy is different from that in model 1. We compute the optimal policy by the three dynamic programming algorithms using the true transition and cost functions, and the returned policies are outlined in Table 5.5. Note that the policies computed by value iteration and policy iteration are identical and are precisely the optimal policy, while the optimal action at state 0 differs between value iteration and direct policy search for model 2. It shows that direct policy search can be unstable, as we calculate the expected total cost only up to a finite number of steps ahead. Therefore, we shall not examine the performance

of this algorithm for model 2.

state	0	1	2	4	5	6
Direct policy search	a	c	b	a	a	a
Value iteration	c	c	b	a	a	a

Table 5.5: Optimal policies returned by direct policy search and value iteration (identical to the policy returned by policy iteration) for model 2.

A plot of the average cumulative absolute expected regret and the mean computation times of the experimented algorithms are given in Figure 5.6 and Table 5.6 respectively. It is observed that the convergence in cumulative absolute expected regret for each algorithm in models 1 and 2 are similar, although the converged values are greater for all experimented algorithms in model 2. This may be explained by the large difference in the mean cost associated with each action, where the agent receives a heavy penalty if action b is executed at states 4, 5, and 6, and hence gives rise to a greater cumulative absolute expected regret. Notice that the cumulative absolute expected regret for Q-value sampling does not converge, according to Figure 5.7. A possible reason is that a Q table captures information about executing an action at each state and we assume that each Q value has a fixed prior distribution for each (current state, action) pair in the Q-value sampling algorithm. Nonetheless, the assumption may not hold if the cost distribution of action b depends also on the successive state. Value iteration and posterior sampling with both value iteration and policy iteration are more efficient in model 2 compared to model 1, in terms of running time, where the time required to generate a simulation for each algorithm here is half that in model 1. We shall point out that Q-learning and double Q-learning require approximately the same amount of time to produce 5000 episodes as in model 1, even when the model becomes more complex.

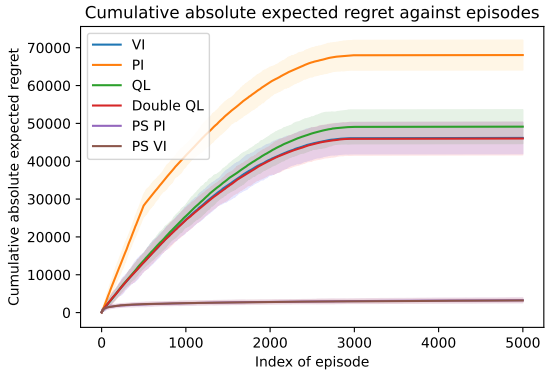


Figure 5.6: The plot of average cumulative absolute expected regret of the experimented algorithms against index of episode for model 2. The shaded region corresponds to a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

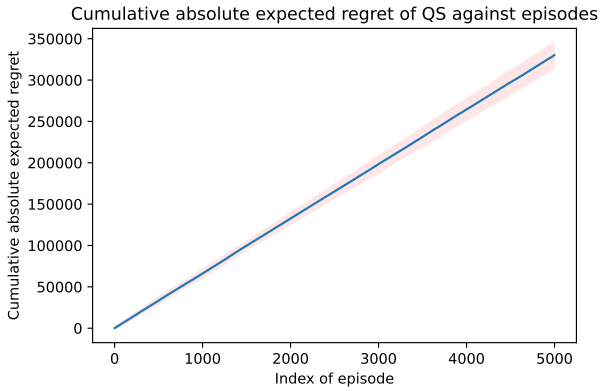


Figure 5.7: The plot of average cumulative absolute expected regret of Q-value sampling against the index of the episode for model 2. The shaded region corresponds to a 99% confidence interval of the cumulative absolute expected regret.

VI	PI	Q-learning	Double Q-learning	PS VI	PS PI	Q-value sampling
24.15	73.44	6.108	6.579	67.92	231.6	27.40

Table 5.6: Average running time (in seconds) of generating 5000 episodes by the experimented algorithms for model 2.

The running times for model-free algorithms that rely on Q values are the shortest among all the experimented algorithms. It is noticed that algorithms that incorporate policy iteration are the most inefficient. In this model, we notice that double Q-learning performs slightly better than Q-learning, as its cumulative absolute expected regret converges at a lower value, according to Figure 5.8. For posterior sampling, the performance for value iteration and policy iteration is highlighted in Figure 5.9. A similar result as in model 1 is observed, where both algorithms perform equally well and posterior sampling with value iteration contributes to a wider confidence interval. This experiment demonstrates a trade-off between computational efficiency and the learning agent's ability to determine the optimal policy. Double Q-learning is preferable if efficiency is more important, while posterior sampling with policy iteration is ideal otherwise.

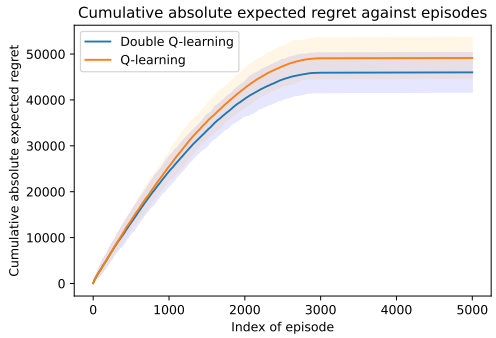


Figure 5.8: The plot of average cumulative absolute expected regret of Q-learning and double Q-learning against the index of the episode for model 2. The shaded region is a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

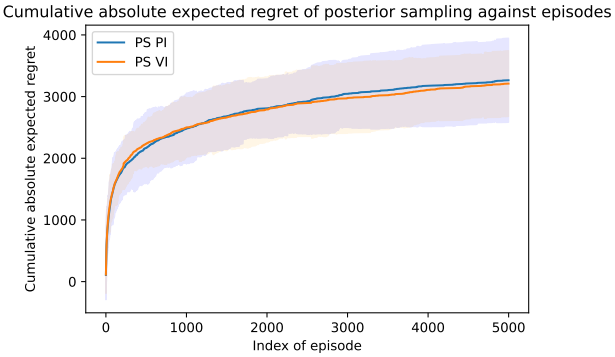


Figure 5.9: The plot of average cumulative absolute expected regret of posterior sampling against the index of the episode for model 2. The shaded region is a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

Model 3

For model 3, experiments are conducted on the same set of algorithms as in model 2. The optimal policies computed using the dynamic programming methods are provided in Table 5.7. The optimal actions in states 10 and 13 of direct policy search deviate from that of value iteration for this model, where the policy returned from value iteration is the true optimal policy. Note that the policies determined by value iteration and policy iteration are the same. We may deduce that direct policy search is unstable and it is unlikely to determine the optimal policy when cost distributions of the actions are not fixed.

state	0	1	2	4	5	6	7	8	9	10	11	12	13
Direct policy search	a	b	b	a	a	a	c	c	a	b	a	b	d
Value iteration	a	b	b	a	a	a	c	c	a	a	a	b	b

Table 5.7: Optimal policies returned by direct policy search and value iteration (identical to the policy returned by policy iteration) for model 3.

A plot of average cumulative absolute expected regret for each examined algorithm is demonstrated in Figure 5.10 and the running times are recorded in Table 5.9. Notice that the cumulative absolute expected regret of Double Q-learning struggles to converge,

which is due to an underestimation of the Q value at state 10 with action a, as shown in Table 5.8. Hence, the agent executes action c at state 10 whenever it exploits its knowledge, which contributes to a great sum of costs. We shall emphasise that the Bayesian approaches perform very well, as their cumulative absolute expected regrets converge at a fast rate and the converged values are much lower than other algorithms. A separate plot of the Bayesian algorithms is displayed in Figure 5.11 for readability.

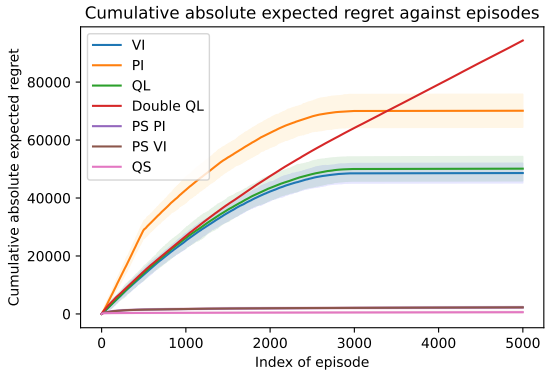


Figure 5.10: The plot of average cumulative absolute expected regret of the experimented algorithms against the index of the episode for model 3. The shaded region is a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

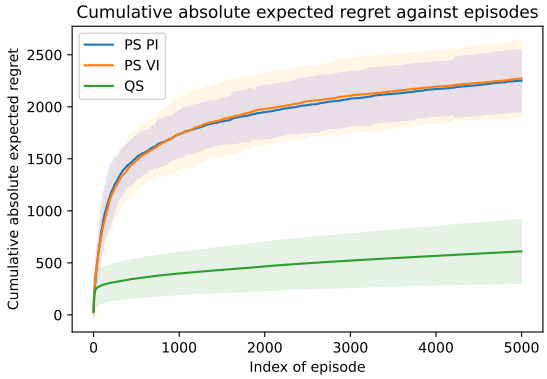


Figure 5.11: The plot of average cumulative absolute expected regret of posterior sampling and Q-value sampling against the index of the episode for model 3. The shaded region is a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

Notice that Q-value sampling outperforms other algorithms in this experiment, while the cost distributions of the model only depend on the current state, so we may deduce that Q-value sampling is sensitive to how the cost distributions are designed. The algorithm also takes a short time to generate a simulation, so we may conclude Q-value sampling has the best performance in this experiment. On the other hand, it is shown in Table 5.9 that value iteration is more efficient than policy iteration in general; Q-learning and double Q-learning, again, account for algorithms with the quickest computation times. Even though the size of the state space in model 3 is twice that of model 2 and the action space is bigger, there are insignificant differences in the efficiency of Q-learning and double Q-learning. In summary, this experiment demonstrates that Q-value sampling is a sensible algorithm to use, in terms of both efficiency and convergence in cumulative absolute expected regret, when the learning environment is large and the cost distributions only depend on the current state and action.

State/action	a	b	c	d
0	-24.64	0	-25.13	0
1	-25.46	-4.022	-25.08	0
2	0	-4.462	-27.97	0
3	0	0	0	0
4	-27.40	-104.4	0	0
5	-33.60	-81.79	0	0
6	-37.13	-102.7	0	0
7	0	-99.42	-5.564	0
8	0	0	-4.812	0
9	-38.94	0	-46.96	-37.26
10	-105.1	-109.0	-24.59	0
11	-5.744	-12.49	-25.45	-14.64
12	-51.07	-29.61	0	-58.97
13	-40.17	-23.35	-29.46	-35.76

Table 5.8: A Q table computed in double Q-learning for model 3.

VI	PI	Q-learning	Double Q-learning	PS VI	PS PI	Q-value sampling
152.4	257.0	8.306	7.240	354.1	1034	16.88

Table 5.9: Average running time (in seconds) of generating 5000 episodes by the experimented algorithms for model 3.

Model 4

The objective of performing an experiment on model 4 is to examine the efficiency of the learning algorithms when the environment has both a large state-action space and a complex design of cost distributions that depend on the current state, action executed, and the successive state. The optimal policy computed by value iteration aligns with that by policy iteration and it is indeed the true optimal policy for model 4. The optimal policies returned by direct policy search and value iteration are outlined in Table 5.10. It is noticed that the policy returned by direct policy search deviates from the true optimal policy at states 2, 4, 9, and 10. Since direct policy search does not accurately determine the optimal policy in model 3, which has the same transition function as in model 4, it is expected that the algorithm also cannot evaluate the correct optimal policy here. We may conclude that direct policy search is capable of finding the optimal policy only when the cost distributions of action are fixed and independent of the current and successive states.

state	0	1	2	4	5	6	7	8	9	10	11	12	13
Direct policy search	a	b	b	a	a	a	c	c	a	b	a	b	d
Value iteration	a	b	c	b	a	a	c	c	d	a	a	b	d

Table 5.10: Optimal policies returned by direct policy search and value iteration (identical to the policy returned by policy iteration) for model 4.

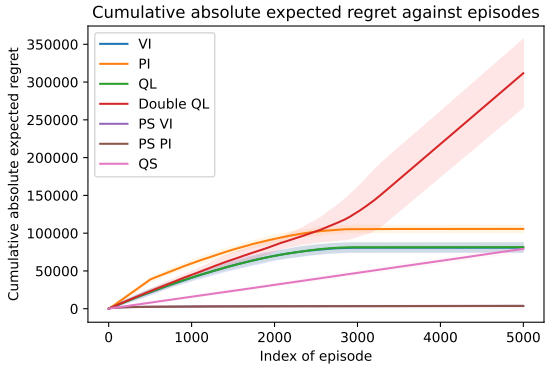


Figure 5.12: The plot of average cumulative absolute expected regret against the index of the episode for model 4. The shaded region is a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

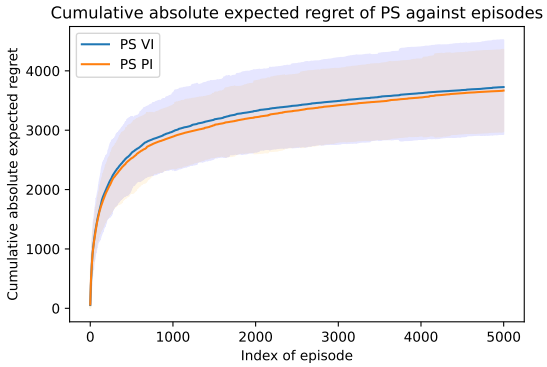


Figure 5.13: The plot of average cumulative absolute expected regret of posterior sampling against the index of the episode for model 4. The shaded region is a 99% confidence interval of the cumulative absolute expected regret of the algorithm represented by the same colour.

The plot of cumulative absolute expected regret for the experimented learning algorithms is displayed in Figure 5.12. Apart from double Q-learning and Q-value sampling, the result is similar to that in previous experiments. The cumulative absolute expected regret for most of the algorithms, apart from posterior sampling, converges at a higher value than in previous experiments. Since the cost distributions depend on the successive state and the difference between the mean costs for some actions is large in model 4, the agent may receive great punishment when a random action is chosen during exploration, which contributes to a high cumulative absolute expected regret. Convergence of the cumulative absolute expected regret for posterior sampling is plotted in Figure 5.13. Recall that model 2 has a smaller environment with smaller state and action spaces and the cost distributions are dependent on the successive state. It is noticed that a similar result is obtained in the experiment for model 2, where the cumulative absolute expected regrets for posterior sampling converge around 3000 and a 99% confidence interval for

value iteration is slightly wider than that for policy iteration. We may conclude that the size of environment does not have a great impact on the performance of posterior sampling. Overall, posterior sampling performs the best in terms of convergence in cumulative absolute expected regret in the experiment for model 4.

Figure 5.12 shows that cumulative absolute expected regrets for Q-value sampling and double Q-learning do not converge, implying the agent in both algorithms fails to determine the optimal policy for model 4. The cumulative absolute expected regret for Q-value sampling diverges in the experiment for models 2 and 4, where they are designed to have cost distributions that depend on the action, the current and successive states. We may deduce that Q-value sampling should be used when the cost distributions are only influenced by the current state and action, as a fixed prior is assigned to the Q value for each current state-action pair. A Q table returned from double Q-learning is given in Table 5.11. The best actions to execute in states 0, 1, 2, 4, 9, 10, and 11 decided by the double Q-learning agent differ from that in the optimal policy, due to an underestimation of the Q values. A similar issue is found in the experiment for model 3, while the cumulative absolute expected regret diverges at a quicker rate in this experiment for model 4. It is plausible that the learning agent finds it more challenging to spot the correct optimal policy when the cost distributions depend on the successive state in a large environment, as a Q value only captures information about the expected total cost given the current state and action and the agent may lack exploring all state-action pairs under a limited time frame. Also notice that double Q-learning gives rise to a wide 99% confidence interval, implying a large variance in its cumulative absolute expected regret.

State/action	a	b	c	d
0	-133.2	0	-49.84	0
1	-13.98	-94.74	-75.37	0
2	0	-99.28	-135.6	0
3	0	0	0	0
4	-187.5	-215.7	0	0
5	-102.4	-115.5	0	0
6	-36.06	-182.3	0	0
7	0	-187.7	-5.114	0
8	0	0	-5.185	0
9	-161.5	0	-168.5	-173.5
10	-204.9	-210.1	-25.72	0
11	-5.737	-4.459	-23.73	-51.59
12	-136.9	-34.85	0	-168.2
13	-39.58	-55.04	-65.23	-35.41

Table 5.11: A Q table computed in double Q-learning for model 4.

The average computation time of producing a simulation for each experimented learning algorithm is provided in Table 5.12. The model-free algorithms, including Q-learning,

double Q-learning, and Q-value sampling, are the most efficient. On the other hand, the running time for posterior sampling is significantly longer, especially with policy iteration. The long running time implies the algorithm may take a lot of iterations to achieve convergence in value functions and policy in a large environment with complex settings of cost distributions. The long computation time for policy iteration with Epsilon-greedy may be due to the same reasoning. This experiment again illustrates a trade-off between computation time and the ability of an agent to minimise long-term costs. Posterior sampling outperforms other algorithms in reducing the expected total cost, as the cumulative expected regret converges at a much smaller value, while it is the least efficient algorithm. Q-learning accounts for the shortest running time while its converged cumulative absolute expected regret value is more than 10 times that for posterior sampling.

VI	PI	Q-learning	Double Q-learning	PS VI	PS PI	Q-value sampling
330.8	387.4	8.756	7.613	670.4	1178	18.82

Table 5.12: Average running time (in seconds) for generating a simulation of 5000 episodes of the experimented algorithms for model 4.

Discussions

To summarise the results for all four experiments, Q-learning and double Q-learning are the most efficient among all the algorithms considered, where they take less than 10 seconds to simulate 5000 episodes on average for the four models. The efficiency of model-free algorithms may be explained by the fact that the learning agent is not required to model the environment by estimating the transition function and cost distributions, while it aims to optimise the policy by observing the collected costs. The performance of Q-learning is consistent throughout the experiments and the uniform overestimation of Q values does not affect the algorithm to return an accurate optimal policy. In the experiment on models 3 and 4 for double Q-learning, however, we notice the algorithm fails to determine the optimal policy due to underestimation of the Q values. As a result, we may deduce that double Q-learning does not appear to be a more advantageous or preferable algorithm over Q-learning.

The experiments for model-based algorithms show that the cumulative absolute expected regret converges at a lower value for value iteration, relative to policy iteration. Moreover, the computation time for value iteration is approximately half of that for policy iteration, so we may deduce value iteration performs better in the experiments. Note that value iteration tends to have a large variance and gives a wide confidence interval on the cumulative absolute expected regret. The instability of direct policy search is demonstrated through failures in computing the optimal policy using the true transition function and cost distributions. Recall from the Methods chapter that the algorithm computes the optimal policy by searching all possible sequences of states the agent may visit in a finite number of time steps. However, a small maximum number of time steps

is usually used in the algorithm due to a lack of computation power, which may lead to an inaccurate result. Hence, we may not recommend applying direct policy search. On the other hand, it is preferable to use a growing ϵ over a constant ϵ in Epsilon-greedy policy, as a constant gives rise to excessive exploration in later episodes and leads to a divergent cumulative absolute expected regret.

Among all the Bayesian algorithms presented in the Methods chapter, we notice that posterior sampling has stable and satisfactory performance, as it assigns a prior cost distribution to every (current state, action, successive state) pair, and hence it can estimate the mean costs accurately regardless of what the cost distribution depends on. Results of the experiments support that posterior sampling should incorporate the Greedy policy, where we observe cumulative absolute expected regret of posterior sampling with Epsilon-greedy policy converges at a much larger value than with Greedy policy. It is believed that Epsilon-greedy forces the agent to perform more explorations than necessary in posterior sampling, which causes the agent to achieve a greater expected total cost at the end of an episode. The convergence in cumulative absolute expected regret of posterior sampling when it is combined with value iteration is almost identical to that when it incorporates policy iteration, although policy iteration causes a narrower confidence interval due to a smaller variance. Q-value sampling outperforms other algorithms in two of the experiments, considering its prompt execution and quick rate of convergence in cumulative absolute expected regret. Nonetheless, we observe from the experiment on models 2 and 4 that the cumulative absolute expected regret of Q-value sampling diverges as the index of episode increases. Therefore, the reader shall be aware that Q-value sampling may not be applied if the cost distributions are known to be dependent on the successive state, as the Q values sampled from the prior distributions are likely to be biased.

6 Conclusions and Recommendations

Summary

The thesis presented an approach for problem resolution in a telecommunications setting with the use of reinforcement learning and the performances of several action selection algorithms. A description of the telecommunications scenario is given in the Introduction chapter, where we aim to minimise the long-term cost while deciding a sequence of partial resolutions to fix a fault. We introduce the multi-armed bandits problems as a simplified reinforcement learning problem, where the environment has a single state, and evaluate why multi-armed bandits may not be preferable in the BT scenario. It is mentioned in the Background chapter that a Markov decision process captures information about the environment of a reinforcement learning problem, such as the transition function and the set of actions. As the telecommunications scenario can be modelled as a Stochastic Shortest Path problem, which is a type of Markov decision process, the learning agent has knowledge about the state space, action space, and the natural stopping point of interaction with the environment. The main learning goal for the agent is to determine the best action to execute in each non-goal state that reduces the expected total cost, through exploring the environment by trial and error and choosing actions based on its knowledge about the cost distributions associated with actions. A variety of action selection strategies is introduced in the Methods chapter, including model-based algorithms such as value iteration and posterior sampling, and model-free algorithms such as Q-learning. The pros and cons of using model-based and model-free algorithms are discussed in the Background chapter. The application of dynamic programming to speed up the computation of optimal policy is highlighted in the Methods chapter, where we also attempt to implement the direct policy search algorithm.

We conduct experiments on four Stochastic Shortest Path models to evaluate the performances of the reinforcement learning algorithms presented in the Methods chapter. We consider a model with a fixed cost distribution for each action, one with cost distributions that depend on the current state, and the other two with cost distributions dependent on both the current and successive states. It is shown in the Results chapter that direct policy search is unstable in computing the optimal policy compared to value iteration and policy iteration, unless the cost distributions are fixed. In direct policy search, the agent determines the best action to take for each non-terminal state by considering the expected total cost in a small number of time steps ahead, so the instability may be due to a lack of computational power to consider a larger number of time steps forwards. Hence, direct policy search may not be an ideal algorithm to solve Markov decision process problems, especially when it is known that the cost distributions are not fixed. The computation times of value iteration are shorter than that of policy itera-

tion in all four experiments. The cumulative absolute expected regret of policy iteration with Epsilon-greedy converges at a greater value than most of the learning algorithms, as heavy punishments may be received due to the agent's active exploration at the beginning of interaction with the environment. An advantage of using policy iteration over value iteration is that policy iteration gives rise to a smaller variance in its cumulative absolute expected regret and a narrower confidence interval.

The results of the experiments illustrate that model-free learning algorithms are more efficient and the size of learning environment has an insignificant impact on computation times, as these algorithms do not involve modelling the transition dynamics of the environment. Among all model-free algorithms presented in this thesis, the cumulative absolute expected regret for Q-learning converges in all experiments, while double Q-learning and Q-value sampling perform poorly when the learning takes place in a large environment and when the cost distributions depend on the successive state, respectively. It is suggested in the Results chapter that Q-value sampling may be the best algorithm to use if the cost distributions are known to be independent of the successive state, as the cumulative absolute expected regret converges very quickly and hence, the agent can effectively minimise the long-term cost. Nonetheless, we often have very limited information about the environment, especially the setting of cost distributions, in reality.

Posterior sampling may be the most preferable reinforcement learning algorithm to apply in most situations, as the experiments demonstrate that the cumulative absolute expected regret quickly converges at a relatively low value in all four models. It follows that the algorithm has stable performance and the size of environment or the design of cost distributions have a minor influence on the agent's ability to reduce the expected total cost. Also, the rapid convergence reflects that the agent acquires knowledge about the dynamics of the environment with a little amount of interactions. However, the application of posterior sampling comes with a trade-off between computational efficiency and minimising the long-term costs. We observe from the experiments that posterior sampling with value iteration and policy iteration account for the longest running times. Although posterior sampling incorporating policy iteration is less efficient than value iteration, value iteration often contributes to a wider confidence interval of cumulative absolute expected regret due to large variance.

In reality, we can only select one reinforcement learning algorithm to interact with the environment in the telecommunications setting, as real-time interaction only takes place once. Furthermore, we are not able to compute the true optimal policy in practice, so we shall be cautious about the decisions made by the agent. For example, further investigations may be needed if the agent tends to choose actions associated with low costs, since the agent may intend to minimise instant cost instead of the long-term cost. Since model-based learning algorithms attempt to model the transition function and cost distributions, we are able to use this information to verify if the agent determines the optimal policy, so it may be more desirable to implement a model-based algorithm. Suppose that we have no information about how the cost distributions are designed, it is recommended to use posterior sampling with value iteration. Although posterior sampling is the least efficient learning algorithm, it is insensitive to the design of the

environment and is capable of reducing the expected total cost. Despite there being a trade-off between computation time and performance of the algorithm, the ability to determine the optimal policy quickly and minimise the long-term cost may be prioritised in a telecommunications setting. Moreover, the inefficiency of the algorithm may be improved by using machines with greater computing power.

Recommendations for Future Work

In this thesis, we discuss the overestimation and underestimation issue in Q-learning and double Q-learning respectively. It is suggested in (Arulkumaran et al., 2017) that combining deep learning and Q-learning may improve the stability of the algorithm, where the combination is called deep Q-network. According to Van Hasselt et al. (2016), a deep Q-network is a multi-layered neural network that returns an array of Q values given a state and the network is updated using the responses from the environment. Unlike Q-learning, the network is updated with observed transitions that are stored a while ago during the interactions and are sampled uniformly from the memory storage, instead of using immediate observations from the environment. This enhances the computational efficiency as fewer interactions are needed for training and smaller variances of the updated values are obtained, as discussed in (Arulkumaran et al., 2017). We assume the state space and action space are of a single dimension in this thesis, while deep Q-network allows these spaces to be multi-dimensional. It might be of interest to compare the efficiency of deep Q-network with the algorithms presented in the Methods chapter and investigate if the Q values predicted by deep Q-network are biased.

The assumptions of the telecommunications scenario ensure that the transition function and cost distributions are independent of the round in an episode of interactions. Therefore, we can model the scenario as a Stochastic Shortest Path problem, which is a type of stationary Markov decision process. This project may be extended by applying reinforcement learning to non-stationary Markov decision processes, where the transition probabilities or cost distributions may change over time in an episode. An illustration in the telecommunications setting is that waiting for more information about the status of fault may cost more at a later stage of fault resolution, given that the fault has not been fixed for a long while. The literature (Mao et al., 2021) proposed a model-free algorithm called Restarted Q-Learning with Upper Confidence Bounds that solves for the optimal policy in a non-stationary Markov decision process. On the other hand, the authors in (Choi et al., 1999) suggested that a non-stationary environment may be modelled by a set of distinct Markov decision processes. Suppose the non-stationary environment changes its transition dynamics and cost distributions at some unknown time steps, we may simulate the environment by a stationary Markov decision process before each change. As the process involves modelling the environment, such an approach motivates the use of model-based algorithms.

Alternatively, further research may be done on the applications of reinforcement learning for partially observable Markov decision processes. The process is a generalised Markov decision process, while the assumption that the learning agent always observes

the current state directly in the environment is removed. An example in the BT scenario is that we may observe a power cut as a fault while the reason of the fault is not known immediately, or we are unable to know the resulting status of the fault immediately after taking an action. The paper (Jaakkola et al., 1994) proposed a model-based algorithm, which is a revised version of policy iteration, for partially observable Markov decision processes. The algorithm first estimates Q values via Monte Carlo method, then devises a policy that contributes to the lowest expected total cost based on the estimated Q values. The optimal policy for a partially observable Markov decision process may also be determined using deep reinforcement learning, which incorporates deep learning in reinforcement learning algorithms, according to (Meng et al., 2021).

7 Appendix

The codes for the experiments can be found in the github repository Thesis-Reinforcement-learning-for-fault-resolution-in-a-telecommunications-setting.

References

Shipra Agrawal and Randy Jia. Optimistic posterior sampling for reinforcement learning: worst-case regret bounds. *Advances in Neural Information Processing Systems*, 30, 2017.

Oguzhan Alagoz, Heather Hsu, Andrew J Schaefer, and Mark S Roberts. Markov decision processes: a tool for sequential decision making under uncertainty. *Medical Decision Making*, 30(4):474–483, 2010.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

Nikolay Atanasov. Ece276b: Planning learning in robotics lecture 9: Infinite horizon problems and stochastic shortest path, Spring 2020.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020.

Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

Dimitri P Bertsekas and Huizhen Yu. Stochastic shortest path problems under weak conditions. *Lab. for Information and Decision Systems Report LIDS-P-2909, MIT*, 2013.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2017.

Samuel Choi, Dit-Yan Yeung, and Nevin Zhang. An environment model for nonstationary reinforcement learning. *Advances in neural information processing systems*, 12, 1999.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*, pages 359–397. MIT press, 2022.

Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. *Aaai/iaai*, 1998:761–768, 1998.

M Milani Fard and Joelle Pineau. Non-deterministic policies in markovian decision processes. *Journal of Artificial Intelligence Research*, 40:1–24, 2011.

Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

Tommi Jaakkola, Satinder Singh, and Michael Jordan. Reinforcement learning algorithm for partially observable markov decision problems. *Advances in neural information processing systems*, 7, 1994.

Mehdi Jafarnia-Jahromi, Liyu Chen, Rahul Jain, and Haipeng Luo. Online learning for stochastic shortest path model via posterior sampling. *arXiv preprint arXiv:2106.05335*, 2021.

Mohsen Jamali. Learning to solve stochastic shortest path problems. *Rapport Technique, Sharif University of Technology*, 2006.

Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE access*, 7: 133653–133667, 2019.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Jiayu Lin. On the dirichlet distribution. *Department of Mathematics and Statistics, Queens University*, 2016.

Jianhua Liu, Xin Wang, Shigen Shen, Guangxue Yue, Shui Yu, and Minglu Li. A bayesian q-learning game for dependable task offloading against ddos attacks in sensor edge cloud. *IEEE Internet of Things Journal*, 8(9):7546–7561, 2020.

Weichao Mao, Kaiqing Zhang, Ruihao Zhu, David Simchi-Levi, and Tamer Basar. Near-optimal model-free reinforcement learning in non-stationary episodic mdps. In *International Conference on Machine Learning*, pages 7447–7458. PMLR, 2021.

Lingheng Meng, Rob Gorbet, and Dana Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5619–5626. IEEE, 2021.

Kevin P Murphy. Conjugate bayesian analysis of the gaussian distribution. *def*, 1($2\sigma 2$): 16, 2007.

Martijn van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement learning*, pages 3–42. Springer, 2012.

Constantin-Valentin Pal and Florin Leon. Brief survey of model-based reinforcement learning techniques. In *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 92–97. IEEE, 2020.

Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for markov decision problem. In *AAAI'96: Workshop on Structural Issues in Planning and Temporal Reasoning*. Citeseer, 1996.

Zhizhou Ren, Guangxiang Zhu, Hao Hu, Beining Han, Jianglun Chen, and Chongjie Zhang. On the estimation bias in double q-learning. *Advances in Neural Information Processing Systems*, 34:10246–10259, 2021.

Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

Linn I Sennott. The convergence of value iteration in average cost markov decision chains. *Operations research letters*, 19(1):11–16, 1996.

Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):4–7, 2019.

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

Jean Tarbouriech, Runlong Zhou, Simon S Du, Matteo Pirotta, Michal Valko, and Alessandro Lazaric. Stochastic shortest path: Minimax, parameter-free and towards horizon-free regret. *Advances in Neural Information Processing Systems*, 34:6843–6855, 2021.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Zihan Zhang and Xiangyang Ji. Regret minimization for reinforcement learning by evaluating the optimal bias function. *Advances in Neural Information Processing Systems*, 32, 2019.