Minimal Exploration in Episodic Reinforcement Learning

ARDHENDU SHEKHAR TRIPATHI
Minimal Exploration in Episodic Reinforcement Learning

ARDHENDU SHEKHAR TRIPATHI
Abstract

Exploration-exploitation trade-off is a fundamental dilemma that reinforcement learning algorithms face. This dilemma is also central to the design of various state of the art bandit algorithms. We take inspiration from these algorithms and try to design reinforcement learning algorithms in an episodic setting. In this work, we develop two algorithms which are based on the principle of optimism in face of uncertainty to minimize exploration. The idea is that the agent follows the optimal policy for a surrogate model, named optimistic model, which is close enough to the former but leads to a higher long-term reward. We show extensively through experiments on synthetic toy MDP’s that the performance of our algorithms is in line (even better in the case where the reward dynamics are known) with the algorithms based on the Bayesian treatment of the problem and other algorithms based on the optimism in face of uncertainty principle. The algorithms suggested in this thesis trump the Bayesian algorithms in terms of the variance of the regret achieved by the algorithms over multiple runs. Another contribution is the derivation of several regret lower bounds, such as a problem-specific (both, asymptotic and non-asymptotic) and a minimax regret lower bound, for any uniformly good algorithm in an episodic setting.
Sammanfattning

Avvägningen mellan upptäckande och utnyttjande är ett grundläggande dilemma som övervakade inlärningsalgoritmer handskas med. Det här dilemmat är också centralt i designen av diverse toppmoderna bandit-algoritmer. Vi inspireras av dessa algoritmer och försöker utforma övervakade inlärningsalgoritmer i en episodisk miljö. I det här arbetet utvecklar vi två algoritmer som är baserade på principen om optimism vid osäkerhet för att minimera upptäckande. Idén är att agenten följer den optimala policyn för en surrogatmodell som kallas optimistisk modell, som är tillräckligt nära ursprungsmodellen men leder till en högre långsiktig belöning. Vi visar utförligt genom experiment på syntetiska leksaks-MDP att algoritmernas prestanda är i linje med (till och med bättre när belöningsdynamiken är känd) algoritmerna grundade på den bayesiska behandlingen av problemet och andra algoritmer baserade på optimism vid osäkerhet. Algoritmerna som föreslås i den här avhandlingen presterar bättre än de bayesiska algoritmerna i varians av den ånger som uppnås av algoritmerna över många körningar. Ett annat bidrag är härledningen av flera nedre gränser, såsom en problem-specifikt nedre gränser (både asymptotisk och icke-asymptotisk) och en nedre gränser enligt minmax-principen, för en godtycklig uniformt god algoritm i en episodisk miljö.
Contents

1 Introduction .................................................. 1
   1.1 Objectives ............................................. 2
   1.2 Outline of the thesis .................................. 4

2 Theoretical Background .................................. 5
   2.1 Problem Formulation ................................... 5
   2.2 A few definitions ...................................... 6
   2.3 Dynamic Programming in MDP’s ....................... 7
   2.4 Performance Measures for RL ......................... 8
     2.4.1 Regret ........................................... 8
     2.4.2 Sample Complexity ................................ 9
   2.5 The Exploration-Exploitation tradeoff ............... 10
   2.6 Kullback–Leibler divergence ......................... 11

3 Related Work ................................................ 12
   3.1 Algorithms based on the OFU principle .............. 12
   3.2 Bayesian treatment of the problem ................. 14

4 Regret Lower Bounds ..................................... 15
   4.1 Notations .............................................. 15
   4.2 Prerequisites ......................................... 16
   4.3 Regret in case of an episodic RL problem ........... 16
   4.4 The problem specific regret lower bound .......... 18
   4.5 Minimax regret lower bound ......................... 22
   4.6 Non-asymptotic regret lower bound ................. 24

5 Algorithms .................................................. 29
   5.1 Unknown Transition Dynamics ......................... 31
      5.1.1 Adaptive Index Policy ........................... 31
      5.1.2 KL-UCB++ inspired optimism ................... 35
5.2 Unknown Rewards .................................. 37

6 Existing Algorithms 38
   6.1 KL-UCRL algorithm .............................. 38
   6.2 UCBVI-BF algorithm ............................. 40
   6.3 PSRL algorithm ................................ 42

7 Results 43
   7.1 Regret ............................................. 44
      7.1.1 River Swim environment ..................... 44
      7.1.2 Six Arm environment ......................... 45
      7.1.3 Random environment ......................... 48
   7.2 Runtime .......................................... 49
   7.3 Discussion ....................................... 51

8 Conclusion 54
   8.1 Future Work ...................................... 55
   8.2 Ethical and Societal aspects ................... 56

Bibliography 57
Chapter 1

Introduction

Reinforcement Learning (RL) is goal oriented learning based on interaction with the environment. RL is said to be the hope of true artificial intelligence. And it is rightly said so, because the potential that RL possesses is immense. The state of the art in RL is growing rapidly and finding its way in applications like driverless cars, self navigating vacuum cleaners, scheduling of elevators are all applications of Reinforcement learning.

To give an intuition about the general RL framework, let us start with a simple example. If you have a pet at home, you may have used this technique with your pet. A clicker (or whistle) is a technique to let your pet know some treat is just about to get served! This is essentially reinforcing your pet to practice good behavior. You click the clicker and follow up with a treat. And with time, your pet gets accustomed to this sound and responds every time it hears the click sound. With this technique, you can train your pet to do good deeds when required. Now let’s make these replacements in the example:

- The pet becomes the artificial agent
- The treat becomes the reward function
- The good behavior is the performed action

The above example explains what reinforcement learning looks like. This is actually a classic example of reinforcement learning. To apply this on an artificial agent, we have a kind of a feedback loop to reinforce your agent. It rewards when the actions performed is right and punishes in-case it was wrong. Basically what we have in our kitty is:
• an internal state, which is maintained by the agent to learn about the environment

• a reward function, which is used to train your agent how to behave

• an environment, which is a scenario the agent has to face

• an action, which is done by the agent in the environment

• and last but not the least, an agent which does all the deeds!

Figure 1.1: A general RL loop (Source: UTCS RL Reading Group)

Figure 1.1 shows how an RL agent interacts with an environment and how the environment dynamics effect the decision taken by the agent. The figure also is symbolic of the many parameters on which the learning by the agent may depend on.

1.1 Objectives

All RL algorithms generally work by balancing the exploration-exploitation trade-off. The exploration-exploitation trade-off is a fundamental dilemma whenever you learn about the world by trying things out. The dilemma is between choosing what you know and getting something close to what you expect (‘exploitation’) and choosing something you aren’t sure about and possibly learning more (‘exploration’). So in case of RL algorithms this transforms to whether we keep playing the
action (in a particular state) which we know are the best upto now ('exploitation') or we explore more (play an action not played much till now). This is a dilemma also faced in the domain of Bandit optimization and the principle of Optimism in the Face of Uncertainty (OFU) is used by many optimal algorithms (in terms of attained regret lower bounds) to mitigate it. Basically, we would like to tap upon the exploration-exploitation trade off optimally to gain performance improvements in our algorithms. In particular, our focus would be on the episodic class of MDPs where the time horizon is divided into episodes of fixed lengths. The formulation would fall in the framework as described in [3].

Real-world Reinforcement Learning (RL) problems often concern systems with large state and action spaces, which makes the design of efficient algorithms extremely challenging. In online RL problems with undiscounted reward, regret lower bounds typically scale as a function of \( S, A \) and \( T \) where they denote the sizes of the state and action spaces and the time horizon, respectively. Hence with large state and action spaces, it is essential to identify and exploit any possible structure existing in the system dynamics and reward function so as to minimize exploration phases and in turn reduce regret to reasonable values. Early work in the direction includes the work by Graves et. al. [10] and Burnetas et. al. [5] where they discuss and design adaptive policies for MDP’s however those algorithms are computationally infeasible. Recently with the advent of Deep Q-Networks (DQN) [14], it has been proven that deep neural network can empower RL to directly deal with high dimensional states like images, thanks to techniques used in DQN. However, there remains a gap between the performance of the network and the time and resource required to train it. Hence, it could be pivotal to exploit structure inherent in the problem to speed up the learning. Ok et. al. [15] address reinforcement learning problems with finite state and action spaces where the underlying MDP has some known structure (Lipschitz in their case) that could be potentially exploited to minimize the exploration of suboptimal (state, action) pairs.

At a high level, the focus of the thesis would be to design a OFU based algorithm that gives comparable or better performance (in some cases) in terms of regret, in an episodic setting.
1.2 Outline of the thesis

The remaining parts of the thesis are structured as follows:

- Chapter 2 provides the reader with the theoretical background that is necessary to understand the details of the thesis.
- Chapter 3 describes some related research and sheds light on the current state of the art.
- Chapter 4 derives the minimax and the problem-specific regret lower bounds for the episodic RL problem.
- Chapter 5 describes the algorithms based on the optimism in face of opportunity designed in the thesis.
- Chapter 6 describes the algorithms against which our algorithm would be compared.
- Chapter 7 presents some results from the numerical experiments for comparison of the proposed algorithm against other methods.
- Finally, Chapter 8 concludes the thesis and discusses possible suggestions for future work.
Chapter 2

Theoretical Background

This chapter provides a theoretical background on which we build upon in the subsequent chapters. Markov Decision Processes (MDP) sit at the very core of RL. In its basic formulation, a Markov Decision Process consists of a set of states, of actions, a stochastic transition function and a stochastic reward function. An interaction consists in observing a current state, choosing an action to play, from which we move to the next state according to the transition function and incur the corresponding reward. This interaction repeats and produces a trajectory of states, actions, rewards. In an MDP, from a RL perspective, the transition and reward functions are unknown, but trajectories are fully observed. The aim of a RL agent is to choose the sequence of actions so as to maximize the cumulated reward (or whatever criteria we need to maximize - this depends on the class of problem we are tackling). The aim of RL algorithms is to learn a Markovian Deterministic (MD) policy $\pi$ maximizing (over all possible policies) given the data.

There are three classes of RL problems namely episodic, discounted and ergodic. Out of these, we are interested in the episodic class of problems and the theory covered henceforth are based on that.

2.1 Problem Formulation

In an episodic RL problem, the agent acts in $K$ episodes of fixed length $H$. To formulate the setup, we consider a MDP $M = (S, A, p, q)$ with finite state space $S$, and action space $A$. $p$ and $q$ denote the transition kernel and the reward distribution of the MDP. We denote by $S$ and $A$ the cardinality of the state and the action space, respectively. The
The probability of going from state $s$ to state $s'$ ($s, s' \in S$) when taking an action $a$ ($a \in A$) is denoted by $p(s'|s, a)$. Let $\Delta_{s,a}$ be the support of the transition probability vector for state $s$ and action $a$ i.e. $p(\cdot|s, a) \in \Delta_{s,a}$. Besides, at a particular time slot $h$, the agent gets a random reward $R_h(s, a)$ (depends on the the state $s$, action $a$ and sometimes also on the index $h$) drawn from the distribution $q_h(\cdot|s, a)$ which is bounded between $[0, 1]$ and has a mean $r_h(s, a)$. Here $q_h(\cdot|s, a) \in \Theta_{s,a}$ which is the support of the reward distribution for state $s$ and action $a$.

The aim of the agent is to choose the sequence of actions so as to maximize the cumulated reward over the $K$ episodes (or whatever criteria we need to maximize - this depends on the class of problem we are tackling). The aim of RL algorithms is to learn a Markovian Deterministic (MD) policy $\pi$ maximizing (over all possible policies) given the data.

### 2.2 A few definitions

**Definition 2.1.** A policy: Written as $\pi$, it describes a way of acting. It is a function that takes in a state $s$ and a time slot $h$ as input and returns an action in state $s$ and slot $h$.

The optimal policy $\pi^*$ is the policy which leads to the highest expected cumulated reward starting in state $s$ in slot $h$.

**Definition 2.2.** Value of a policy: the value of a policy $\pi$ in state $s$ is the expected reward collected in an episode when starting in state $s$. The value function for a fixed time domain $H$ is given by:

$$V^\pi_H(s) = E^\pi\left[\sum_{h=1}^{H} r_h(S_h, A_h) + r_H(S_H) | S_0 = s\right].$$

Here, $r_h(s, a)$ is the random reward collected by the agent when it executes the action $a$ at state $s$ in slot $h$. It is important to note here that the action $A_h$ in state $S_h$ at slot $h$ are random variables and are decided based on the policy $\pi$. The state evolution happens based on the action played and the transition dynamics of the system. Also, $r_H(s)$ denotes the reward of being at state $s$ at slot $H$. The takeaway is that the value $V^\pi_H(s)$ is actually the expected reward when starting in state $s$ over a time domain of $H$. The definition can be easily adapted for infinite
time horizon using a discount factor. Another important thing to note here is that we are indexing everything starting from 1 here.

**Definition 2.3.** Q-function: $Q_h(s,a)$ is the maximal expected cumulated reward starting in state $s$ and performing action $a$ in slot $h$ (remember the episode length is $H$).

So basically, to calculate $Q_h(s,a)$, we take an action $a$ in state $s$, and after that always continue with the given policy (usually the optimal policy). The difference to value function is that we do not execute the given/optimal policy in state $s$, slot $h$, but choose to perform action $a$ instead. One can think of a modified policy where one is executing action $a$ in the first step and then is always following the policy. It is important to note that:

$$V^*_H(s) = \max_{a \in A(s)} Q_1(s,a). \quad (2.2)$$

### 2.3 Dynamic Programming in MDP’s

Richard Bellman was an American applied mathematician who derived the following equations which allow us to start solving these MDPs. The Bellman equations are ubiquitous in RL and are necessary to understand how RL algorithms work. Since, we would be dealing with a time domain divided into episodes of fixed length, a sequential decision making problem is what we are required to solve. Before we get solving this problem, let us define a utility function $U^\pi_h(s)$ as the average reward starting at slot $h$ and state $s$ when following policy $\pi$.

$$U^\pi_h(s) = \mathbb{E}^\pi \left[ \sum_{i=h}^{H} r_i(S_i, A_i) + r_H(S_H) | S_h = s \right]. \quad (2.3)$$

The way to find the optimal policy $\pi^*$ is to start with: $U_{H+1}(s) = r_H(s)$ for all $s$ and then by backward recursion compute $U_h$ from $U_{h+1}$:

$$U_h(s) = \sup_{a \in A(s)} \left[ r_h(s,a) + \sum_{j \in S} p(j|s,a) U_{h+1}(j) \right]. \quad (2.4)$$

The optimal action to take at slot $h$ given the MDP dynamics and the policy $\pi_h$ for step $h$ is determined by:

$$\pi_h(s) = \arg \max_{a \in A(s)} \left[ r_h(s,a) + \sum_{j \in S} p(j|s,a) U_{h+1}(s) \right]. \quad (2.5)$$
The Q-function provides a nice way to encode both, the value function and the policy.

\[ \pi_h(s) = \arg\max_{a \in \mathcal{A}(s)} Q_h(s, a). \tag{2.6} \]

Algorithm 1 calculates the Q-function for the \(k\)-th episode and slot \(h\) using the Dynamic Programming (DP) paradigm. In the algorithm listed, \(Q_{kh}\) denotes the Q-function at slot \(h\) in episode \(k\) whereas \(U_{kh}\) is the utility function at slot \(h\) in episode \(k\). \(Q_k\) and \(U_k\) denote the Q-function and the utility function for episode \(k\). It is pivotal to note here that \(M\) encompassed the transition and reward dynamics (\(p\) and \(r\), respectively).

<table>
<thead>
<tr>
<th>Algorithm 1</th>
<th>Dynamic Programming when true (M) is known</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: procedure DP((M))</td>
<td></td>
</tr>
<tr>
<td>2: Initialize (U_{k,H+1} = 0)</td>
<td></td>
</tr>
<tr>
<td>3: for (h = H, H - 1, \ldots, 1) do</td>
<td></td>
</tr>
<tr>
<td>4: for ((s, a) \in S \times A) do</td>
<td></td>
</tr>
<tr>
<td>5: (Q_{kh}(s, a) = r(s, a) + p(\cdot</td>
<td>s, a))(T)(U_{k,h+1})</td>
</tr>
<tr>
<td>6: (U_{kh}(s) = \max_{a \in \mathcal{A}(s)} Q_{kh}(s, a))</td>
<td></td>
</tr>
<tr>
<td>7: end for</td>
<td></td>
</tr>
<tr>
<td>8: end for</td>
<td></td>
</tr>
<tr>
<td>9: return (Q_k)</td>
<td></td>
</tr>
<tr>
<td>10: end procedure</td>
<td></td>
</tr>
</tbody>
</table>

### 2.4 Performance Measures for RL

The performance of all RL algorithms are measured and compared in terms of two quantities:

#### 2.4.1 Regret

The regret of an algorithm is defined as the difference between the cumulative reward of the optimal policy and that gathered by the policy \(\pi\) learned by the algorithm. The regret quantifies the exploration-exploitation trade-off showcased by the algorithm (generally on-policy) in question. Figure 2.1 shows a how a typical plot for regret may look like.
There are two types of theoretical regret lower bounds that we are interested in when we study the proposed algorithms. Those are problem specific regret bounds and the minimax regret bound. In case of the problem specific regret bound, we will make statements such as \( \forall M, \forall \pi, R_\pi^M(T) \geq F(M, T) \) where \( F(M, T) \) is a function of the MDP \( M \) and the time domain \( T \). For the minimax lower bound on the other hand, we will make statements along the lines of \( \exists M \text{ such that } \forall \pi, \forall T, R_\pi^M(T) \geq G(S, A, T, \ldots) \). Here \( G(S, A, T, \ldots) \) is a distribution independent function.

### 2.4.2 Sample Complexity

The sample complexity of an algorithm is defined as the time required by the algorithm to find an approximately optimal policy which is well defined for any kind of RL problem.

In the episodic case, an on-policy algorithm returns, after the end of the \((k - 1)^{th}\) episode, a policy \( \pi_k \) to be applied in the \( k^{th} \) episode. The sample complexity of an algorithm \( \pi \) is defined as the minimum number of episodes \( K^{\pi}_{SP} \) such that for all \( k \geq K^{\pi}_{SP}, \pi_k \) is \( \epsilon \)-optimal with probability at least \( 1 - \delta \), i.e., for \( k \geq K^{\pi}_{SP}, \)

\[
P[V_{H_k}^{\pi_k} \geq V_{H}^{\pi^*} - \epsilon] \geq 1 - \delta. \tag{2.7}
\]

Here, \( V_{H_k}^{\pi_k} \) is the expected cumulated reward over a time \( H \) by the agent when following policy \( \pi_k \) and similarly \( V_{H}^{\pi^*} \) is the expected cumulated reward over a time \( H \) by the agent when following policy \( \pi^* \).
2.5 The Exploration-Exploitation tradeoff

An important thing to note in Algorithm 1 is that it takes the MDP $M$ as an input. This means that the algorithm assumes that we know the MDP dynamics namely the reward and the transition function. But, in real scenarios we generally do not know both or at least one of them. That is when it becomes pivotal to estimate the MDP $M$. A greedy method of estimating the MDP generally leads to a suboptimal regret hence an intelligent strategy to toggle exploration and exploitation should work well.

The exploration-exploitation trade-off is a fundamental dilemma whenever you learn about the world by trying things out. The dilemma is between choosing what you know and getting something close to what you expect (‘exploitation’) and choosing something you aren’t sure about and possibly learning more (‘exploration’). So in case of RL algorithms this transforms to whether we keep playing the action (in a particular state) which we know are the best up to now (‘exploitation’) or we explore more (play an action not played much till now). This is a dilemma also faced in the domain of Bandit optimization and the principle of optimism in the face of uncertainty is used by many optimal algorithms (in terms of attained regret lower bounds) to mitigate it. We are interested in these class of algorithms in RL setting. This dilemma in case of the bandit optimization takes shape in the form of the decision - whether play the arm which produces the highest average reward till now or to try out an arm which has not been pulled enough till now.

![Clinical trial: a typical bandit problem](Source: Reinforcement Learning: A Graduate course)

Basically, a multi-armed bandit problem can be seen as an MDP with only one state ($S = 1$) and with independent arms. In this setting the actions $a \in A$ are often called “arms” and the optimal average
reward is simply the average reward of the best arm. The regret in this case is the cost of trying to find which the best arm is to pull.

Figure 2.2 shows a typical problem which can be phrased as a bandit problem. The goal is to design a treatment selection scheme $\pi$ maximising the number of patients cured after treatment. There are two available treatments with unknown rewards (‘Live’ or ‘Die’). After administering the treatment, whether she survives or dies is the bandit feedback. We also talk about a few optimal algorithms for designing an efficient strategy to select the best treatment in the next chapter.

2.6 Kullback–Leibler divergence

We use the notion of Kullback–Leibler (KL) divergence in the thesis often. In mathematical statistics, the KL divergence is a measure of how one probability distribution diverges from a second, expected probability distribution. It is a distribution-wise asymmetric measure and thus does not qualify as a statistical metric of spread.

For discrete probability distributions $I$ and $J$ ($I$ is continuous w.r.t $J$), the KL divergence from $I$ to $J$ is defined as:

$$KL(I,J) = \sum_x I(x) \log \frac{I(x)}{J(x)}.$$

In other words, it is the expectation of the logarithmic difference between the probabilities $I$ and $J$, where the expectation is taken using the probabilities $I$. For distributions $I$ and $J$ of a continuous random variable, the KL divergence is defined to be the integral:

$$KL(I,J) = \int_{-\infty}^{+\infty} i(x) \log \frac{i(x)}{j(x)} \lambda(dx),$$

where $i$ and $j$ denote the densities of $I$ and $J$. We also assume that $I$ and $J$ are also continuous with respect to some measure $\lambda$ on $\mathbb{R}$. 
Chapter 3

Related Work

Since we are considering the problem of optimal exploration in reinforcement learning for finite horizon MDPs, it is important to discuss the work done in the field till now. We model the environment as a MDP whose transition and reward dynamics are unknown to the agent. As the agent interacts with the environment it observes the states, actions and rewards generated by the system dynamics. This leads to a fundamental trade off: should the agent explore poorly-understood states and actions to gain information and improve future performance, or exploit its knowledge to optimize short-run rewards.

The most common approach to this learning problem is to separate the process of estimation and optimization. Naive optimization can lead to premature exploitation. Dithering approaches to exploration (e.g., \( \epsilon \)-greedy) address this failing through random action selection. However, since this exploration is not directed the resultant algorithms may take exponentially long to learn [12]. In order to learn efficiently it is necessary that the agent prioritizes potentially informative states and actions. Moreover, Garivier et. al. [9] illustrate that strategies based on an exploration phase (up to a stopping time) followed by exploitation are necessarily suboptimal. Hence, we probably need to switch between the exploration and exploitation phase adaptively to design efficient algorithms.

3.1 Algorithms based on the OFU principle

To combat the above mentioned failings, the majority of provably efficient learning algorithms employ the OFU principle. This principle
has been exploited in many bandit algorithms like UCB \cite{2} and KL-UCB \cite{8} among others. Recently, Menard et. al. \cite{13} propose the KL-UCB++ algorithm for regret minimization in stochastic bandit models with exponential families of distributions. The authors prove that it is simultaneously asymptotically optimal (in the sense of Lai and Robbins’ lower bound) and minimax optimal. This is the first algorithm proved to enjoy these two properties at the same time.

In OFU class of algorithms for RL, each state and action is afforded some “optimism” such that its imagined value is as high as statistically plausible. The agent then chooses a policy under this optimistic view of the world. OFU allows for efficient exploration since poorly-understood states and actions are afforded higher optimistic bonus. As the agent resolves its uncertainty, the effects of optimism will reduce and the agent’s policy will approach optimality. Almost all reinforcement learning algorithms with polynomial bounds on sample complexity employ optimism to guide exploration \cite{12} \cite{6} \cite{4} \cite{21}.

From the OFU side, the work by Azar et. al. \cite{3} contains two key insights. They use careful application of concentration inequalities to the optimal value function as a whole, rather than to the transitions probabilities (to improve scaling in $S$), and then define Bernstein-based “exploration bonuses” that use the empirical variance of the estimated values at the next states (to improve scaling in $H$). Their algorithm, Upper Confidence Bound Value Iteration (UCBVI) is similar to model-based interval estimation (MBIE-EB) \cite{20}. A drawback though in their work is that they only handle the case where the transition dynamics are unknown. Importantly, the upper bound for regret derived by them matches the lower bound for this problem which we will derive in this thesis, up to logarithmic factors. They demonstrate that it is possible to design a simple and computationally efficient optimistic algorithm that simultaneously address both the loose scaling in $S$ and $H$ to obtain the first regret bounds that match the $\Omega(HSAT)$ (established in this thesis) lower bounds as $T$ becomes large.

For ergodic MDP’s, Jaksch et. al. \cite{11} suggest an algorithm Upper Confidence Reinforcement Learning (UCRL2) where the optimism in the reward and the transition probabilities is bounded by an $L^1$ norm around the respective empirical estimates. In order to describe the transition structure of an MDP, they propose a new parameter $D$: An MDP has diameter $D$ if for any pair of states $s, s'$ there is a policy which moves from $s$ to $s'$ in at most $D$ steps (on average). They also show that
the UCRL2 algorithm attains a total regret of $\tilde{O}(DS\sqrt{AT})$ after $T$ steps for any unknown MDP with $S$ states, $A$ actions per state, and diameter $D$. A corresponding lower bound of $\Omega(\sqrt{DSAT})$ on the total regret of any learning algorithm is given in the paper as well. Filippi et. al. [7] modified the UCRL2 algorithm by using the KL measure instead of the $L_1$ norm to bound the optimism in the reward and transition dynamics. KL-UCRL provides the same guarantees as UCRL2 in terms of regret. However, numerical experiments on classical benchmarks show a significantly improved behavior, particularly when the MDP has reduced connectivity.

3.2 Bayesian treatment of the problem

At the other end of the spectrum is the principle motivated by the Bayesian treatment of the problem (inspired by Thompson Sampling [23]) has emerged as a practical competitor to optimism. The algorithm Posterior Sampling Reinforcement Learning (PSRL) [18] maintains a posterior distribution for MDPs and, at each episode of interaction, follows a policy which is optimal for a single random sample. Experiments in [16] also reveal that the PSRL algorithm performs better than all existing OFU based algorithms in terms of the average regret attained by the algorithm. Previous works have argue for the potential benefits of such PSRL methods over existing optimistic approaches [18] but they come with guarantees on the Bayesian regret only. However a very recent work [1] have shown that an optimistic version of posterior sampling (using a max over several samples) achieves a frequentist regret bound $\tilde{O}(H\sqrt{SAT})$ (for large T) in the more general setting of weakly communicating MDPs.

In this thesis, we try to design OFU inspired algorithms which derive ideas from a few past algorithms and specifically from bandit literature.
Chapter 4

Regret Lower Bounds

This chapter provides the lower bounds on regret for reinforcement learning in the episodic scenario. In particular, in this chapter, we:

- derive an expression for regret;
- derive a constraint minimization problem for the minimization of regret for any uniformly good algorithm;
- derive an asymptotic and a non-asymptotic problem specific regret bounds, and
- derive a minimax regret bound.

4.1 Notations

It is important first to define the notations we use in this chapter. The following list contains the meaning of the notations we use. Other notations are defined at the place of their use.

- \( N_k(s, a) \): total number of visits of pair \((s, a)\) in the first \(k\) episodes
- \( N_{kh}(s, a) \): total number of visits of pair \((s, a)\) at slot \(h\) in the first \(k\) episodes
- \( n'_{kh}(s, a) \): total number of visits of pair \((s, a)\) at slot \(h\) in the \(k^{th}\) episode
- \( N_{kh}(s) \): total number of visits to state \(s\) at slot \(h\) in the first \(k\) episodes
• $\pi^*_M$: optimal policy for MDP $M$
• $O(s, h, M)$: optimal action at state $s$ in slot $h$ according to an optimal oracle policy for MDP $M$
• $\pi_k$: policy followed in episode $k$ under the algorithm $\pi$

We assume that there is a single optimal policy $\pi^*_M$ for MDP $M$.

4.2 Prerequisites

To present the regret lower bound, we introduce the notion of uniformly good (UG). An algorithm $\pi$ is uniformly good if from all starting states $s_0 \in S$ (in each episode), and all MDPs $M$, the expected number of times a suboptimal action $a$ in state $s$ at slot $h$ till episode $K$ satisfies $E^\pi_M[N_{kh}(s, a)] = o(T^\alpha)$ as $T$ grows large, for all $\alpha > 0$. Let $\Pi_G$ denote the set of all UG algorithms.

Also, let us introduce a random vector $X_{kh} = (S_{kh}, A_{kh}, R_{kh})$ representing the state, the action, and the collected reward at slot $h$ of episode $k$. A policy $\pi$ selects an action, denoted by $\pi(s, h)$, in step $h$ when the system is in state $s$ based on the history captured through $H^\pi_k$ (till episode $k$), the $\sigma$-algebra generated by $(X_{11}, X_{12}, \ldots, X_{k,h-1}, X_{kh}, S_{k,h+1})$ observed under $\pi$.

4.3 Regret in case of an episodic RL problem

The regret in case of the episodic RL setup (for any algorithm $\pi$) is defined as:

$$R^\pi_M(T) = E\left[\sum_{k=1}^K (V^*_{H^\pi_k}(S_1) - V^\pi_k(S_1))\right]. \quad (4.1)$$

We assume that the starting state of each episode $S_1$ is drawn from a distribution $\zeta$. $V^*_{H^\pi_k}(S_1)$ represents the optimal average expected reward (we are following policy $\pi^*$) over a time period $t$ when starting from state $S_1$ (a random variable) in each episode. $S_i$ denotes the random state at slot $i$ from here on. Similarly $V^\pi_k(S_1)$ is the average expected reward when following the policy $\pi_k$ in the episode $k$ (starting from $S_1$ state) over $t$ slots.
**Theorem 4.3.1.** Let $M$ be an MDP in an episodic setting with $K$ episodes and the length of each episode being $H$, the regret ($R_{\pi M}^T$) for any algorithm $\pi$ is given by:

$$R_{\pi M}^T = \sum_{h=1}^H \sum_{(s,a)} E_{M}^h[N_{Kh}(s,a)]\phi^*(s, a, h, M).$$

$\phi^*(s, a, h)$ is defined as the suboptimality gap of action $a$ in state $s$ at time slot $h$ and is given by:

$$\phi^*(s, a, h, M) = \{r_h(s, a^*) - r_h(s, a)\} + [p(\cdot|s, a^*) - p(\cdot|s, a)]^T V_{H-k}^{\pi^*}.$$  

**Proof.** Let's drop the summation over $K$ episodes from Equation (4.1) for now and consider the expression of regret for the $k$th episode starting at state $S_1$. It is given by:

$$D_{\pi}^{\pi_k}(S_1) = V_{H}^{\pi^*}(S_1) - V_{H}^{\pi_k}(S_1). \tag{4.2}$$

The expressions for $V_{H}^{\pi^*}(S_1)$ and $V_{H}^{\pi_k}(S_1)$ take the form:

$$V_{H}^{\pi^*}(S_1) = r_1(S_1, A_1^*) + E_{S_1}[E_{S_2}[V_{H-1}^{\pi^*}(S_2)|S_1]] \tag{4.3}$$

$$V_{H}^{\pi_k}(S_1) = r_1(S_1, A_1^{\pi_k}) + E_{S_1}[E_{S_2}^{\pi_k}[V_{H-1}^{\pi_k}(S_2^{\pi_k})|S_1]]. \tag{4.4}$$

$S_t^*$ and $A_t^*$ are random variables representing the state and action at slot $t$ when following the optimal policy $\pi^*$. Similarly $S_t^{\pi_k}$ and $A_t^{\pi_k}$ are the random variables representing the state and action at slot $t$ when following the optimal policy $\pi_k$. Substituting Equations (4.3) and (4.4) in Equation (4.2):

$$D_{H}^{\pi_k}(S_1) = r_1(S_1, A_1^*) + E_{S_1}[E_{S_2}[V_{H-1}^{\pi^*}(S_2)|S_1]] \tag{4.5}$$

$$- r_1(S_1, A_1^{\pi_k}) - E_{S_1}[E_{S_2}^{\pi_k}[V_{H-1}^{\pi_k}(S_2^{\pi_k})|S_1]].$$

Now adding and subtracting $E_{S_1}[E_{S_2}^{\pi_k}[V_{H-1}^{\pi_k}(S_2^{\pi_k})|S_1]]$ from the right hand side of Equation (4.5). $D_{H}^{\pi_k}(S_1)$ takes the following form:

$$D_{H}^{\pi_k}(S_1) = Y + Z$$

where
18 CHAPTER 4. REGRET LOWER BOUNDS

\[ Y = r_1(S_1, A_1^*) + E_{S_1}[E_{S_2}^* [V_{H-1}^*(S_2^*)|S_1]] \]
\[ - r_1(S_1, A_1^T) - E_{S_1}[E_{S_2}^{\pi_k} [V_{H-1}^{\pi_k}(S_2^{\pi_k})|S_1]] \]
\[ Z = E_{S_1}[E_{S_2}^{\pi_k} [V_{H-1}^{\pi_k}(S_2^{\pi_k}) - V_{H-1}^{\pi_k}(S_2^{\pi_k})|S_1]]. \]

\[ Y = \{ r_1(S_1, A_1^*) - r_1(S_1, A_1^T) \} + \big[p(\cdot|S_1, A_1^*) - p(\cdot|S_1, A_1^T)\big]^T V_{H-1}^{\pi_k}. \]

Carefully analyzing Z, it takes a recursive form:
\[ Z = E_{S_1}[D_{H-1}^{\pi_k}(S_2^{\pi_k})]. \]

Iterating down to \( H = 1 \), we have:
\[ D_{H-1}^{\pi_k} = Y + \sum_{h=2}^{H} E_{S_1}\{ r_h(S_h^{\pi_k}, A_h^*) - r_h(S_h^{\pi_k}, A_h^T) \} \]
\[ + \big[p(\cdot|S_h^{\pi_k}, A_h^*) - p(\cdot|S_h^{\pi_k}, A_h^T)\big]^T V_{H-h}^{\pi_k} \]
\[ = \sum_{h=1}^{H} \sum_{(s,a)} E_{S_1}[n_{kh}^*(s,a)]\phi^*(s,a,h,M). \]

Remember, this expression is for a single episode. The regret \( \mathfrak{R}_M^\pi(T) \) for any algorithm \( \pi \) using the result obtained for a single episode and the expression for regret in Equation (4.1) is given by:
\[ \mathfrak{R}_M^\pi(T) = \sum_{h=1}^{H} \sum_{(s,a)} E_M[N_{Kh}(s,a)]\phi^*(s,a,h,M). \]

4.4 The problem specific regret lower bound

To state our lower bound, we introduce the following notations. For \( M = (S, A, p, q) \) and \( M' = (S, A, p', q') \), we denote \( M \ll M' \) if \( M \) is absolutely continuous with respect to that of \( M' \), i.e. \( \forall E, P_M[E] = 0 \).
if \( P_{M'}[E] = 0 \). For \( M \) and \( M' \) such that \( M \ll M' \), we define the KL-divergence between \( M \) and \( M' \) in state \( s \), action \( a \) and slot \( h \) 
\( (KL_{M|M'}(s,a,h)) \) as the KL-divergence between the distributions of the next state and collected reward if in state \( s \), action \( a \) is selected under these two MDP’s at slot \( h \):

\[
KL_{M|M'}(s,a,h) = \sum_{y \in S} p(y|s,a) \log \frac{p(y|s,a)}{p'(y|s,a)} + \int_{0}^{1} q_h(r|s,a) \log \frac{q_h(r|s,a)}{q'_h(r|s,a)} \lambda(dr).
\]

The first term on the right hand side denotes the KL divergence between the transition dynamics of the two MDP’s \((M, M')\) and the second term denotes KL divergence between the reward distributions of \( M \) and \( M' \) at slot \( h \). It is important to note here that we are not considering the case of non-stationary transition dynamics (we can extend all our arguments to that case as well without loss of generality). We further define the set of all confusing MDP’s:

\[
\Gamma_\Phi(M) = \{ M' \in \Phi : M \ll M', (i) KL_{M|M'}(s,a,h) = 0, \forall s, \forall a \in O(s,h,M); (ii) \pi^*_M \cap \pi^*_M' = \emptyset \}.
\]

Here \( \Phi \) is the set of all plausible MDP’s with the same state and action sets (only the reward and the transition kernel can change). The set \( \Gamma_\Phi(M) \) consists of MDP’s \( M' \) that (i) coincide with \( M \) for (state, slot) tuple where the actions are optimal (the kernels of \( M \) and \( M' \) cannot be statistically distinguished under an optimal policy); and (ii) that the optimal policies under \( M' \) are not optimal under \( M \). Hence, \( \Gamma_\Phi(M) \) can be interpreted as a set of confusing MDP’s to identify the optimal policy.

**Theorem 4.4.1.** For any uniformly good algorithm \( \pi \in \Pi_G \), and for any MDP \( M = (S,A,p,q) \), we have:

\[
\lim_{T \to \infty} \inf \frac{R^\pi_M(T)}{\log T} \geq C_\Phi(M)
\]
where \( C_\Phi(M) \) is the value of the optimization problem given by:

\[
\min_{\eta_h(s,a) \in F_0(M)} \sum_{h=1}^{H} \sum_{(s,a)} \eta_h(s,a) \phi^*(s,a,h,M)
\]

\[
\text{s.t.} \sum_{h=1}^{H} \sum_{(s,a) \in S \times A} \eta_h(s,a) KL_{M|M'}(s,a,h) \geq 1 \forall M' \in \Gamma_\Phi(M).
\]

Here we define,

\[
F_0(M) = \{ \eta_h(s,a) \geq 0 : \eta_h(s,a) = 0, \forall (s,a,h) \text{ s.t. } a \in O(s,h,M) \}.
\]

**Proof.** Let \( M \) be a MDP represented by \((S,A,p,q)\) and \( \pi \in \Pi_G \) be a uniformly good algorithm. From Theorem 4.3.1, we know that if we lower bound \( \mathbb{E}_\pi^M[N_{Kh}(s,a)] \), this in turn provides a lower bound on the regret.

Let us consider an observation sequence \( O = S_{k1}, A_{k1}, R_{k1}, S_{k2}, A_{k2}, R_{k2}, \ldots S_{k,H+1} \) where \( k \in \{1,2,3 \ldots K\} \). Here \( S_{ki}, A_{ki} \) and \( R_{ki} \) denote the random states, actions and reward received at episode \( k \) and slot \( i \). Basically, we are observing how the system evolves for \( K \) episodes (each episode of length \( H \)). The MDP model \( M \) is characterized by a transition kernel \( p(\cdot|s,a) \) and a reward distribution \( q_h(\cdot|s,a) \) (slot dependent). The probability of observing the mentioned sequence conditional on model \( M \) is given by:

\[
P_M(O) = \left\{ \prod_{i=1}^{K} \zeta(S_{i1}) \right\} \prod_{j=1}^{K} \prod_{l=1}^{H} p(S_{j,l+1}|S_{jl}, A_{jl}) q_l(R_{jl}|S_{jl}, A_{jl}).
\]

Here \( \zeta(S_{i1}) \) is the probability of starting in state \( S_{i1} \) at slot 1 in episode \( i \). Let us now consider another model \( M' \) characterized by a transition kernel \( p'(\cdot|s,a) \) and a reward distribution \( q'_h(\cdot|s,a) \) (slot dependent). Defining the log-likelihood ratio of data \( (O) \):

\[
L = \log \frac{P_M(O)}{P_{M'}(O)}
\]

the log-likelihood ratio \( L \) takes the form:

\[
L = \sum_{j=1}^{K} \sum_{l=1}^{H} \left[ \log \frac{p(S_{j,l+1}|S_{jl}, A_{jl})}{p'(S_{j,l+1}|S_{jl}, A_{jl})} + \log \frac{q_l(R_{jl}|S_{jl}, A_{jl})}{q'_l(R_{jl}|S_{jl}, A_{jl})} \right].
\]
We can use the same techniques as in [9] (essentially an extension of Wald’s lemma) to write:

$$E^\pi_M[L] = \sum_{h=1}^{H} \sum_{s,a \notin O(s,h,M)} E^\pi_M[N_{Kh}(s,a)] KL_{M|M'}(s,a,h). \quad (4.10)$$

Additionally, the following data processing inequality holds $\forall M'$ and we have a uniformly good algorithm $\pi$:

$$E^\pi_M[L] \geq KL(P^\pi_M(E), P^\pi_{M'}(E)). \quad (4.11)$$

$E$ is any event selected so that it leverages our definition of uniformly good algorithm. We first estimate the right hand side of Equation (4.11). According to our definition of uniformly good, lets define an event $E = \{N_{Kh}(s,a) \leq \rho(T - \sqrt{T}) : a = O(s,h,M)\}$ for some constant $\rho$ and some state $s$, slot $h$. We also know that $a \neq O(s,h,M')$ because of the way we have defined our set of confusing MDP’s $\Gamma_\phi(M)$. Since, we have a uniformly good algorithm $\pi$, $P^\pi_M[E] = 0$ and $P^\pi_{M'}[E] = 1$ as $T \to \infty$. Hence as $T \to \infty$,

$$\frac{KL(P^\pi_M(E), P^\pi_{M'}(E))}{\log(T)} \sim \frac{1}{\log(T)} \log \frac{1}{P^\pi_{M'}[E^C]}. \quad (4.12)$$

Here, the event $E^C = \{N_{Kh}(s,a) \geq \rho(T - \sqrt{T}) : a = O(s,h,M)\}$. We can write using the Markov inequality:

$$P^\pi_{M'}[E^C] \leq \frac{E^\pi_{M'}[N_{Kh}(s,a)]}{\rho(T - \sqrt{T})} \frac{1}{\log(T)} \log \frac{1}{P^\pi_{M'}[E^C]} \geq \frac{\log(\rho(T - \sqrt{T}))}{\log(T)} \frac{E^\pi_{M'}[N_{Kh}(s,a)]}{\log(T)} \quad (4.13)$$

Since, $\pi$ is a uniformly good algorithm and action $a$ is suboptimal for state $s$ and slot $h$ for $M'$ and we have a uniformly good algorithm $\pi$, $\frac{E^\pi_{M'}[N_{Kh}(s,a)]}{\log(T)} \to 0$ as $T \to \infty$. Hence, $KL(P^\pi_M(E), P^\pi_{M'}(E)) = \log(T)$ as $T \to \infty$. Using this result and Equations (4.10) and (4.11), we get a constraint for our minimization problem:

$$\liminf_{T \to \infty} \frac{1}{\log(T)} \sum_{h=1}^{H} \sum_{s,a \notin O(s,h,M)} E^\pi_M[N_{Kh}(s,a)] KL_{M|M'}(s,a,h) \geq 1. \quad (4.15)$$
Combining the above constraints valid for any $M' \in \Gamma_\phi(M)$ (additionally, $E_M^\pi[N_{Kh}(s, a)] \geq 0$ for all $(s, a, h)$) and Equation (4.8) concludes the proof of the theorem.

\[\square\]

## 4.5 Minimax regret lower bound

In this section, we derive the minimax regret lower bound for any uniformly good algorithm. Jaksch et. al. \cite{Jaksch2010} derive a minimax regret lower bound in case of an ergodic RL problem. Menard et. al. \cite{Menard2015} state that \cite{Kazerone2014} derive the minimax regret lower bound for the episodic RL problem but \cite{Kazerone2014} does not contain any proof for the minimax regret lower bound for the episodic RL case. To the best of our knowledge, the proof provided here is the only comprehensive proof of the minimax regret lower bound for an episodic RL problem.

**Theorem 4.5.1.** There exists an MDP $M$ with $S$ states, $A$ actions such that for any uniformly good algorithm $\pi \in \Pi_G$ and $T \geq 0.23 \times HSA$, the regret ($R_M^\pi(T)$) incurred by $\pi$ over a time domain $T$ is $\Omega(\sqrt{HSAT})$.

**Proof.** Let us consider two MDP’s ($M$ represented by $(S, A, p, q)$ and $M'$ represented by $(S, A, p', q')$). The number of states and actions for both the MDP’s is $S$ and $A$. For each state all actions are possible. The transition dynamics for both the models is $p(s^+|s, a) = 1/S$ for all states $s \in S$ ($s^+ \in S$ denotes the next state) and $a \in A$. For $M$, $R_t(s, a) \sim Be(\delta) \forall (s, a, t)$ and $t \in \{1, 2, \ldots, H\}$. $M'$ differs from $M$ only in the reward distribution function at state $s' \in S$ for an action $a' \in A$ at slot $h$. The reward distribution function for $M'$ at $(s', a', h)$ is given by $R'_h(s', a') \sim Be(\delta + \epsilon)$. Moreover, here we assume $0 < \delta \leq 0.5$ and $\epsilon \leq 1 - 2\delta$.

The expected regret for any uniformly good algorithm $\pi' \in \Pi_G$ for MDP $M'$ is:

$$R_M^\pi(T) = \epsilon(E_M^\pi[N_{Kh}(s')] - E_M^\pi[N_{Kh}(s', a')]).$$

From Equation (4.10) and using an extension of the dataprocessing inequality derived in \cite{Kazerone2014}, it can be deduced that:

$$E_M^\pi[N_{Kh}(s', a')]KL(\delta, \delta + \epsilon) \geq KL(E_M^\pi[Z], E_{M'}^\pi[Z]).$$
$Z$ is any $\mathcal{H}_K$ measurable random variable. By the design of our MDP’s, we know that $E_{M'}[N_{Kh}(s')] = K \times (1/S) = T/HS$. The expected regret then takes the following form:

$$R_{M'}(T) \geq T \epsilon \left( \frac{1}{HS} - \frac{E_{M'}[N_{Kh}(s', a')]}{T} \right).$$

To lower bound the expected regret, we need to upper bound $E_{M'}[N_{Kh}(s', a')]$. To this aim, we set $Z = \frac{N_{Kh}(s', a')}{T}$. For this $Z$, the Pinsker inequality leads us to:

$$KL\left(\frac{E_{M'}[N_{Kh}(s', a')]}{T} \left| \frac{E_{M'}[N_{Kh}(s', a')]}{T}\right.\right) \geq 2 \left(\frac{E_{M'}[N_{Kh}(s', a')]}{T} - \frac{E_{M'}[N_{Kh}(s', a')]}{T}\right)^2.

(4.16)$$

Using the results in (4.16) and (4.5), we obtain:

$$E_{M'}[N_{Kh}(s', a')] \leq \frac{E_{M'}[N_{Kh}(s', a')]}{T} \leq \frac{1}{2} E_{M'}[N_{Kh}(s', a')] KL(\delta, \delta + \epsilon).$$

This inequality when used in expression for the expected regret in (4.5) bounds it:

$$R_{M'}(T) \geq T \epsilon \left( \frac{1}{HS} - \frac{E_{M'}[N_{Kh}(s', a')]}{T} \right) - \sqrt{\frac{1}{2} E_{M'}[N_{Kh}(s', a')] KL(\delta, \delta + \epsilon)}. \tag{4.17}$$

Also, since the reward distribution function for $M$ is same for all slots, states and actions, following policy $\pi'$ for $M$, we have:

$$\frac{E_{M'}[N_{Kh}(s', a')]}{E_{M'}[N_{Kh}(s')]} \leq \frac{1}{A}$$

$$E_{M'}[N_{Kh}(s', a')] \leq \frac{E_{M'}[N_{Kh}(s')]}{A}$$

$$E_{M'}[N_{Kh}(s', a')] \leq \frac{K}{SA} = \frac{T}{HS A}.$$

Using the above result the expected regret is:
\[ \mathcal{R}_{M'}^\pi(T) \geq T \epsilon \left( \frac{1}{H S} - \frac{1}{H S A} - \sqrt{\frac{1}{2} H S A} KL(\delta, \delta + \epsilon) \right). \]

Jaksch et. al. [11] in Lemma 20 prove that \( KL(\delta, \delta + \epsilon) \leq \frac{\epsilon^2}{\delta \log 2} \) for the constraints defined on \( \delta \) and \( \epsilon \). Using their result:

\[ \mathcal{R}_{M'}^\pi(T) \geq T \epsilon \left( \frac{1}{H S} - \frac{1}{H S A} - \sqrt{\frac{1}{2} H S A} \frac{\epsilon^2}{\delta \log 2} \right). \quad (4.18) \]

Setting \( \delta = 0.45 \) and \( \epsilon = \sqrt{\frac{H S A}{T}} \) and enforcing the constraint that \( \epsilon < 1 - 2\delta \), we have \( T \geq 0.23 \times H S A \). Substituting these values in (4.18),

\[ \mathcal{R}_{M'}^\pi(T) \geq \sqrt{H S A T} \times \sqrt{0.0045} \left( \frac{1}{H S} - \frac{1}{H S A} - \sqrt{\frac{1}{2} \log 2} \right). \quad (4.19) \]

Hence, there exists some constant \( C \) so that \( \mathcal{R}_{M'}^\pi(T) \geq C \times \sqrt{H S A T} \forall T > 0.23 \times H S A \). This proves Theorem 4.5.1.

\[ \square \]

### 4.6 Non-asymptotic regret lower bound

We have already derived an asymptotic regret lower bound for any uniformly good algorithm in Section 4.4. In this section, we derive a non-asymptotic regret lower bound for any uniformly good algorithm. The proof of the theorem is on the same lines as those of the proof provided in [9] for the bandit case.

**Definition:** A strategy is \( \pi \in \Pi_G \) smarter than the uniform strategy if for all MDP models \( M \), for the optimal action \( a^* \) in any state \( s \) at slot \( h \), for all \( T \geq 1 \), the following holds:

\[ \frac{E_M^\pi[N_{kh}(s, a^*)]}{E_M^\pi[N_{kh}(s)]]} \geq \frac{1}{A} \quad (4.20) \]

\[ E_M^\pi[N_{kh}(s, a^*)] \geq \frac{E_M^\pi[N_{kh}(s)]}{A} \quad (4.21) \]

\[ E_M^\pi[N_{kh}(s, a^*)] \geq \frac{K \times f_M(h, s, \pi)}{A}. \quad (4.22) \]
Here $f_M(h, s, \pi)$ is the probability of the system ending in state $s$ at slot $h$ when following the policy $\pi$ (assuming the starting state in each episode is the same) for MDP $M$. From the regret expression (4.8) for the episodic RL case, we know that lower bounding $E_\pi^M[N_{Kh}(s, a)]$ is enough to lower bound the total regret. Theorem 4.6.1 puts a lower bound on $E_\pi^M[N_{Kh}(s, a)]$ hence, lower bounding the regret for any uniformly good algorithm in an episodic setting. To state the non-asymptotic regret bound, we define:

$\mathcal{K}_{inf}(q_h(\cdot|s, a), q'(\cdot|s, a)) = \inf \{KL(q_h(\cdot|s, a), q'(\cdot|s, a)) : q_h(\cdot|s, a), q'(\cdot|s, a) \in \Theta(s, a) \text{ and } a \neq O(s, h, M), a = O(s, h, M')\}$.

Here, $q_h(\cdot|s, a)$ is the reward distribution function for the MDP $M$ for state $s$, action $a$ in slot $h$ and $a$ is not optimal at state $s$ in slot $h$ in $M$. And, $q'(\cdot|s, a)$ is the reward distribution function at state $s$, action $a$ in slot $h$ in another MDP $M'$ (keeping everything same as in $M$) such that action $a$ becomes the optimal action in state $s$ at slot $h$ in $M'$.

**Theorem 4.6.1.** For all algorithms $\pi \in \Pi_G$ that are smarter than the uniform strategy, for any MDP $M$, $\forall(s, a, h)$, and $T \geq 1$,

$$\frac{E_\pi^M[N_{Kh}(s, a)]}{K} \geq \frac{f_M(h, s, \pi)}{A} - \sqrt{\frac{2K * f_M(h, s, \pi) * K_{inf}(q_h(\cdot|s, a), q'(\cdot|s, a))}{A^2}}.$$  

Here $q_h(\cdot|s, a)$ is the reward distribution function for $M$ for state $s$ and action $a$ at slot $h$.

**Proof.** Let us consider an MDP $M = (S, A, p, q)$ and a uniformly good algorithm $\pi \in \Pi_G$. Recalling the expression for $E_\pi^M[L]$ from (4.10) and the fact that $E_\pi^M[L] \geq KL(E_\pi^M(Z), E_{M'}^\pi(Z))$ ($Z$ is a random variable which is $\mathcal{H}_K^\pi$ measurable),

$$E_\pi^M[L] = \sum_{h=1}^{H} \sum_{s,a \not\in O(s,h,M)} E_{M'}^\pi[N_{Kh}(s, a)]KL_{M|M'}(s, a, h) \geq KL(E_\pi^M(Z), E_{M'}^\pi(Z)).$$  

The above inequality holds for all $M' \in \Gamma_\Phi(M)$. Choosing a specific $M'$ such that $M' = M$, $\forall(\text{state, action, slot}) \neq (s, a, h)$. The reward
distribution at slot $h$ and state-action pair $(s, a)$ for $M'$ is changed to $q'(\cdot | s, a)$ such that action $a \not\in O(s, h, M)$ becomes the optimal action in state $s$ at slot $h$ for $M'$. It is also important to note that here the reward at each slot maybe drawn from a different slot specific distribution.

From (4.23), we have:

$$E^\pi_M[N_{Kh}(s, a)]KL(q_h(\cdot | s, a), q'(\cdot | s, a)) \geq KL(E^\pi_M[Z], E^\pi_{M'}[Z]). \quad (4.24)$$

We are assuming the system to demonstrate static transition dynamics (independent of the slot $h$). If this was not the case, extension to that case is can be done along the lines of the proof we are going to present. Setting $Z = N_{Kh}(s, a)/K$,

$$KL(E^\pi_M[Z], E^\pi_{M'}[Z]) = KL(E^\pi_M\left[\frac{N_{Kh}(s, a)}{K}\right], E^\pi_{M'}\left[\frac{N_{Kh}(s, a)}{K}\right]). \quad (4.25)$$

Any uniformly good policy $\pi$ is going to make the system end up in state $s$ at slot $h$ more often for $M'$ than for $M$ so that action $a$ can be taken more often in state $s$ at slot $h$ for $M'$ (because of the way we defined a specific $M'$). Hence, we have:

$$E^\pi_{M'}\left[\frac{N_{Kh}(s, a)}{K}\right] \geq E^\pi_M\left[\frac{N_{Kh}(s, a)}{K}\right].$$

Using the results in Equations (4.20)-(4.22), we have

$$E^\pi_{M'}\left[\frac{N_{Kh}(s, a)}{K}\right] \geq \frac{f_M(h, s, \pi)}{A}.$$  

Remember $a$ is not optimal in state $s$ at slot $h$ for $M$. We now know:

$$KL\left(E^\pi_M\left[\frac{N_{Kh}(s, a)}{K}\right], E^\pi_{M'}\left[\frac{N_{Kh}(s, a)}{K}\right]\right) \geq KL\left(E^\pi_M\left[\frac{N_{Kh}(s, a)}{K}\right], \frac{f_M(h, s, \pi)}{A}\right) \quad (4.26)$$

since $KL(p, q)$ is an increasing function for a fixed $p$ and $q \in [p, 1]$. Lemma 2 in [9] states:

$$KL(p, q) \geq \frac{1}{2q}(p - q)^2 \text{ where } 0 \leq p \leq q \leq 1.$$
Using the above result,

\[ KL\left(E_M^\pi \left[ \frac{N_{Kh}(s,a)}{K} \right], \frac{f_M(h,s,\pi)}{A} \right) \geq \frac{A}{2 \cdot f_M(h,s,\pi)} \left[ \frac{f_M(h,s,\pi)}{A} \right]^2 - E_M^\pi \left[ \frac{N_{Kh}(s,a)}{K} \right]^2. \] (4.27)

Using Equations (4.24), (4.26), (4.27):

\[ E_M^\pi[N_{Kh}(s,a)]KL(q_h(\cdot|s,a), q'_h(\cdot|s,a)) \geq \frac{A}{2 \cdot f_M(h,s,\pi)} \left[ \frac{f_M(h,s,\pi)}{A} \right]^2 - E_M^\pi \left[ \frac{N_{Kh}(s,a)}{K} \right]^2. \] (4.28)

Rearranging the terms and using the fact that \( E_M^\pi[N_{Kh}(s,a)/K] \leq f_M(h,s,\pi)/A \) for a non-optimal action \( a \) in state \( s \) at slot \( h \) for \( M \), we conclude:

\[ E_M^\pi\left[ \frac{N_{Kh}(s,a)}{K} \right] \geq \frac{f_M(h,s,\pi)}{A} \sqrt{ \frac{2K \cdot f_M^2(h,s,\pi) \cdot KL(q_h(\cdot|s,a), q'_h(\cdot|s,a))}{A^2} }. \]

Taking the infimum over all \( q'_h(\cdot|s,a) \):

\[ E_M^\pi\left[ \frac{N_{Kh}(s,a)}{K} \right] \geq \frac{f_M(h,s,\pi)}{A} \sqrt{ \frac{2K \cdot f_M^2(h,s,\pi) \cdot K_{\text{inf}}(q_h(\cdot|s,a), q'_h(\cdot|s,a))}{A^2} }. \] (4.29)

This proves Theorem 4.6.1. We can further deduce from Theorem 4.6.1 that the regret in the episodic setting for any uniformly good algorithm \( \pi \) for small \( T \) is linear.

\[ \square \]

The essence of these lower bounds derived in this chapter is that no matter what learning algorithm you choose, the regret bounds (both distribution dependent and independent) cannot be lesser than what we derived in this chapter. This is a pretty powerful result, since it
means that if we can design an algorithm with upper bounds on regret which matches the derived bounds, then those algorithms will be in some sense near-optimal. Moreover, using the methodology for the derivation of lower bounds for regret (specially for the problem specific case) could be used to design optimal algorithms in terms of regret.
Chapter 5

Algorithms

In this chapter, we suggest algorithms that are based on the OFU principle for the episodic setting. The idea comes from the optimal adaptive policies which were first suggested by Burnetas et. al. [5]. The optimistic index calculation algorithm proposed in [5] involves the calculation of an optimistic estimate of the transition dynamics which is not discussed in the paper and hence makes the method computationally implausible. By studying the linear maximization problem under the KL constraints, [7] provide an efficient algorithm for solving KL-optimistic extended value iteration. We adapt the method suggested by them to calculate optimistic utility and Q-functions for the episodic scenario. Further, we propose an algorithm by altering the radius of the KL-ball which basically quantifies our optimism for index calculation. This approach is inspired by the KL-UCB++ algorithm proposed by Menard et. al. [13] for bandits which is simultaneously asymptotically optimal and minimax optimal.

To start with, we describe a generic algorithm (Algorithm 2) that forms the basic structure for all the algorithms based on the OFU principle we discuss in the thesis. In Algorithm 2, we start with an empty history variable $H$ which is updated as we make more observations. The history is used to find empirical estimates of the transition and reward dynamics, respectively. At the start of the $k^{th}$ episode, the Q-functions (both, the empirical Q-function - $Q_k$ and the optimistic Q-function - $Q_{k}^{opt}$) are estimated for the $k^{th}$ episode. These are then used for action selection at each time step in the episode. By $\hat{M}$ (defined by $\hat{p}$ and $\hat{r}$), we denote the empirical estimate of the MDP (which can be computed using $H$). In the algorithm, $a_{kh}$ is the action taken at
slot \( h \) of episode \( k \). Similarly, \( s_{kh} \) is the state at slot \( h \) of episode \( k \). In the algorithm, the functions \( \text{CalculateEmpiricalQFunction()} \) and \( \text{CalculateOptimisticQFunction()} \) calculate the empirical and the optimistic Q-function. These are dummy functions and are replaced by the respective substitutes for different algorithms. Similarly, the \( \text{PlayAction()} \) function selects the action to be played at slot \( h \) and episode \( k \) \( (a_{kh}) \) in state \( s_{kh} \). Again, we use different algorithm specific action selection paradigms for different algorithms. Moreover, \( \text{GetNextState()} \) is a simulator specific function that generates the next state given the current state and the action selected based on the true transition dynamics.

We list the greedy action selection paradigm in Algorithm 3. The algorithm selects the best action \( (a_{kh}) \) greedily at a state \( s \) in slot \( h \) given the Q-function for episode \( k \) and slot \( h \) \( (Q_{kh}) \) and the count of the number of occurrences of \( (s, a) \) tuples in the first \( k \) episodes at slot \( h \) \( (N_{kh}) \). It is important to note here that the Q-function which is passed in to the algorithm can be the empirical estimate or an optimistic estimate (depends on the algorithm that calls this function).

Algorithm 2: Episodic RL Generic Algorithm

1: \begin{algorithmic}
2: \Procedure{M}{AIN} \Comment{Control center for all OFU algorithms}
3: \State \( \mathcal{H} = \emptyset \) \Comment{The history}
4: \For{\( k = 1 \colon K \)}
5: \State \( Q_k = \text{CalculateEmpiricalQFunction}(\hat{M}) \)
6: \State \( Q_k^{\text{opt}} = \text{CalculateOptimisticQFunction}(\hat{M}, k) \)
7: \For{\( h = 1 \colon H \)}
8: \State \( a_{kh} = \text{PlayAction}(\hat{M}, Q_{kh}, Q_{kh}^{\text{opt}}, s_{kh}) \) \Comment{According to algorithm of choice}
9: \State \( s_{k,h+1} = \text{GetNextState}(s_{kh}, a_{kh}) \)
10: \State \( \mathcal{H} = \mathcal{H} \cup (s_{kh}, a_{kh}, s_{k,h+1}) \) \Comment{Update history}
11: \EndFor
12: \EndFor
13: \EndProcedure
\end{algorithmic}

For calculating the Q-function from the empirical estimate of MDP \( M \) i.e. \( \hat{M} \) (defined by \( \hat{p} \) and \( \hat{r} \)), we just pass \( \hat{M} \) in place of \( M \) in the dynamic programming method (Algorithm 1) discussed in Chapter 2.

Further, we handle the case where the transition and reward dynamics are unknown, unlike the algorithm suggested in [5] where only the unknown transition dynamics case is considered.
Algorithm 3: Greedy action selection

1: procedure SELECTACTIONGREEDY($N_{kh}, Q_{kh}, s$)
2:     if $\{a \in \mathcal{A}(s) : N_{kh}(s, a) = 0\} \neq \emptyset$ then
3:         $W = \{a \in \mathcal{A}(s) : N_{kh}(s, a) = 0\}$
4:         $a_{kh} = \text{Choose a random action from set } W$
5:     else
6:         $a_{kh} = \arg\max_{a \in \mathcal{A}(s)} Q_{kh}(s, a)$
7:     end if
8: return $a_{kh}$
9: end procedure

5.1 Unknown Transition Dynamics

Let us first consider the case where only the transition dynamics ($p$) is unknown. We propose an index based policy where the choice of actions at each state $s$ and time slot $h$ is based on the indices that are inflations of the right hand side of the following DP recursive equation.

$$Q_{kh}(s, a) = r(s, a) + p^T (\cdot | s, a) U_{k, h+1}.$$  (5.1)

A backward induction equation of the above form is used for estimating the empirical Q-function ($Q_{kh}$) as well as the optimistic Q-function ($Q_{opt, kh}$) for episode $k$ and slot $h$. The recursion is initialized by setting $U_{k, H+1} = 0$ for all $s$. Recollecting from Chapter 2, $U_{kh}(s) = \max_{a \in \mathcal{A}(s)} Q_{kh}(s, a)$.

5.1.1 Adaptive Index Policy

The OFU based algorithm we design here (Algorithm 5) is inspired by the paradigm first suggested in [5] for ergodic MDP’s. Burnetas et. al. in [5] proposed an index policy whose expected regret asymptotically matches the regret lower bound for the ergodic RL case. When the transition dynamics are unknown, to calculate the optimistic estimate of the Q-function for slot $h$, episode $k$, we modify the backward
induction equation (Equation (5.1)) to the following form:

\[ Q_{kh}^{opt}(s, a) = \max_{p_{opt} \in \Delta_{s, a}} \{ r(s, a) + p_{opt}^T U_{k, h+1} : KL(\hat{p}(\cdot|s, a), p_{opt}) \leq \frac{\log(k + 1)}{N_{kh}(s, a)} \}. \] (5.2)

Algorithm 4 describes the way we calculate the optimistic Q-function \((Q_{kh}^{opt})\) for episode \(k\). Equation (5.2) is implemented in the algorithm in the form of a maximization problem constrained by a KL-ball whose radius is defined by \(\epsilon = \frac{\log(k + 1)}{N_{kh}(s, a)}\). The function takes as input the utility function for episode \(k\) \((U_k)\), the empirical estimate of the transition dynamics \((\hat{p})\) (the reward \(r(s, a)\) for each state-action pair is known), the episode number \(k\) and \(N_{kh}\) which as earlier contains the information on how many times a particular \((s, a)\) occur at slot \(h\) till episode \(k\). Note, that we still have not described how the optimistic transition dynamics \(p_{opt}\) is calculated in the algorithm. We come back to that part later.

Algorithm 4 Calculate AIP optimistic indices

1: procedure CALCULATEAIPINDICES\((U_k, \hat{p}, k, N_{kh})\)
2: \hspace{1em} for \(h = H, H - 1, \ldots, 1\) do
3: \hspace{2em} for \((s, a) \in S \times A\) do
4: \hspace{3em} \(\epsilon = \frac{\log(k + 1)}{N_{kh}(s, a)}\)
5: \hspace{3em} \(p_{opt} = \text{MaxKLBall}(U_{k, h+1}, \hat{p}(\cdot|s, a), \epsilon)\)
6: \hspace{3em} \(Q_{kh}^{opt}(s, a) = r(s, a) + p_{opt}^T U_{k, h+1}\)
7: \hspace{2em} \(U_{kh}(s) = \max_{a \in A(s)} Q_{kh}^{opt}(s, a)\)
8: \hspace{1em} end for
9: \hspace{1em} end for
10: return \(Q_{kh}^{opt}\)
11: end procedure

Now, moving on to the action selection paradigm (Algorithm 5) for the AIP algorithm. The algorithm takes as an input the empirical and the optimistic Q-functions \((Q_k\) and \(Q_{kh}^{opt}\) respectively) for episode \(k\) and the matrix \(N_{kh}\). To state the algorithm, we introduce the set of relatively frequently sampled actions for any state \(s\) at slot \(h\) in episode \(k\):

\[ D_{kh}(s) = \{a \in A(s); N_{kh}(s, a) \geq \log^2(N_{kh}(s))\}. \] (5.3)
Any action \( a \not\in D_{kh}(s) \) is referred to as relatively under sampled in state \( s \) at time slot \( h \) till episode \( k \). It is important to note that since we do not know the true MDP \( M \), we estimate \( \hat{M} \) based on the observations till that episode.

The algorithm basically selects an action to be played for state \( s \) at slot \( h \) in episode \( k \) as follows. We consider the action selection dynamics for episode \( k \). At time slot \( h \), if an action has not been played even a single time in a state \( s \) till the current episode, we play that action. Otherwise, we find the set of oversampled actions \( D_{kh}(s) \) at slot \( h \) and the set of optimal actions \( O \) obtained from the empirical estimate of the Q-function at state \( s \) and slot \( h \). We then construct two sets of actions \( \Gamma_1 \) and \( \Gamma_2 \). \( \Gamma_1 \) has actions which belong to \( O \) and may be under sampled in the next episode at slot \( h \) whereas \( \Gamma_2 \) contains actions that are optimal according to the optimistic estimate of the Q-function (we use Equation (5.2) for calculating \( Q_{kh}^{opt} \)). If all the actions in \( O \) may become under sampled in the next episode at slot \( h \), we select one of the actions from \( \Gamma_1 \) randomly otherwise we choose an action from \( \Gamma_2 \).

**Algorithm 5 AIP action selection**

1. procedure SELECTACTIONAIP\( (N_{kh}, Q_k, Q_{kh}^{opt}, s) \)
2. if \( \{ a \in A(s) : N_{kh}(s,a) = 0 \} \neq \emptyset \) then
3. \( W = \{ a \in A(s) : N_{kh}(s,a) = 0 \} \)
4. \( a_{kh} = \text{Choose a random action from set } W \)
5. else
6. \( D_{kh}(s) = \{ a \in A(s) ; N_{kh}(s,a) \geq \log^2(N_{kh}(s)) \} \)
7. \( O = \arg\max_{a \in A(s)} Q_{kh}(s,a) \)
8. \( \Gamma_1 = \{ a \in D_{kh}(s) ; N_{kh}(s,a) < \log^2(N_{kh}(s,a)) + 1 \} \)
9. \( \Gamma_2 = \{ a \in A(s) ; \arg\max_a Q_{kh}^{opt}(s,a) \} \)
10. if \( O == \Gamma_1 \) then
11. \( a_{kh} = \text{a random action from set } \Gamma_1 \)
12. else
13. \( a_{kh} = \text{a random action from set } \Gamma_2 \).
14. end if
15. end if
16. return \( a_{kh} \)
17. end procedure

Selecting an action from \( \Gamma_1 \) is in a way a forced selection since if we do not do that, in the next episode, none of the optimal actions in state
s at slot h will be visited unless one of the actions is visited now. Intuitively, selecting an action from \( \Gamma_1 \) is the exploitation phase where we are exploiting the best known actions from what we know about the true MDP using the empirical estimates. On the other hand, choosing an action from the \( \Gamma_2 \) corresponds to the exploration phase. It is important to note how the exploration-exploitation phases are inter-twined with each other in the algorithm at each step and the algorithm does not necessarily explore first and then commit (this kind of a strategy has been shown to be suboptimal in [9]). In every episode the empirical transition law can in principle be used to estimate an optimal policy. However, it is easy to see that this policy results in a positive probability of converging to a non-optimal solution [19]. The remedy to this is to keep taking seemingly inferior actions from time to time.

Now, let us revisit Equation (5.2). Since, here we are assuming that the reward is known, Equation (5.2) actually takes the following form:

\[
Q^{\text{opt}}_{kh}(s,a) = r(s,a) + \max_{p_{\text{opt}} \in \Delta_{s,a}} \left\{ p_{\text{opt}}^T U_{k,h+1} : KL(\hat{p}_k(s,a), p_{\text{opt}}) \leq \frac{\log(k+1)}{N_{kh}(s,a)} \right\} \tag{5.4}
\]

The use of a KL metric is advantageous in many ways, specially when comparing to the \( L^1 \) metric used in [11]. Filippi et. al. [7] discuss in their work how using a KL metric alleviates many issues which occur when using the \( L^1 \) metric specially in terms of the smoothness and continuity induced by the metric. More importantly, they show that the KL-optimistic model results from a trade-off between the relative value of the most promising state and the statistical evidence accumulated so far regarding its reachability. [7] provide an efficient procedure, based on one-dimensional line searches, to solve the linear maximization problem under KL constraints.

Equation (5.4) involves a maximization problem of the following form:

\[
\max_{p_{\text{opt}} \in \Delta_{s,a}} p_{\text{opt}}^T U \text{ s.t. } KL(\hat{p}, p_{\text{opt}}) \leq \epsilon. \tag{5.5}
\]

\( U \) and \( p_{\text{opt}} \) are vectors here. The radius of the neighborhood \( \epsilon \) controls the size of the confidence ball and \( \hat{p} \) is the empirical estimate of the transition dynamics till the current episode. This convex maximization problem is studied in the appendix of [7] leading to the efficient algorithm presented below. Detailed analysis of the Lagrangian
of Equation (5.5) done in [7] shows that the solution of the maximization problem essentially relies on finding roots of the function $f$ (that depends on the parameter $U$), defined as follows: for all $\nu \geq \max_{i \in \bar{Z}} U_i$, with $\bar{Z} = \{i : \hat{p}_i > 0\}$,

$$f(\nu) = \sum_{i \in \bar{Z}} \hat{p}_i \log(\nu - U_i) + \log \left( \sum_{i \in \bar{Z}} \hat{p}_i \nu - U_i \right).$$  \hspace{1cm} (5.6)$$

Here $\hat{p}_i$ and $U_i$ refer to the $i$th elements of the $\hat{p}$ and $U$ vectors respectively. In the special case where the most promising state $s$ has never been reached from the current state-action pair (i.e. $\hat{p}_s = 0$), the algorithm makes a tradeoff between the relative value of the most promising state $U_s$ and the statistical evidence accumulated so far regarding its reachability. Algorithm 6 provides a detailed procedure to find the optimistic transition estimate $p_{opt}$ in state $s$, action $a$ and slot $h$. The algorithm takes the vectors $U$ and $\hat{p}$ and the KL-ball radius $\epsilon$ as inputs.

Algorithm 6 Find $p_{opt}$ to maximize $U^T p_{opt}$ inside a KL ball

1: procedure MAXKLBALL($U$, $\hat{p}$, $\epsilon$) \hspace{1cm} \triangleright Calculates the optimistic transition vector estimate
2: \hspace{1cm} Let $Z = \{i : \hat{p}_i = 0\}$ and $\bar{Z} = \{i : \hat{p}_i > 0\}$
3: \hspace{1cm} Let $I^* = Z \cap \arg\max_i U_i$
4: \hspace{1cm} if $I^* \neq \emptyset$ and there exists $i \in I^*$ such that $f(U_i) < \epsilon$ then
5: \hspace{2cm} Let $\nu = U_i$ and $r = 1 - \exp(f(\nu) - \epsilon)$
6: \hspace{2cm} $\forall i \in I^*$, assign values of $p_{opt,i}$ such that $\sum_{i \in I^*} p_{opt,i} = r$
7: \hspace{2cm} For all $i \in Z/I^*$, let $p_{opt,i} = 0$
8: \hspace{1cm} else
9: \hspace{2cm} For all $i \in Z$, let $p_{opt,i} = 0$, Let $r = 0$
10: \hspace{2cm} Find $\nu$ such that $f(\nu) = \epsilon$ \hspace{1cm} \triangleright Use Newton’s method
11: \hspace{1cm} end if
12: \hspace{1cm} $\forall i \in \bar{Z}$, let $p_{opt,i} = \frac{(1-r)\tilde{q}_i}{\sum_{i \in \bar{Z}} \tilde{q}_i}$ where $\tilde{q}_i = \frac{\hat{p}_i}{\nu - U_i}$
13: \hspace{1cm} return $p_{opt}$
14: end procedure

5.1.2 KL-UCB++ inspired optimism

This algorithm that we propose is a reinforcement learning adaptation of the bandit version of the KL-UCB++ algorithm [13] which has been
shown to be simultaneously minimax and problem specific optimal by
the authors. The authors use a different KL-ball radius inside which
they calculate the inflated reward for each arm based on the empirical
estimates they collect for each arm. We adapt the KL-ball radius
discussed in [13] for the episodic RL case (RL-KLUCB++). The back-
ward induction equation for estimating the optimistic Q-function
\(Q_{kh}^{\text{opt}} \) for any state-action pair \((s, a)\) at slot \(h\) in episode \(k\) when the reward is
known can be written as:

\[
Q_{kh}^{\text{opt}}(s, a) = r(s, a) + \max_{p_{\text{opt}} \in \Delta_{s,a}} \{ p_{\text{opt}}^T U_{k,h+1} : KL(\hat{p}_k(\cdot|s,a), p_{\text{opt}}) \leq Z \} \tag{5.7}
\]

where \(Z\) is defined as:

\[
Z = \frac{1}{N_{kh}(s,a)} \log_+ \left( \frac{\sum_{a \in A(s)} N_{kh}(s,a)}{Y \cdot N_{kh}(s,a)} \left( \log_+ \left( \frac{N_{kh}(s)}{Y \cdot N_{kh}(s,a)} \right) + 1 \right) \right). \tag{5.8}
\]

Here \(\log_+(x) = \max(0, \log(x))\) and \(Y = HSA\).

**Algorithm 7** Calculate RL-KLUCB++ optimistic indices

1: procedure CALCULATE-RLLKLUCE++INDICES(\(\hat{p}, N_{kh}\))
2: Initialize \(U_{k,H+1} = 0\)
3: \(Y = HSA\)
4: for \(h = H, H-1, \ldots, 1\) do
5:     for \((s, a) \in S \times A\) do
6:         \(\epsilon = \frac{1}{N_{kh}(s,a)} \log_+ \left( \frac{\sum_{a \in A(s)} N_{kh}(s,a)}{Y \cdot N_{kh}(s,a)} \left( \log_+ \left( \frac{N_{kh}(s)}{Y \cdot N_{kh}(s,a)} \right) + 1 \right) \right) \triangleright \log_+(x) = \max(0, \log(x))\)
7:         \(p_{\text{opt}} = \text{MaxKLBall}(U_{k,h+1}, \hat{p}(\cdot|s,a), \epsilon)\)
8:         \(Q_{kh}^{\text{opt}}(s, a) = r(s, a) + p_{\text{opt}}^T U_{k,h+1}\)
9:         \(U_{kh}(s) = \max_{a \in A(s)} Q_{kh}^{\text{opt}}(s, a)\)
10:     end for
11: end for
12: return \(Q_k^{\text{opt}}\)
13: end procedure

In Algorithm 7, we provide an algorithm for the calculation of the
optimistic Q-function for episode \(k\) for the RL-KLUCB++ algorithm.
The algorithm only differs from Algorithm 4 in terms of the radius of
the KL-ball considered. For action selection, we use the greedy action selection paradigm (Algorithm 3) based on our optimistic Q-function estimates.

## 5.2 Unknown Rewards

Till now we have assumed that the reward function was known. The discussion in this section handles the case where the true reward function is unknown. The first approach was based on a UCB kind of a bonus. Here for each empirical estimate of the reward \( \hat{r}(s, a) \), we add a UCB bonus such that:

\[
\begin{align*}
    r_{opt}(s, a) &= \hat{r}(s, a) + \sqrt{2 \cdot \log(N_{kh}(s)) / N_{kh}(s, a)}.
\end{align*}
\]  

(5.9)

It is important to note that the bonus depends on the slot \( h \) and state \( s \). But, it has been shown in case of bandits that UCB kind of optimism gives a suboptimal regret as compared to the regret achieved by the KL-UCB kind of optimism [8]. The KL-UCB kind of an optimism in this case can be represented as:

\[
\begin{align*}
    r_{opt}(s, a) &= \sup\{\mu \in \Theta_{s,a} : KL(\hat{r}(s, a), \mu) \leq \epsilon_1\}.
\end{align*}
\]  

(5.10)

\( \epsilon_1 \) here depends on the algorithm in question (AIP or RL-KLUCB++). In case of AIP, \( \epsilon_1 = \log(k+1) / N_{kh}(s, a) \) and for RL-KLUCB++, \( \epsilon_1 \) is given by Equation (5.8). Since, we know that \( KL(a, b) \) is an increasing function for a fixed \( a \) and \( 0 \leq a \leq b \leq 1 \), \( r_{opt}(s, a) \) in this case is estimated using the Newton’s method. To use these inflated estimates of the reward, we just need to replace \( r(s, a) \) in Algorithms 4 and 7 by \( r_{opt}(s, a) \).

It was observed during experiments that KL-UCB kind of optimism for estimating the optimistic reward function performs better than the UCB kind of optimism in terms of the regret, hence we consider the KL-UCB kind of optimism for the case where the rewards are unknown from here on.
In this chapter we describe the existing algorithms for comparison against the algorithms suggested in this thesis. These algorithms broadly belong to two classes - the OFU based algorithms and the algorithms that are motivated by Bayesian treatment of the problem. From the OFU side, we consider the Upper Confidence Bound - Value Iteration (UCB-VI) algorithm \[3\] with the Bernstein-Freedman bonus and the Kullback Leibler measure based Upper Confidence Reinforcement learning (KL-UCRL) algorithm \[7\]. From the Bayesian side, we discuss the Posterior Sampling Reinforcement Sampling (PSRL) algorithm suggested by Osband et al. \[18\].

### 6.1 KL-UCRL algorithm

In \[7\] Filippi et al. consider communicating MDPs in an ergodic setting, i.e., MDPs such that for any pair of states \((s, s')\), there exists policies under which \(s'\) can be reached from \(s\) with positive probability (communicating MDPs). KL-UCRL is an optimistic algorithm based upon the UCRL algorithm \[11\] that works in episodes of increasing lengths. This algorithm first build confidence balls for the reward and transition probabilities, and then identifies an optimistic Q-function. We modify the algorithm to use in case of the episodic setting that we are investigating.

The algorithm is based on the idea of the agent following the optimal policy for a surrogate model, named optimistic model, which is close enough to the former but leads to a higher long-term reward. Algorithm 8 lists the steps involved for calculation of the optimistic Q-function.
Q-function for episode $k$. The algorithm takes the empirical estimate of the transition and reward dynamics till episode $k$ ($\hat{p}$ and $\hat{r}$, respectively) and $N_k$ which contains the information about the number of times the tuple $(s,a)$ is observed till episode $k$ as inputs. For action selection, we use the greedy action selection in Algorithm 3 based on our optimistic Q-function estimates.

Algorithm 8 Calculate KL-UCRL optimistic indices

1: procedure $\text{CALCULATEKL-UCRLINDICES}(\hat{p}, \hat{r}, N_k)$
2: $U_{k,H+1} = 0$
3: for $h = H, H - 1, \ldots, 1$ do
4: for $(s,a) \in S \times A$ do
5: $\epsilon = \frac{C_P}{\sqrt{N_k(s,a)}}$
6: $r_{opt}(s,a) = \hat{r}(s,a) + \frac{C_R}{N_k(s,a)}$
7: $p_{opt} = \text{MaxKLBall}(U_{k,h+1}, \hat{p}(\cdot | s,a), \epsilon)$
8: $Q_{kh}^{opt}(s,a) = r_{opt}(s,a) + p_{opt}T_{kh}$
9: $U_{kh}(s) = \max_{a \in A(s)} Q_{kh}^{opt}(s,a)$
10: end for
11: end for
12: return $Q_k^{opt}$
13: end procedure

Here $C_R$ and $C_P$ are defined in the paper as:

$$C_R = \sqrt{\frac{\log(4SA \log(T)/\delta)}{1.99}},$$

and

$$C_P = S \left( B + \log(B + \frac{1}{\log(T)}) \left[ 1 + \frac{1}{B + \frac{1}{\log(T)}} \right] \right).$$

The authors define $B = \log(\frac{2eS^2A \log(T)}{\delta})$. This basically puts a bound on how far our optimistic estimates can be from the empirical estimates of the reward and transition functions respectively. Here $\delta$ is an adjustable parameter which is used to specify the confidence with which the regret bounds derived in [7] holds. It is important to note that the calculation of the optimistic transition probability vector in the algorithm takes the same form as discussed in the previous chapter (Equations (5.5) and (5.6)). Moreover, to handle unknown rewards,
as suggested in the original algorithm, we simply add a bonus to the empirical reward estimate ($\hat{r}(s,a)$).

Like UCRL, the KL-UCRL algorithm has also been shown to obtain near optimal regret bounds (logarithmic). This algorithm tries to mitigate the drawbacks mentioned for the UCRL algorithm by using the KL pseudo-distance instead of the $L_1$ metric. As compared to the UCRL algorithm, the regret bounds and the numerical complexity are comparable whereas a significant performance improvement is achieved using the KL metric. Theorem 1 in [7] provides an upper bound for regret for the KL-UCRL algorithm for an ergodic RL problem. It states that the regret for the KL-UCRL algorithm is $\tilde{O}(D(M)S\sqrt{AT})$. Here, $D(M)$ represents the diameter of the MDP $M$ under consideration.

### 6.2 UCBVI-BF algorithm

UCB-VI [3] is an extension of Value Iteration, guaranteeing that the resulting value function is a (high-probability) upper confidence bound (UCB) on the optimal value function. This algorithm unlike the UCRL and the KL-UCRL algorithms directly build a confidence ball for the Q-function based on the empirical estimates of the model (the transition probabilities and the reward estimates). Algorithm 9 shows how the optimistic Q-function is calculated for episode $k$. At the beginning of episode $k$, the algorithm computes state-action values using empirical transition kernel and reward function. In step $h$ of backward induction to update $Q_{kh}(s,a)$ for any $(s,a,h)$, an optimistic bonus $b_{kh}(s,a)$ to the value. There are 2 kinds of bonuses that the paper discusses (the 2 variants of the algorithm corresponding to each bonus called UCBVI-CH and UCBVI-BF). We consider the bonus based on the Bernstein-Freedman “exploration bonuses” (Algorithm 10) that use the empirical variance of the estimated values at the next states (to improve scaling in $H$ as compared to the Chernoff-Hoeffding’s based exploration bonus). In Algorithm 9, $N_{kh}(s,a)$ denotes the number of times we observe state-action pairs $(s,a)$ at slot $h$ till episode $k$ and $N_k(s,a)$ denotes the number of times we observe state-action pairs $(s,a)$ till episode $k$. $\delta$ is an adjustable parameter which is used to specify the confidence with which the regret bounds derived in [3] holds. $Q_{k-1}^{opt}$ is the optimistic estimate of the Q-function for the previous episode. For action
selection, we use the greedy action selection paradigm (Algorithm 3) based on our optimistic Q-function estimates.

Algorithm 9 UCBVI Q-function computation

```
1: procedure UCBVI(Q^opt_{k−1}, \bar{p}, \hat{r}, N_k, N_{kh})
2:    Initialize U_{k,H+1} = 0
3:    Q^opt_k = H
4:    for h = H, H−1, . . . , 1 do
5:        for (s, a) ∈ \{(S × A) : N_k(s, a) > 0\} do
6:            b_{kh}(s, a) = Bonus(\bar{p}, U_{k,h+1}, N_k(s, a), N_{kh}(s, a))
7:            Q^opt_{kh}(s, a) = \min(H, \max_{\hat{r} \in A(s)} Q^opt_k(s, a) + b_{kh}(s, a) + \hat{r}(s, a)^T U_{k,h+1})
8:        end for
9:    end for
10:   return Q^opt_k
11: end procedure
```

The idea is that if we had knowledge of the optimal utility U^*_h, we could build tight confidence bounds using the variance of the optimal utility function at the next state in place of the loose bound of H. Since however U^*_h is unknown, here a surrogate empirical variance of the estimated values is used. As more data is gathered, this variance estimate will converge to the variance of U^*_h. To make sure that the estimates of U_{kh} are optimistic (i.e., that they upper bound U^*_h) at all times an additional bonus (last term in b_{kh}(s, a)) is added, which guarantees that the variance of U^*_h is upper bounded. In Algorithm 10, N_k,h+1(s) is the count of the occurrences of state s at slot h + 1 up to episode k.

Algorithm 10 Bernstein-Freedman bonus calculation

```
1: procedure BONUS(\bar{p}_k, U_{k,h+1}, N_k, N_{kh})
2:    L = log(5SAT/\delta)
3:    b = \sqrt{\frac{8LVar_{Y \sim \hat{p}_k(\cdot|s,a)}(U_{k,h+1}(Y))}{N_k(s,a)}} + \frac{14HL}{N_k(s,a)} \left[ \min_{\hat{r} \in A(s)} \hat{p}_k(\cdot|s,a) \sum_{\hat{r}} \frac{\log^{2} H^{3} S A^{2} T_{h+1}}{N_{kh}(\hat{r}, h+1)} \right]
4:    return b
5: end procedure
```

Theorem 2 in [13] states that the bound for the regret for UCBVI-BF algorithm is \( \tilde{O}(\sqrt{HSA}) \) for \( T > H^3S^3A \) and \( SA \geq H \) which matches
the minimax regret lower bound \( \Omega(\sqrt{HSA T}) \) which we derived in Chapter 2 up to logarithmic factor.

### 6.3 PSRL algorithm

PSRL is a sampling based algorithm which proceeds in repeated episodes of known duration. At the start of each episode, PSRL updates a prior distribution (calculated from the history variable \( H \)) over Markov decision processes and takes one sample \( \tilde{M} \) from this posterior \( f \). PSRL then follows the policy that is optimal for this sample during the episode. The optimal policy is calculated using the dynamic programming paradigm from the sampled reward function and the transition dynamics respectively. PSRL always selects policies according to the probability they are optimal. Uncertainty about each policy is quantified in a statistically efficient way through the posterior distribution. We express the prior in terms of Dirichlet and normal-gamma distributions over the empirical estimate of transitions and rewards respectively. Algorithm 11 describes the paradigm. Additionally, Theorem 1 in \[18\] establishes an \( \tilde{O}(HSA/AT) \) bound on expected regret for the PSRL algorithm.

**Algorithm 11 Posterior Sampling Reinforcement Learning**

1: procedure PSRL  
2: \( H = \emptyset \)  
3: for \( k = 1 : K \) do  
4: \( \tilde{M} \sim f(H) \)  
5: \( Q_k = DP(\tilde{M}) \)  
6: for \( h = 1 : H \) do  
7: \( a_{kh} = \arg\max_{a \in A(s)} Q_k(s, a) \)  
8: \( s_{k,h+1} = \text{getNextState}(s_{kh}, a_{kh}) \)  
9: update \( H = H \cup (s_{kh}, a_{kh}, s_{k,h+1}) \)  
10: end for  
11: end for  
12: end procedure

In this chapter we discussed three existing algorithms from the OFU and the Bayesian sides. In the next chapter we run experiments to compare the performance of the algorithms on a few toy MDP’s in an episodic setting.
Chapter 7

Results

In this chapter, we run numerical experiments on three different MDP’s, two of which are commonly used as benchmarks (the River Swim environment and the Six Arm environment) \cite{21} and a randomly generated MDP to compare the performance (in terms of regret) of the algorithms. It was ensured that the mean rewards were normalized in the interval $[0, 1]$ for each of the toy MDP’s. We consider two scenarios for our regret plots - (1) the case where the mean rewards are known for each state-action pair and (2) the case where the mean rewards are unknown for each state-action pair. In the second case, the rewards for each state-action pair are drawn from a Bernoulli distribution with the mean being the values specified in the figure’s for the respective MDP’s. For the UCBVI-BF algorithm and the KL-UCRL algorithms, we only plot regrets for the case where the rewards are unknown since it was observed that the regret for these algorithms is order of magnitudes greater than the PSRL, AIP or the RL-KLUCB++ algorithm and it was irrelevant to plot the case where the rewards are known. At all times we assume that the transition dynamics are unknown to the agent. The parameter $\delta$ was set to 0.05 for the KL-UCRL and the UCBVI-BF algorithms. 100 simulations for each algorithm were run on each MDP. This was done to facilitate us to plot the 95% confidence interval for the regret plots. It is important to state here that to run multiple simulations of the algorithms for plotting the mean regrets incurred by the respective algorithms, the Joblib library in python was used. The library facilitates us to use the multithreading construct in python.

We also compare algorithms in terms of the execution time to draw
a tradeoff between their performance and the cost of computation for the optimal policy by the algorithm.

### 7.1 Regret

#### 7.1.1 River Swim environment

The River Swim environment consists of six states. The agent starts from the left side of the row and, in each state, can either swim left or right ($A = 2$). Swimming to the right (against the current of the river) is successful with probability 0.35; it leaves the agent in the same state with a high probability equal to 0.6, and leads him to the left with probability 0.05 (see Figure 7.1). On the contrary, swimming to the left (with the current) is always successful. The agent receives a small reward when he reaches the leftmost state, and a much larger reward when reaching the rightmost state – the other states offer no reward. This MDP requires efficient exploration procedures, since the agent, having no prior idea of the rewards, has to reach the right side to discover which is the most valuable state-action pair. The episode length ($H$) was set to 12.

![Figure 7.1: The River Swim MDP](image)

Figure 7.1 shows the average regret plots (with 95% confidence intervals also plotted) for the UCBVI-BF, KL-UCRL and the simple dynamic programming (using the empirical estimates of the reward and transition dynamics) algorithms on the River Swim MDP. The reward and the transition dynamics are unknown in this case. We also plot $\sqrt{HSAT}$ to represent the minimax bound we derived in Chapter 4.
CHAPTER 7. RESULTS

Figure 7.2: Regret plots: River Swim environment (unknown reward and transition dynamics)

7.1.2 Six Arm environment

The Six Arm environment consists of seven states, one of which (state 0) is the initial state. From the initial state, the agent may choose one among six actions: the action $a \in \{1, \ldots, 6\}$ ($A = 6$) leads to the state $s = a$ with probability $p_a$ (see Figure 7.4) and let the agent in the initial state with probability $1 - p_a$. From all the other states, some actions deterministically lead the agent to the initial state while the others leave it in the current state. Staying in a state $s \in \{1, \ldots, 6\}$, the agent receives a reward equal to $R_s$ (see Figure 7.4), otherwise, no reward is received. The episode length was set to 14.

Figure 7.3 shows the average regret plots with 95% confidence interval for the PSRL, RL-KLUCB++ and the AIP algorithm. For these algorithms we consider both the scenarios, when the reward is known and when the reward is unknown (the transition dynamics is always unknown).
(a) unknown transition dynamics

(b) unknown reward and transition dynamics

Figure 7.3: Regret plots: River Swim environment

Similar experiments as in the case of the River Swim MDP were carried out for the Six Arm MDP as well. Figure 7.5 shows the average regret plots (with 95% confidence interval) for the UCBVI-BF, KL-UCRL and the simple dynamic programming (using the empirical estimates of the reward and transition dynamics) algorithms on the
CHAPTER 7. RESULTS

Six Arm MDP. The reward and the transition dynamics are unknown in this case.

Figure 7.5: Regret plots: Six Arm environment (unknown reward and transition dynamics)

Figure 7.6 shows the average regret plots with 95% confidence interval for the PSRL, RL-KLUCB++ and the AIP algorithm. For these algorithms again two scenarios are considered, when the reward is known and when the reward is unknown.
7.1.3 Random environment

In addition to the benchmark environments, a generator of sparse environments was used to create a 5-states and 3-actions environments with random rewards in $[0, 1]$. In these random environments, each
state is connected with 4 other random states for each action (with transition probabilities drawn from a Dirichlet distribution). The episode length was set to 10.

We reproduced the same experiments as in the case of the previous environments. Figures 7.7 and 7.8 show the average regret plots for the algorithms.

7.2 Runtime

We also compare the computation time for a single run of the PSRL, RL-KLUCB++ and the AIP algorithm. This is important since we want to know whether the improvements achieved in terms of mean regret and the variance of regret has a tradeoff in terms of the time required to compute the optimal policy. Table 7.1 lists the times in seconds for a single run of each algorithm to run through $1.5 \times 10^6$ episodes for each of the environments described in the previous section (the reward and the transition dynamics are unknown).

To evaluate the running time, we use a single machine since the performance can significantly vary across different platforms. In particular, the machine has the following specifications. We use a 64 bit
(a) unknown transition dynamics

(b) unknown reward and transition dynamics

Figure 7.8: Regret plots: Random environment
<table>
<thead>
<tr>
<th>MDP</th>
<th>AIP</th>
<th>RL-KLUCB++</th>
<th>PSRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>River Swim MDP</td>
<td>2383.38 sec</td>
<td>1734.82 sec</td>
<td>823.92 sec</td>
</tr>
<tr>
<td>Six Arm MDP</td>
<td>5487.62 sec</td>
<td>4126.37 sec</td>
<td>1756.87 sec</td>
</tr>
<tr>
<td>Random MDP</td>
<td>3614.21 sec</td>
<td>2591.95 sec</td>
<td>1348.78 sec</td>
</tr>
</tbody>
</table>

Table 7.1: Run time for a single simulation of algorithms on different MDP’s

machine with Intel Xeon(R) Silver 4110 CPU @ 2.10 GHz×16. The machine had a 48 GB RAM.

7.3 Discussion

From the results of the numerical experiments on the three MDP’s, it is easy to infer that the UCBVI-BF and the KL-UCRL algorithms are sub-optimal in terms of the average regret when compared to algorithms like PSRL, AIP and RL-KLUCB++. In some cases (River Swim MDP and Random MDP), these algorithms even perform worse than a simple dynamic programming based paradigm using the empirical estimates of the transition and reward dynamics. This can be attributed to the large exploration bonuses used in both the algorithms ($C_P$ and $C_R$ in case of the KL-UCRL algorithm and the Bernstein-Freedman bonus in case of the UCBVI-BF algorithm). It is important to note here that $C_P$ and $C_R$ define the radius of the KL-ball inside which we try to find the optimistic estimates of the transition and the reward dynamics respectively. The KL-UCRL algorithm also shows a non-convergent behavior because of the the large values of $C_P$ and $C_R$.

Additionally, just changing the radius of the optimistic KL-ball to a value inspired by the KL-UCB++ algorithm in the KL-UCRL algorithm [13], we get an algorithm (RL-KLUCB++) which gives a comparable performance in terms of average regret to the PSRL algorithm but still the nature of the algorithm remains a bit non-convergent (the regret plot does not flatten out perfectly as $T$ increases). Though, if we compare the variance of the regrets obtained by the two algorithms (PSRL and RL-KLUCB++), the RL-KLUCB++ algorithm trumps the PSRL algorithm which being a sampling based algorithm has a high variance. This is the case both, when the reward dynamics is known and unknown. These results reinforce the need to be optimistic in the right
amount during the exploration phase of the algorithm. Too much optimism can be harmful in practice, extending the exploration phase far beyond the point where the algorithm has enough evidence to choose the optimal action in a given state and slot.

On the other hand, the AIP algorithm achieves a significantly lower average regret in comparison to the PSRL algorithm when the reward is known. This behavior of the AIP algorithm can be attributed to the way in which the optimistic transition dynamics is calculated for the AIP algorithm. In case of the PSRL algorithm, while sampling from the prior (depends only on the empirical estimate of the transition dynamics), the transitions probabilities to states which have not been visited much are assigned lesser probabilities most often and hence in cases where we need to explore a bit to find the optimal policy, the PSRL algorithm lags behind. On the other hand, for the AIP algorithm, the way we estimate the optimistic transition dynamics using Algorithm 6 in the special case where the most promising state has never been reached from the current state-action pair, the algorithm makes a trade-off between the relative value of the most promising state and the statistical evidence accumulated so far regarding its reachability. Though, when the reward is unknown, this advantage of the AIP algorithm is mitigated and we achieve comparable performance to the PSRL algorithm in terms of the average regret, the variance of the PSRL algorithm still remaining quite high compared to the AIP algorithm. A similar line of reasoning explains the reason for RL-KLUCB++ algorithm achieving less average regret for the known reward case in the River Swim and the Six Arm environment.

Shifting our focus on the runtime’s for the PSRL, AIP and the RL-KLUCB++ algorithms (Table 7.1). Since PSRL is a sampling based algorithm, the algorithm calculates the Q-function for episode $k$ using the sampled estimates of the transition and the reward dynamics and hence the most computational heavy part of the PSRL algorithm is the dynamic programming paradigm. On the other hand the RL-KLUCB++ algorithm and the AIP algorithm are required to calculate an optimistic estimate of the Q-function. Hence, both the algorithms depend on the optimistic estimates of the transition and reward dynamics. Both, the calculation of an optimistic estimate of the reward function and the optimistic estimate of transition dynamics involves finding roots using the Newton’s method. Hence, RL-KLUCB++ and the AIP algorithm demonstrate a greater runtime as compared to the
PSRL algorithm. Moreover, for the AIP algorithm, an empirical estimate of the Q-function is also required to find the set of optimal actions in state $s$ and slot $h$ which run the risk of being undersampled in the next episode at slot $h$. Hence, the AIP algorithm requires the maximum runtime among the three algorithms that have been shown to perform well in our numerical experiments.

The respective runtime’s for the algorithms and the results from the numerical experiments emphasize the fact that AIP is a feasible algorithm from the OFU side that betters the PSRL algorithm in terms of the average regret (specially when the reward dynamics are known) while completing in reasonable amount of time. And even for the case where the reward dynamics are not known, the low variance of the AIP algorithm makes it a better algorithm than the PSRL algorithm to deploy.
Chapter 8

Conclusion

Results of the empirical experiments conducted reveal that the algorithms suggested in this thesis (AIP and RL-KLUCB++) have lower average regret compared to the PSRL algorithm when the reward dynamics are known and have comparable regret when the reward dynamics is unknown. In all experiments we assume that the transition dynamics are unknown. Both, AIP and RL-KLUCB++ have a lesser variance than the PSRL algorithm. The AIP and the RL-KLUCB++ algorithms to our knowledge are the only algorithms inspired by the OFU principle which are comparable to the PSRL algorithm in terms of average regret. Other OFU inspired algorithms like KL-UCRL and UCBVI-BF were shown to have a sub-optimal average regret with respect to the average regret incurred by the PSRL algorithm.

There is a tradeoff though in terms of the computation times when it comes to using the AIP or the RL-KLUCB++ algorithms. These algorithms have a higher computation time than the PSRL algorithm and the time highly depends on the number of actions and the episode length, hence these two parameters must be taken into consideration when deploying the algorithms. It is important to state here that for a given episode $k$, slot $h$ and state $s$, calculating $Q_{kh}(s, a)$ and $Q_{kh}^{opt}(s, a)$ for any action $a$ is independent of any other action and hence the operation can be easily parallelized. We also showed in the experiments that too much optimism (in the case of KL-UCRL and UCBVI-BF) also yields suboptimal regret.
8.1 Future Work

We compared various algorithms through empirical experiments against the AIP and the RL-KLUCB++ algorithm. Though, to fully understand the scaling of regret in terms of $S, A, T$ and $H$, a thorough theoretical analysis should be conducted. Such an analysis should also provide upper and lower bounds on the regret of our algorithm. We expect the theoretical analysis to not be very different compared to the one conducted in [5] and [13], respectively. We expect the AIP algorithm to attain a upper regret bound that matches the problem specific bound derived in Chapter 4 whereas for the RL-KLUCB++ algorithms, we expect the regret upper bound to match both, the problem specific bound and the minimax bound derived in Chapter 4.

Moreover, the count-based methods discussed in the thesis are difficult to employ in cases when the state space is continuous or is very large. Tang et. al. [22] describe how we can employ the count based RL methods to environments with continuous state spaces by designing a hashing function which maps the continuous states to discrete values. Hashing can be used in such cases with the methods suggested in the thesis. Further, we assume that episode length is fixed for each episode which may not be the case in scenarios like gameplay. In such cases, the Q-function can be estimated using neural nets as function approximaters [14] and then using those values to calculate the optimistic Q-function.

Another drawback that the count-based algorithms face is the high computational costs of the algorithms (specifically true for OFU based algorithms). The traditional approaches do not take into account the intrinsic structure of the MDP. For instance, in many situations, the transition distributions are not arbitrarily different between two state action pairs. Ok et. al. [15] devise a Directed Exploration Learning (DEL) based algorithm that matches the derived regret lower bounds for ergodic setting. They further provide a simplified algorithm for Lipschitz MDPs, and show that the simplified version is still able to efficiently exploit the structure. The authors in the paper show that the regret under the version of DEL exploiting the Lipschitz structure does not seem to grow with the number of states, A paradigm on the similar lines can be designed for the episodic case as well. Exploiting structure in RL remains to be a promising avenue specially to reduce the sample complexity and computation time of RL algorithms.
8.2 Ethical and Societal aspects

Given the vast number of RL algorithms that will be used in the future in industry, video games, robotics, and research, the moral stakes are high. Hence, it is important that scientists and altruists work towards a more humane approach to reinforcement learning. Present-day RL algorithms have minimal emotional self-awareness e.g. in the effort to explore more, a robot may cause damage to its surroundings. Though, the algorithms and the results discussed in the thesis are more of fundamental and theoretical nature, on large scales they may begin to add up to something significant. Hence, it is important that before employing these algorithms, we take into consideration its societal and ethical implications. In addition, as the algorithms are refined and combined with higher-level cognition, they will become more morally urgent. Moreover, since in the thesis we run experiments on toy MDP’s, the topic of data privacy and data ownership does not arise here.
Bibliography


