LEARNING TO MAKE DECISIONS WITH INCOMPLETE INFORMATION: REINFORCEMENT LEARNING, INFORMATION GEOMETRY, AND REAL-LIFE APPLICATIONS

DEBABROTA BASU
(B. Eng. (Hons.), Jadavpur University)

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2018

Supervisors:
Associate Professor Stéphane Bressan, Main Supervisor
Professor Pierre Senellart, École Normale Supérieure, Co-Supervisor

Examiners:
Professor Tan Kian Lee
Dr. Jonathan Mark Scarlett
Dr. Cappé Olivier, French National Centre for Scientific Research (CNRS)

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

DEBABROTA BASU

24 November 2018

Acknowledgements

None of a human's endeavour is a journey in isolation. Rather, like himself/herself, all the milestones traversed by a human is a collective construction. Thus, we become a human instead of being born into it. Similarly, I have sailed across my journey to the doctorate degree by constant support and influence of several people and institutions. I verbalise my gratefulness to some of them, and for the rest, let there be my silent gratitude and respect.

I would like to pay regards to my parents, Ma and Baba, to believe in myself, ideas, and ideals. Thanks for ushering the boon of knowledge and education on me under the most adverse conditions of life. I wish that I could share this with my late Dida and Mama for loving unconditionally and standing as the silent pillars. I would like to thank Munia for standing by me throughout the gravest failures and the brightest memories, and waiting patiently till it ended.

I convey due respect and gratitude to my supervisors Prof. Sté phane Bressan and Prof. Pierre Senellart. I would like to thank Prof .Sté phane for your guidance and advice throughout the wavy PhD journey. I would like to thank Prof. Pierre for teaching the academic skills and being the inspiration. It has been an enriching experience to be your supervisee that has transformed a lot of aspects personally and academically. I would also thank all the present and ex-members of our research group Ashish, Naheed, Agus, Remmy, Liu Qing, Fajrian, and Yann to be the brothers in arms. I wish all of you to succeed in your academic journeys with flying colours. I also like to thank all my collaborators of Prof. haibo Chen's group is SJTU, Prof. Talel's group

in Telecom Paristech, and Prof. Giulia's group in Arizona State University for their help and support to fulfil the missions that we shared together.

No journey is complete without friends. Thus, last but not the least, I would like to thanks my brothers Rahulda, Tarunda, Kaustavda, Subhabratada, Subhasankarda, Abhishekda, Arka, and Subhodip. Thanks for always standing with each other when everything seemed to fall apart and when everyone lost hope on themselves. In a distant land, you have built my shelter from the storm.

I would also like to convey my respect to my poet friends of Migrant Writers' Singapore, my second home that taught me the meaning of freedom and resistance- Jadavpur University, and other academic instiutions that kindly hosted me- ENS, NUS, NTU, ISI, SJTU, and ETH-Zurich.

To the endless journey to reach knowledge, freedom, and the beauty of geometry.

– Melencolia I by Albrecht Dürer.

Contents

Abstract

Reinforcement learning is a machine learning framework where an agent learns to make decisions by interaction with an uncertain environment. Effective information accumulation, processing, and learning lead to efficient decision making in reinforcement learning. This framework requires a reward function to assess the goodness of a decision and knowledge of the underlying dynamics of the environment to manoeuvre the long-term effect of learning and decision making. In real-world applications, the reward function and/or the underlying dynamics of the decision making process are often not known a priori. The agent tries to learn these information through exploration while she also tries to reach a sequence of decisions returning maximal reward through exploitation of the present knowledge. Thus, balancing exploration-exploitation while efficiently managing the large space of plausible decisions takes the central stage in designing a reinforcement learning algorithm.

In this thesis, we investigate scenarios of reinforcement learning where the reward function or the underlying process dynamics are not accurately known to the agent. We approach the problem in three settings with increasing complexity. Each setting involves a separate theoretical analysis and a real-life application for experimental validation.

In the first setting, we develop two algorithms, COREIL [Basu et al., 2015a,b] and rCOREIL [Basu et al., 2016], that learn the reward function simultaneously as they take decisions in a known-transition, unknown-reward Markov decision process and apply them to the problem of self-driving database management systems.

In the second setting, we develop an algorithm, Megh [Basu et al., 2017b,c], that works for the known-cost and unknown-transition Markov decision processes. It learns the transition matrix while taking decisions on-the-go. We apply this for live VM migration in medium-scale data centres and prove that it saves 14% of operation cost while completing each step in approximately 1 second.

In the third setting, we develop an information-geometric approach to the case where we do not know reward and transition functions. We generalise the representations of reward and transition functions, and consider them to be unknown distributions. We develop a framework, BelMan [Basu et al., 2018c], and theoretically analyse it for the multi-armed bandits. This framework provides a unified algorithm and analysis for pure exploration, exploration–exploitation and risk-aware scenarios for bandits. We also show an application of BelMan for online scheduling of jobs in a multiple-queue, multiple-server system with known arrival rates and unknown service rates. We prove that BelMan theoretically achieves asymptotic convergence while experimentally outperforms the state-of-the-art algorithms for Bernoulli service rates.

Finally, we sketch an extension of the information geometric approach to Markov decision processes with unknown transition functions and reward distributions [Basu et al., 2018d]. This extension builds a link among the proposed approach, linearly solvable Markov decision processes, and curiosity driven reinforcement learning. This analysis motivates further investigation of the exploration-exploitation trade-off in variants of reinforcement learning and their applications.

List of Contributions

The list of publications that I have worked on and am working during the period of PhD is documented here. The research works are broadly categorised as the ones incorporated to this thesis and the ones not incorporated in it. Under each of these categories, we present the published papers, the paper under review in conferences and journals, and the papers we are planning to submit by the time of thesis defence.

- Research works incorporated in the thesis:
  - Published:
    1. **Cost-model oblivious database tuning with reinforcement learning.** Debabrota Basu, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. In International Conference on Database and Expert Systems Applications, pages 253–268. Springer, 2015.
    2. **Apprentissage par renforcement pour optimiser les bases de donnéees indépendamment du modèle de coût.** Debabrota Basu, Qian Lin, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. In Proceedings of BDA. 2015.
    3. **Regularized cost-model oblivious database tuning with reinforcement learning.** Debabrota Basu, Qian Lin, Weidong Chen, Hoang Tam Vo, Zihong Yuan, Pierre Senellart, and Stéphane Bressan. In Transactions on Large-Scale Data and Knowledge-Centered Systems, volume 28, pages 96–132, 2016.

4. **Learn-as-you-go with Megh: Efficient live migration of virtual machines.** Debabrota Basu, Xiayang Wang, Yang Hong, Haibo Chen, and Stéphane Bressan. In IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pages 2608–2609. IEEE, 2017.
5. **BelMan: Bayesian bandits on the belief–reward manifold.** Debabrota Basu, Pierre Senellart, and Stéphane Bressan. arXiv preprint arXiv:1805.01627, 2018.
6. **Learn-as-you-go with Megh: Efficient live migration of virtual machines.** Debabrota Basu, Xiayang Wang, Yang Hong, Haibo Chen, and Stéphane Bressan. Accepted in IEEE Transactions on Parallel and Distributed Systems (TPDS), 2018.

– Under review:

1. **BelMan: An information geometric approach to multi-armed bandit problems.** Debabrota Basu, Pierre Senellart, and Stéphane Bressan. October, 2018. Submitted to International Conference on Artificial Intelligence and Statistics (AISTATS).

– Working papers:

1. **BelMan in queue: An information geometric approach to queueing bandits with general distributions.** Debabrota Basu, Giulia Pedrielli, Pierre Senellart, and Stéphane Bressan. 2018. Submitting to ACM SIGMETRICS.
2. **MoQ: A model oblivious learning algorithm to sequentially optimize speed of a vessel under uncertain weather.** Debabrota Basu, Abdul Rahman, and Stéphane Bressan. 2018. Submitting to IEEE Transactions on Intelligent Transportation Systems.
3. **An information geometric analysis of exploration–exploitation trade-off in reinforcement learning.** Debabrota Basu, Pierre Senel-

lart, and Stéphane Bressan. 2019. Submitting to International Conference on Machine Learning (ICML).

- Research works not incorporated in the thesis:
  - Published:
    1. **Top-k queries over uncertain scores.** Qing Liu, Debabrota Basu, Talel Abdessalem, and Stéphane Bressan. In Proceedings of OTM 2016 Conferences:CoopIS, C&TC, and ODBASE 2016, pages 245262, 2016.
    2. **How to find the best rated items on a Likert scale and how many ratings are enough.** Qing Liu, Debabrota Basu, Shruti Goel, Talel Abdessalem, and Stéphane Bressan. In International Conference on Database and Expert Systems Applications, pages 351359. Springer, 2017.
    3. **Sequential vessel speed optimization under dynamic weather conditions.** Debabrota Basu, Giulia Pedrielli, Weidong Chen, Szu Hui Ng, Loo Hay Lee, and Stéphane Bressan. In International Maritime-Port Technology and Development Conference. 2017.
    4. **An investigation of instability in differential privacy mechanisms for regularised linear regression.** Ashish Dandekar, Debabrota Basu, and Stéphane Bressan. In International Conference on Database and Expert Systems Applications. Springer, 2018.
  - Under review:
    1. **Differential privacy at risk.** Ashish Dandekar, Debabrota Basu, and Stéphane Bressan. October, 2018. Submitted to International Conference on Extending Database Technology (EDBT).
    2. **Graph topological data analysis using landmarks.** Naheed Anjum Arafat, Debabrota Basu, and Stéphane Bressan. November, 2018. Submitted to International Conference on Database Systems for Advanced Applications (DASFAA).

3. **Evaluation of differentially private non-parametric machine learning as a service.** Ashish Dandekar, Debabrota Basu, and Stéphane Bressan. November, 2018. Submitted to International Conference on Database Systems for Advanced Applications (DASFAA).

– Working paper:

1. **TL-NQS: Augmenting neural quantum states with transfer learning.** Remmy Augusta Menzata Zen, Debabrota Basu, My Duy Hoang Long, and Stéphane Bressan. 2018. Submitting to Neural Computation.

List of Tables

List of Figures

List of Notations

Notation	Description
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^{+}$	The set of positive real numbers
$\mathbb{R}^{\geq 0}$	The set of non-negative real numbers
$\mathbb{R}^{\leq 0}$	The set of non-positive real numbers
$\mathbb{Z}$	The set of integers
$\mathbb{N}$	The set of natural numbers $= \{1, \ldots\}$
$\mathbb{N}_0$	The set of natural numbers and zero, i.e. $\{0, 1, \ldots\}$
$[n]$	$\{1, 2, \ldots, n-1, n\}$
$\text{Ber}(X; \mu)$	Bernoulli distribution of random variable X with probability of success μ
$\text{Beta}(X; \alpha, \beta)$	Beta distribution of random variable X with parameters α and β
$\lvert A \rvert$	Cardinality, i.e. the number of elements, of a set A
$\lceil x \rceil$	The least integer greater than or equal to x
δ_{x_0}	Dirac's delta function evaluated at x_0
$\text{Dom}(f)$	The set of inputs on which f is defined
$\mathbb{E}$	Expectation of a random variable
$\lfloor x \rfloor$	The greatest integer less than or equal to x
$\Gamma(x)$	Gamma function $\Gamma(x) = \int_0^\infty z^{x-1} \exp(z) dz$
$\nabla f(x)$	Gradient of a function f evaluated at x
$\nabla^2 f(x)$	Hessian of a function f evaluated at x

Notation	Description
$\mathbb{1}(C)$	The function returns 1 is the clause C inside is true, and 0 otherwise
$\mathcal{N}(X;\mu,\sigma)$	Gaussian or normal distribution of random variable X with mean μ and variance σ^2
2^A	The powerset, i.e. the set of all subsets, of a set A
$\mathbb{P}(\mathcal{E})$	Probability of an event $\mathcal{E}$
$\mathrm{Ran}(f)$	The set of outputs of f
$\mathrm{span}(e_1,\ldots,e_n)$	The vector space spanned by the basis $e_1,\ldots,e_d$
Supp	Support of a distribution or a random variable
$\mathcal{U}(X;a,b)$	Uniform distribution of random variable X with minimum value a and maximum value b
$\mathbb{V}$	Variance of a random variable

Chapter 1

Introduction

By seeking and blundering we learn.

— *Johann Wolfgang von Goethe*[1]

Human intelligence is the ability of accumulating information, processing this information in form of general constructions, and learning these constructions to adapt with the environment. Specifically, in the course of human life, the components of intelligence – information accumulation, processing, and learning – lead to efficient and effective decision making. Sequences of such decisions imbibe an experience and, in turn, influence the process of learning. These two phenomena go hand-in-hand through the course of human life. This process of learning and decision-making on-the-go is hard to efficiently and effectively optimise due to two major bottlenecks. One bottleneck is the enormous variety of phenomena in this world and the even larger number of underlying factors influencing them. The other bottleneck is the uncertainty surrounding the amount of information needed to effectively learn and efficiently leverage to take decisions. The field of artificial intelligence, in general, projects simulation of this organic notion of intelligence in artificial machines as its grand goal. The field of machine learning specifically deals with the realisation of this learning process in machines with computational power. Thus, these fields of research also face the prob-

[1]Extracted from "in conversation with Johann Peter Eckermann", 1825.

lems caused by the plethora of influencing factors regarding a phenomenon, and also the uncertainty regarding the amount of information required to learn and to decide.

Since the research endeavour described in this thesis belongs to the domain of machine learning, specifically, and artificial intelligence, broadly, we address these two issues mathematically, algorithmically, and experimentally through various problem models, solution methodologies, and real-life applications. The problem models investigated in the scope of this thesis are multi-armed bandits (Chapters 5 and 6) and Markov decision processes (Chapters 3 and 4). The solution methodologies followed to investigate them are mainly online functional approximation and optimisation (Chapters 3 and 4) and information geometry (Chapters 5 and 6)). The real-life application addressed with the developed methodologies are automated database tuning (Chapter 3), energy- and performance-efficient live virtual machine migration in clouds (Chapters 4), and online scheduling of jobs arriving in queues to servers (Chapter 6).

Before delving into the details of individual works, we use the scope of this chapter to provide a technical background and to posit the contribution of this thesis in this context.

1.1 Learning to Make Good Decisions under Uncertainty: Reinforcement Learning

Artificial intelligence as a field has multiple branches and tributaries due to different conflicting definitions of intelligence, and eclectic bottlenecks to model and to implement its real-world applications. Further diversity in the literature is added due to the research works coming from different fields like computer science, statistics, psychology, engineering, and philosophy. In this thesis, we narrow our scope down to the computer science literature but with frequent interaction with statistical and mathematical methodologies. Specifically, we look towards the research problems from a paradigm of machine learning called *reinforcement learning*.

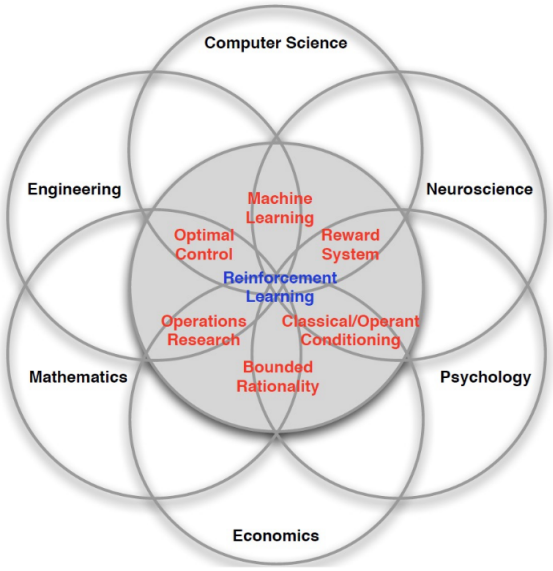


Figure 1.1: Diverse aspects of Reinforcement Learning as a field. [Silver, 2015]

In reinforcement learning (RL), an agent tries to learn to make a good sequence of decisions through interactions with an uncertain environment. This subject has originated during the infant years of cybernetics. Since then, it has been independently developed in different branches of science like psychology, statistics, and computer science. Although the works in each of these fields have resemblance in their basic approaches, they widely differ in their interpretations and applications of 'reinforcement'. Figure 1.1 [Silver, 2015] depicts this complex origin and diverse interpretations of reinforcement learning as a field. As we mentioned previously, we mostly stick with the computer science approach to reinforcement learning in the scope of this thesis.

The goal of reinforcement learning is to *learn* to make *good sequence of decisions* under *uncertainty*. Thus, a generic reinforcement learning algorithm should involve four key factors: an optimiser maximising *goodness* of decisions, a component to *learn about the uncertain environment*, a measure to quantify *impact of the present decisions on future sequence of decisions*, and inherent ability to *generalise these learnings* in form of a function of gained experience. Reinforcement learning measures the goodness of a decision by a real-valued function, called *reward*. The agent in the reinforcement learning tries to maximise the total sum of rewards obtained through the sequence

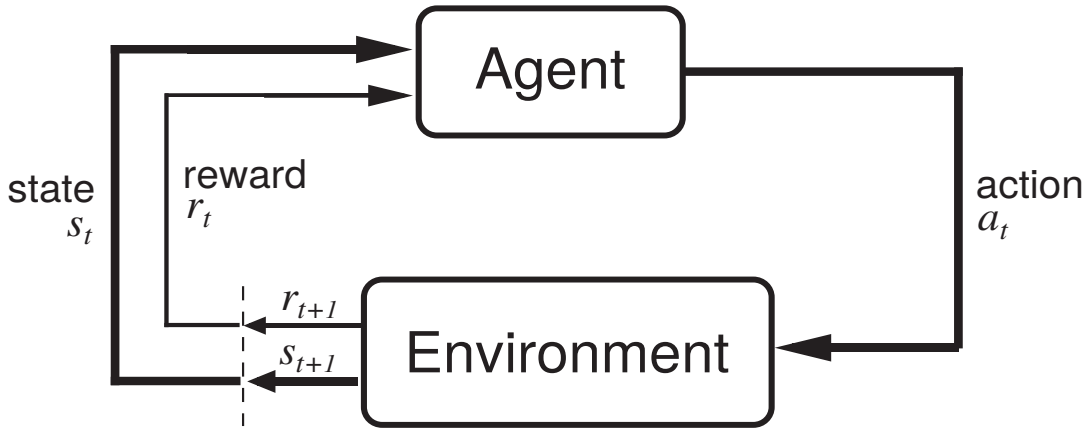


Figure 1.2: A block diagram of reinforcement learning [Sutton and Barto, 1998].

of decisions. This poses an optimization problem. The agent learns about the uncertainty of the environment by exploring different decisions and observing the rewards obtained from them. This poses a learning problem that deals with information accumulation and processing. Evaluating the effect of the present decisions on the future poses a prediction and planning problem. Thus, the agent tries to maximise the long-term effect by maximising the expected sequence of rewards obtained by the sequence of decisions if the present learning is exploited. This expected sum of sequence of rewards is called *utility* of a decision. This imposes a layer of planning on top of the optimisation problem. The agent has to generalise the mapping between the learning and the decision making components and to balance them. This mapping is done in form of a probabilistic function between decisions and past experiences, and is referred to as *policy*. Constructing and improving an effective and efficient policy involves the question of balancing learning and decision making.

Solving these problems of optimisation, learning, planning, and generalising makes generic reinforcement learning a challenging and interesting research problem to investigate. As mentioned earlier, these problems become hard if the number of possible decisions and their effects comes in a plethora. The space of possible solutions for these problems becomes too huge to find out the optimal solution. This problem is called the **curse of dimensionality** [Bellman and Kalaba, 1965]. The other bottleneck is balancing the learning and the optimisation problems, which in turn controls

the planning and generalising problems. This is called the **exploration–exploitation trade-off** [Macready and Wolpert, 1998]. Though there are further problems specific to different formulations and models of reinforcement learning, we focus on addressing these two issues through the theoretical and application-based developments described in this thesis.

For instance, the process of reinforcement learning is similar to training children in schools through examinations and evaluations. This provides them some skill sets to understand and interpret nature and society. But while educating them no one, not even the educators, knows how they are going to use these skills or whether they will ever use them in their professional or mundane lives. Still, this is the age-old way, we, human beings, learn and educate ourselves. Although this methodology has been followed for a long time, we still do not know the best way to educate people. Similarly, the generalisation of the learning scenario, specifically taking away the problem-specific details about *how* to do it, brings formidable computational challenges with it. This aspect brings an interesting dimension and a body of research works on reinforcement learning from the field of education and psychology. These challenges make reinforcement learning an interesting paradigm to look into.

In the following sections, we illustrate the basic elements and the problem settings of reinforcement learning before technically positing our contributions.

1.1.1 Basic Elements of Reinforcement Learning

The fundamental blocks of reinforcement learning are *the agent* and *the environment* [Kaelbling et al., 1996; Sutton and Barto, 1998]. The interaction between the agent and the environment is modelled using *the state* and *the action*. In order to model the goodness of a decision, long term impact of a decision, and the mapping from past experience to present decision, the reinforcement learning literature uses *a reward function*, *a value function*, and *a policy* respectively.

A **state** is a representation of the collective condition of the agent and the environment as perceived by the agent. A decision of the agent that facilitates her interaction

with the environment is called an **action**. For example, if we imagine that an agent is playing tic-tac-toe with an adversary, the condition of the grid at a certain instance is her state and putting a cross (X) or zero (0) in the following step are the possible actions.

A **reward function** quantifies the effect of taking an action at a certain state. A reward is a map from a state-action pair to a real number where a state-action pair represents the interaction between the agent and environment. Hence, the reward is used for defining the desirability of an action at a state of the agent.

Hypothesis 1 (Reward hypothesis)**.** *The benefit of any interaction between an agent and its environment can be represented using a real-valued reward function.*

The reinforcement learning paradigm relies on this *reward hypothesis*. The intuitive question is, whether it is effective to use numerical rewards from the same scale for all the interactions between agent and environment. Theoretically, North [North, 1968] proved that if we abide by four simple axioms imposing transitivity of decisions and rationality of the agent, a real-valued reward function would uniquely exist. Although in real-life human nature deviates from rationality, Hypothesis 1 holds appropriately in most of the problems that we face in statistics and computer science.

This reward function considers only the instantaneous effect of a decision but considering the short-term benefit is not the best way to make decisions. It is better to take an action that gives long-term benefits than a temporary one. For example, let us consider the scenario of travelling to Marina Bay Sands from the School of Computing, NUS at the peak hours in the afternoon. Assume that you want to complete this journey in the least time. If you can get a bus as soon as you come out of your office, is it be better to take this bus or to go for the train (MRT)? The answer of an experienced Singaporean is the MRT, though it is not giving you the highest reward at that point of time. This incompleteness of reward function introduces us to the utility (or value) function.

In reference with the previous discussion, we introduce a **utility (or value) function** that is used to quantify goodness of a decision in the long run. The utility

function indicates the long-term desirability of a state after considering the possible reward gains that can be obtained from the future states, which would probably be followed after taking a certain action in a certain environment. Thus efficiently solving a reinforcement learning problem requires the agent to look into a trade-off between the short-term and the long-term profits. In general, we represent the utility function by the expected sum of rewards that an agent expects to accumulate over a given horizon in the future. Thus, solving a reinforcement problem involves finding the sequence of actions with the highest utility. If all possible actions and their effects are known, this is equivalent to finding the action with the highest utility at the initial state and continuing its execution.

Hypothesis 2 (Utility hypothesis)**.** *All goals to be reached by an agent can be described by the maximisation of expected cumulative reward, i.e. the utility function.*

Theoretically, if there are m states and n actions, it is possible to solve the $m \times n$ equations describing the dynamics of the agent with n^m unknowns to get such an optimal utility function. In practice, it is not feasible for a large number of states and actions. Such a large state-action space is impractical to solve with limited computational resources in a reasonable time. This is the effect of the curse of dimensionality. In real-life, it is also impossible to know all the future scenarios that the agent is going to experience after taking a certain action. This is the effect of uncertainty that the agent tries to resolve by exploring and learning.

A **policy** is a mapping from the environment perceived by the agent through the observed states, actions, and rewards to the action to take at a state. It is basically the function representing behaviour of an agent at a given scenario. Depending on the interactions and the environment, a policy can be either dynamic or static. It is the core decision variable of a reinforcement learning algorithm. Thus, the problem of reinforcement learning is reducible to the search for a policy that simulates the behaviour optimally. The intrinsic uncertainty of environment led researchers to different approximations of the utility function that differ in dealing with the following questions.

1. How much information do we have to accumulate for calculating the utility function?
2. Can we formulate a simulator that mimics the environment to give us a basic idea of the environment and its evolution?

The first question leads to the formulation of different exploration and solution methodologies, such as Q-learning [Watkins and Dayan, 1992], TD-learning [Sutton, 1988], R-max [Singh et al., 2000], ϵ-greedy [Sutton and Barto, 1998] etc., and different representations, such as state space, state-action space etc. This also leads us to our work in Chapter 5.4.4. The second question leads to formulation of broadly two categories of algorithms: model-dependent, and model-free [Sutton and Barto, 1998]. In the model-dependent algorithms, there exists a mathematical representation, or simulator, which is used by the agent to mimic the behaviour of environment. The model helps the agent to predict the future of the environment and approximate the utility function. The accuracy of the model-based algorithms depends on how efficient and compatible the model is. A poor model may cause a grand failure, whereas the model-free algorithms come with the beguiling promise of doing the best without a simulator of environment. The model-free algorithms sample from the environment to learn about the state-action pairs and the obtained rewards. Thus, the design of model-free algorithms proves to be computationally challenging due to the duelling effects of learning the uncertainty of the environment and maximising the utility function. This projects the question of uncertainty modelling, and in turn the exploration–exploitation trade-off as a central issue. In this thesis, we try to address this questions of uncertainty modelling and exploration–exploitation trade-off while taming the problem of the curse of dimensionality wherever needed.

1.1.2 Settings of Reinforcement Learning

Before delving into the technical contributions of the thesis, we want to elaborate the settings of reinforcement learning in which we are going to frame our problems and solutions. Depending on the formulation of the environment and its effect on

formulating the state, the action, and their evolution with time, we find three broad settings of reinforcement learning in the literature. The settings are: *multi-armed bandits* [Bubeck et al., 2012], *Markov decision processes* [Szepesvári, 2010], and *partially observable Markov decision processes* [Cassandra et al., 1994]. Multi-armed bandits generally look into scenarios involving a single state representing the environment and multiple actions. Markov decision processes look into multiple states representing the environment and multiple actions. In this setting, the agent changes state over time but this evolution is completely observable and measurable. Partially observable Markov decision processes look into multiples states and multiple actions scenario, as for Markov decision processes, but the states are not completely observable and accurately measurable. This imposes another layer of uncertainty in learning and decision making. In this thesis, we focus on the first two settings, i.e., multi-armed bandits and Markov decision processes. We describe the details of these settings next.

Multi-Armed Bandits

In multi-armed bandits, the environment consists of a set of reward distributions. In the classical formulation of bandit problems, the reward distributions are stationary over time. Thus, it is considered as a single state formulation of reinforcement learning [Sutton and Barto, 1998]. Choosing a reward distribution and sampling from it constitutes an action. Sampling a reward distribution yields a reward to the agent. The utility function is the sum of accumulated rewards in a given time horizon. The problem would be simpler if the agent were to know the reward distributions. She could then always sample the distribution with the highest expected reward to maximise the expected utility. The problem becomes challenging when the reward distributions and the expected rewards are unknown to the agent. The agent has to simultaneously learn the reward distributions and to sample the distributions with higher expected rewards in order to achieve the optimal utility. This leads to the problems of uncertainty modelling and exploration–exploitation trade-off. Thus, researchers use the setting of multi-armed bandit to investigate the problems of uncertainty modelling and

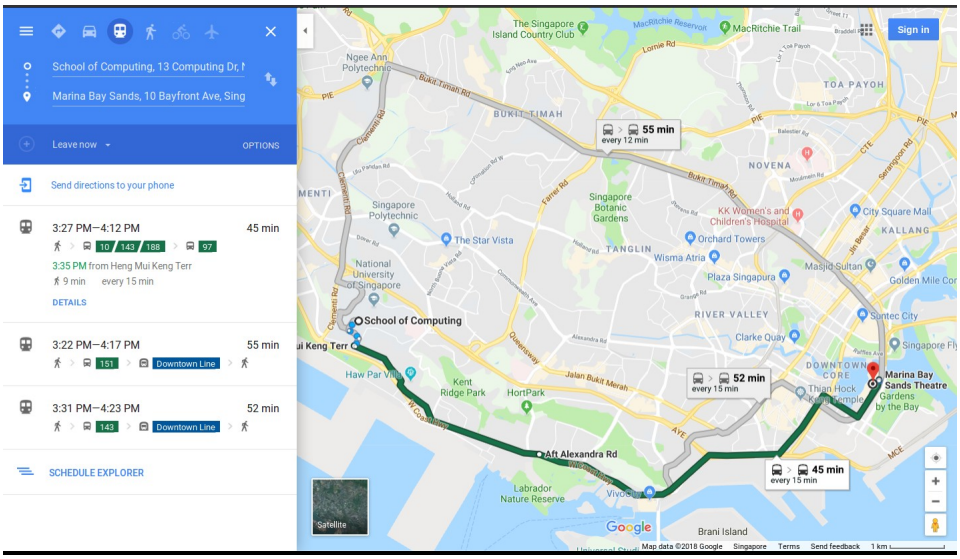


Figure 1.3: Finding a path in the map as a multi-armed bandit problem.

exploration–exploitation trade-off, disregarding the other problems caused by huge number of states and actions.

We illustrate the problem setting using the example of Figure 1.3. Suppose the agent is staying at Marina Bay Sands for a year and wants to go back home daily from the School of Computing with minimum travel time. Google Maps proposes her three different ways, which we see as three actions. The travel time of each day is the reward and the total travel time throughout the year is the utility. The three available ways are listed by Google Maps on the left side of the figure with the corresponding expected travel times. If we assume them to be correct, the agent can always take the path with minimum expected travel time (in green) to maximise the utility. Since this is an estimate at a certain time of a certain day, it may vary depending on the traffic conditions and other events throughout the day and throughout the year. Thus, in order to solve the problem with generality, let us assume that she does not have a knowledge of the expected travel time. A probable way in which she can decrease her travel time is by exploring these available options enough to learn their expected travel times and then to use the path with minimum travel time. The first phenomenon of learning by trying different actions is called exploration and the second phenomenon of using the learned knowledge to optimise the utility function is

called exploitation. Though initial exploration is needed to take an optimal decision, it is also essential to exploit eventually. Results proved that only exploration, or only exploitation, or fixed windows of exploration followed by exploitation do not lead to an optimal growth of cumulative regret. A randomised and adaptive mixture of exploration and exploitation is essential to achieve that. Thus, the multi-armed bandit provides a scope to investigate the exploration–exploitation trade-off. We use multi-armed bandits to investigate this issue in Chapters 5 and 6.

In the terminology of the bandit literature, each reward distribution is referred as an arm. The collection of them, i.e., the environment is called the bandit. The utility function is called the cumulative reward. Though we follow these terminologies in the corresponding chapter, we explained it here within the general reinforcement learning terminologies for the sake of continuity.

Markov Decision Processes

Markov decision processes increase the complexity of the multi-armed bandits by considering multiple states of the environment that evolves with time. Each of the states represents and encodes the information about the environment. The actions are the possible ways the agent have to transit between these states. This introduces an additional notion of transition function that determines the probability of transiting to a state from the other through a certain action. This formulation of reinforcement learning induces a planning problem of state transitions for the Markov decision processes. The reward is obtained from either a distribution or a deterministic function over the state-action pairs. The utility is computed as the sum of accumulated rewards obtained till a given horizon. There are alternative formulations of the utility where they consider unavailability of such a horizon and thus, compute a modified sum of the rewards. The goal is to determine a sequence of decisions that lead to optimisation of utility. Alternatively, it can be rephrased as finding a policy that balances the exploration, planning, and exploitation in order to optimise the utility.

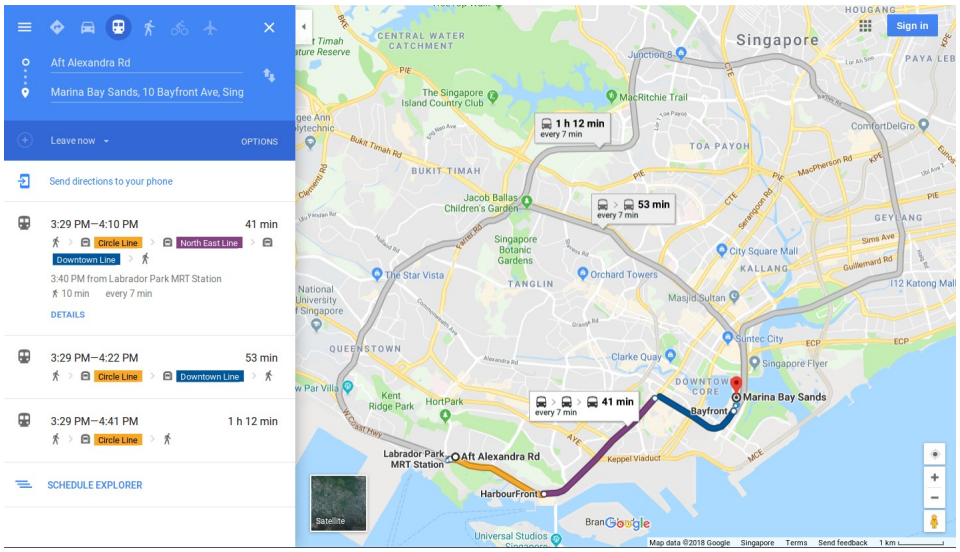


Figure 1.4: Finding and planning a path in the map as a Markov decision process.

We illustrate the problem setting by extending the example of Figure 1.3 and using the information of Figure 1.4. Suppose the agent in the example of Figure 1.3 is travelling from School of Computing to Marina Bay Sands. As she reaches the point 'Aft Alexandra Rd', her state of spatial position changes from School of Computing to 'Aft Alexandra Rd'. Now, if the road gets blocked for some emergency reason, she cannot take the regular bus route. She has to take the MRT which has unknown options and unknown travel delays for her. Thus, she has to check the map in Figure 1.4 and decide her available actions to be the three paths shown in the map or a mixture of them depending on the future scenarios. Each of them would transit her to a new MRT station, which is a new state. Then she can further decide which sequence of MRTs to take to reach Marina Bay Sands. These transitions are unknown to her and also the rewards obtained through each of the transitions. Thus, this would not only involve the previous problems of exploration and exploitation but also that of the planning depending on the state transitions. In addition, if the number of possible states are too huge, it is hard for the agent to explore all of them in one journey and to decide which sequence of transitions would lead her to the optimal travelling time. Thus, the planning problem would enhance the hardness by adding the curse of dimensionality problem.

In classical Markov decision processes, though we assume the knowledge of a reward function and a transition function, we observe from the above example that it is not known to the agent. This makes the learning and decision making more complex by adding another layer of uncertainty. These are the Markov decision processes with incomplete information. Thus, the corresponding solutions require additional estimation techniques for the reward and transition functions through exploration and accumulation of information. Any error in estimating these functions would lead to further error in the solution of the Markov decision process. We develop techniques to learn and to estimate these functions which are stable, accurate and compatible with the decision process to solve the problem efficiently and effectively. We address these problems using the functional approximation and online optimisation techniques in Chapters 3 and 4.

1.2 Motivations and Contributions

In this section, we discuss the theoretical and application-centric motivations and contributions of the proposed works.

1.2.1 Theoretical Aspects

Figure 1.5 illustrates the core theoretical ideas and contributions of this thesis. As we stated earlier, the inherent complexity of learning and decision making in an uncertain environment gives us a mathematical structure to explore along with challenging problems to solve.

Multi-armed bandits deal with two issues: accumulation of information to reduce uncertainty of decision making (exploration) and leveraging present knowledge to gain higher rewards (exploitation). Since the agent starts with incomplete information about the stochastic reward structure and gradually discovers more through actions, exploration is necessary. Investigating the pure exploration problem [Bubeck et al., 2009] allows us to focus on these aspects, equally significant for exploration–

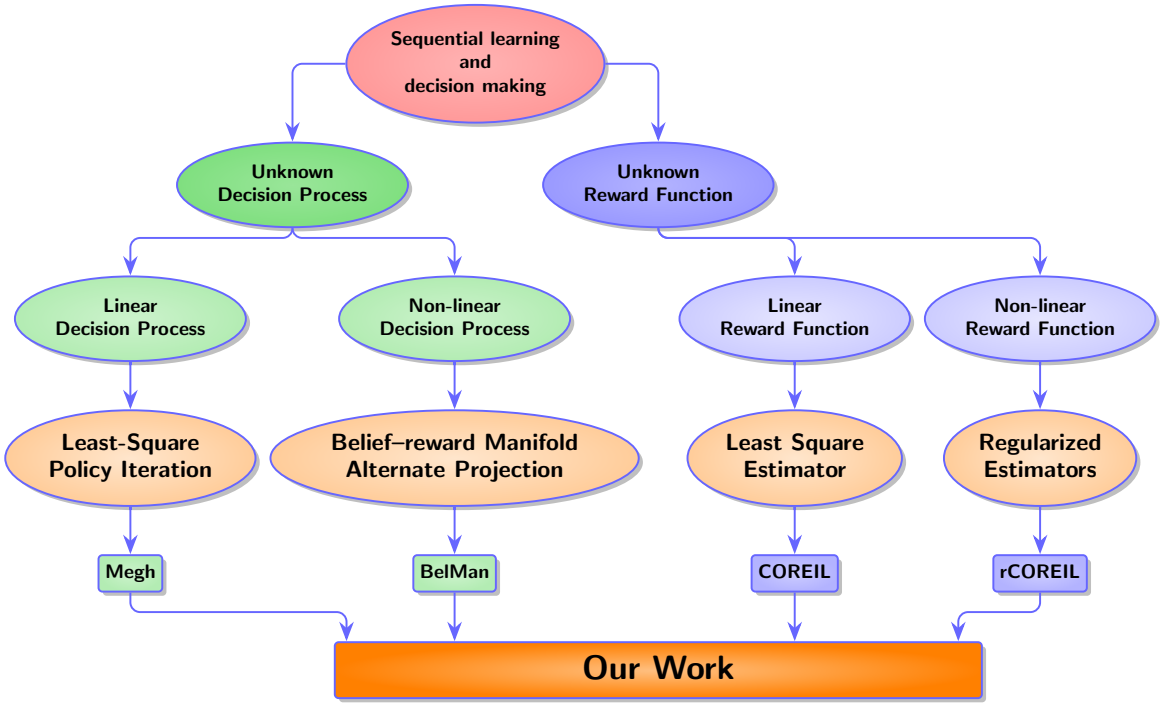


Figure 1.5: Positing the theoretical contribution of the thesis.

exploitation bandits [Bubeck et al., 2012]. On the other hand, in the exploration–exploitation problem, exploration alone is not sufficient: the gambler has to exploit available information to draw the optimal arm. The trade-off between exploration and exploitation emerges as a central question.

In Chapter 5, we analyse a Bayesian, information-geometric algorithm for multi-armed bandits, BelMan [Basu et al., 2018c], in light of the analysis of the state-of-the-art algorithms like UCB [Auer et al., 2002] and Thompson sampling [Thompson, 1933]. We analyse the geometrical constructions developed in BelMan, which are the pseudobelief-reward distribution and the focal distribution. We prove existence, uniqueness, and convergence in estimation of the pseudobelief–reward distribution. These results establish the pseudobelief-reward as a valid summary of the information over the arms for bandit problems. We show that it facilitates exploration. Following this, we show that use of pseudobelief-focal distribution leads to a balance between exploration and exploitation. We prove that BelMan is asymptotically consistent

for any bounded reward distribution like the state-of-the-art algorithms. We also empirically and comparatively argue that BelMan reaches logarithmic regret bound like the optimal bandit algorithms. We provide arguments and a conjecture to reach the formal proof. In Chapter 6, we apply this method to the online scheduling in multiple queue and multiple server systems with unknown service rates [Basu et al., 2018a]. Experiments show that the proposed algorithm outperforms the state-of-the-art for queueing systems both Bernoulli and exponential service rates. We want to leverage the generality of BelMan and extend it to other general service distributions.

When we deal with Markov decision processes, it poses both the problems of exploration–exploitation trade-off and planning [Puterman, 2009]. In real-life scenarios, often the state space becomes too huge to plan. For example, if we again consider an agent is playing tic-tac-toe in a 3-by-3 grid, it has to calculate over 2^9 states for getting the optimal results. Thus for any n-by-n grid the agent has to go through 2^{n^2} possible states to take the optimal decision. This exponential blow-up of the state space, even in case of simple learning scenarios like tic-tac-toe, shows us how computationally hard it can be to deal with a learning problem. Even with the modern resources majority of these learning problems remain computationally intractable. This phenomenon is popularly termed as **the curse of dimensionality** in learning literature.

These phenomena require us to dive into the mathematical structure of the state and action space. Often we try to prune the space and to find out a smaller subspace of interest by using the problem based knowledges. Like if you are playing tic-tac-toe and you want to take the decision of putting a cross (X), you are not going to put it in an already filled up space. So there is no meaning of looking into the possibilities related to those places. Though it may not stop the exponential blow-up, it will reduce the state space to be explored by the agent before taking the decision. These kind of strategies have inspired researchers to develop several heuristics and algorithms to solve reinforcement learning problems efficiently.

The most popular approach to solve Markov decision processes is to use dynamic programming [Bellman, 1957a]. This approach is based on simple but powerful 'divide-and-conquer' policy to solve problems. Firstly the problem is divided into smaller and easier pieces. Then each of those pieces are solved that in turn solves the bigger problem when juxtaposed together in a proper manner. So it is basically a bottom-up approach but reaching to the bottom level from the original problem is also tricky. From Hypothesis 2, we know that finding an optimal policy is analogous to finding the optimal utility function. Using the dynamic programming approach, the optimal utility or value function can be determined by a simple iterative algorithm called *value iteration* [Bellman, 1957a; Bertsekas, 1987]. In another version of dynamic programming based algorithm, called *policy iteration* [Puterman, 2009], the agent directly tries to find out optimal policy rather than the value function. We will discuss more about these two algorithms and their theoretical and practical importances in the following chapters.

But this approach requires a pre-defined model to learn and use it to simulate the learning procedure. There is another class of algorithms that tries to learn the environment without simulating a model. They are called 'model-free' algorithms whereas the previous class is called 'model-based' algorithms. In the model-free approach, the feedback or learning loop is considered as an adaptive critic that tries to analyse the pros and cons of the actions of the agent in order to achieve a goal. This critic provides reward for the actions taken if the result was good and punish if the result was bad. But the question is, what is the good time to judge the result or in other words to *observe and evaluate* it? Temporal difference algorithms, like TD(λ) [Sutton, 1988], Q-learning [Watkins and Dayan, 1992], use insight from value iteration to adjust the estimated value of a state based on the immediate reward and the estimate value of the next state. In another approach, we can consider this feedback loop as an actor-critic game [Kimura and Kobayashi, 1998], where they constantly evaluate each other to gain a complete knowledge of the environment.

In Chapter 3, we solve a Markov decision process with unknown reward function by using a regularised function estimator with compatible stability guarantees in parallel with an online reinforcement learning algorithm [Basu et al., 2015a,b, 2016]. In Chapter 4, we solve a Markov decision process with unknown transition function by projecting the observed transition on a well-designed feature space with convergence guarantees in parallel with an online reinforcement learning algorithm [Basu et al., 2017c,b]. We are now assembling these methodologies for the unknown reward function and unknown transition function. We presented some basic results showing efficiency of the algorithms developed to solve this problem.

1.2.2 Application Aspects

While reinforcement learning raises theoretical questions as mentioned in the previous section, it also has a vast field of applications in real-life problems. Specially with increase of automation and advent of intelligent systems in every domain of modern life, application of reinforcement learning is becoming a part and parcel of every domain of studies. Beside developing theoretical works and algorithms, another aim of this piece of research is to apply them to problems from eclectic domains like automatic index tuning, automatizing virtual machine migration, or online scheduling in queues. Each of these problems and their relations with reinforcement learning are described briefly in the following paragraphs.

Automatic Database Tuning

Database tuning describes processes used to optimize and homogenize the performance of a database (DB). It refers to design of the database files, selection of the database management system (DBMS) application, and configuration of resources (OS, CPU, etc.) used by the DB. Database tuning aims to optimise the use of system resources to perform work as efficiently and rapidly as possible. Generally database administrators (DBA) manage to use of system resources. But the growing complexity of database applications, management systems, and infrastructures is a challenge for their admin-

istration. Researchers and vendors have been studying the design and implementation of self-managing database systems [Luhring et al., 2007; Schnaitter et al., 2007]. We propose a learning approach to adaptive performance tuning of database applications. The objective is to validate the opportunity to devise a tuning strategy that does not need prior knowledge of a cost model. Instead, the cost model is learned through reinforcement learning. We instantiate our approach to the use case of index tuning. We model the execution of queries and updates as a Markov decision process whose states are database configurations, actions are configuration changes, and rewards are functions of the cost of configuration change and query and update evaluation. During the reinforcement learning process, we face two important challenges: the size of the state space and the unavailability of a cost model. In order to address the former, we devise strategies to prune the state space, both in the general case and for the use case of index tuning [Basu et al., 2015a,b]. In order to address the latter, we iteratively learn the cost model, in a principled manner, using regularization to avoid overfitting [Basu et al., 2016]. We empirically and comparatively evaluate our approach on a standard OLTP dataset [Raab, 1993]. We show that our approaches are more efficient and effective with respect to the state-of-the-art adaptive index tuning [Chaudhuri and Narasayya, 1998], which is dependent on a cost model.

We elaborate on this in Chapter 3.

Virtual Machine Migration

Cloud computing is emerging as the new paradigm of computing because of its exciting features like resource pooling, on demand computing, flexibility, scalability, pay-as-per-use, high availability and low capital infrastructure management [Alsarhan et al., 2018]. In cloud data centre generally a network of physical machines (PM) are kept with virtual machines (VM) installed on them. Cloud providers leverage live migration of virtual machines [Clark et al., 2005] to reduce energy consumption and allocate resources efficiently in data centers [Xu et al., 2012]. Each migration decision depends on three questions: when to move a virtual machine, which virtual machine to move

and where to move it? Dynamic, uncertain and heterogeneous workloads running on virtual machines make such decisions difficult. Knowledge-based and heuristics-based algorithms are commonly used to tackle this problem. Knowledge-based algorithms, such as MaxWeight scheduling algorithms [Maguluri et al., 2012], are dependent on the specifics and the dynamics of the targeted Cloud architectures and applications. Heuristics-based algorithms, such as MMT algorithms [Beloglazov et al., 2012], suffer from high variance and poor convergence because of their greedy approach. We propose a reinforcement learning approach. This approach does not require prior knowledge. It learns the dynamics of the workload as-it-goes. In [Basu et al., 2017c,b], we formulate the problem of energy- and performance-efficient resource management during live migration as a Markov decision process. While several learning algorithms are proposed to solve this problem, these algorithms remain confined to the academic realm as they face the curse of dimensionality. They are either not scalable in real-time, as it is the case of MadVM, or need an elaborate offline training, as it is the case of Q-learning [Watkins and Dayan, 1992]. We propose an actor-critic algorithm [Grondman et al., 2012b], Megh, to overcome these deficiencies. Megh uses a novel dimensionality reduction scheme to project the combinatorially explosive state-action space to a polynomial dimensional space with a sparse basis. Megh has the capacity to learn uncertain dynamics and the ability to work in real-time. Megh is both scalable and robust. We implement Megh using the CloudSim toolkit [Calheiros et al., 2011] and empirically evaluate its performance with the PlanetLab [Park and Pai, 2006] and the Google Cluster workloads [Reiss et al., 2011]. Experiments validate that Megh is more cost-effective, incurs smaller execution overhead and is more scalable than MadVM [Han et al., 2016] and MMT [Beloglazov et al., 2012]. We explicate our choice of parameters through a sensitivity analysis. We discuss this work in Chapter 4.

Scheduling in Queues

We formulate the allocation problem in a multiple-server multiple-queue system with known arrival rate and unknown service rates as a multi-armed bandit problem [Basu

et al., 2018a]. The queue-server pair and the allocation algorithms are analogous to the arms and the gambler of multi-armed bandits. In classical queueing theory literature, both the arrival rates and the service rates are assumed to be accurately known and it leads to an analytical solution for queue performance. For the aforementioned systems, due to the lack of such knowledge of accurate parameters, the proposed multi-armed bandit formulation provides a model-free framework to explore the queue parameters and to exploit them on-the-go. In order to resolve the exploration–exploitation trade-off in the bandit formulation, we adapt our information geometric algorithm, BelMan, for multi-armed bandits. This approach does not only solve the proposed formulation but also caters for an information geometric analysis of allocation in a queueing system.

We comparatively evaluate our approach with the state-of-the-art bandit algorithms, such as Thompson sampling [Thompson, 1933], Q-ThS, and Q-UCB [Krishnasamy et al., 2016]. Experiments validate the similitude of the behaviour of our approach with that of the existing analysis that shows an initial increase in queue-regret followed by a sub-logarithmic decrement. We theoretically prove the asymptotic consistency of our approach while experimentally verifying its optimal finite-time behaviour as the optimal algorithms.

We illustrate this work in detail in Chapter 6.

1.3 Structure of the Thesis

We discuss the background of the reinforcement learning settings and algorithms like multi-armed bandits, and Markov decision processes in Chapter 2. In this chapter, we also contextualise the contributions of this thesis in the light of the existing literature. Following this, we illustrate the contributions into two parts depending on the underlying methodology which are functional approximation and information geometry respectively.

In Part I, we focus on dealing with both the uncertainty modelling and the planning components of the Markov decision processes with unknown reward and transition

functions. We discuss the problem of solving Markov decision processes without a known reward function, and instantiate it for the automated database tuning problem in Chapter 3. We elaborate the problem of solving Markov decision processes without a known transition function, and instantiate it for energy and performance efficient migration of virtual machines in Chapter 4. These chapters theoretically rely on techniques of functional approximation and online optimisation.

In Part II, we proceed beyond the realm of functional approximation and consider the reward function and transition function to be a probability distribution. In order to develop the intended methodologies, we narrow our focus first to the multi-armed bandits that imbibes the exploration–exploitation problem. We use information theory and geometry as our tools of investigation as they intuitively probe the process of information accumulation, representation and exploitation. In Chapter 5, we discuss our solution to the multi-armed bandits and evaluate the proposed algorithm, BelMan, theoretically and experimentally. In Chapter 6, we illustrate application of BelMan in online scheduling of jobs arriving at multiple queues to multiple servers.

Chapter 7 draws a closure to this discourse with a discussion of the presented work, the perspectives learned from them, and a description of the future works following this research endeavour.

Chapter 2

A Primer on Reinforcement Learning

. . . you join the conversation – first by listening to what is being said, and then formulating a comment designed to advance the dialogue.

— Linda Bloomberg and Marie Volpe, Developing and Presenting the Literature Review', 2008.

Reinforcement learning [Sutton and Barto, 1998] is a sub-field of machine learning where an agent (or learner) sequentially learns to decide between actions with uncertain payoffs. Actions constitute interaction of the agent with the environment that is initially unknown to the agent. The goal of the agent is to maximise the total gain over a time horizon. The inherent uncertainties of the actions and the lack of information about the environment create a need for sequential accumulation of knowledge about the payoffs of different actions, and the interactions with environments, respectively. Thus, a reinforcement learning problem has two façades: learning, and decision making. Gaining more knowledge about the environment and different actions leads to better and informed decisions while forcing the agent to choose actions with suboptimal payoff. Hence, the challenge of the agent is to adaptively balance learning and maximisation of payoffs. We elaborate these two perspectives and their trade-off as the central conundrum of the field of reinforcement learning as well as this thesis.

Different formalisms of reinforcement learning, such as multi-armed bandits (MAB) [Bubeck et al., 2012] and Markov decision processes (MDP) [Puterman, 2009], have

been developed and researched. They fundamentally differ in the problem descriptions and fundamental assumptions. In the following sections, we elaborate on the multi-armed bandits (MAB) and Markov decision processes (MDP) in incremental order of complexity.

In the first section, we formulate the multi-armed bandit problem. We specifically focus on the stochastic variant [Auer et al., 2002] of it due to our interest in this thesis. Following this, we discuss the notion of regret of a bandit algorithm, and fundamental lower and upper bounds on it [Lai, 1988]. The lower bound of regret depicts the fundamental limitation of the sequential learning and decision making. The upper bound of regret leads us to the notion of optimality of a bandit algorithm. This also leads us to a singular string to discuss the state-of-the-art algorithms, and contextualise the adaptive trade-off as the central conundrum.

In the second section, we extend this formalism of multi-armed bandits to discuss Markov decision processes. We describe the major families of algorithms to solve Markov decision processes, such as dynamic programming (DP) and temporal-difference (TD) methods [Powell, 2007]. These methods are based on the memoisation trick, i.e., storing the data sequentially in tables and accessing them efficiently. We explicate the connection between these methods and functional approximation techniques that leads to improved algorithms. These functional approximation based algorithms are used to efficiently solve large Markov decision problems [Gordon, 1999]. Following this, we conclude the chapter with different variants of these formulations which will be relevant in further discourses of this thesis.

2.1 Multi-Armed Bandits

Multi-armed bandit problems constitute the archetypal formalism of sequential learning and decision-making. These problems reveal the fundamental issues of reinforcement learning. The multi-armed bandit problems, or in short bandit problems, were introduced by William Thompson in an article on adapting medical trials on-the-

go [Thompson, 1933]. In the following 85 years, this problem engendered an eclectic set of research works in the fields of mathematics, statistics and computer science, and still creating more contributions after its use in AlphaGo [Silver et al., 2016].

The name 'bandit' came from the experiments conducted by statisticians Frederick Mosteller and Robert Bush on mice and humans to build a stochastic model of learning [Bush and Mosteller, 1953; Mosteller et al., 1956]. In case of humans, they provided a gambler with a slot machine that has two arms. Playing each arm provides a random payoff sampled from two different distributions of payoffs. These distributions are initially unknown to the gambler. Thus, the gambler faces this dilemma to play both the arms enough number of times to learn about their pay-off distributions while keeping the total gain in payoff as high as possible. These experiments were conducted to build a primary stochastic model of how humans learn to make decisions [Bush and Mosteller, 1953; Mosteller et al., 1956]. These slot machines are called bandits (Figure 2.1) in popular terminology as gamblers loose money by playing them. Hence, the problem of sequential learning and decision making to choose the arms with distributional pay-offs is termed as the multi-armed bandit problem, or bandit problem in short.

Though the bandit problem looks benign and simple, it took 79 years for researchers to propose a theoretical explanation [Agrawal and Goyal, 2012] of the optimal performance of the algorithm proposed by Thompson, called Thompson sampling [Thompson, 1933]. The study of bandit problems originated research works in the field of concentration inequalities [Boucheron et al., 2013], Markov chains [Gittins, 1979], stochastic processes [Seldin et al., 2012], information theory [Russo and Van Roy, 2016], statistics [Lai and Robbins, 1985], convex optimisation [Agarwal et al., 2011], computational complexity [Mannor and Tsitsiklis, 2004], and machine learning [Silver, 2015]. We are going to discuss some of these developments and the state-of-the-art algorithms in the following section.

Bandit problems are not only of theoretical and intellectual interest. We find out application of bandit problems and corresponding algorithms for design of ethical clini-

Figure 2.1: A one-armed bandit.

cal trials [Press, 2009], internet advertising [Langford et al., 2008] and recommendation engines [Li et al., 2010] developed by companies such as Google and Yahoo, designing decision making systems like AlphaGo [Silver et al., 2016], building online tutoring platform [Nguyen and Bourgine, 2014], and so on.

In the following section, we discuss the mathematical formulation of the bandit problem, the notion of regret and optimality in bandits, and the state-of-the-art algorithms developed to solve it.

2.1.1 Finite-Armed Stochastic Bandit

A multi-armed bandit problem consists of two main components: an agent $\mathcal{A}$ and an environment $\mathcal{E}$. The environment $\mathcal{E}$ constitutes of a set of probability density functions over real numbers $\{f_1, f_2, \ldots, f_K\}$. Each distribution corresponds to *an arm* of the bandit. Following bandit literature, we call the payoffs of the arms to be *the reward* $X \in \mathbb{R}$. Since the distributions are over the payoffs, a distribution f_i is termed as *the reward distribution* of the arm i.

Figure 2.2: The sequential interaction of the agent and the environment in a multi-armed bandit problem.

Algorithm 1 Finite-Armed Stochastic Bandit

1: Time horizon T, number of arms K
2: **for** $t \in [1, \ldots, T]$ **do**
3: The agent chooses an action $A_t \leftarrow \pi_t(H_{t-1})$
4: Environment samples a reward $X_t \in [0, X_{\max}]$ from the distribution f_{A_t}. The agent update the history with A_t and X_t.
5: **end for**

As shown in Figure 2.2, the agent $\mathcal{A}$ interacts sequentially with the environment. At each time t, the agent chooses an arm $A_t \in \{1, \ldots, K\}$ from the environment. The environment samples a reward X_t from the distribution f_{A_t} and reveals it to the agent. Sequential interaction of the agent with the environment creates a sequence of arms pulled and the corresponding rewards obtained. They together form *the history* of the agent till time $t-1$, which is $H_{t-1} \triangleq (A_1, X_1, \ldots, A_{t-1}, X_{t-1})$. Our goal is to compute a randomised map from agent's history to her actions that will maximise her total gain in reward for a given time horizon $T \in \mathbb{N}$. A randomised map from the agent's history to her action is called *a policy* $\pi : H_{t-1} \to \mathbb{P}(A_t)$. The total gain in reward of the agent $\mathcal{A}$ till time horizon T is called *the cumulative reward* $S(\mathcal{A}, T) \triangleq \sum_{t=1}^{T} X_t$. We illustrate this process in Algorithm 1.

Algorithm 1 implicates two fundamental assumptions of this problem formulation. Firstly, the conditional distribution of obtaining the reward R_t given the history

H_{t-1} and action A_t is the reward distribution f_{A_t}. Mathematically, $f_{A_t} = \mathbb{P}(X_t \mid A_t) = \mathbb{P}(X_t \mid A_t, H_{t-1})$. This implies the Markov property on the interaction with the environment. Secondly, the probability distribution of choosing an action is a function of the agent's history. Mathematically, for a given policy π_t at time t, $\mathbb{P}(A_t = a \mid H_{t-1}) = \pi_t(H_{t-1})$. Such a sequence of policies $[\pi_t]_{t=1}^T$ characterises the agent and determines her cumulative reward. This assumption implies that the agent cannot use future rewards to improve current decisions. Following the probability theory and philosophy literature, we argue that this keeps the causality of the agent and decision-making process intact. In this thesis, we assume the bandit consists of a finite number of arms, i.e., $K \in \mathbb{N}$, and the rewards are bounded and non-negative, i.e., $X \in [0, X_{\max}]$. This classical formalism of the bandit problem is called the *finite-armed stochastic bandit* problem.

This is also an archetypal setting of reinforcement learning as illustrated in Figure 1.2. An action in reinforcement learning corresponds to an arm in the bandit problem. There is only one state that corresponds to the set of reward distributions that does not change with time. Thus, it is often called the single-state reinforcement learning problem. We would further discuss this and extend to Markov decision processes in Section 2.2.

Information accumulation, representation, and exploitation

The agent's goal is to maximise the cumulative reward $S(\mathcal{A}, T)$. The cumulative reward $S(\mathcal{A}, T)$ is a random variable that depends on both the sequence of policies and the reward distributions. Thus, a measure of utility has to be defined on the cumulative reward that the agent can optimise and utilise to decide the effect of decision making. This utility function depends on the agent's intention and goal of the decision-making process. Typically, we use expected reward as the measure of utility of choosing a reward distribution. Thus, we loosen the goal to maximise the expected cumulative reward, i.e., $\mathbb{E}\left[\sum_{t=1}^T X_t\right]$, rather than the cumulative reward. Following the literature, we misuse the terminology a bit to refer to the expected

cumulative reward as the cumulative reward $S(\mathcal{A}, T)$. Though expectation acts as an intuitive choice of utility metric, it does not completely incorporate the randomness of the cumulative reward. Specifically, expectation fails to serve as a proper metric when the reward distribution is bimodal or has heavy tails. Beside this, expectation does not satisfy the interest of a user who tries to minimise the risk or wants to make more stable decisions. This reasoning has led to research works concerning risk-aware bandits [Galichet et al., 2013; Sani et al., 2012; Maillard, 2013; Huo and Fu, 2017] and stability analysis of utilities in bandit algorithms [Oneto et al., 2016; Das and Kamenica, 2005]. Researchers have also proposed different utility metrics on the cumulative reward such as the value at risk (VaR)[1] [Galichet et al., 2013], conditional value at risk (CVaR)[2] [Huo and Fu, 2017], mean-variance measure[3] [Sani et al., 2012], and minimax risk[4] [Bather, 1983].

Even after that fixing a goal does not reduce the bandit problem to an optimisation problem as the true reward distributions are unknown to the agent. We elaborate on that in Example 1.

Example 1. *Figure 2.3 shows an example of the bandit problem. The agent has three coins in front of her with* 50%*,* 75% *and* 80% *chances of getting heads but these probabilities are unknown to her a priori. We can visualise this environment formally as a set of three Bernoulli distributions with probability of success* $p_1 = 0.50$*,* $p_2 = 0.75$*, and* $p_3 = 0.80$*. Since the probabilities* p_1*,* p_2*, and* p_3 *are unknown to the agent, she starts playing the coins and counts the outcomes. As shown in Figure 2.3, she has*

[1]Value at risk measures the potential gain in value of cumulative reward over a defined time horizon for a given confidence interval. Mathematically, we can define it as $\text{VaR}_\alpha(X) = \inf\{x \in \mathbb{R}^{\geq 0} \mid \mathbb{P}(X \geq x) > \alpha\}$ for a given confidence parameter α.

[2]Conditional value at risk is the expected reward obtained from a reward distribution below a certain confidence interval. Mathematically, we define CVaR for a confidence level $\gamma \in [0, 1]$ as $\text{CVaR}_\gamma(X) = \frac{1}{\gamma}\int_0^\gamma \text{VaR}_\alpha(X)d\alpha = \mathbb{E}[X \mid \mathbb{P}(X) < \gamma]$. Thus, maximising the CVaR is analogous to maximising the minimum reward for a given confidence interval.

[3]Mean-variance measure is a weighted combination of empirical mean and empirical variance [Sani et al., 2012]. Minimisation of mean-variance measure leads to minimisation of variance and maximisation of mean of the cumulative reward.

[4]Minimax risk is a weighted difference between maximum reward and the expected cumulative reward. This risk quantifies the deficit in the obtained cumulative reward because of ignoring the rewards above the expected reward.

Figure 2.3: An example of multi-armed bandit

played the first coin 45 times, the second coin 40 times, and the third coin 30 times respectively. Thus, her estimate of the probability of success for three coins are $\frac{20}{45} = 0.45$, $\frac{25}{40} = 0.625$, *and* $\frac{20}{30} = 0.67$ *respectively. Playing the coins a larger number of times can lead her to a better estimate of the success probabilities of the coins, and thus to better choices of coins. Based on this, she would choose the third coin relatively more often, which would lead her to accumulation of higher number of heads. Hence, accumulating higher number of heads in a given time does not only need optimisation but an adaptive combination of learning the probabilities of success of the coins.*

From Example 1, we observe that an efficient optimisation of the cumulative reward requires an effective information accumulation about the reward distributions. It is also essential to provide a succinct representation to the accumulated information about the reward distributions. This gives us different approaches to model the environment.

The most common approach is to model the reward distributions. Following the statistics literature, it can be done in two ways such as parametric and non-parametric. In the parametric approach, the reward distributions are modelled as a statistical distribution that can be expressed using a vector of random variables and a set of parameters. The most common examples of such parametric statistical distributions are Bernoulli, Gaussian, uniform, and exponential [Brown, 1986]. For example, if the agent known the outcomes are binary, like head and tail in Example 1, it is rational to assume the reward distribution to be Bernoulli. Thus, the goal of the agent's learning becomes estimating the parameters of reward distribution accurately because a unique set of parameters uniquely define a reward distribution. Often in the bandit literature and also in this thesis, we follow the parametric approach to bandits due to the structure of output available to the agent and the mathematical tractability the parametric assumption provides. Though the parametric assumption may not be applicable to all environments, rather less information is available about the structure of the outcome. In such a scenario, it is intuitive to go for a compatible non-parametric family to investigate the properties of reward generation. Some of the well-discussed non-parametric bandits are subgaussian bandits and bandits with heavy tails. The advantage of the non-parametric approach over parametric one is that it has no constraint on the degrees of freedom. The disadvantage is the non-parametric approach takes away significant structure in the reward distribution and thus makes the learning harder for the agent.

Beside this, there are structured and unstructured approaches to reward generation. In the unstructured approach, knowledge about the distribution of one arm does not restrict the range of possibilities for other arms. Thus, the only way to learn about the distribution for an arm is to play it. This is the typical structure followed in the finite-arm stochastic bandit literature. In the structured approach, knowledge about the distribution of one arm restricts or facilitates the knowledge about the other arms. This is the typical structure followed in covariate bandits and contextual bandits. In this thesis, we follow the unstructured bandit approach as our target is to address the

exploration–exploitation problem engendered by the trade-off between learning and decision making. Further constraints and structures can be incrementally built upon this.

Regret

As we follow the parametric unstructured formalism of finite-arm stochastic bandit with expectation as the final utility metric to be served, the question that arrives is how to measure goodness of a sequence of decisions. In this formalism, the goal of the agent is to maximise the expected cumulative reward. We misuse the notation a bit to call it cumulative reward.

Definition 1 (Cumulative Reward)**.** The cumulative reward of an agent $\mathcal{A}$ for a given time horizon T is the expectation of sum of the rewards accumulated at each time step t.

$$S(\mathcal{A}, T) \triangleq \mathbb{E}\left[\sum_{t=1}^{T} X_{A_t}\right]. \tag{2.1}$$

The goal of the agent is to choose a sequence of arms $[A_t]_{t=1}^{T}$ in order to maximise $S(\mathcal{A}, T)$. We can use the Equation (2.1) to find out the two layers of uncertainty involved in the multi-armed bandit problem. If we look into the contribution of each of the arms, the reward accumulated until time step T by playing arm a depends on its expected reward μ_a and the number of times the arm is played. Mathematically, we can express it as

$$\begin{aligned}
\mathbb{E}\left[\sum_{t=1}^{T}\left(X_{a_t} \times \mathbb{1}(a_t = a)\right)\right] &= \sum_{t=1}^{T} \mathbb{E}\left[X_a\right] \times \mathbb{E}\left[\mathbb{1}(a_t = a)\right] \\
&= \mu_a \times \sum_{t=1}^{T} \mathbb{E}[\mathbb{1}\left(a_t = a\right)] \\
&= \mu_a \times \mathbb{E}[n_a(T))].
\end{aligned}$$

Here, $n_a(T)$ is the number of time the arm a is played till horizon T. It is also a non-negative random variable and decided by the set of policies followed by the agent.

This allows us to express the cumulative reward till time T as the sum of the rewards accumulated from all the arms till that time. Hence,

$$S(\mathcal{A}, T) = \mathbb{E}\left[\sum_{t=1}^{T} X_{A_t}\right] = \sum_{a=1}^{K} \mathbb{E}\left[\sum_{t=1}^{T} (X_{a_t} \times \mathbb{1}(a_t = a))\right] = \sum_{a=1}^{K} \left[\mu_a \times \mathbb{E}[n_a(T))]\right].$$

Thus, maximising the cumulative reward tackles two level of uncertainties. The first one is induced by the randomness of the reward distribution of the arm and is quantified by the expected cumulative reward μ_a. This is not controllable by the agent because it is inherent to the environment. A perfect knowledge of μ_a's for all the arms can lead the agent to choose the arm with maximum expected reward for every time step t. Since the expected rewards are not known to the agent, the number of times the arms are played are not deterministic rather they depend on the policies used by the agent. Thus, we treat $n_a(T)$ as the random variable controlled by the agent and its sequence of policies. This reduces the problem of maximising the cumulative reward to optimal choices of $n_a(T)$ which lead to optimal learning of the reward distribution and maximisation of the cumulative reward.

If we assume that there exists at least one arm with maximum expected reward $\mu^* \triangleq \max_{k \in \{1,\ldots,K\}} \mu_a$, the optimal policy of an agent will be to pull this arm always. We call this agent with full information OPT. For ease of notation, let us refer to the arm $a = 1$ as the arm with the maximum expected reward μ^*, i.e., $\mu^* = \mu_1$. Thus, the cumulative reward of the agent with full information would be

$$S(\mathrm{OPT}, T) = \mathbb{E}\left[\sum_{t=1}^{T} X_1\right] = \mu^* \times T.$$

We call it the optimal cumulative reward. Any agent with incomplete information about the reward distributions cannot achieve the optimal cumulative reward since she has to play the other arms a few times to find out the optimal arm. This provides us a guideline to design the desired set of policies for the agent such that the expected

reward obtained at each step would be asymptotically close to the optimal expected reward μ^*. [Robbins, 1952] formalised this notion as Definition 2.

Definition 2 (Asymptotic consistency)**.** A bandit algorithm is asymptotically consistent if it allows the agent to choose a sequence of arms that would asymptotically lead to expected cumulative reward gain per-step to the maximal expected reward. Mathematically,

$$\lim_{T\to\infty} \frac{S(\mathcal{A}, T)}{T} = \mu^*. \tag{2.2}$$

This implies that if an agent follow an asymptotically consistent bandit algorithm for a large enough number of times, she will find out the optimal arm and will continue playing it. Equation (2.2) leads us to another interesting metric to evaluate the performance of a bandit algorithm. The statement of Equation (2.2) is equivalent to stating that the per-step difference between the optimal cumulative reward and the cumulative reward of a consistent bandit algorithm would asymptotically be 0. We derive this notion in Lemma 1. Before this, we would like to define this new metric of performance that quantifies the difference between the optimal cumulative reward and the cumulative reward of a bandit algorithm. We call this performance metric regret of the agent $\mathcal{A}$.

Definition 3 (Regret)**.** The regret of an agent is defined as the difference between the optimal cumulative reward and the cumulative reward obtained by her till a given time horizon T.

$$R(\mathcal{A}, T) \triangleq S(\mathrm{OPT}, T) - S(\mathcal{A}, T) = \mu^* \times T - \sum_{a=1}^{K} \left[\mu_a \times \mathbb{E}[n_a(T)]\right]. \tag{2.3}$$

An alternative definition of regret of an agent $\mathcal{A}$ at time T is the expected value of deficit of cumulative reward because of not playing the optimal arm always. This

is obtained from Equation (2.3) through the following derivation.

$$\begin{aligned} R(\mathcal{A}, T) &= \mu^* \times T - \sum_{a=1}^{K} [\mu_a \times \mathbb{E}[n_a(T)]] \\ &= \mathbb{E}\left[\sum_{t=1}^{T} X_1\right] - \mathbb{E}\left[\sum_{t=1}^{T} X_{a_t}\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T} X_{a^*} - \sum_{t=1}^{T} X_{a_t}\right]. \end{aligned}$$

This alternative definition of regret places it as the metric that evaluates the amount of cumulative reward lost due to unavailability of complete information about the reward distributions. Thus, the lower and upper bounds on the regret of a bandit algorithm provide us respectively the minimum sacrifice in cumulative rewards to learn enough before taking optimal decisions and the maximum loss the algorithm may incur due to its design.

Before going into further details, we would illustrate some of the properties of regret. The first property tells us that regret is the weighted sum of expected action-counts, where the weights are the difference between optimal expected reward and the expected reward of the arm. The second property says for a bandit algorithm the regret is always positive due to unavailability of information. The third property states that the only policy that would have regret zero is the one with full information and this is the optimal policy choosing the optimal arm with probability one. The fourth property implies that for a bandit algorithm to be optimal the regret has to grow sublinearly with time.

Lemma 1 (Properties of regret)**.** *If $\mathcal{E} = \{f_1, \ldots, f_K\}$ is the environment of a finite-arm stochastic bandit with the optimal expected reward μ^* such that the reward distributions have bounded and real expectations, then*

a.

$$R(\mathcal{A}, T) = \sum_{a=1}^{K} \Delta_a \times \mathbb{E}[n_a(T)], \tag{2.4}$$

where $\Delta_a \triangleq \mu^* - \mu_a$ *is the suboptimality gap of arm* a *with respect to the optimal arm.*

b. $R(\mathcal{A}, T) \geq 0$ *for all agents* $\mathcal{A}$*.*

c. An agent $\mathcal{A}$ *satisfies* $R(\mathcal{A}, T) = 0$ *if and only if it chooses the arm with the optimal expected reward with probability one i.e* $\mathbb{P}(\mu_{A_t} = \mu^*) = 1$*.*

d. For an agent satisfying the asymptotic consistency of Definition 2, the growth of regret would be sublinear in time i.e $R(\mathcal{A}, T) = \mathrm{O}\big(T^p\big)$ *for* $p \in [0, 1)$*.*

The first statement tells us that to keep the regret small, the agent should try to play an arm with a larger suboptimality gap proportionally fewer times. The fourth statement of Lemma 1 is often generalised in the learning theory literature as Equation (2.5):

$$R(\mathcal{A}, T) \leq C(\mathcal{E}) g(T) \quad \forall \;\; T \in \mathbb{N} \;\; \text{and} \;\; \mathcal{E}, \tag{2.5}$$

where $C(\mathcal{E}) : \mathcal{E} \to \mathbb{R}^{\geq 0}$ is a constant dependent on the environment and $g(T) : \mathbb{N} \to \mathbb{R}^{\geq 0}$ is a function of horizon that grows sublinearly with time T. These properties, specifically the first and fourth statements, provide us a guideline to design the bandit algorithms as we will see in the upcoming sections.

There is an alternative Bayesian formulation of regret. If we assume existence of a prior B on the environment $\mathcal{E}$, Bayesian regret is defined as the average regret with respect to the prior. In finite-arm stochastic bandits, the environment is defined as the product of the set of possible reward distributions of the arms $F_1 \times \ldots \times F_K$, where $F_i = \{f_i\}$. This definition assumes that drawing from the reward distribution of an arm does not affect the rewards obtained from another arm. This is implied by the formulation of unstructured multi-armed bandits.

Definition 4 (Bayesian regret)**.** If the environment $\mathcal{E} = F_1 \times \ldots \times F_K$ is equipped with a σ-algebra $\mathcal{F}$ and there exists a prior probability measure B defined over the possible environments $\nu \in \mathcal{E}$, the Bayesian regret is the expected regret with respect

to the prior.

$$\mathrm{BR}(\mathcal{A}, T, B) \triangleq \int_{\mathcal{E}} R(\mathcal{A}, T \mid \nu) dB(\nu). \tag{2.6}$$

$\mathrm{BR}(\mathcal{A}, T, B)$ is defined assuming that $R(\mathcal{A}, T \mid \nu)$ is a measurable function with respect to $\mathcal{F}$.

Interestingly, maximising Bayesian regret reduces to an optimisation problem if the prior B is exactly known. This strategy is used in some special variants of stochastic bandits, such as Markovian bandits [Bellman, 1956], and some specific algorithms, such as Gittins index [Gittins, 1979]. However, in reality these prior distributions are often not known and have to be learned or constructed on-the-go.

Lower bound: Fundamental limit of bandit algorithms

Following the construction of regret and cumulative reward, researchers began investigating the achievable limits of an asymptotically consistent bandit algorithm. Since the typical bandit algorithm does not have knowledge of the true reward distributions, a certain amount of arm plays has to be spent for learning them. The seminal paper [Lai and Robbins, 1985] first formulated such a lower bound for the finite-arm stochastic bandits where the reward distributions belong to the Bernoulli family. They proved that if we have an unstructured, finite-arm, stochastic bandit with Bernoulli reward distributions, the minimum regret achieved in time T by an asymptotically consistent bandit algorithm $\mathcal{A}$ would be lower bounded by a logarithmic factor of T. They also proved the corresponding constant factor that dictates the ratio of the growth of regret and $\log(T)$ would be dependent on the sum of suboptimality gaps of the other arms in the environment. Later on, [Garivier et al., 2016a] improved it for the family of reward distributions with finite expected rewards and a unique support. We illustrate this result mathematically in Theorem 1.

Theorem 1. *If the environment $\mathcal{E} = F_1 \times \ldots \times F_K$ and the bandit algorithm $\mathcal{A} \in \Pi_{cons}(\mathcal{E})$ belongs to the set of asymptotically consistent policies for the environment $\mathcal{E}$,*

then for all $\nu \in \mathcal{E}$

$$\liminf_{T\to\infty} \frac{R(\mathcal{A}, T)}{\log T} \geq C(\nu) = \sum_{a:\Delta_a(\nu)>0} \frac{\Delta_a(\nu)}{\inf\limits_{f\in F_a(\nu)} \{D_{\mathrm{KL}}\left(f_a \| f\right) \mid \mu(f) > \mu^*\}}. \tag{2.7}$$

Here, $D_{\mathrm{KL}}\left(f_a \| f\right)$ *is the KL-divergence [Kullback, 1997] between two reward distributions* f_a *and* f *defined over the same support set.*

In the RHS of Equation (2.7), there are two factors in the numerator and denominator that dictate the lower bound $C(\nu)$. The numerator consists of the suboptimality gap. If it is bigger, the minimum achievable reward is higher. It means that the agent has to sacrifice more cumulative reward if it plays the arm with bigger suboptimality gap more number of times. The denominator consists of $\inf\limits_{f\in F_a(\nu)} \{D_{\mathrm{KL}}\left(f_a \| f\right) \mid \mu(f) > \mu^*\}$. It is the KL-divergence of the reward distribution f in the set of possible estimates of true reward distribution F_a which is most similar to the true reward distribution f_a but has expected reward $\mu(f)$ more than the optimal reward μ^*.

Example 2. *If the reward distributions are Bernoulli distributions, the set of possible estimates* $F_a \triangleq \{Ber(\mu_a) \mid \mu_a \in [0, 1]\}$. *If there exists an optimal arm with probability of success* μ^*, *then* $\inf\limits_{f\in F_a(\nu)} \{D_{\mathrm{KL}}\left(f_a \| f\right) \mid \mu(f) > \mu^*\} = \inf\limits_{\mu_a>\mu^*} \mu \log\left(\frac{\mu_a}{\mu^*}\right) + (1 - \mu_a) \log\left(\frac{1-\mu_a}{1-\mu^*}\right)$. *This quantity achieves smaller values as* μ_a *tends towards* μ^*, *and thus causes a higher value for the lower bound constant* $C(\nu)$.

This dictates that if there are reward distributions among the possible estimates of the true reward distribution of an arm which are quite similar to the reward distribution of the optimal arm, it is hard to distinguish such arms from the optimal arm. Thus, the agent has to play a higher number of times and to consume more regret in order to detect the optimal arm and to reach the optimal regret. Since the constant in the lower bound $C(\nu)$ is dependent on the environment, i.e., the exact set of distributions that the agent is accessing, this bound is called as an *instance-dependent bound*. This provides an additional guideline for designing bandit algorithms that the best achievable regret grows logarithmically with time T.

There are further investigations on obtaining instance-independent bounds where the constant $C(\nu)$ is not dependent on the environment. Instead the constant depends on the number of arms K only. Thus, we have to compute the lower bound of worst-case regret possible in an environment. This generated the notion of minimax regret which is the infimum achievable by a bandit algorithm for the worst case regret of an environment.

Definition 5 (Minimax regret)**.** For a set of bandit algorithms $\Pi = \{\mathcal{A} \mid \mathcal{A}\text{is a feasible policy}\}$ and an environment $\mathcal{E} = \{\nu \mid \nu\text{is a set of K reward distributions}\}$, the minimax regret is

$$R^*(\mathcal{E}, T) \triangleq \inf_{\mathcal{A}\in\Pi} R(\mathcal{A}, T \mid \mathcal{E}) = \inf_{\mathcal{A}\in\Pi} \sup_{\nu\in\mathcal{E}} R(\mathcal{A}, T \mid \nu). \tag{2.8}$$

[Audibert and Bubeck, 2009] has proved that for subgaussian bandit with variance less than or equal to one and mean reward in $[0, 1]$ then $R^*(\mathcal{E}, T) \geq c\sqrt{KT}$ for $K > 1$ and $T \geq K$. The subgaussian setup implies that the mean has to bounded and the tail of the reward distribution should be upper bounded by a limit. This setup is non-parametric and includes commonly used families of distributions such as Bernoulli, Gaussian, exponential, etc. This provides another proof that the growth of worst case regret would be at least $\sqrt{T}$.

Upper bound: Optimality of bandit algorithms

From the previous sections, we learn that a bandit algorithm should choose the arms inversely proportionally to the suboptimality gap, an asymptotically consistent bandit algorithm should incur sublinear regret, and the minimum regret growth achievable is at the logarithmic order of time because it has to learn the reward distributions of arms by playing them. This sheds light on two important aspects of any agent which are learning by exploring the arms and maximising the cumulative reward by playing the arm with maximum expected reward as per the present knowledge. The first phenomenon where the agent learns about the reward distributions of the arms by playing them a certain number of times is called *exploration*. The second phenomenon

Algorithm 2 Pure exploration

1: Time horizon T, number of arms K, an environment $\mathcal{E}$ with reward distributions $f_1, \ldots, f_K$.
2: **for** $t \in [1, \ldots, T]$ **do**
3: The agent uniformly randomly chooses an action $A_t \leftarrow a \in \{1, \ldots, K\}$ such that $\mathbb{P}(A_t = a) = \frac{1}{K}$
4: The environment samples a reward R_t from the distribution f_{A_t}. The agent updates the history with A_t and R_t.
5: **end for**

is *exploitation* where the agent uses its present knowledge to choose the empirically best arm, i.e., the arm that is presently known to have maximum expected reward and keep on playing it. This allows us to test these basic strategies of exploration and exploitation. Further regret analysis helps us to devise the bound on regret for each one of them which projects a guideline to design bandit algorithms.

Algorithm 2 depicts the pseudocode of pure exploration algorithm. In the pure exploration algorithm, the agent uniformly chooses the arms with equal probability. Thus, the agent on average plays the arms equal number of times. This leads to learning of the reward distributions of arms but incurs a linear growth in regret with time. Since $\mathbb{P}(A_t = a) = \frac{1}{K}$, $\mathbb{E}[n_a(T)] = \frac{T}{K}$. From Equation (2.4), we get the regret $R(\mathcal{A}, T) = \sum_{a=1}^{K} \Delta_a \times \frac{T}{K} = \left(\mu^* - \frac{1}{K}\sum_{a=1}^{K} \mu_a\right) T$. Thus, the regret grows linearly with time horizon T.

Algorithm 3 depicts the pseudocode of pure exploitation algorithm. In the pure exploitation algorithm, the agent plays the arm once to get some basic knowledge about them and then keep on playing the arm with best empirical average of reward. Thus, the agent plays each arm except the chosen one only once. This hinders learning of the true reward distributions of arms and gets stuck in one arm only. It also causes linear regret in worst case when the arm that the agent sticks to is not an optimal arm. Under such condition, $n_a(T) = T$ for the chosen arm, say i, and $n_a(T) = 0$ for all other arms. From Equation (2.4), we get the regret $R(\mathcal{A}, T) = \Delta_i T$. Thus, the worst case regret grows linearly with time horizon T.

Algorithm 4 depicts the pseudocode of Explore-then-Exploit algorithm. It is also called Explore-then-Commit (ETC) in bandit literature [Garivier et al., 2016a]. Here, Algorithms 2 and 3 are merged sequentially. The agent decides a fixed window of time wT for which it plays the arms uniformly randomly. Playing them equally likely gives her a chance to learn about the corresponding reward distributions. After that, the agent leverages this present knowledge about the arms to find out the empirically best arm and commits to play it. For a given window w, the regret for Explore-then-Exploit also grows linearly with time T.

Theorem 2. *The regret of Algorithm 4 grows linearly with* $T \in \mathbb{N}$ *and* $w \in \{1, \ldots, \lceil T/K \rceil - 1\}$.

$$R(\mathcal{A}, T) \leq w(\sum_{a=1}^{K} \Delta_a) + (T - wK)(\sum_{a=1}^{K} \Delta_a \exp(\frac{-w\Delta_a^2}{4}))$$

[Garivier et al., 2016a] showed that the Explore-then-Commit can reach a logarithmic regret bound if the suboptimality gaps and the time horizon are known a priori, and the agent fixes the the window as a function of them. This knowledge of suboptimality gaps is impractical to obtain a priori. This shows that we need an adaptive

Algorithm 3 Pure exploitation

1: Time horizon T, number of arms K, an environment $\mathcal{E}$ with reward distributions $f_1, \ldots, f_K$.
2: **for** $t \in [1, \ldots, K]$ **do**
3: The agent chooses the arm t

$$A_t \leftarrow t$$

4: The environment samples a reward R_t from the distribution f_{A_t}. The agent updates the history with A_t and R_t.
5: **end for**
6: **for** $t \in [K+1, \ldots, T]$ **do**
7: The agent chooses the arm with best empirical average of reward $\hat{\mu_a}$.

$$A_t \leftarrow \underset{a \in \{1,\ldots,K\}}{\arg\max}\, \hat{\mu_a}$$

8: The environment samples a reward R_t from the distribution f_{A_t}. The agent updates the history with A_t and R_t.
9: **end for**

Algorithm 4 Explore then Exploit (or Explore then Commit)

1: Time horizon T, number of arms K, an environment $\mathcal{E}$ with reward distributions $f_1, \ldots, f_K$, an exploration window $w \in \{1, \ldots, \lceil T/K \rceil - 1\}$.
2: **for** $t \in [1, \ldots, wK]$ **do**
3: The agent uniformly randomly chooses an action $A_t \leftarrow a \in \{1, \ldots, K\}$ such that $\mathbb{P}(A_t = a) = \frac{1}{K}$
4: The environment samples a reward R_t from the distribution f_{A_t}. The agent updates the history with A_t and R_t.
5: **end for**
6: **for** $t \in [wK + 1, \ldots, T]$ **do**
7: The agent chooses the arm with best expected reward as per present knowledge

$$A_t \leftarrow \underset{a \in \{1,\ldots,K\}}{\arg\max}\, \hat{\mu_a}$$

8: The environment samples a reward R_t from the distribution f_{A_t}. The agent updates the history with A_t and R_t.
9: **end for**

interaction between exploration and exploitation to obtain sublinear regret bound. A fixed window of exploration and exploitation without any dependence on the time horizon cannot do so.

Lai and Robbins [Lai and Robbins, 1985], and Burnetas and Katehakis [Burnetas and Katehakis, 1997] pushed this sublinear growth of regret further to prove that the optimal growth of regret should be logarithmic with time. Lai and Robbins [Lai and Robbins, 1985] proved the logarithmic regret growth for Bernoulli bandits. Burnetas and Katehakis [Burnetas and Katehakis, 1997] extended it to parametric reward distributions with same support set and bounded reward. We formally state this logarithmic regret bound for parametric reward distributions in Theorem 3.

Theorem 3. *If a bandit algorithm satisfies the following sufficient conditions*

a. There is no suboptimal arm whose expected reward is infinitesimally close to the optimal expected reward and whose reward distribution is similar to the reward distribution of the optimal arm. For a suboptimal arm a, $\lim_{\epsilon \to 0} \inf_{f \in F_a(\nu)} \{D_{\mathrm{KL}}(f_a \| f) \mid \mu(f) > \mu^* - \epsilon\} = \infty$.

b. The parameter θ_a of the reward distribution f_a is estimated as $\widehat{\theta_a^t}$ with error ϵ with high probability and the accuracy increases inversely with the number of times the arm is played. Mathematically, $\mathbb{P}\left(|| \hat{\theta_a^t} - \theta_a ||\leq \epsilon\right) = 1 - o(1/t)$ as $t \to \infty$ for $\epsilon > 0$.

c. The expected reward of an arm a is not underestimated with a high probability and the accuracy increases inversely with the number of times the arm is played. Mathematically, $\mathbb{P}\left(\sup_{\theta}\{\mu(f(\theta)) \mid D_{\mathrm{KL}}\left(f_a(\theta)\|f_a(\hat{\theta_a^t})\right) < \frac{\log t}{t}\} \geq 1 - \mu(f_a(\theta_a)) - \epsilon\right) = 1 - o(1/t)$ as $t \to \infty$ for $\epsilon > 0$.

then for all parametric reward distributions with non-zero support in an environment ν

$$\limsup_{T\to\infty} \frac{R(\mathcal{A}, T)}{\log T} \leq C(\nu) \triangleq \sum_{a:\Delta_a(\nu)>0} \frac{\Delta_a(\nu)}{\inf_{f\in F_a(\nu)}\{D_{\mathrm{KL}}\left(f_a\|f\right) \mid \mu(f) > \mu^*\}}. \tag{2.9}$$

Previously mentioned logarithmic lower bound along with the logarithmic upper bound for asymptotically optimal algorithms dictate the logarithmic growth of regret for an optimal bandit algorithm. Algorithms satisfying this logarithmic regret growth are called *asymptotically optimal* bandit algorithms. The sufficient conditions of achieving such bound provides a guideline to develop optimal bandit algorithms such as UCB [Auer et al., 2002]. This discussion also shows that the algorithms with deterministic window of exploration and exploitation are not asymptotically optimal policies. Rather an adaptive combination of exploration and exploitation is needed to achieve the logarithmic regret. Thus, exploration–exploitation trade-off plays a central role in design of an optimal bandit algorithm.

2.1.2 Bandit Algorithms

Following the fundamental theoretical results about the unstructured, finite-arm, stochastic bandits, we illustrate some of the state-of-the-art algorithms developed to solve this

Bandit Algorithms	Frequentist	Bayesian
Deterministic	UCB [Auer et al., 2002], KL-UCB [Garivier and Cappé, 2011] MOSS [Audibert and Bubeck, 2009]	Bayes-UCB [Kaufmann et al., 2012a], **BelMan** [Basu et al., 2018c]
Randomised	Adaptive ϵ greedy [Auer et al., 2002]	Thompson sampling [Thompson, 1933]

Table 2.1: Classification of Asymptotically Optimal [Lai, 1988] Bandit Algorithms.

problem effectively. Table 2.1 depicts a classification of the state-of-the-art bandit algorithms.

Frequentist Algorithms

There are two principal genres of algorithms developed from the frequentist perspective. The first family includes the randomised ϵ-greedy algorithms. [Watkins, 1989] proposed it as a solution to the equivalent one-state Markov decision process problem. Later on, it is further extended for multiple-state Markov decision processes [Sutton and Barto, 1998]. This family of algorithms perform pure exploration, i.e., play the arms with equal probability, with a given probability ϵ, and to perform exploitation, i.e., to play the arm with maximum empirical expected reward, with a given probability $1-\epsilon$. ϵ-greedy algorithms that maintain an ongoing distinction between exploitation and exploration phases are called semi-uniform. Hence, following the proof structure of Algorithm 4, it can be shown that ϵ-greedy algorithm incurs linear regret growth with time. There are other variants of ϵ-greedy algorithms, such as adaptive ϵ-greedy [Auer et al., 2002], can achieve optimal regret bound. [Auer et al., 2002] proposed to set $\epsilon(t) = \min\{1, \frac{cK}{d^2\ T}\}$, where $d \in (0, \min_a \Delta_a]$. This leads to logarithmic growth of regret. Since it requires a prior knowledge of the differences of expected rewards of the arms, it is not always practical to use such algorithm in reality. Beside this, it does not reach asymptotic optimality as dictated by Equation (2.9) because the constant in RHS is greater than that of $C(\nu)$.

Bandit Algorithm	Decision Function ($l(a,t)$)	Finite-time regret bound
UCB1 [Auer et al., 2002]	$\widehat{\mu_a}(t) + \sqrt{\frac{2\log t}{n_a(t)}}$	$8\left[\sum_{a:\mu_a<\mu^*} \frac{\log t}{\Delta_a}\right] + (1+\frac{\pi^2}{3})(\sum_{a=1}^{K}\Delta_a)$
UCB2 [Auer et al., 2002]	$\widehat{\mu_a}(t) + \sqrt{\frac{c(1+\log t - n_a^{\text{epoch}}\log c)}{2c^{n_a^{\text{epoch}}}}}$	$\sum_{a=1}^{K} \frac{c^2\log(2e\Delta_a^2 t)+C(c)}{2\Delta_a}$
UCB-tuned [Auer et al., 2002]	$\widehat{\mu_a}(t) + \sqrt{\frac{\log t}{n_a(t)}\min\{0.25, \hat{\sigma_a}(t)\}}$	–
MOSS [Audibert and Bubeck, 2009]	$\widehat{\mu_a}(t) + \frac{\log T}{K n_a(t)}$	$25\sqrt{TK}$
OC-UCB [Lattimore, 2015]	$\widehat{\mu_a}(t) + \sqrt{\frac{c}{n_a(t)}\log\left(\frac{\psi T}{t}\right)}$	$\sum_{a=1}^{K} \frac{C(\psi,c)}{\Delta_a}\log\left(\frac{t}{n_a(t)}\right)$
KL-UCB [Garivier and Cappé, 2011]	$D_{\text{KL}}\left(\widehat{\mu_a}(t)\|M\right) \leq \frac{\log t + c\log\log t}{n_a(t)}, M=\{\mu\}$	$\left(\sum_{a=1}^{K}\frac{\Delta_a}{D_{\text{KL}}(f_a\|f^*)}\right)\log t + C\left(\sum_{a=1}^{K}\Delta_a\right)\log\log t$
KL-UCB++ [Ménard and Garivier, 2017]	$D_{\text{KL}}\left(\widehat{\mu_a}(t)\|M\right) \leq \log\left(\frac{T}{Kt}\right) + 2\log\log\left(\frac{T}{Kt}+1\right)$	$76\sqrt{\sigma_{max}KT} + (\mu_{max}-\mu_{min})K$
Bayes-UCB [Kaufmann et al., 2012a]	$Q(1-\frac{1}{t}, \text{Posterior}_a^{t-1}(X))$	$\left(\sum_{a=1}^{K}\frac{\Delta_a}{D_{\text{KL}}(f_a\|f^*)}\right)(\log t + C\log\log t)$

Table 2.2: The decision functions and finite-time regret bounds of the 'optimism in face of uncertainty' bandit algorithms.
5

Frequentist, deterministic algorithms such as UCB [Auer et al., 2002] and KL-UCB [Garivier and Cappé, 2011] are state-of-the-art for the exploration–exploitation bandit problem. They operate with *optimism in the face of uncertainty* principle, i.e., they choose to play an arm with maximum empirical mean plus a converging exploration bonus. In Table 2.2, we demonstrate the decision functions for different variants of *the optimism in the face of uncertainty algorithms* and the corresponding finite-time regret bounds. All of them reach logarithmic regret growth. UCB1, UCB2, MOSS, and OC-UCB are still not asymptotically optimal as the constant obtained for them are greater than that of $C(\nu)$ in Equation (2.9). KL-UCB, KL-UCB++, and Bayes-UCB algorithms reach asymptotic optimality for bounded rewards. OC-UCB and KL-UCB++ algorithms reach both the optimal $\sqrt{T}$ growth for minimax regret and the optimal $\log T$ growth for expected regret.

Frequentist algorithms are often not supportive to assimilation of *a priori* knowledge about the arms or specially, the underlying process. Use of such *a priori* knowledge in form of a prior distribution improves performance in applications where an underlying model for the reward distributions can be constructed [Kawale et al., 2015].

Bayesian Algorithms

A variant of the Bayesian formulation was introduced by [Bellman, 1956] with a discounted reward setting. Unlike S_T, the discounted sum of rewards $D_\gamma \triangleq \sum_{t=0}^{\infty} [\gamma^t X_{t+1}]$ is calculated over an infinite horizon. Here, $\gamma \in [0, 1)$ ensures convergence of the sequential sum of rewards for infinite horizon. Intuitively, the discounted sum implies the effect of an action decay with each time step by the discount factor γ. This setting assumes K independent priors on each of the arms and also models the process of choosing the next arm as a Markov process. Thus, the bandit problem is reformulated as maximising

$$\int \dots \int \mathbb{E}_{\boldsymbol{\theta}}[D_\gamma] \mathrm{d}b^1(\theta_1) \dots \mathrm{d}b^K(\theta_K)$$

where, b^a is the independent prior distribution on the parameter θ_a for $a = 1, \dots, K$. [Gittins, 1979] showed the agent can have an optimally indexed policy by sampling from the arm with largest Gittins index

$$G^a(s^a) \triangleq \sup_{\tau > 0} \frac{\mathbb{E}\left[\sum_{t=0}^{\tau} \gamma^t x^a(S_t^a) \mid S_0^a = s^a\right]}{\mathbb{E}\left[\sum_{t=0}^{\tau-1} \gamma^t \mid S_0^a = s^a\right]}$$

where s^a is the state of arm a and τ is referred to as the stopping time i.e, the first time when the index is no greater than its initial value. Though Gittins index [Gittins, 1979] is proven to be optimal for discounted Bayesian bandits with Bernoulli rewards, explicit computation of the indices is not always tractable and does not provide clear insights into what they look like and how they change as sampling proceeds [Nino-Mora, 2011].

Thus, researchers developed approximation algorithms [Lai, 1988] and sequential sampling schemes [Thompson, 1933]. [Kaufmann et al., 2012a] also proposed a Bayesian analogue of the UCB algorithm. Bayesian algorithms assume the existence of a set of distributions over the parameter of reward distributions of the arms. These distributions representing the uncertainty of the parameters are called *belief*

distributions. Unlike the original, it uses belief distributions to keep track of arm uncertainty and update them using Bayes' theorem, computes UCBs for each arm using the belief distributions, and chooses the arm accordingly. Thus, [Kaufmann et al., 2012a] tried to amalgamate the benefit of Bayesian modelling with performance and efficiency of UCB algorithms. They constructed Bayes-UCB algorithm on the basis of this proposition. Thus, we can trace back Bayes-UCB as an *optimism in the face of uncertainty algorithm* where the loss function is a quantile of a posterior reward distribution updated in a Bayesian manner.

The oldest but the simplest Bayesian algorithm for bandits is Thompson sampling [Thompson, 1933]. Thompson sampling is widely used for its simplicity and optimality [Agrawal and Goyal, 2012]. At any iteration, Thompson sampling samples K parameter values from the belief distributions. Following that, it chooses to play the arm that has maximum expected reward for the sampled parameter values. Despite of its simplicity, Thompson sampling is proved to reach the logarithmic regret bound [Agarwal et al., 2011] and is asymptotically optimal [Kaufmann et al., 2012b].

2.1.3 Pure Exploration Bandits

In the pure exploration setup proposed by [Bubeck et al., 2009], the agent tries to minimise the simple regret rather than cumulative regret. *Simple regret* is the difference between the expected difference between the best reward that could be achieved and the reward which has been achieved at any instance. Mathematically, the *simple regret* after n iterations is

$$r_t(\boldsymbol{\theta}) \triangleq \mu^*(\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{\theta}}[X_{A_t}]. \tag{2.10}$$

Unlike minimisation of cumulative regret that requires simultaneous exploration and exploitation, minimising simple regret depends only on exploration and the number of available rounds to do so. This setup represents the situation where the agent seeks to know more about the environment rather than exploiting it. It is also representative of applications where the cost of pulling an arm is expressed in terms of resources

rather than rewards. [Bubeck et al., 2009] also proved for Bernoulli bandits, that if an explore–exploit algorithm achieves an upper bounded regret, it cannot reduce the expected simple regret by more than a fixed lower bound. This establishes the fundamental difference between behaviour of efficient algorithms for explore-exploit bandits and pure exploration bandits. They also proposed a modification of UCB for the pure exploration setting of Bernoulli bandits. [Audibert and Bubeck, 2010] identified this pure exploration problem as *best arm identification* problem and proposed the Successive Rejects algorithm under fixed budget constraints. [Bubeck et al., 2013] extended this algorithm for finding m-best arms and proposed the Successive Accepts and Rejects algorithm. In another endeavour to adapt the UCB family to pure exploration scenario, the LUCB family of frequentist algorithms are proposed [Gabillon et al., 2012; Kaufmann and Kalyanakrishnan, 2013]. In the beginning, they sample all the arms. Following that, they sample both the arm with maximum expected reward and the one with maximum upper-confidence bound till the algorithm can identify each of them separately. But these algorithms do not provide us an intuitive and rigorous explanation of how a unified framework would work for both the pure exploration and the explore–exploit scenario and what would be the critical construction needed to do so. Furthermore, there is a clear void in the Bayesian literature to provide such a pure exploration solution [Kaufmann and Kalyanakrishnan, 2013]. As discussed in previous section, both Thompson sampling and Bayes-UCB lack this feature of constructing a single successful structure for both pure explore and explore-exploit. In another trend of literature, [Chen et al., 2014, 2016] focus on exploiting the underlying structure of the bandit problem or the constraints to develop better ways to explore the reward distributions of arms.

2.1.4 Our Contribution: BelMan

BelMan [Basu et al., 2018c], presented in detail in Chapter 5, proposes a Bayesian, information geometric algorithm that concurrently addresses the pure exploration problem, the exploration–exploitation problem and their concatenation in the form of the

two-phase reinforcement learning problem [Putta and Tulabandhula, 2017a]. BelMan maintains a distribution over the parameter of the reward distribution, and a joint distribution over both the parameter of the reward distribution and reward itself. We refer to the distribution over the parameter as the *belief distribution*, and to the joint one as the *belief-reward distribution*. Belief-reward distributions quantify the total uncertainty of the underlying process. We further investigate the *belief-reward manifold* of all possible belief-reward distributions. BelMan leverages the statistical manifold [Amari and Nagaoka, 2007] structure of the space of belief-reward distributions to respond to the questions of information accumulation, summarisation and exploration–exploitation trade-off. As rewards are accumulated, belief-reward distributions are updated using Bayes' theorem. This is equivalent to a displacement in the belief space, and in turn, in the belief-reward space. The underlying structure of statistical manifold allows us to rigorously formulate this assimilation of information in the belief distribution of each arm. Exploiting the belief-reward distribution of the arms independently would lead to an effective estimate of the reward distribution of the most played arm but would get myopically stuck in it. Efficient exploration requires a collective representation of the accumulated knowledge. BelMan uses the *pseudobelief-reward* distribution as the geometric representation of this summarised knowledge-base. *Pseudobelief-reward* minimises the sum of KL-divergences from the belief–rewards of the arms. We show the pseudobelief-reward is a weighted barycentre of the belief-reward distributions of the arms. In each iteration, the agent plays an arm to update this pseudobelief and leverage this to choose the next arm. Though pseudobelief-reward deals with exploration, in the exploration–exploitation problem, it is essential to exploit the present knowledge of rewards and to gradually increase exploitation in order to achieve higher cumulative reward [Macready and Wolpert, 1998]. BelMan constructs a *focal distribution* in the reward space that incrementally focuses on higher rewards with each iteration. This evolution towards higher values of reward depends on a time-variant factor, called *exposure*. Exposure decreases with time and its variation decides the exploration–exploitation trade-off. Amalgamation

of focal distribution with the pseudobelief-reward distribution leads to pseudobelief-focal distribution on the belief-reward manifold. Pseudobelief-focal provides the scope to exploit the present knowledge of rewards and to gradually increase exploitation in order to achieve higher cumulative reward [Macready and Wolpert, 1998]. BelMan sequentially takes decision to pull an arm and update the pseudobelief-focal distribution using the *alternating information projection* [Csiszár and Tusnády, 1984]. In the *information* (I-) projection, an arm is chosen such that its belief-reward distribution is at minimal distance with respect to the pseudobelief-focal distribution. In the *reverse information* (rI-) projection, the pseudobelief-focal distribution is updated by assimilating the observed reward of the last played arm such that its distance from all the arms' belief-reward distributions would be minimum. Thus, BelMan iteratively updates its knowledge about the reward distributions of the arms and decides the arm to be explored to maximise the cumulative reward as well as the information. By convergence of alternating information projection [Csiszár and Tusnády, 1984], BelMan asymptotically estimates the 'true' reward distributions for the arms and converges to the choice of the optimal arm.

2.2 Markov Decision Processes

Although bandit environments provide a useful formalism for many problems, they are limited in scope as the actions do not have long term consequences, and the environment remain unaffected by the actions taken by the agent. Such long term consequences, and the dynamic interaction of the agent and the environment are expressed through the notion of an environment state which changes as a function of the action taken. The interactions and changes can be both deterministic and stochastic. If after every transition the state becomes observable to the agent, we arrive at *fully-observable Markov decision processes*, or in short *MDPs*. Following the schema of bandits in Figure 2.2, we illustrate a scheme for the MDP in Figure 1.2.

Though the notion of agent, environment, action, reward function, history and policy as constructed for bandits would be here in extended forms, we would need to define new components like the transition function and the value function. We describe that in the problem setup of finite-state finite-action MDPs. Following that, we describe some of the basic algorithms developed to solve it and are related with the methodologies developed in this thesis.

2.2.1 Finite-State Finite-Action MDPs

Under its broadest definition, a Markov decision process formalism covers an eclectic set of models. This generality makes it a powerful formalism but provides too little structure to find efficient solutions. Thus, we define here the variant of Markov decision process interesting to our problem formats and for AI practitioners. In the corresponding chapters, we add the structures to this definition according to the requirement of the real-life applications to solve them efficiently and effectively.

A *finite-state finite-action discrete-time fully-observable Markov decision process* $\mathcal{M}$ is defined as a tuple $\langle S, A, [T], \mathcal{P}, \mathcal{R} \rangle$, where

a. S is the finite set of all possible states of the environment. S is called the *state space* with cardinality $\mid S \mid < \infty$.

b. A is the finite set of all possible actions that the agent take. A is called the *action space* with cardinality $\mid A \mid < \infty$.

c. $[T]$ is the finite or infinite sequence of natural numbers $\{1, 2, \ldots, T\}$ with time horizon $T \in \mathbb{R} \cup \{\infty\}$. If $T < \infty$, it is called *finite-horizon MDP*. If $T = \infty$, it is called *infinite-horizon MDP*.

d. $\mathcal{P} : S \times A \times S \times [T] \to [0, 1]$ is the *transition function*. $\mathcal{P}$ can be both stochastic and deterministic. For the stochastic variant, $\mathcal{P}(s_1, a, s_2, t)$ quantifies the probability of reaching state s_2 if the agent performs action a being in the state s_2 at time t. For the deterministic variant, range of $\mathcal{P}$ becomes $\{0, 1\}$ such that it

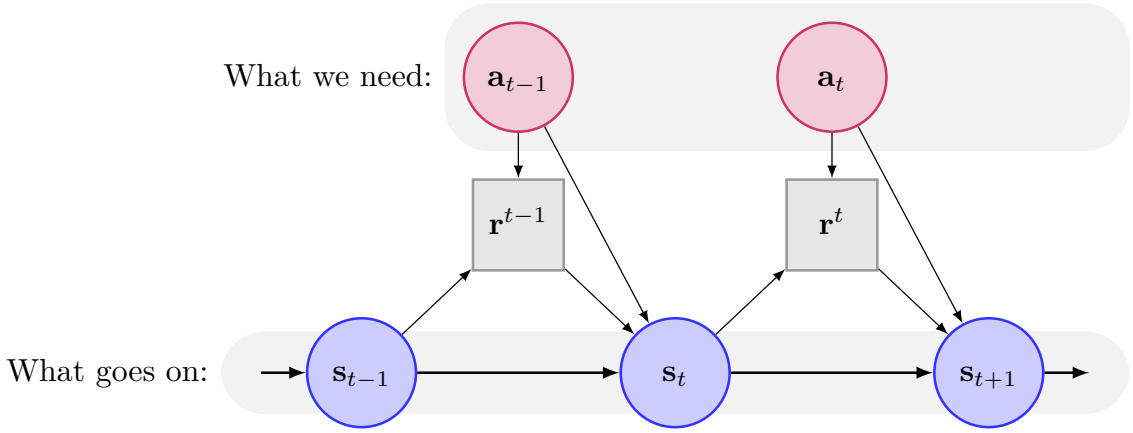


Figure 2.4: The probabilistic graph representing the temporal flow of an MDP.

deterministically dictates whether the agent moves to, or not, to a state s_2 if the agent performs action a being in the state s_2 at time t.

e. $\mathcal{R} : S \times A \times S \times [T] \to \mathbb{R}$ is the *reward function* that returns a finite numeric value to quantify the goodness of an action at a certain time at a certain state. $\mathcal{R}(s_1, a, s_2, t)$ quantifies the reward obtained as the environment transits to state s_2 from state s_1 while the agent performs action a at time t. $\mathcal{R}$ can generate both a deterministic value or a random variable.

Since we almost always discuss *finite-state finite-action discrete-time fully-observable Markov decision process* in this thesis, we will refer to them as *MDPs* for the rest of the thesis.

Markovian assumption. We depict the above mentioned dynamics of MDPs through the probabilistic graphical model of Figure 2.4. This figure depicts one of the fundamental observations of the MDP formalism. The transition function $\mathcal{P}(s_{t-1}, a_t, s_t, t)$ deciding transition from state s_{t-1} to state s_t does not depend on the previous states or actions. Similarly, the reward obtained at time t, i.e., $r_t \triangleq \mathcal{R}(s_{t-1}, a_t, s_t, t)$ does not depend on past states and actions such as s_{t-1} and a_{t-1}. This assumption of depending on the present state but not on the past sequence of state and actions in the

process is called the first-order Markov property. This assumption lays at the base of this formalism and its nomenclature.

History. As shown in the Figure 2.4, the agent takes an action a_t at each time t while she is in a state s_t. This returns her a reward r_t. The agent sequentially accumulate these observations to construct a history. The history $\mathcal{H}_t$ at time $t \in [T]$ is defined as an ordered set $\{(s_1, a_1, r_1), \ldots, (s_{t-1}, a_{t-1}, r_{t-1}), s_t\}$. Let us call the space of all possible histories the observed space of the agent $\mathcal{H} \triangleq \cup_{t=1}^{[T]}$. The agent tries and constructs functions that processes the history to decide the next action.

Policy. The function that determines the action of an agent given a history is called a policy $\pi : \mathcal{H} \times A \to [0, 1]$. It can be a probabilistic function or a deterministic function. Thus, it may outcome a probability distribution over A, or may choose a certain action at a certain point. Unluckily, this observed space can be very large due to large number of possible histories. Such eclectic possibilities and uncertainty makes it vary hard to deal with arbitrary history-dependent policies. Thus, in the literature, another layer of first-order Markov assumption is added. It is assumed that the choice of an action under a policy depends only on the current state and time step. This family of policies are called Markovian policies. Hence, if π is a Markovian policy and an action a is taken at state s_t, $\pi(\mathcal{H}_t, a) = \pi(\mathcal{H}'_t, a)$ for any two histories $\mathcal{H}_t$ and $\mathcal{H}'_t$ having s_t as the present state. Thus, Markovian policies are only functions of the present state, time step and the present action, i.e., $\pi : S \times [T] \times A \to [0, 1]$.

Fortunately, disregarding non-Markovian policies is hardly a limitation in practice. For most variants of MDPs and objective functions there exists at least one optimal Markovian policy [Szepesvári, 2010]. Thus, by restricting attention to Markovian policies we are not eliminating the opportunity to solve these MDPs optimally. Instead of that, we are making our solution space tractable enough to reach the optimal solution faster.

If $[T]$ is finite, the number of possible policies grow linearly with T. This can be really bad if T is large in a practical application. Beside this, there are infinite horizon

MDPs which are quite common in real-life applications since the horizon is not known. For example, if the agent is a scheduler of jobs in a factory, one cannot predefine the amount of time it has to work continuously or the days when it does not have to work at all. Rather it is practical to design it irrespective of the time horizon and assuming that the effect of a job schedule transcends far after it is finished. For such scenarios, the stationary assumption is imposed on the Markovian policy. A Markovian policy π is stationary if for any state s and action a, $\pi(s, t_1, a) = \pi(s, t_2, a)$ for any two time steps t_1 and t_2 in $[T]$. It implies that the decisions made by following π are independent of time. Thus, we can write stationary Markov policies as functions of state and action, i.e., $\pi : S \times A \to [0, 1]$. Though this makes the space of feasible policies even narrower, they construct a practical set of solutions for infinite horizon MDPs. [Szepesvári, 2010] shows that there exists at least one stationary Markovian policy as an optimal solution of the infinite horizon MDPs. If not explicitly mentioned, we would refer to the stationary Markovian policies as policies for any further discussion in this thesis.

Value function. When we were discussing policies, we mentioned about the optimal solutions of MDPs. This naturally evokes the question of quantifying optimality of a policy solving MDPs. This indicates the need of defining a utility metric on the accumulated reward obtained through a sequence of states and actions in an MDP. In reinforcement learning, we always assume existence of such a function quantifying the goodness of a sequence of states and actions numerically. This is called utility function hypothesis. It is originated from the prescriptive theory of decision making. Specifically, von Neumann and Morgenstern proved that under reasonable axioms of rational behaviour under uncertainty, a rational agent must choose amongst alternatives by computing the expected utility of the outcomes. Though this assumption of rationality of the agent is debated by the descriptive theorists of decision making, we stick to this formalism as we are dealing with computer systems and not the human beings. Thus, we define a value function $V : \mathcal{H} \to \mathbb{R} \cup \{-\infty, +\infty\}$ as a numeric quantification of goodness of a history. As we have discussed in case of policies, Markovian

and stationary assumptions are imposed on value functions to define Markovian and stationary Markovian value functions respectively. Markovian value functions are independent of the history. Thus, they are functions of present state and time only, i.e., $V : S \times [T] \to \mathbb{R} \cup \{-\infty, +\infty\}$. Stationary value functions are independent of the time. Thus, they are functions of present state only, i.e., $V : S \to \mathbb{R} \cup \{-\infty, +\infty\}$.

This definition of value function provides little structure to the value function. Specifically, this definition does not explicitly connect it with the sequence of actions that the agent has to take, and thus the policy she has to compute. Hence, the value function of a policy π is defined and often dealt with in practical algorithms. Thus, value function of a policy[6] is defined as a numeric utility metric computed on the future sequence of rewards obtained by using the policy. If we denote the reward obtained after time t by following the policy π from state s as $R^\pi(s_t)$, the value function is defined as $V^\pi(s) = u(R^\pi(s_t), \ldots, R^\pi(s_T))$ for an utility metric u. Rigorously speaking, this is the stationary Markovian value function $V^\pi(s)$ of a stationary Markovian policy π at state s. In this thesis, we would use this as the default definition of value function. Sometimes the value function is called utility function or cost-to-go function too. Generally in the literature and specifically in this thesis, all of these terminologies refer to the same quantity.

Now, the question reduces to how to define the utility metric and how does it effect the optimality of the policy computed to solve an MDP. Like the bandits, it is intuitive to define it as the sum or rewards obtained from state s at time t. Thus, $u(R^\pi(s_t), \ldots, R^\pi(s_T)) = \sum_{i=t+1}^{T} R^\pi(s_i)$. Since $R^\pi(s_i)$'s are random variables, they may vary in different iterations and they belong to a predictive future of the agent This definition of utility turns out to be a random variable which is not the same for different runs of even same policy. Thus, it is not a proper metric to use to compare between the performance of different policies. In order to resolve this, the expected linear additive utility is defined. An expected linear additive utility metric is a function u that computes the utility of reward sequence following a state and a policy as the

[6]Markovian stationary policy

expected sum of discounted reward of an MDP as

$$u(R^{\pi}(s_t), \ldots, R^{\pi}(s_T)) \triangleq \mathbb{E}_{\mathcal{P},\pi}\left[\sum_{i=t+1}^{T} \gamma^{i-t} R^{\pi}(s_i)\right]. \tag{2.11}$$

Here, $\gamma \in [0, 1]$ is the discount factor. The expectation is calculated with respect to the transition function and the the policy that dictated the reward generation process. Defining expectation as the utility metric is dependent on the application and intention of the agent as we have discussed for the bandits. In the scope of the thesis and the discussed application, we would stick to expectation as the eligible metric.

This expected linear additive utility is chosen from the intuition that the utility of a reward sequence is it expected sum. If the discount factor $\gamma = 1$, all the reward in a sequence are equally valued. For infinite horizon MDPs, $\gamma = 1$ makes the expected sum of reward divergent and thus immeasurable for different policies. Thus, the discount factor is assumed to be less than 1. Above this, the discount factor being zero means that no future effect is recognised while computing the value function. A reasonable discount factor belongs to $(0, 1)$. This implies that the utility of a state is captured better by its recent rewards than the rewards obtained in distant future and the effect of a state in reward generation decreases by the discount factor per-step.

This discussion provides us a valid formulation of a value function of a policy given an initial state as an expected linear additive function of the obtained rewards. Though the choice is not universal and unique, the discussion clarifies the intuition supporting the choice. Specifically, the Belman's optimality principle which we will elaborate now provides it the additional eligibility as the measure of optimality of a sequence of actions in an MDP.

The optimal solution. As we have defined all the component to construct the formalism of an MDP and its solutions, we want to discuss the optimality of a policy solving an MDP.

A policy π^* is an optimal solution of an MDP $\mathcal{M} \triangleq \langle S, A, [T], \mathcal{P}, \mathcal{R} \rangle$ for a given value function V such that the value function V^{π^*} dominates the value functions of all other feasible policies for all the initial states $s \in S$. This means $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $s \in S$ and for all $\pi \neq \pi^*$. π^* is called an *optimal policy*.

For the choice of expected linear additive value function, we obtain the Belman's optimality principle. This is a cornerstone of MDP literature, and in general, of the dynamic programming approach to problem solving. The MDP version of this can be stated as:

If goodness of every feasible policy π can be measured for every initial state by the corresponding expected linear additive value function, there exists a policy that is optimal in each step. The policy which is optimal in every step is also a globally optimal policy π^* and vice-versa.

This result assure that search for an optimal policy under such formalism is not a dead end. The existence of the solution helped the researchers to focus only on the solution methodologies. Though the result is general enough, the condition is important to check while designing MDP for a real-life problem because there are exception [Puterman, 2009]. It also reminds us of the fact that though the optimal policy exists, it is not unique. This fact plays an important role in designing more specific variants with unique and global optimal solution.

Infinite-horizon MDPs. Although finite-horizon MDPs have simple mathematical properties and suited for some mathematical analysis, it is quite restrictive and also not practical as the time horizon is not known often. For example, in the virtual machine migration setting of Chapter 4, the scheduling algorithm has to keep on deciding which virtual machine to migrate to which physical machine in the data center till the data center or the algorithm is shut down. In such scenarios, the reward is accumulated over a virtually infinite sequence of time steps. Thus, infinite-horizon MDP is the formalism followed in the applications discussed in this thesis.

Solving the infinite horizon MDPs require three specific design assumptions. Since the time horizon is infinite, $[T]$ is an uncountable set. Thus, the domains of the transition function $\mathcal{P} : S \times [T] \times A \to [0, 1]$ and the reward function $\mathcal{R} : S \times [T] \times A \to \mathbb{R} \cup \{-\infty, +\infty\}$ are uncountably large. This makes any computation involving them intractable. It leads to imposition of stationarity on the transition function and reward function as a necessary design practice. Transition function $\mathcal{P}$ is stationary if $\mathcal{P}(s_1, a, s_2, t) = \mathcal{P}(s_1, a, s_2, t')$ for any states $s_1, s_2 \in S$, action $a \in A$, and time $t, t' \in [T]$. Similarly, the reward function $\mathcal{R}$ is stationary if $\mathcal{R}(s_1, a, s_2, t) = \mathcal{R}(s_1, a, s_2, t')$ for any states $s_1, s_2 \in S$, action $a \in A$, and time $t, t' \in [T]$. This implies that they are independent of time and are only functions of present state, future state, and present action. We call the MDPs with stationary transition and reward functions as stationary MDPs.

The other design constraint is to keep the expected linear additive value function finite. As we have discussed earlier, this is done by introducing a discount factor $\gamma \in [0, 1)$. Generally the value of γ is hard to know a priori. It is commonly set through analysis of experimental results as we will see in Chapter 3 and 4. The only plausible information we have for assigning the value is not to set it to zero as it makes the value function oblivious to the future effects. The other application specific information is available for finance problems, where γ can be modelled as the rate of inflation or the interest as computed from economic data.

Additionally, we are assuming use of stationary Markovian policies and stationary Markovian value functions as the default design choices. Though they are not a fundamental requirement to design and solve an infinite-horizon MDP, we follow them due to the reasons mentioned in earlier discussions.

Thus, we can define an infinite-horizon Markov decision process $\mathcal{M}^\infty$ as a tuple $\langle S, A, \mathcal{P}, \mathcal{R}, \gamma \rangle$ with value function $V^\pi(s) = \mathbb{E}_{\mathcal{P},\pi} \left[\sum_{t=0}^{\infty} \gamma^t R^\pi(s_t) \mid s_0 = s\right]$ defined for a policy $\pi \in \Pi$. Here, S is a finite state space, A is a finite action space, $\mathcal{P}$ is a stationary transition function, $\mathcal{R}$ is a stationary reward function with bounded rewards, γ is the discount factor, and Π is the space of feasible Markov stationary policies. They are also

called infinite-horizon discounted-reward MDPs [Puterman, 2009]. Since this structure provides an effective formalism for the large scale problems with indefinite endpoints, we will use it as the default structure throughout Chapter 3 and 4.

As we have defined the problem setup, we would elaborate the optimality principle derived for this specific case.

Theorem 4 (Optimality principle of infinite-horizon MDP)**.** *Let us formulate an infinite horizon MDP as* $\mathcal{M}^{\infty} \triangleq \langle S, A, \mathcal{P}, \mathcal{R}, \gamma \rangle$ *with value function* $V^{\pi}(s) = \mathbb{E}_{\mathcal{P},\pi}\left[\sum_{t=0}^{\infty} \gamma^t R^{\pi}(s_t) \mid s_0 = s\right]$ *and* $\gamma \in [0, 1)$*. Under such a formalism, there exists a stationary Markovian optimal value function* V^* *solving* $\mathcal{M}^{\infty}$ *such that for all* $s \in S$*,*

$$V^*(s) = \max_{a \in A} \mathbb{E}_{s' \sim \mathcal{P}(s,a,\cdot)} \left[\mathcal{R}(s, a, s') + \gamma V^*(s')\right]. \tag{2.12}$$

The corresponding optimal policy π^* *is unique and deterministic stationary Markovian, and satisfies for all* $s \in S$

$$\pi^*(s) = \arg\max_{a \in A} \mathbb{E}_{s' \sim \mathcal{P}(s,a,\cdot)} \left[\mathcal{R}(s, a, s') + \gamma V^*(s')\right]. \tag{2.13}$$

2.2.2 Functional Abstraction of MDP

[Bertsekas, 2013, 2017] proposed an abstraction of this formulation of optimality principle using the techniques of functional analysis and calculus of variations [Gelfand et al., 2000]. They proposed this framework from a general perspective which is valid for optimal control problems, stochastic control problems and MDPs. We would express it using the formalism of infinite-horizon MDPs only [Chapter 4 of [Bertsekas and Shreve, 2004;Chapter 4]. If we assume the value function for a policy π is $V^{\pi} : S \to \mathbb{R} \cup \{-\infty, +\infty\}$ and $\mathcal{V}(S)$ is a set of such feasible value functions, we can

define another mapping $J : S \times A \times \mathcal{V}(S) \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$ such that

$$J(s, a, V^{\pi}(s)) \triangleq \mathbb{E}\left[\mathcal{R}(s, a, s') + \gamma V^{*}(s')\right],$$

Using this construction, we can express Equation (2.12) using the Bellman operators $\mathcal{T}$. Bellman operator $\mathcal{T}_{\pi}$ for a policy π is defined as

$$(\mathcal{T}_{\pi} V)(s) \triangleq J(s, \pi(s), V^{\pi}(s)) \qquad \forall s \in S. \tag{2.14}$$

This, in turn, defines the Bellman operator $\mathcal{T}$ as

$$(\mathcal{T} V)(s) \triangleq \inf_{a \in \pi(s)} J(s, a, V^{\pi}(s)) \qquad \forall s \in S. \tag{2.15}$$

Let us we assume that there exists a termination stage where the value function return $\bar{V}(s) = 0$ for all $s \in S$. We can always construct such a situation for infinite-horizon MDPs [Bertsekas and Shreve, 2004]. Under mild conditions guaranteeing that Fubini's theorem holds [Section 2.3.2 of [Bertsekas and Shreve, 2004]], it reconstructs the value function of infinite-horizon MDP as

$$\begin{aligned} V^{\pi}(s_0) &= \limsup_{T \rightarrow \infty} \quad \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, \pi(s_t), s_{t+1})\right] \\ &= \limsup_{T \rightarrow \infty} \quad (\mathcal{T}_1 \ldots \mathcal{T}_T \bar{V})(s_0). \end{aligned}$$

If $\gamma < 1$ and the reward function is bounded such as $|\mathcal{R}(s, a, s')| < R_{max}$, the Bellman operator $\mathcal{T}$ and Bellman operator $\mathcal{T}_{\pi}$ for policy π are contraction mappings with respect to the the standard sup-norm. Mathematically, we can show that for all $\pi \in \Pi$ and for all $s \in S$, $|| \mathcal{T} V_1 - \mathcal{T} V_2 ||_{\infty} \leq \gamma || V_1 - V_2 ||$, where $|| \, . \, ||_{\infty}$ is the functional sup-norm[7] and the discount factor $\gamma < 1$. Following the results of Chapter 2 of [Bertsekas, 2013], we conclude that the Bellman's equation of an infinite-horizon MDP (Equation (2.12)) has a unique solution. This solution corresponds to the unique fixed points in $\mathcal{V}(S)$

[7] $|| \, . \, ||_{\infty}$ is the sup-norm such that $|| V ||_{\infty} \triangleq \max_{s \in S} V(s)$.

of the Bellman operators $\mathcal{T}$ and $\mathcal{T}_\pi$ since $\mathcal{T}V^* = V^*$ and $\mathcal{T}_{\pi^*}V_\pi = V_\pi$. These fixed points are the optimal value function V^* and the value function of the stationary policy V_π respectively. The solution V^* corresponds to the stationary policy π^* such that $\mathcal{T}_{\pi^*}V^* = \mathcal{T}V^*$.

This functional approach coupled with the Bellman's optimality principle constructs the theoretical basis and design guideline for MDP algorithms. Value iteration algorithm is equivalent applying a sequence of Bellman operators $\mathcal{T}_t$ starting from a $V_0 \in \mathcal{V}(s)$ such that $V^* = \lim_{t\to} \mathcal{T}_t V$ is satisfied. Policy iteration algorithm is equivalent to the scheme of starting from an initial stationary policy π_0 and generating a sequence of stationary policies π_t by solving $\mathcal{T}_{\pi_{t+1}}V_{\pi_t} = \mathcal{T}V_{\pi_t}$. Here, V_{π_t} is obtained as the fixed point of the Bellman operator $\mathcal{T}_{\pi_t}$. This is done through different optimisation methods even through value iteration. Such a combination of policy update and policy evaluation together constructs the actor-critic algorithms. This is the principle family of methods which is used to solve MDP problems in Chapter 3 and 4.

If the dimension of $\mathcal{V}(S)$ is really large which means that the MDP has large state-action space and/or large policy space, it is computationally expensive to compute the fixed points of the Bellman operators. It is often found that for real-life applications with some problem structure the fixed point of the Bellman operator is often based on the solution of lower-dimensional equations defined on the subspace $\mathcal{V}^{approx}(S)$. $\mathcal{V}^{approx}(S)$ is defined as a projection plane $\{\Phi\tilde{V} \mid \tilde{V} \in \mathbb{R}^{|S|}\}$ which is spanned by the columns of a $d \times |S|$ matrix Φ for $d < |S|$. There are different methods of performing this approximation. Some methods perform it through multi-step version of the mapping, such as TD(λ), LSTD (λ) and other temporal difference algorithms [Sutton, 1988]. Some methods perform this through functional approximation of the Bellman operator and the value function such as LSPI, FDD, iFDD. Some methods combine these two methodologies to perform the functional approximation through sequential steps such as Gradient SARSA, Batch-iFDD augmented LSTD. The algorithms developed by us in Chapter 3 and 4 fall into this category due to their sequential functional

approximation schemes as solving the large MDPs on-the-go requires both of these techniques to be merged.

2.2.3 Dynamic Programming

Dynamic programming is a problem solving methodology inspired by the Bellman's optimality principle. In the dynamic programming, we generally divide a bigger problem into smaller overlapping problem such that their combined solution answers the original bigger problem. The smaller subproblems are solved sequentially and their results are stored in some tabular format. This is often referred to memoisation. Then, they are assimilated according to the optimality corresponding Bellman operator to get the global solution. As mentioned in Theorem 4 and the concluding discussion of the preceding section, we observe that dynamic programming is one of the basic and intuitive approaches to design algorithms for solving MDPs. Here, we discuss two such basic algorithms such as value iteration and policy iteration. Following that, we modify this technique to discuss the temporal difference algorithms. We conclude this discussion by stating the practical limitations of these memoisation based dynamic programming algorithms because they consume large memory and computational time for large MDPs.

Value iteration

Value Iteration (VI) forms the basis of most of the MDP algorithms. It was originally proposed by Richard Bellman in 1957 [Bellman, 1957a]. It is an indirect method motivated by Equation (2.12) that searches in the value function space rather than the policy space, and computes the current policy based on current estimates of value function. VI computes the possible improvement of a value function for each of the states, and choose the one with maximum increment as the updated value function for that state. The pseudo-code is described in Algorithm 5. VI tabulates the Q value for each of the state-action pairs (s, a), where $Q : S \times A \to \mathbb{R}$. Thus, each iteration of value iteration takes time $\mathrm{O}\big(|S|^2 |A|\big)$ in the worst case. Though it is still

Algorithm 5 Value Iteration Algorithm

1: Initialise: a value function V_0, say $V_0(s) = 0$ for all $s \in S$.
2: **if** $|| V_t - V_{t-1} ||_\infty > \epsilon$ **then**
3: **for all** $s \in S$ **do**
4: **for all** $a \in A$ **do**
5:

$$Q_t(s,a) = \sum_{s' \in S} \mathcal{P}(s,a,s') \left[\mathcal{R}(s,a) + \gamma V_{t-1}(s')\right] \tag{2.16}$$

6: **end for**
7: $V_t(s) = \max_{a \in A} Q_t(s,a)$
8: **end for**
9: **end if**

polynomial time on the dimensions of the state and action spaces, it can be quite computationally costly and impractical for MDPs with large state-action spaces and real-time applications.

Though we obtain the computational complexity for each time step, the next question is whether it converges to a nearly-optimal enough value function. [Williams, 1993] proved that if the maximum difference between two successive value functions is less than ϵ, the value of the greedy policy obtained by Algorithm 5 differs from the optimal value by not more than $\frac{2\epsilon\gamma}{1-\gamma}$. Mathematically,

Theorem 5. *If* $|| V_{t+1} - V_t ||_\infty \leq \epsilon$ *for* $\epsilon \geq 0$*, then we get a value function* V_t *which is* $\frac{2\epsilon\gamma}{1-\gamma}$ *close to the optimal value function, i.e.,* $|| V_t - V^* ||_\infty \leq \frac{2\epsilon\gamma}{1-\gamma}$.

This proves convergence of this algorithm for infinite-horizon MDPs. The proof technique uses the fact that VI is equivalent to applying a sequence of Bellman operators $\mathcal{T}_t$ satisfying $V^* = \lim_{t \to} \mathcal{T}_t V$, and the property that the Bellman operator for the infinite-horizon MDP is a contraction mapping with a contraction factor γ.

Policy iteration

Policy iteration (PI) is another fundamental algorithm for solving MDPs that searches directly in the policy space than optimising the value function. Policy iteration con-

sists of two processes, policy evaluation and policy improvement. Policy evaluation computes the value functions consistent with a given policy. Policy improvement greedily updates the policy with respect to the value function obtained in policy evaluation. Using the functional formulation of dynamic programming, we illustrate it in Algorithm 6.[8] Using this construction of the policy iteration algorithm, the monotonicity

Algorithm 6 Policy Iteration Algorithm

1: Initialise: a policy π_0, say $\pi_0(s) = \text{Unif}(A)$ for all $s \in S$.
2: **if** $\pi_t \neq \pi^*$ **then**
3: Policy Evaluation: Compute V_{π_t} as the unique solution of $V_{\pi_t} = \mathcal{T}_{\pi_t} V_{\pi_t}$
4: Policy Improvement: Find a policy π_{t+1} such that $\mathcal{T}_{\pi_{t+1}} V_{\pi_t} = \mathcal{T} V_{\pi_t}$
5: **end if**

of the mapping J, and the contraction property of the Bellman operator $\mathcal{T}$, we get that PI asymptotically converges to the optimal value function.

Theorem 6. *If $[\pi_t]$ is the sequence of policies generated by PI, then for all t there would be a certain improvement in the value function until PI reaches the optimal value function. This means $V_{\pi_{t+1}} \leq V_{\pi_t}$. The equality holds if and only if $V_{\pi_t} = V^*$. Additionally, PI asymptotically converges to optimal value function*

$$\lim_{t \to \infty} || V_{\pi_t} - V^* ||_\infty = 0.$$

In practice, the policy evaluation is done by some iterative optimisation method, and policy improvement is also not performed indefinitely. In the simplest format, policy is evaluated through iterative usage of Bellman's equation till an error threshold δ. The policy improvement is carried forward till the corresponding value function improvement is bounded by an error threshold ϵ. We illustrate the pseudocode in Algorithm 7.

Following a similar scheme as Theorem 6, it can be proved that the policy obtained through PI returns a value function which is $\frac{\epsilon + 2\gamma\delta}{1-\gamma}$ close to the optimal value function.

[8] $\text{Unif}(A)$ is a uniform distribution over the finite space A such that the probability of choosing each element is $\frac{1}{|A|}$.

Algorithm 7 (Approximate) Policy Iteration Algorithm

Initialise: a policy π_0, say $\pi_0(s) = \text{Unif}(A)$ for all $s \in S$.
if $(\pi_{t+1} \neq \pi_t) \wedge \left(|| V_{\pi_{t+1}} - V_{\pi_t} ||_\infty > \epsilon\right)$ **then**
  if $|| \hat{V_{t+1}} - V_{\pi_t} ||_\infty > \delta$ **then**
    for all $s \in S$ **do**
      $\hat{V_{t+1}}(s) = \sum_{s' \in S} \mathcal{P}(s, \pi_t(s), s') \left[\mathcal{R}(s, \pi(s)) + \gamma \, V_t(s')\right]$
    end for
  end if
  for all $s \in S$ **do**
    $\pi_{t+1}(s) = \arg\max_a \sum_{s' \in S} \mathcal{P}(s, a, s') \left[\mathcal{R}(s, a) + \gamma \, \hat{V_{t+1}}(s')\right]$
  end for
end if

This shows convergence and optimality of PI. PI is often used than VI due to the feasible policy space is often smaller than the value function space, and in practical problems, we are more interested to find out the optimal policy than the value function.

Curse of dimensionality and approximate dynamic programming

The dynamic programming methodologies uses full-backup scheme which means that they store all the history of the MDP till a stage. For each backup, they compute the value functions for each successor state and action while also use the complete knowledge of reward and transition functions to update the value function of the present state. Thus, the memoisation trick proves to be computationally expensive both in terms of space and time. This phenomenon makes dynamic programming techniques applicable for medium-sized MDPs with a knowledge of the underlying reward and transition function model. Though in reality, the number of states $|S|$ grows exponentially with the number of state variables, i.e., the degrees of freedom of the state space. Thus, the dynamic programming methodologies become intractable for large MDPs as even one backup takes too long to be performed. This bottleneck of reinforcement learning problems is called the *curse of dimensionality*.

Solving curse of dimensionality played a central role in the MDP literature. This endeavour created the works on approximate dynamic programming [Powell, 2007]. The general idea is to approximate the value function using eclectic methodologies such as sampling, linear projection, decision trees, Fourier or wavelet basis, and neural networks [Silver et al., 2016]. The basic idea behind this methodology is discussed in the final paragraph of Section 2.2.2 which is nothing but approximating the Bellman operators and the value function in a lower dimensional subspace where computations are less expensive to carry on. There are two principle families of such approximation techniques such as the temporal difference methods and functional approximation methods. We discuss the corresponding methodologies and algorithms in Sections 2.2.5 and 2.2.6 respectively. Generally, construction of such approximation takes extensive training of the algorithm. The algorithms that perform elaborate training are called offline algorithms. The algorithms that start from a tabula rasa and learns as it goes are called online or real-time algorithms. In this thesis, we develop online algorithms that work in real-time application to maintain the generality and efficiency of the solutions.

2.2.4 On-Policy and Off-Policy Learning

Before delving into the approximation techniques, we would like to shed light on another central challenge of reinforcement learning: the exploration–exploitation trade-off [Macready and Wolpert, 1998]. The agent has to learn the optimal policy while behaving non-optimally by exploring all actions. This dilemma engenders two approaches for learning action values: on-policy and off-policy [Sutton and Barto, 1998].

In on-policy methods, the agent learns the best policy while using the same to make decisions. Off-policy methods separate it into two policies. The agent learns a policy different from what currently dictates her behavior. The policy that she learns about is called the target policy. The policy that dictates her behaviour of action choice is called the behaviour policy. Since learning is from experience *off* the target policy, these methods are classified as off-policy. The on-policy methods are

generally simpler than off-policy methods but they learn action values not for the optimal policy, but for a near-optimal policy that the agent explores. The off-policy methods learn the optimal policy, and they are considered more powerful and general. They often cause greater variance and slower convergence. While on-policy methods learn policies depending on actual behaviour, off-policy methods learn the optimal policy independent of agents actual behaviour, i.e., the policy actually used during exploration.

For example, SARSA [Rummery and Niranjan, 1994] updates action values using a value of the current policys action a in next state s. Thus, it is an on-policy algorithm. Q-learning [Watkins and Dayan, 1992] updates its action-value using the greedy (or optimal) action a of next state s but the agent selects an action by ϵ-greedy policy. Here, the target policy is the greedy policy and the behaviour policy is commonly the ϵ-greedy policy. Thus, Q-learning is an off-policy algorithm. We discuss these algorithms in further details in the following section.

2.2.5 Temporal-Difference Algorithm

As mentioned in the final paragraph of Section 2.2.2, temporal difference (TD) methods try to sequentially estimate the value function and the Bellman operator. They do not assume any model for the transition function or the reward function but try to estimate them through sequentially accumulated state, action, and reward data. Thus, these methods are called model-free reinforcement learning algorithms. They operate in on-line and fully incremental manner. TD methods are the widely used due to their computational simplicity, on-line learning approach, and learning directly from experience as they go.

In TD algorithms, the total learning period is partitioned into episodes. Each episode consists of a finite horizon or transitions from an initial state to a terminal state. At each time step t of an episode, the agent takes an action a_t from the present state s_t by following its behaviour policy. This leads the agent to another state s_{t+1} with reward r_t. By accumulating these information, the algorithm updates

Dynamic Programming with Full Backup	TD Algorithms with Sequential Backup
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	TD Learning (TD(0)) $V(s) \leftarrow V(s) + \alpha(R + \gamma V(S') - V(s))$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in A} Q(S', a') \mid s, a\right]$	Q-Learning $Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma \max_{a' \in A} Q(S', a') - Q(s,a)\right)$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S', A') \mid s, a\right]$	SARSA $Q(s,a) \leftarrow Q(s,a) + \alpha(R + \gamma Q(S', A') - Q(s,a))$

Table 2.3: Comparing dynamic programming and temporal difference algorithms.

the value function of s_t to $V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$. Here, the value function is updated using the difference between the current estimate of value function at s_t, and the updated value depending on the observed reward and estimated value of the next state s_{t+1} which is $r_t + \gamma V(s_{t+1})$. This quantity is called the TD error $\delta_t \triangleq r_t + \gamma V(s_{t+1}) - V(s_t)$. With repetitive update of the value function of each state, their estimates become more and more accurate. Here, $\alpha \in (0, 1]$ is called the step-size or the learning rate parameter. For $\alpha = 1$, the TD algorithm becomes myopic and considers only the one step information. Thus, for convergence of a TD algorithm, α has to be in $(0, 1)$. Specifically, we see in the proofs that if α is properly reduced over time, the algorithm converges to optimal value function.

The relation between the full-backup dynamic programming methods and the temporal difference algorithms is shown in Table 2.3.

Q-learning [Watkins and Dayan, 1992]. Q-learning is one of the off-policy TD algorithms. This method is called Q-learning because it proposed to deal with the value iterated of state-action pairs than that of the states only. The value functions for state-action pairs are named as the Q-values $Q : S \times A \to \mathbb{R}$. Hence, the algorithm is called Q-learning. The pseudocode of Q-learning is illustrated in Algorithm 8. This method learns the Q-values based on transitions from a state-action pair to a state. The actions

Algorithm 8 Q-learning

1: Initialise: $Q(s,a) > 0$ for all $s \in S$ and $a \in A$, $Q(terminalstate, a) = 0$ for all $a \in A$.
2: **for** Each episode **do**
3: Choose an initial state s
4: **for all** Time step t in the episode **do**
5: **if** s is not a terminal state **then**
6: Choose a from the present state using a behaviour policy that uses Q-values (e.g Boltzmann exploration)
7: Take action a
8: Observe the reward r and new state s'
9: Update the Q-value of (s,a) to $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_a Q(s',a)]$
10: $s \leftarrow s'$
11: **end if**
12: **end for**
13: **end for**

are chosen using a behaviour policy that induces the exploration–exploitation trade-off. The behaviour policy can be a purely greedy policy which means that the empirically best action is greedily selected all the time. The behaviour policy can be a purely exploratory policy which means that all the actions are chosen uniformly randomly all the time. Generally, the behaviour policies are no designed to belong to one of these extremes rather to keep a balance between exploration and exploitation. We discuss such methods in Section 2.2.8. If the state-action pairs are visited infinitely often and the step-size $\alpha \in [0,1)$, Q-learning is proved to converge to the optimal Q-values Q^* with probability 1 [Jaakkola et al., 1994].

Though this method can learn irrespective of the underlying model and the behaviour policy used (e.g. ϵ-greedy or Boltzmann exploration), it suffers from slow convergence and instability. Specifically, the optimal policy does not traverse through all the state-action pairs or even the MDP is not stationary, which are the basic criteria for the convergence of Q-learning.

SARSA [Rummery and Niranjan, 1994]. SARSA is proposed as the on-policy version of Q-learning. This method learns Q-values based on transitions from a state-

Algorithm 9 SARSA

Initialise: $Q(s,a) > 0$ for all $s \in S$ and $a \in A$, and $Q(\text{terminal state}, a) = 0$ for all $a \in A$.
for Each episode **do**
    Choose an initial state s
    Choose a from the state s using a policy that uses Q-values (e.g Boltzmann exploration)
    for all Time step t in the episode **do**
        if s is not a terminal state **then**
            Take action a
            Observe the reward r and new state s'
            Choose a' from the new state s' using a policy that uses Q-values (e.g Boltzmann exploration)
            Update the Q-value of (s,a) to $Q(s,a) \leftarrow (1-\alpha)Q(s,a)+\alpha[r+\gamma Q(s',a')]$
            $s \leftarrow s'$
            $a \leftarrow a'$
        end if
    end for
end for

action pair to another state-action pair. Since it uses a 5-tuple of station-action-reward-state-action $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ for each update, it is named SARSA. Similar to the Q-learning, if all state-action pairs are visited infinitely often and $\alpha \in [0,1)$, SARSA is proved to converge to the optimal Q-values Q^* with probability 1 [Jaakkola et al., 1994]. The real-life performance of the algorithm highly depends on the α and the behaviour policy itself.

TD(λ) algorithm [Sutton, 1988]. The TD algorithm is further generalised to TD(λ) algorithm with $\lambda \in (0,1)$. Instead of looking into one-step into future like TD algorithm, the TD(λ) algorithms try to look into n-steps into future and ideally till the infinite horizon. They try to predict using some sampling method, such as Monte Carlo sampling, up to n steps in future. This allows them to predict the n-step future return at time t as $G_t^n \triangleq R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$. Thus, the λ return to the infinite-horizon future is defined as $G_t(\lambda) \triangleq (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^n$. This allows us to define the TD(λ)-error as $\delta_t(\lambda) \triangleq G_t(\lambda) - V(s_t)$. This value is now

used than the standard TD error to update the value function. Additionally, we can conclude the original TD algorithm to be a special case of TD(λ) algorithm for $\lambda = 0$.

From a functional point of view, we can think of this as approximating the Bellman operator T with a multistep operators $T^{(\lambda)} : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ such that

$$(T^{(\lambda)}V)(s) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}(T^n V)(s) \qquad \forall s \in S. \tag{2.17}$$

Here, λ is a predefined parameter in $(0, 1)$ and T^n is the n-fold composition of Bellman operator T with itself for n times. Following this, the mapping $T^{(\lambda)}$ is used in place of T in the Bellman optimality equation to determine the optimal value function. This provides a framework to design model-free, online, incremental method to solve the MDP problems.

Still there is one more glitch that appears in case of large state-action space, it is really expensive to compute the exact value of λ return or the TD(λ) error. Beside this still the dimension of the value function is as large as the state space, and that of the Q-value function is as large as the state-action space. This takes us to the problem of approximating the value functions or Q-value functions to a subspace of smaller dimension that would lead to efficient and fast computation. Following this queue, we describe the functional approximation methods in the next section, and illustrate the state-of-the-art algorithms to respond to this issue.

2.2.6 Functional Approximation Algorithms

Classical reinforcement-learning algorithms are mostly applied to small finite and discrete state spaces. They represent the value functions using a tabular form that stores the state-action values. In small and discrete problems, a lookup table represents all state-action values of a learning space. However, in real-life applications with large and continuous state spaces, the size of the table grows exponentially. In such problems, a major challenge is to represent and store value functions. Thus, the tabular methods typically used in reinforcement learning are improved using functional approximation

algorithms to apply to such large problems. The approximate value function is represented as a parametrised function of the weight vector $\Phi \in \mathbb{R}^d$. $\hat{V}(s, \Phi) \approx V_\pi(s)$ denotes the approximate value of state s given weight vector Φ.

Mean-square error algorithms

In tabular algorithms like value iteration, learning at a certain state yields an update to its value function, and the values of the other states remain unchanged. An update is applied only to the current state and it does not affect value functions of the other states. In approximation algorithms, the number of states is larger than the number of weights w. Thus, an update at one state affects the estimated values of other states. Updating one state makes the estimated value more accurate while it turns the values of other states less correct. Hence, the functional approximation algorithms aim to balance the errors in different states rather than trying to make zero error of value functions for all states. Thus, it is necessary to specify a state weight or distribution $p(s) \geq 0$ and $\sum_s p(s) = 1$ to weigh the error in each state s. The squared difference between the approximate value $\hat{V}(s, \Phi)$ and the true value $V_\pi(s)$ is averaged with weights spanning over the state space by p. The mean squared value error is denoted by $VE : \mathbb{R}^d \to \mathbb{R} \cup \{0\}$, such that

$$VE(\Phi) \triangleq \sum_{s \in S} p(s) \left[V_\pi(s) - \hat{V}(s, \Phi) \right]^2 . \tag{2.18}$$

Gradient-descent algorithms

We consider gradient-descent algorithms to minimise the mean squared error of the observed data. The gradient-descent algorithms are commonly used for functional approximation. The approximate value function $\hat{V}(s, \Phi)$ is a differentiable function of the weight vector Φ for all $s \in S$. We update the weight vector Φ. By the gradient descent method, the weight vector is changed by in the direction that minimises VE, i.e. the error between true value function under policy π and the approximate value

function $\hat{V}(s, \Phi)$).

$$\Phi_{t+1} \triangleq \Phi_t + \alpha[V_\pi(s_t) - \hat{V}(s_t, \Phi_t)]\nabla\hat{V}(s_t, \Phi_t). \tag{2.19}$$

α is a positive parameter called learning rate. $\nabla\hat{V}(s_t, \Phi_t)$ is the gradient of $\hat{V}$ with respect to Φ. In Table 2.4, we enlist the gradients used for different algorithms solving MDP.

Practically, this gradient is calculated sequentially or in batch by collecting data points. If the update is done on a single example, the update is called a 'stochastic gradient-descent update. When more than one example is used for the update, the gradient-descent algorithm is called 'batch.

Feature mapping

In linear functional approximation, the value is computed as a sum of features times corresponding weights. The computation relies on features. Appropriate design of features facilitate estimation of values, but if the features are selected improperly, it may result poor performance. Features are designed to represent the state-action space and to process the information necessary to learn the environment's dynamics. Selecting appropriate features, i.e., feature engineering, is a challenge as it needs domain-specific knowledge. Representational design is based on the system designers knowledge and intuition. Additionally, the linear form has a limitation that it cannot take into account any interactions between features or even nonlinearity of features. Linear approximations assume that each feature is linearly independent of other features. It is not computationally tractable for a designer to choose features with considering all interaction between features. Research works have addressed this problem to construct features automatically. These methods are based on errors of the value function and add features to improve the estimation of value estimation. Incremental Feature Dependency Discovery (iFDD) is an online feature expansion method that facilitates a linear function approximation. iFDD sequentially creates features

that eliminate error of the value function approximation. The process begins with an initial set of binary features. iFDD identifies all conjunctions of existing features as potential features and increases the relevance of each potential feature by the absolute approximation error. If a potential features relevance crosses a threshold, the feature is added to the pool of 'good' features used for future approximation. iFDD also has a variant for the batch setting. Batch-iFDD is a Matching Pursuit algorithm with guaranteed rate of error-bound reduction. Like iFDD, Batch-iFDD does not require a large pool of features at initialization but expands the pool of potential features incrementally which are the conjunction of previously selected features. Batch-iFDD runs LSTD algorithm to estimate the TD-error over all samples. Following that, it adds the most relevant feature to the 'good' feature set. Batch-iFDD experimentally outperforms the previous state-of-the-art Matching Pursuit algorithm. Even though features are constructed in an online manner and these methods overcome an imperfect initial selection of features, it is still crucial to provide good initial features and designing them intelligently.

2.2.7 Actor, Critic, and Actor-critic Algorithms

Critic-only methods

Critic-only algorithms rely on value function approximation. They aim to learn an approximate solution to the Bellman equation such that it will return a near-optimal policy. Critic-only algorithms are indirect as they do not try to optimise directly over a policy space. Even after constructing a good approximation of the value function, these algorithms lack reliable guarantees in terms of near-optimality of the final policy. For example, SARSA [Sutton and Barto, 1998] and Q-learning [Watkins and Dayan, 1992] are critic-only algorithms.

Actor-only methods

Actor-only algorithms operate on a parametrised family of policies. These algorithms directly estimate the gradient of the performance with respect to the actor parameters by simulation. In turn, the parameters of the policies are updated in a direction of improvement. The major bottleneck of such algorithms is large variance of the gradient estimators. Furthermore, in the original actor-only methods, a new gradient of policy is estimated independently of past estimates. Hence, there is no learning in form of accumulation and consolidation of older information. Though policy gradient assures optimality almost always due to the properties of gradient descent, it has slow convergence for large enough policy space and also the learning rate is hard to choose. For example, REINFORCE [Williams, 1992] is an actor-only algorithm. Now, with the advent of deep learning algorithms in reinforcement learning, policy gradient methods got back the popularity. [Montgomery and Levine, 2016], and [Schulman et al., 2015] proposed different variants of policy gradient methods for deep reinforcement learning applications.

Actor-critic methods

Actor-critic algorithms combine the strengths of actor-only and critic-only algorithms. The critic uses an approximation scheme and a simulation to learn a value function. The actor uses this approximation to update the policy parameters towards performance improvement. Such algorithm, as long as they are gradient-based, show desirable convergence properties, unlike critic-only algorithms that guarantees convergence in limited settings. These algorithms deliver faster convergence, due to variance reduction, in comparison with the actor-only algorithms. [Grondman et al., 2012b] provides a survey of the actor-critic algorithms. We provide a unified view of these algorithms in Table 2.4.

Algorithm	Policy gradient
REINFORCE	$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \quad V_t]$
Q actor-critic	$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \quad Q^\pi(s,a)]$
Advantage actor-critic	$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \quad A^\pi(s,a)]$
TD actor-critic	$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \quad \delta]$
TD(λ) actor-critic	$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \quad \delta e]$
Natural actor-critic	$G_\theta \Phi$

Table 2.4: Policy gradients for different Actor-critic algorithms.

2.2.8 Exploration in MDPs

In order to maximise the value function, the agent should choose the action with highest reward (exploitation), but to discover such action she has to play and learn the actions not selected before (exploration). Exploration enables to learn about the reward generation of new actions. It may also increase the greater total reward in the long run because we would discover better actions. In the optimisation sense, it does not let the algorithms to be trapped in the local minima by myopically choosing an action performing better according to the present knowledge. The trade-off between exploitation and exploration is one of the challenges in reinforcement learning [Sutton and Barto, 1998]. We have discussed that in multi-armed bandits. Here, we present some well-known exploration methods for MDPs that we would use in Chapter 3 and 4. We also discuss the present research works to resolve this issue in MDPs and extend this well-discussed phenomenon from bandits to MDPs. This would later provide us the scope to extend the work in BelMan to MDPs and its implications.

Learning in MDPs

There are some research works that follows the investigation of regret upper bounds in stochastic bandits to provide the basic limits of learn-ability in MDPs. Specifically, if the MDP is unknown, the problem of finding an optimal policy is no longer

just an optimization problem. The regret is introduced to measure the price of tackling uncertainty. The regret of a policy π is often defined as the deficit of rewards with respect to the expected average reward of an optimal policy. Mathematically, $R(\pi, T) = V^* - \sum_{t=1}^{T} \mathcal{R}(a_t, s_t)$. For a fully connected MDP $\mathcal{M}$ with diameter D and bounded reward, the expected reward is upper bounded by $\mathbb{E}[R(\pi, T)] \leq 1 + C\ D\ |S| \sqrt{2\,|A|\,T \log T}$. For a similar setup of MDP with sufficiently large diameter and sufficiently large horizon to cover the state-action space, the expected reward is lower bounded by $\mathbb{E}[R(\pi, T)] \geq C\sqrt{D\,|S|\,|A|\,T}$. The upper and lower bounds are separated by a factor of at least $\sqrt{D\,|S| \log T}$ that allows a large window of variance. There are recent works to focus on the exploration–exploitation aspect of MDPs that leads to better theoretical bounds on regret and to more efficient algorithms.

Exploration strategies in MDP

Now, we illustrate three well-known exploration strategies in MDPs, such as ϵ-greedy, Boltzmann, and R_{max} exploration.

ϵ-greedy exploration [Sutton and Barto, 1998]. The ϵ-greedy strategy for MDPs works as same as that of the bandits. With probability $1 - \epsilon$, the agent chooses an action with the highest estimated value, but with small probability $\epsilon > 0$ she chooses an action uniformly at random. The drawback of ϵ-greedy is to choose equally among all actions. This may cause a suboptimal exploration and increase the number of samples required to learn enough about the comparatively better actions. If $\epsilon = 1$, it becomes a random exploration strategy. If $\epsilon = 0$, it becomes a purely greedy strategy.

Boltzmann exploration [Sutton and Barto, 1998]. An alternative strategy is Boltzmann exploration. It is also called the softmax exploration. This is a popular strategy among researchers working on deep reinforcement learning. Boltzmann exploration gives exponentially more weight to the actions having higher value estimate.

An action a at state s is chosen according to the probability

$$p(a \mid s) \triangleq \frac{\exp\left[Q(s,a)/\tau\right]}{\sum_{a \in A} \exp\left[Q(s,a)/\tau\right]}.$$

The parameter $\tau > 0$ is called the temperature. The temperature controls the degree of exploration. As $\tau \to \infty$, the actions are selected uniformly at random. As $\tau \to 0$, the actions are chosen in a completely greedy manner.

Both of these strategies are undirected strategies of exploration [Thrun, 1992]. These strategies do not use any information from the environment to make an informed exploratory action. They predominantly rely on randomness to do exploration. It is hard to theoretically determine the exact values of ϵ and τ to perform these methods optimally. [Singh et al., 2000] tried to propose a theoretical guideline to set the exploration in MDPs. This guideline is called *Greedy in the Limit with Infinite Exploration* that suggests every stage should be explored infinitely often while the algorithm should be asymptotically greedy. This guides us to set an ϵ or τ which is inversely proportional with time. This still does not guarantee any well-behaving property such as unique or global solution to these strategies [Littman, 1996]. Thus, researchers proposed several variants of the Boltzmann exploration and ϵ-greedy, such as mellowmax [Asadi and Littman, 2016], Boltzmann with montone learning rate [Cesa-Bianchi et al., 2017] and so on. Though these two strategies are naïve approaches towards the exploration–exploitation problem in MDPs, but we still use it in large MDPs because of its low resource requirements compared to the sophisticated alternatives.

R_{max} exploration [Brafman and Tennenholtz, 2002]. It is an optimistic value initialization strategy. Initially, the value of each state-action is assigned as the maximal discounted value function. This forces the agent to chooses uniformly at random the initial action to perform, since there is no decision better than another. The transitions provided by the environment are used to update a model. When a state-action pair has been observed exactly m times, the current model is used to update the value

function through value iteration. Here, $m > 0$ is a predefined parameter. Till then, observing the same state-action pair again does not modify the model nor the value function. Thus, exploration does not stop until each reachable state-action pair is chosen m times. Then, the exploration stops and the agent starts to exploit the knowledge about the explored state-action pairs. Since these methods use information about the environment, they are called directed strategies of exploration [Thrun, 1992].

2.2.9 Balancing Exploration and Exploitation in MDPs

Instead of constructing two different on-policy and off-policy in an algorithm, researchers began to investigate design of a single algorithm that can unify the optimal policy finding and exploration–exploitation dilemma. [Schmidhuber, 1991a] quoted, "The same complex mechanism which is used for 'normal' goal-directed learning is used for implementing curiosity and boredom. There is no need for devising a separate system which aims at improving the world model."

Intrinsically motivated Reinforcement Learning

The first inspiration came from the side of animal learning and psychology [Berlyne, 1966]. This lead to classification of reward obtained by an agent in extrinsic and intrinsic rewards. Extrinsic reward is the reward obtained from the decision process. The reward function discussed till now constitute this category. While the intrinsic reward is something inherent to the agent such as curiosity, exploratory joy, surprise and so on. The idea is that the agent is motivated by this internal reward, and in the process of maximizing this reward, they learn a collection of skills. This idea was transferred to the domain of reinforcement learning by Schmidhuber in [Schmidhuber, 1991b]. [Schmidhuber, 1991b] proposed different kinds of internal reward signals like adaptive curiosity and adaptive confidence. [Chentanez et al., 2005] approached this problem from a hierarchical learning perspective. They use both external and internal rewards to learn a behaviour policy. The extrinsic reward is simultaneously used to learn multiple temporally extended behaviours. The intrinsic reward is attempting

to provide a novelty bonus. The paper assumes that there are certain events in the world that are salient and which the agent will be motivated to seek. [Oudeyer et al., 2007; Konidaris and Barto, 2006] implemented these ideas in robotics problems. proposed a novel approach to creativity and exploration using tools of information theory. This family of works refer to this approach as intrinsically motivated reinforcement learning [Oudeyer and Kaplan, 2009].

Curiosity-driven Reinforcement Learning

We can simultaneously look into this problem from a learning theoretic point of view. If an MDP is not completely known, learning is required and optimisation is not sufficient to find out the optimal policy. If too much exploration is conducted while choosing the actions, too much information about suboptimal actions will be accumulated. This would lead to higher sample complexity and a suboptimal policy. If too less exploration is conducted due to extra attention to optimisation, enough information would not be accumulated. This would lead to overfitting, and thus, to a suboptimal policy. In learning theory, this problem is tackled by adding a regularisation term with the loss function of the original problem. These regularisation terms are commonly exploration bonuses for states with higher potential of learning [Şimşek and Barto, 2006; Baranes and Oudeyer, 2009; Bellemare et al., 2016] or with higher uncertainty [Peters et al., 2010; Filippi et al., 2010; Still and Precup, 2012; Lopes et al., 2012; Neu et al., 2017]. This term is often called curiosity and this series of works are called curiosity-driven reinforcement learning.

Information-regularised MDPs. In an approach to the regularisation term, researchers constructed terms inspired by information theory. Variants of entropy are used as the regularisation terms for the classic Bellman equations in order to induce safe exploration [Fox et al., 2015], and to formulate risk-sensitive policies [Howard and Matheson, 1972; Ruszczyński, 2010], or to model behaviours of imperfect agents [Ziebart et al., 2008; Ziebart, 2010]. [Still and Precup, 2012] proposed a Q-learning algorithm

where a learner should chose a policy that maximises its predictive power that is the information about the future carried by the most recent state-action pair. Here, we get a regularisation term similar to the KL-divergence between the state-action distribution induced by updated policy and the empirical state distribution constructed from the past data. [Peters et al., 2010] proposed a policy search algorithm in which new policies are penalized by the relative entropy between the new state distribution and the empirical state distribution constructed from the past data. [Montgomery and Levine, 2016; Schulman et al., 2015; O'Donoghue et al., 2016] proposed other variants of policy search algorithms with entropy-regularised value functions that lead to safe online exploration in an unknown Markov decision processes. The state-of-the-art deep reinforcement learning algorithms, such as [Mnih et al., 2016; Silver et al., 2016], also use a form of such entropy-regularised value functions to balance the exploration–exploitation trade-off. [Neu et al., 2017] unified these methods using the Lagrangian duality of entropy-regularised Bellman's equation, and proved them to be equivalent to online convex optimization methods [Hazan et al., 2016]. [Neu et al., 2017] also provided methods to extend these regularised methods to infinite-horizon MDPs while keeping the stability and optimality criteria intact.

Optimism in the Face of Uncertainty. Optimism in the face of uncertainty (OFU) covers a family of algorithms as we have already described for multi-armed bandits. They work on the philosophy that the Q-value of a state-action pair would not only be its estimated Q-value which is computed from the previous experience but also another time dependent function that leads to optimistic choices. We can think of this extra factor as a function or uncertainty involved with a state-action pair. If a state-action pair has not been visited sufficiently for the agent to be familiar with it, the uncertainty factor assigns a high value to it such that it is considered good and the agent will be driven towards it. This ensures that the agent will explore new options in the state-action space. As time goes on and the state-action pairs are explored enough, the uncertainty on the current Q-value estimation decreases and the Q-value reflects

an informed approximation to the actual value. The first algorithm for MDP using this idea, Explicit Explore and Exploit (E^3), was developed by Kearns and Singh [Kearns and Singh, 2002]. Later, it was extended by [Brafman and Tennenholtz, 2002] to design the R_{max} algorithm that we have already discussed. These set of methods are sometimes called optimistic initial value methods [Szita and Lőrincz, 2008].

The other set of OFU methods are directly inspired by the upper confidence bound algorithms (UCB) in bandits. [Auer and Ortner, 2007] provided an extension of UCB called UCRL for MDPs that incur logarithmic growth in regret with respect to time for finite-horizon ergodic MDPs. [Filippi et al., 2010] improved this to propose KL-UCRL algorithm that extends the KL-UCB algorithm to finite-horizon MDPs. KL-UCRL also achieves logarithmic regret for finite-horizon ergodic MDPs. [Lattimore and Hutter, 2014] extended the UCRL approach for infinite-horizon MDPs and proposed UCRL-γ algorithm that achieves nearly-optimal PAC bounds of regret.

Another family of OFU methods are called the model-based interval estimation methods. Initially, all confidence intervals are wide. They shrink gradually towards the true Q- values. IEQL+ [Meuleau and Bourgine, 1999] algorithm directly estimates confidence intervals of Q -values. [Wiering and Schmidhuber, 1998] compute confidence intervals for the transition and reward functions to obtain the Q-value bounds indirectly. [Strehl et al., 2009] improved this method and prove a polynomial-time PAC convergence bound.

The problem with OFU methods is when the size of state-action pair gets too large, its hard to store all the information for successful exploration. Furthermore, this way of exploration seeks to explore the entire space of state-action pair. This might be computationally expensive, useless, and highly dependent on the initial estimate. Osband and Van Roy illustrated in [Osband and Van Roy, 2017] that "In principal, there should be optimistic approaches that fare well relative to randomized approaches, but that would require intractable computation. Optimistic approaches that have been proposed in the literature sacrifice statistical efficiency for the sake

of computational efficiency. Randomized approaches, on the other hand, may enable simultaneous statistical and computational efficiency."

Bayesian Exploration Methods. The other set of methods are Bayesian exploration methods. Here, the MDP is assumed to be drawn form a parametrised prior distribution $\mathcal{B}_0$. The distributions are further updated to $\mathcal{B}_t$ using the prior distribution $\mathcal{B}_0$ and the collected history. [Strens, 2000] proposed a framework to compute the policy that minimises the uncertainty of the parameters of the distributions. The drawback is the exact computation of the optimal policy is infeasible under this Bayesian formalism and the optimal policies are computationally demanding even under simplifying assumptions. [Bellemare et al., 2017] recently proposed an alternative formulation of this distributional approach that seem to be compatible with the present deep reinforcement learning and experimentally proved to generate optimal algorithms for different problems [Dabney et al., 2017]. [Osband et al., 2013] propose a posterior sampling algorithm for MDPs which is an extension of the information sampling algorithm for bandits, while [Gopalan and Mannor, 2015] extended Thompson sampling to solve the MDP problems. In [Osband and Van Roy, 2016], the authors illustrate the posterior sampling methods provide better solutions for general MDPs than the OFU methods. Till now, these methods are theoretically analysed and operated on comparatively smaller MDPs where the memoisation is not a central issue for performance. Extension of these methods to large scale MDPs is still an open and active research problem.

Linear MDPs

Interestingly, the linear MDP [Todorov, 2007; Dvijotham and Todorov, 2012] approach came from the optimal control theory, specifically differential dynamic programming. Though the motivation behind the approach is very different, the results obtained from it is very similar to that of the entropy-regularised MDPs [Malek et al., 2014]. Linear MDP addresses the problem of finding closed-form solutions to reinforcement learning

problems, by reformulating the optimization function. The system is assumed to have a 'passive' or 'base' dynamics induced by a default policy. The optimization criterion includes both the value function and a regularisation term. The regularisation term is the KL-divergence between the state distributions induced by the updated policy and the default policy. This term penalises any deviation form this base dynamics. Such a regularisation leads to the Bellman optimality equations characterizing the optimal policy that take the form of a system of linear equations. Linear MDPs generalise the existing duality between optimal control and Bayesian inference [Todorov, 2008]. The belief propagation algorithm used in dynamic probabilistic graphical models is equivalent to the power iteration method in linear MDPs [Kappen et al., 2012]. This duality shows that the continuous formulation of linear MDP in both space and time lead to path integral control proposed by [Kappen, 2005]. [Neu and Gómez, 2017] proved its connection with online convex optimisation problem and has shown that the follow-the-leader approach for stochastic linear bandits lead to regret of order $\log^2 T$ for the linear MDP.

This body of research opens up an interesting direction to investigate the connection between linear MDPs, entropy-regularised MDPs, online convex optimisation, and bandit algorithms. In the concluding chapter, we would discuss how our information geometric approach of BelMan indicates towards a path of bridging these lines of work.

Part I

A Functional Approximation Approach to Learning with Unknown Reward and Unknown Transition Function

Chapter 3

Learning with Unknown Reward: Automated Database Tuning

If people are good only because they fear punishment, and hope for reward, then we are a sorry lot indeed.

— *Albert Einstein*[1]

In this chapter, we propose an algorithm design methodology for Markov decision processes without a known reward function, and develop it in the context of the adaptive performance tuning of database applications. The objective is to devise, and to validate a tuning strategy that does not need prior knowledge of a cost model. Instead, the cost model is learned through a reinforcement learning algorithm. We instantiate our approach to the use case of adaptive index tuning. We model the execution of queries and updates as a Markov decision process whose states are database configurations, actions are configuration changes, and rewards are functions of the cost of configuration change and query and update evaluation. While solving the Markov decision process (MDP), we encounter two important challenges, which are 'unavailability of a cost model' and 'size of the state space'. In order to address the former, we sequentially learn the cost model, in a principled manner. In the initial endeavour, we

[1]Quoted in "All the Questions You Ever Wanted to Ask American Atheists" by Madalyn Murray O'Hair, 1982.

design an algorithm, COREIL that adapts a least square estimator and also facilitates the convergence of the MDP solving algorithm. Following that, we improve COREIL using regularisation to avoid overfitting, in order to propose rCOREIL that achieves a logarithmic regret in estimation. In order to address the large size of the state space, we devise strategies to prune the state space, both in the general case and for the use case of index tuning. We empirically and comparatively evaluate our approach on a standard OLTP dataset. We show that the proposed algorithms are competitive with state-of-the-art adaptive index tuning, which is dependent on a cost model.

3.1 Introduction

In a recent SIGMOD blog entry [Lohman, 2014], Guy Lohman asked "Is query optimization a *solved* problem?". He argued that current query optimisers and their cost models can be critically wrong. Instead of relying on wrong cost models, [Stillger et al., 2001] have proposed LEO-DB2, a *learning* optimiser. Enhanced performance of LEO-DB2 with respect to the classical query optimisers strengthens the claim of discrepancies introduced by the predetermined cost models. This is our perspective when we choose automated performance tuning of database applications as an MDP with unknown reward function. By performance tuning, we mean selection of an optimal physical database configuration in view of the workload. In general, configurations differ in the indexes, materialised views, partitions, replicas, and other parameters. While most of the existing tuning systems and literature [Bruno and Chaudhuri, 2007a; Schnaitter et al., 2007; Schnaitter and Polyzotis, 2012] rely on a predefined cost model, the objective of this work is to validate the opportunity for a tuning strategy to do without.

In order to achieve this objective, we propose a formulation of automated database tuning as a *reinforcement learning* problem (see Section 3.3). The execution of queries and updates is modelled as an MDP whose states are database configurations, whose actions are configuration changes, and whose rewards are functions of the cost of

configuration change and query/update evaluation. This formulation does not rely on a pre-existing cost model but learns it.

We present a solution to the MDP formulation that tackles the *curse of dimensionality* (Section 3.4). In order to resolve this issue, we reduce the search space by exploiting the quasi-metric properties of the configuration change cost, and we approximate the cumulative cost with a linear model. We formally prove that assuming such a linear approximation is sound as our approach converges to an optimal policy even after estimating the cost. In Section 3.5, we tackle the problem of overfitting. In order to avoid and the corresponding instability, we add a regularisation term in the objective function while learning the cost model. We formally derive a bound on the total regret of the regularised estimation, that is logarithmic in the time step (i.e., in the size of the workload). We instantiate our approach to the use case of index tuning (Section 3.6), developing in particular optimisations specific to this use case to reduce the search space. Approaches developed in Sections 3.4 and 3.5 provide us with two algorithms COREIL and rCOREIL to solve the index tuning problem.

We use this case to demonstrate the validity of a cost-model oblivious database tuning with reinforcement learning, through experimental evaluation on a TPC-C workload [Raab, 1993] (see Section 3.7). We compare the performance with the *Work Function Index Tuning* (WFIT) algorithm [Schnaitter and Polyzotis, 2012]. Results show that our approach is competitive yet does not need knowledge of a cost model.

Comparison of WFIT with COREIL establishes reinforcement learning as an effective approach to automatise the index tuning problem. Performance evaluation of rCOREIL with respect to COREIL demonstrates that the learning performance is significantly enhanced by a crisp estimation of the cost model. The theoretical and experimental results along with the algorithm design methodology shows us a functional approximation approach to develop algorithm to solve MDPs with unknown reward function.

3.2 Literature Review and Contextualisation

In this section, we introduce a brief review of related works and highlight the contributions of our proposed approach for online automated database configuration. First, we discuss existing work related to automated database configuration. Later, we review how learning algorithms is exploited in the field of data management.

3.2.1 Automated Database Configuration

There is a large body of work in the field of automated physical database design. Table 3.1 provides a brief classification of these related works in terms of various dimensions. The classification considers whether the algorithm is online or offline. We also consider the physical design aspects, such as index selection [Agrawal et al., 2000; Zilio et al., 2004b], vertical partitioning [Lightstone and Bhattacharjee, 2004; Rasin and Zdonik, 2013], or mixed design together with horizontal partitioning and replication [Bruno and Nehme, 2008; Bruno and Chaudhuri, 2008].

Offline Algorithms

Traditionally, automated database configuration is conducted in an offline manner. In this approach, database administrators (DBAs) identify representative workloads from the trace of historical database queries and updates. This task can be done either manually or with the help of sophisticated tools provided by database vendors. Based on these representative workloads, new database configurations are realised. For example, new beneficial indexes are created [Agrawal et al., 2000; Zilio et al., 2004b; LeFevre et al., 2014], smart vertical partitioning is done for reducing I/O costs [Rasin and Zdonik, 2013; Hammer and Niamir, 1979; Lightstone and Bhattacharjee, 2004], or possibly a combination of index selection, partitioning and replication for both standalone databases [Papadomanolakis et al., 2007a; Bruno and Nehme, 2008; Bruno and Chaudhuri, 2008; Zilio et al., 2004a] and parallel databases [Agrawal et al., 2004; Rao et al., 2002; Nehme and Bruno, 2011; Pavlo et al., 2012].

	Index selection	Vertical partitioning	Mixed (index+part.+repl.)
Offline	[Agrawal et al., 2000; Zilio et al., 2004b] [LeFevre et al., 2014]	[Rasin and Zdonik, 2013] [Lightstone and Bhattacharjee, 2004] [Hammer and Niamir, 1979]	*Stand-alone databases*: [Bruno and Nehme, 2008; Bruno and Chaudhuri, 2008] [Papadomanolakis et al., 2007a; Zilio et al., 2004a] *Parallel databases*: [Agrawal et al., 2004; Rao et al., 2002] [Nehme and Bruno, 2011; Pavlo et al., 2012]
Online	[Schnaitter and Polyzotis, 2012; Schnaitter et al., 2007] [Sattler et al., 2003; Luhring et al., 2007] [Bruno and Chaudhuri, 2007a, 2010, 2007b]	[Li and Gruenwald, 2013] [Rösch et al., 2012] [Alagiannis et al., 2014]	**COREIL: [Basu et al., 2015a,b]** **rCOREIL: [Basu et al., 2016]**

Table 3.1: Classifying automated database design literature.

Online Algorithms

The complication and agility of database applications is growing with the introduction of modern database environments such as database as a service. Though the aforementioned manual task of database administrators can be done in an offline fashion, it has become even more tedious, problematic, and prone to errors. Therefore, it is desirable to design automated solutions to database design and tuning problems. Specifically, there is a need of solutions which are able to continuously monitor changes in the workload, and to react in real-time by adapting the database configuration to the new workload. In fact, the problem of online index selection is studied in [Schnaitter and Polyzotis, 2012; Bruno and Chaudhuri, 2007a; Malik et al., 2009; Luhring et al., 2007; Schnaitter et al., 2007; Bruno and Chaudhuri, 2010, 2007b; Sattler et al., 2003; Schnaitter et al., 2006]. Generally, these techniques adopt the same working model in which the system continuously tracks the incoming queries for identifying candidate indexes, profiles the benefit of the indexes, and realises the ones that are most useful for query execution. Specifically, an online approach to database tuning, such as index selection, was proposed in [Bruno and Chaudhuri, 2007a]. The algorithm progressively chooses the optimal plan at each step by using a case-by-case analysis on the potential benefits that we may lose by not implementing relevant candidate indexes. A new database configuration is selected only when a physical change, such as creation or deletion of an index, would be helpful in improving system performance. Similarly, a framework for continuous online physical tuning was proposed in [Schnaitter et al., 2007]. [Schnaitter et al., 2007] creates and deletes effective indexes in response to the shifting workload. This framework is able to self-regulate its performance by providing explicit mechanism for controlling the overhead by profiling the benefit of indexes.

One of the key components of an index selection algorithm is profiling indexes' benefit. These algorithms evaluate the cost of executing a query workload with the new indexes as well as the cost of configuration change like creation and deletion of indexes. In order to realise these functionalities, most of the online algorithms exploit the what-if optimiser [Chaudhuri and Narasayya, 1998] that returns such estimated

costs. For example, the what-if optimiser of DB2 was used in [Schnaitter and Polyzotis, 2012]. The what-if optimiser of SQL Server is leveraged in [Bruno and Chaudhuri, 2007a]. The classical optimiser of PostgreSQL is extended to support what-if analysis in [Schnaitter et al., 2007]. However, it is well-known that invoking the optimiser for estimating the cost of each query is computationally expensive [Papadomanolakis et al., 2007b]. In this work, we propose an algorithm that is free from the what-if optimiser and also able to adaptively provide a better database configuration in the end. We validate these properties in the experimental results.

Recently column-oriented databases have attracted a great deal of attention in both academia and industry. This development has invoked online algorithms for automated vertical partitioning which are critical for this emerging breed of database systems [Rösch et al., 2012; Alagiannis et al., 2014; Li and Gruenwald, 2013]. Specifically, a storage advisor for SAP, called HANA in-memory database system, is proposed in [Rösch et al., 2012]. It takes advantage of both columnar and row-oriented storage layouts. At the core of this storage advisor, there is a cost model that estimates and compares query execution times for different stores. Similarly, a continuous layout adaptation is introduced in [Alagiannis et al., 2014]. [Alagiannis et al., 2014] supports multiple storage layouts in a single engine which is able to adapt to changing data access patterns. This adaptive store monitors the access patterns of incoming queries through a dynamic window of N queries, and devises cost models for evaluating the workload and layout transformation cost. Furthermore, in order to efficiently find a near optimal data layout for a given workload, the hybrid store exploits proper heuristic techniques to prune the immense search space of alternative data layouts. On the contrary, the algorithm introduced in [Li and Gruenwald, 2013] uses data mining techniques for vertical partitioning in database systems. The technique is based on closed item sets mining from a query set and system statistic information at run-time. Hence it is able to automatically detect changing workloads and to perform a re-partitioning action without the need of interaction with DBAs.

Context of Our Contributions

Compared to the above state-of-the-art in online automated database design, our proposed approach is more general and can be applied for various problems such as index selection, horizontal/vertical partitioning design, and a combination with replication as well. As our proposed online algorithm is able to learn the estimated cost of queries sequentially, it does not need the what-if optimiser for estimating query cost. Therefore, our proposed algorithm is applicable to a wider range of database management systems which may not implement what-if-optimiser or expose its interface to the users.

3.2.2 Reinforcement Learning in Data Management

As we describe in Chapter 2, reinforcement learning [Sutton and Barto, 1998] is about determining *the next best thing to do* in order to reach a goal while the knowledge of the environment evolves continuously. This goal is represented as maximisation of the cumulative reward obtained through a sequence of actions. Here each action leads to an individual reward and to a new state, usually in a stochastic manner. We mentioned in Chapter 2 that *Markov decision process* (MDP) [Puterman, 2009] is a model of reinforcement learning problem. In this model each action leads to a new state and a given reward according to a probability distribution that must be learned. This implies an inherent trade-off between exploration, i.e. trying out new actions leading to new states and to potentially high rewards, and exploitation, i.e. performing actions already known to yield high rewards [Audibert et al., 2009]. Despite of these flexibilities under uncertain environments use of MDP in data management applications has been limited so far, for the following reasons,
(i) The state space is large as it represents all possible partial knowledge of the world. This is mentioned as the curse of dimensionality.
(ii) States have complex structures due to the data, or, in our case, the indexes.
[Benedikt et al., 2006] uses MDP for modelling the data cleaning tasks. The authors

discuss about the absence of a straightforward technique to do that because of the large state space. Other common issues in using MDP for database applications are – (i) Rewards can be delayed, i.e. obtained after a long sequence of state transitions. For example, we observe such issues in focused Web crawling [Gouriten et al., 2014]. Still a variant of multi-armed bandits have been successfully applied [Gouriten et al., 2014] in this problem.
(ii) Due to data uncertainty, there may be only *partial observability* of the current state. This leads a partially observable Markov decision process [White, 1993].
Fortunately, last two problems are hardly prevalent in the online tuning problem discussed here. This let us formulate online tuning problem as an infinite-horizon MDP and to emphasise into the problems of exploration-exploitation, and curse of dimensionality.

3.3 Automated Database Tuning as a Learning Problem

In this section, we model the process of executing queries and changing configurations as a Markov decision process. We formalise the problem of adaptive performance tuning as a stochastic optimisation problem. A policy is a selection strategy of the next configuration given a current configuration and query. The problem is to find a policy that maximises the expected performance. For the sake of simplicity, we focus on response time as our performance metric. We define an infinite horizon problem in which the cost of future events is amortised by a discount factor.

Let R be a logical database schema. We can consider R to be the set of its possible database instances. Let S be the set of physical database configurations of the instances of R. For a given database instance, configurations may differ in the indexes, materialised views, partitions, replicas, and other parameters and they are logically equivalent if they yield the same results for all queries and updates.

Problem parameters	
$R \triangleq \{r\}$	A Database schema
r	Database instances of R
s	Physical configuration of r
$S \triangleq \{s\}$	Set of physical database configurations
$\delta : S \times S \to \mathbb{R}^{\geq 0}$	Cost function of changing configuration
$t \in \mathbb{N}$	Time step
q_t	Query at time t
$Q \triangleq [q_t]_{t \geq 0}$	Workload which is an ordered set of queries
T	Number of queries in the workload
Design variables	
$cost : S \times Q \to \mathbb{R}^{\geq 0}$	Cost function for executing a query at a configuration
$C : S \times S \times Q \to \mathbb{R}^{\geq 0}$	Per-stage cost of configuration change and query execution
$\gamma \in (0, 1)$	Discount factor
$\boldsymbol{\phi}(s)$	Feature vector of $V^{\pi}(s)$
$\boldsymbol{\eta}(s, q)$	Feature vector of $cost(s, q)$
n	Dimension of feature vectors
Decision variables	
$\pi : S \times Q \to S$	Policy
$\mathcal{U} \triangleq \{\pi\}$	Policy space
$V^{\pi} : S \to \mathbb{R}^{\geq 0}$	Cost-to-go function

Table 3.2: Notations used in Chapter 3.

The cost of changing configuration from $s \in S$ to $s' \in S$ is denoted by the function $\delta\colon\ S \times S \to \mathbb{R}^{\geq 0}$. The function $\delta(s, s')$ has the following properties:

- Non-negativity: for all $s, s' \in S$, $\delta(s, s') \geq 0$.
- Identity of indiscernible: for all $s, s' \in S$, $\delta(s, s') = 0$ if and only if $s = s'$.
- Triangle inequality: for all $s, s', s'' \in S$, $\delta(s, s'') \leq \delta(s, s') + \delta(s', s'')$.

The first and second property imply that there is no free configuration change. Whereas the triangle inequality means that it is always cheaper to do a direct configuration change than going through an intermediate state. The function δ is not necessarily symmetric. This means for all $s, s' \in S$ the equality $\delta(s, s') = \delta(s', s)$ may not hold because the cost of changing configuration from s to s' and the reverse may not be the same. Therefore, δ is a quasi-metric on S.

Let Q be the workload which is defined as a sequential schedule of queries and updates. For brevity, we simply refer both of them as queries. Query $q_t \in Q$ is the t^{th} query in the schedule which is executed at time t. We model a query q_t as a random variable. The generating distribution is not known *a priori*, and q_t is only observable at time t. The cost of executing query $q \in Q$ on configuration $s \in S$ is denoted by the function $cost\colon\ S \times Q \to \mathbb{R}^{\geq 0}$. Though we are aware of the aforementioned properties of δ and $cost$, the exact functional form of them are not known in this application. For the sake of simplicity and without significant loss of generality, we assume the issue of concurrency control [Bhargava, 1999] is orthogonal to the current presentation.

Let $s_0 \in S$ be the initial configuration of the database. At any time t the configuration changes from s_{t-1} to s_t with the following events in order:

1. Arrival of query q_t. We call $\hat{q}_t$ the observation of q_t[2].
2. Choice of the configuration $s_t \in S$ based on $\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_t \in Q$ and $s_{t-1} \in S$.
3. Change of configuration from s_{t-1} to s_t. If no configuration change occurs at time t, then $s_t = s_{t-1}$.
4. Execution of observed query $\hat{q}_t$ under the configuration s_t.

The cost of configuration change and query execution at time t, referred as the *per-stage cost*, is the sum of the cost of configuration change, and the cost of query execution.

$$C(s_{t-1}, s_t, \hat{q}_t) \triangleq \delta(s_{t-1}, s_t) + cost(s_t, \hat{q}_t). \tag{3.1}$$

We phrase the stochastic decision process of choosing the configuration changes as an MDP [Puterman, 2009], where states are database configurations, actions are configuration changes, and costs (negative rewards) are the per-stage cost of the action. In this case, transitions from one state to another on an action are deterministic. In

[2] q_t is a random variable that can possibly be evaluated as one of the logical statements. The observation $\hat{q}_t$ is an instantiation of the random variable q_t in form of a particular instruction or logical statement. Generally, capital letters are used for random variables in this thesis. q_t is an exception of notation format.

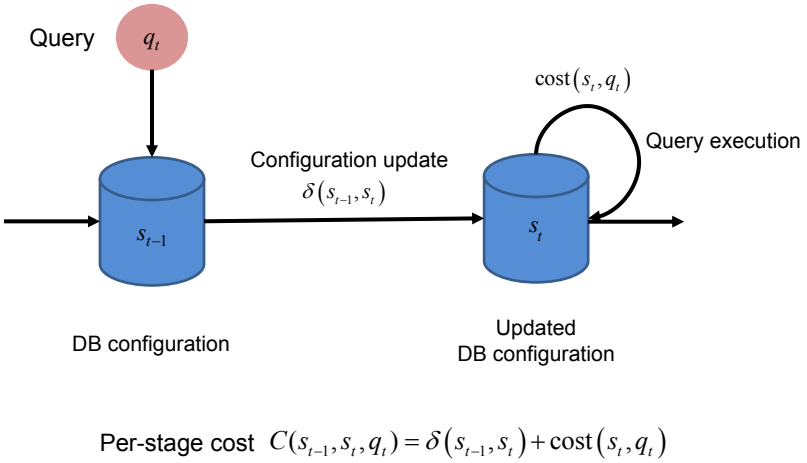


Figure 3.1: The events involved in a step of database transition.

contrast to the general framework of MDPs, update to the new configuration is certain when a configuration change is decided. On the other hand, costs are both *stochastic* and *uncertain*. The stochastic property is caused due to the dependence on the query, which is a random variable. The cost of a query is uncertain as it is is not known in advance, specifically in the absence of a reliable cost model. This transition is shown in Figure 3.1.

Ideally, the problem would be to find the sequence of configurations that minimises the sum of future per-stage costs. Assuming an infinite horizon [Sutton and Barto, 1998] makes the sum of future per-stage costs infinite. One practical way to circumvent this problem is to introduce a *discount factor* γ that gives more importance to immediate costs than to costs distant in the future, and to try and minimise a *cumulative cost* defined with γ. Under Markov assumption, a sequence of configuration changes is determined by a Markovian *policy* $\pi\colon\ S \times Q \to S$, which, given the current configuration s_{t-1} and a query $\hat{q}_t$, returns a configuration $s_t \triangleq \pi(s_{t-1}, \hat{q}_t)$.

Let Π be the set of all feasible Markovian policies. Our objective is to find a policy that minimises the expectation of cumulative sum of per-stage cost.

$$\begin{aligned} \arg\min_{\pi \in \Pi} \quad & \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \left(\delta(s_{t-1}, s_t) + cost(s_t, q_t)\right)\right] \\ \text{such that,} \quad & s_t = \pi(s_{t-1}, q_t), \qquad \forall t \in \mathbb{N}. \end{aligned} \tag{**P1**}$$

We define the discounted utility function of a policy π as the discounted sum of costs obtained by following it. In this chapter, we call it the cost-to-go function to differentiate it from the classical value function that deals with rewards than the costs (Section 2.2). The *cost-to-go* function V^{π} of following policy π is defined as

$$\begin{aligned} V^{\pi}(s) \triangleq & \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \left(\delta(s_{t-1}, s_t) + cost(s_t, q_t)\right)\right] \\ \text{such that,} \quad & \pi \in \Pi, s_0 = s, \\ & s_t = \pi(s_{t-1}, cost(q_t)), \qquad \forall t \in \mathbb{N}. \end{aligned}$$

The value of $V^{\pi}(s)$ represents the expected cost following policy π from current time to the future with the current configuration being s. The problem (**P1**) is equivalent to:

$$\begin{aligned} & \textit{argmin} \;\; V^{\pi}(s_0) \\ & \textit{such that,} \;\; \pi \in \mathcal{U}. \end{aligned}$$

Given a cost-to-go function V, an policy π can be recovered by solving the following Bellman equation at each step t.

$$\pi_t = \arg\min_{s \in S} \left[\delta(s_t, s) + cost(s, q_t) + \gamma V(s)\right]. \tag{3.2}$$

Let π^* be the optimal policy of **P1** and V^* be the cost-to-go function following π^*. This implies the corresponding Bellman's equation:

$$V^*(s) = \mathbb{E}\left[\min_{s' \in S} \left(\delta(s, s') + cost(s', q) + \gamma V^*(s')\right)\right]. \tag{3.3}$$

This indicates that $V^*(s)$ is an expectation of the optimised cost summation of changing configuration, query execution under the new configuration, and the discounted optimal cost-to-go value for the new configuration.

3.4 Automated Database Tuning with Cost-Model Learning

3.4.1 Algorithmic Framework

In the MDP framework, we generally provide the algorithm a cost function that indicates whether the learning agent is doing well or poorly. It will be the learning algorithm's job to figure out how to choose sequence of configurations over time that minimises the cost. We begin the design of our algorithm with the policy iteration algorithm as described in Section 2.2. For Sections 3.4.1, 3.4.2 and 3.4.3, we assume the cost functions are known or at least accessible for evaluation. In Section 3.4.4, we introduce the scheme of cost model learning on top of the algorithmic framework developed till Section 3.4.3.

The *policy iteration* framework iteratively tries and updates the policy and its estimated value function using Bellman Equations (3.2) and (3.3). Let $\bar{V}$ be an estimation of cost-to-go function. If the probability distribution of q_t was known in advance, the policy iteration algorithm would operate as per Algorithm 10. We begin with an arbitrary policy and iteratively improve it using the current estimate of cost–to–go function. The algorithm terminates when there is no change in the policy. The proof of optimality and convergence of Algorithm 10 is described in [Powell, 2007]. We also

Algorithm 10 Policy iteration algorithm [Sutton and Barto, 1998]

1: Initialisation an arbitrary policy π^0.
2: Set $t = 1$.
3: **for all** $s \in S$ **do**
4:

$$\bar{V}^t(s) = \min_{s \in S} \left(\delta(s_t, s) + \mathbb{E}\left[cost(s, q_t)\right] + \gamma \bar{V}^{t-1}(s) \right) \tag{3.4}$$

5: **end for**
6: $\pi^t \leftarrow$ applying (3.2) on $\bar{V}^t$.
7: **if** $\pi^t = \pi^{t-1}$ **then**
8: **return** π^t
9: **else**
10: $t \leftarrow t + 1$.
11: Go to line 3.
12: **end if**

discuss the implications of this algorithm and its relation with Bellman operators in Section 2.2.

Unfortunately, Algorithm 10 suffers from the curse of dimensionality [Powell, 2007]. First, solving (3.4) for every $s \in S$ (line 3 of Algorithm 10) is intractable when the set S is combinatorial (discussed in Section 3.6.1). Therefore it might be impractical to solve (3.4) for every $s \in S$. Second, if the size of Q is large, it would become impractical to compute the expectation of the query execution $\mathbb{E}\left[cost(s', q)\right]$ in (3.4). Moreover, the probability distribution of queries is not known a priori, which makes the expectation not explicitly computable. Third, while solving (3.4), we need to enumerate the cost-to-go function for all possible $s \in S$. This leads to an intractable search space for solutions.

The basic framework of our algorithm is shown in Algorithm 11. In line 5 of Algorithm 11, we overcome the issues of the curse of dimensionality by amalgamating the original problem and approximating the cost-to-go function. We map a configuration to a vector of associated features. This is called a feature mapping. It has to be designed succinctly and in accordance with the application. Let $\boldsymbol{\phi}$ represent a feature mapping and V be the cost-to-go function. We design a linear projection with the

Algorithm 11 Algorithmic Framework

1: Initialisation an arbitrary policy π.
2: **for** **do**$t \geq 1$
3: Observe the query $\hat{q}_t$ and the current configuration s_{t-1} .
4: Estimate δ and *cost* from the previous observations as δ_t and $cost_t$.
5: $\hat{V}_t \leftarrow \min_{s \in S} (\delta_t(s_{t-1}, s) + cost_t(s, \hat{q}_t) + \gamma V_{t-1}(s))$
6: $\bar{V}_t \leftarrow$ using $\hat{V}_t$ and feature mapping.
7: $\pi_t \leftarrow$ applying (3.2) on $\bar{V}_t$.
8: **end for**

feature mapping such that $V(s) \approx \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for some vector $\boldsymbol{\theta}$. We update $\boldsymbol{\theta}$ and π iteratively and make decisions based on the approximation. At time t, the state s_t minimises the right-hand side of line 5 of Algorithm 11. We update the parameter $\boldsymbol{\theta}$, and hence, the approximation $\bar{V}$ by using the observation $\hat{V}_t$. By using the feature mapping, when the value of $\bar{V}$ for one configuration is updated, the values of $\bar{V}$ for the other configurations are also updated automatically. This solves the first curse of dimensionality. Also by using this method, computing the expectation of query execution becomes unnecessary and thus we can get rid of the second curse of dimensionality.

In line 5 of Algorithm 11, the size of the searching set needs to be reduced. This will be discussed in Section 3.4.2. The function δ and the function *cost* need to be evaluated. We assume the availability of a cost model, to the degree of sophistication. Since it is unknown to the algorithm, it learns the cost function through sequential estimation. Quality of learning will depend on the cost model and accuracy.

3.4.2 Reducing the Search Space

In order to reduce the size of the search space in (5), we filter the configurations that satisfy certain necessary conditions which is derived from an optimal policy.

Lemma 2. *Let $s \in S$ be a database configuration and $\hat{q} \in Q$ be any observed query. Let $\pi^* \in \Pi$ be an optimal policy. If s' is the state reached from state s through the*

optimal policy i.e. $\pi^*(s, \hat{q}) = s'$*, then*

$$cost(s, \hat{q}) - cost(s', \hat{q}) \geq 0.$$

Furthermore, if the cost of a configuration update $\delta(s, s') > 0$ *,*

$$cost(s, \hat{q}) - cost(s', \hat{q}) > 0.$$

Proof. Since $\pi^*(s, \hat{q}) = s'$, we have

$$\begin{aligned}\delta(s, s') + cost(s', \hat{q}) + \gamma V(s') &\leq cost(s, \hat{q}) + \gamma V(s) \\ &= cost(s, \hat{q}) + \gamma \mathbb{E}\left[\min_{s''} \left(\delta(s, s'') + cost(s'', \hat{q}) + \gamma V(s'')\right)\right] \\ &\leq cost(s, \hat{q}) + \gamma \delta(s, s') + \gamma V(s').\end{aligned}$$

The first inequality holds as $\delta(s, s') > 0$ for all $s \neq s' \in S$. The second inequality is derived from the triangle inequality $\delta(s, s'') \leq \delta(s, s') + \delta(s', s'')$ for all $s, s', s'' \in S$. This infers that $cost(s, \hat{q}) - cost(s', \hat{q}) \geq (1 - \gamma)\delta(s, s') \geq 0$. The assertion follows. □

By Lemma 2, if π^* is an optimal policy and $s' = \pi^*(s, \hat{q}) \neq s$, then $cost(s, \hat{q}) > cost(s', \hat{q})$. This implies that if a policy is optimal, it would update to such a database configuration where the cost of executing the query is less than or equal to the cost of executing it at the present configuration. Thus, while we are evaluating the value function of a policy, it is not necessary to evaluate it for the configurations that have higher query execution cost than the present configuration. This allows us to define a narrower state space $S_{s,\hat{q}}$ to search for such that

$$S_{s,\hat{q}} \triangleq \{s' \in S \mid cost(s, \hat{q}) > cost(s', \hat{q})\}.$$

Hence at time t, it is sufficient solve the modified Bellman equation

$$\hat{V}_t = \min_{s \in S_{s_{t-1}, \hat{q}_t}} \left(\delta(s_{t-1}) + cost(s, \hat{q}_t) + \gamma \bar{V}_{t-1}(s)\right) \tag{3.5}$$

instead of (5). This smaller search space reduces the computational cost per-step which is important to solve the real-time problem. We devise an algorithm that converges to an optimal policy while searching in the reduced set $S_{s_{t-1},\hat{q}_t}$ for all t.

3.4.3 Reducing the Dimensionality in Policy Iteration

Narrowing down the search space is not sufficient to design an online MDP solver that works fast enough for a real-life database tuning scenario. Thus, we follow the sequence of tricks described in the Section 2.2. We take an actor-critic approach to solve the problem. As we discussed in Section 2.2.7, the actor-only and critic-only methods have their own pros and cons. Critic-only methods approximates the value function and tries to solve an approximate Bellman equation using this. This method is helpful to compute the value function fast. Specifically, the TD algorithms provide an approximate dynamic programming framework to perform this efficiently. Since we are facing the large state space in this problem, TD algorithm alone does not suffice. Thus, we use functional approximation technique of feature mapping and solving Least Square Temporal Difference (LSTD) [Bradtke and Barto, 1996] algorithm using such a map to evaluate the value functions (Section 2.2.6). Actor-only methods use the policy iteration technique with a set of parametrised policies. As we follow the policy iteration methodology in our framework, as described in Algorithm 11, this functionality of actor-only methods is an integral part of our algorithms. Thus, we use the actor-only style to update the evaluated policy at each step. In order to tackle the curse of dimensionality, we also use the functional approximation technique of Least Square Policy Iteration (LSPI)[Lagoudakis and Parr, 2003a] here. Merging these two methods in a single actor-critic algorithm provides us the strengths of both paradigms which are stability and fast convergence respectively. Additionally, we develop the online version of this actor-critic method such that it can work in a sequential manner for the real-time application.

Hence, our algorithm evaluates the cost-to-go function of a policy by projecting it into a feature space and approximating it in that space as well as possible. Following

that, the cost-to-go function is sent to evaluate the policy and its next possible update using the Bellman equation that searches in the restricted state space $S_{s_{t-1},\hat{q}_t}$ at time t. This step provides us a policy update. Till now, we are assuming the cost function is known and it is evaluated accurately at each step. We would add the extra layer of computation for unknown reward in the next section.

Cost-to-go function evaluation. In order to perform the functional approximation in a lower dimensional subspace, we project the value function for a given policy and observed for a present state in a feature space. Though a feature space has to maintain some properties, such as linearity and existence of an orthogonal basis, construction of the feature space is dependent on the application. We would discuss about the design of such a feature space in Section 3.6 in case of index tuning. For now, let us assume that we have such a feature space with basis functions $\boldsymbol{\phi}(s) \in \mathbb{R}^d$ for all $s \in S$ and a dimension $d \in (0, 2^{|S|})$. This feature space allows us to approximate the value function for a given policy as $V(s) \cong \langle \boldsymbol{\theta}, \boldsymbol{\phi}(s) \rangle$ for all $s \in S$. We use the notation $\boldsymbol{\phi^t}$ to represent the vector $\boldsymbol{\phi}(s_t)$ evaluated at configuration s_t. Since the exact projection is not known and the projections $\boldsymbol{\phi^t}$ are not known till the states are explored, the projection specifically the parameter vector $\boldsymbol{\theta}$ has to be computed depending on the samples observed.

An online learning model needs to be build and updated iteratively. The observations to evaluate the cost-to-go function V come from a sequential process. Hence, the effect of a recent action on the observed cost-to-go function would be more than that of an action in distant past. It is intuitive to assign higher weights on the recent observations than the old ones but still to incrementally build a model of value function from the whole experience [Powell, 2007]. We use LSTD [Bradtke and Barto, 1996] as shown in Algorithm 12 to update $\boldsymbol{\theta}$. LSTD is very efficient for extracting information from the training experiences. LSTD is based on the theory of linear least-squares function approximation. It shows that the parameter vector $\boldsymbol{\theta}$ at time t can be computed as $\boldsymbol{\theta}^t = B^{t-1}\boldsymbol{\xi}^t$. Here, the projected vector $\boldsymbol{\xi}^t \in \mathbb{R}^d$ is defined as

a weighted sum of the feature basis vectors where the weights are proportional to the rewards collected from the corresponding states. Strictly speaking, $\boldsymbol{\xi}^t \triangleq \frac{1}{t}\sum_{i=1}^{t} C^t \boldsymbol{\phi^t}$. The projection matrix $B^{t-1} \in \mathbb{R}^{d\times d}$ captures the effect of the cumulative difference of auto-covariance of visited states, and the covariance of the visited state with the next future step. The projection matrix is defined as $B^{t-1} = \left(\frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\phi^t}\ \left(\boldsymbol{\phi^t} - \gamma\boldsymbol{\phi^{t+1}}\right)\right)^{-1}$. In Algorithm 12, we present an online version of this approximation algorithm that uses Sherman-Morrison formula [Hager, 1989][3] to perform the inversion and the update on-the-go. In Algorithm 12, $\hat{C}^t$ is the one-step contribution function given as $\delta(s_{t-1}, s_t) + cost(s_t, \hat{q}_t)$. We can show that the time complexity of this step is $\mathrm{O}\big(d^2\big)$, or $\mathrm{O}\big(m\big)$ to be specific where m is the number of non-zero entries in B^{t-1}. This gain in computational speed can be achieved by a proper design of the feature mapping.

Algorithm 12 Least squares temporal differencing.

1: **procedure** LSTD($\hat{C}^t, B^{t-1}, \boldsymbol{\theta}^{t-1}, \boldsymbol{\phi}^{t-1}, \boldsymbol{\phi^t}$)
2: $\quad\epsilon^t \leftarrow \hat{C}^t - (\boldsymbol{\phi}^{t-1} - \gamma\boldsymbol{\phi}^t)^T\boldsymbol{\theta}^{t-1}$
3: $\quad B^t \leftarrow B^{t-1} - \frac{B^{t-1}\boldsymbol{\phi}^{t-1}(\boldsymbol{\phi}^{t-1}-\gamma\boldsymbol{\phi}^t)^T B^{t-1}}{1+(\boldsymbol{\phi}^{t-1}-\gamma\boldsymbol{\phi}^t)^T B^{t-1}\boldsymbol{\phi}^{t-1}}$
4: $\quad\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} + \frac{\epsilon^t B^{t-1}\boldsymbol{\phi}^{t-1}}{1+(\boldsymbol{\phi}^{t-1}-\gamma\boldsymbol{\phi}^t)^T B^{t-1}\boldsymbol{\phi}^{t-1}}$
5: $\quad$**return** $B^t, \boldsymbol{\theta}^t$.
6: **end procedure**

Updating the policy. As we introduce the online LSTD algorithm with feature mapping $\boldsymbol{\phi^t}$ for value function evaluation, now we solve **P1** by using the LSPI algorithm [Lagoudakis and Parr, 2003a]. LSPI has the convergence guarantee if for any policy $\pi \in \Pi$, there exists a vector $\boldsymbol{\theta}$ such that $V^\pi(s) = \boldsymbol{\theta}^T\boldsymbol{\phi}(s)$ for any configuration s. Algorithm 13 shows a modified online LSPI algorithm for our infinite horizon MDP problem **P1**. The main difference between Algorithm 13 and the original LSPI algo-

[3]Sherman-Morrison formula states that the new matrix obtained after one-rank update of an invertible matrix would also be invertible under some technical condition. Mathematically, if A is a square invertible matrix and u, v are column vectors, the matrix $A + uv^T$ is invertible if and only if $1 + v^T A^{-1}u \neq 0$ and is given by

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1}u}$$

.

rithm lies in (3.6), and the online update used to speed-up. In (3.6), we search in a reduced set $S_{s_{t-1},\hat{q}_t}$ instead of searching all the possible configurations in S.

Algorithm 13 Modified least squares policy iteration.

1: Initialise the configuration s_0.
2: Initialise $\boldsymbol{\theta}^0 = \boldsymbol{\theta} = \mathbf{0}$ and $B^0 = \epsilon I$.
3: **for** t=1,2,3,... **do**
4: Observe the query $\hat{q}_t$ at time t.
5:

$$s_t \leftarrow \underset{s \in S_{s_{t-1},\hat{q}_t}}{\arg\min} \left(\delta(s_{t-1}, s) + cost(s, \hat{q}_t) + \gamma \boldsymbol{\theta}^T \boldsymbol{\phi}(s)\right) \tag{3.6}$$

6: Change the configuration to s_t.
7: Execute query $\hat{q}_t$.
8: Evaluate the cost of the present actions $\hat{C}^t \leftarrow \delta(s_{t-1}, s_t) + cost(s_t, \hat{q}_t)$.
9: Update the value function approximation

$$(B^t, \boldsymbol{\theta}^t) \leftarrow LSTD(\hat{C}^t, B^{t-1}, \boldsymbol{\theta}^{t-1}, \boldsymbol{\phi}(s_{t-1}), \boldsymbol{\phi}(s_t))$$

10: **if** $(\boldsymbol{\theta}^t)$ converges **then**
11: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^t$.
12: **end if**
13: **end for**

In Algorithm 13, the vector $\boldsymbol{\theta}$ determines the current policy. The decision at time t is made by solving (3.6). The values of $\delta(s_{t-1}, s)$ and $cost(s, \hat{q}_t)$ are obtained from the cost model. The vector $\boldsymbol{\theta}^t$ is used to approximate the cost-to-go function following the current policy. If $\boldsymbol{\theta}^t$ converges, we update the current policy (Lines 10-12). Figure 3.2 illustrates the workflow of Algorithm 13, where the model parameter refers to $\boldsymbol{\theta}$.

Now, we prove Theorem 1 that guarantees the convergence of the modified online LSPI algorithm (Algorithm 13).

Proposition 1. *If for any Markovian policy $\pi \in \Pi$ and feature mapping $\phi(s) \in \mathbb{R}^d$ for all $s \in S$, there exists a vector $\boldsymbol{\theta}$ such that $V^\pi(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for any configuration s, then Algorithm 13 will converge to an optimal policy $\pi^* \in \Pi$.*

Proof. Let us define $\mathcal{V}$ to be the set of all feasible cost-to-go functions $V^\pi(s)$ for any $s \in S$. The cost-to-go-functions are bounded for bounded cost functions. Specifically,

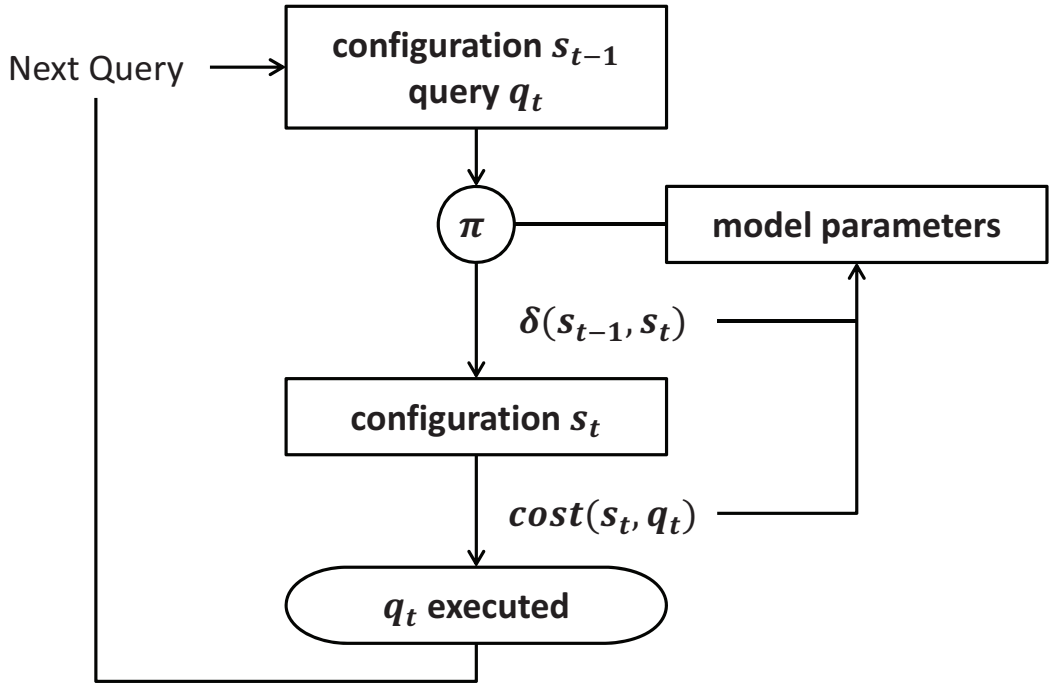


Figure 3.2: Workflow of automated database tuning with online actor-critic algorithm.

if the upper bound on the per-step cost is $\sup_{s,q,s'} C(s,q,s') = C_{max} \in \mathbb{R}$, then the upper bound of the cost-to-go function would be $\frac{C_{max}}{1-\gamma}$. Thus, $\mathcal{V}$: $S \to \mathbb{R}$ is a set of bounded, real-valued functions. This structure makes $\mathcal{V}$ a Banach space with the norm $\|v\| = \|v\|_\infty = \sup_s |v(s)|$ for any $v \in \mathcal{V}$.

If we modify **P1** by limiting the search space at time t to the configurations can be changed to in each iteration i.e $S_{s_{t-1},\hat{q}_t}$, we get

$$\begin{aligned} \arg\min_{\pi \in \Pi} \quad & \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \left(\delta(s_{t-1}, s_t) + cost(s_t, q_t)\right)\right] \\ \text{such that,} \quad & s_t = \pi(s_{t-1}, q_t), s_t \in S_{s_{t-1},q}, \quad \forall t \in \mathbb{N}. \end{aligned} \tag{3.7}$$

Under this problem statement of Equation (3.7), Algorithm 13 is equivalent to the original LSPI. Additionally, Algorithm 13 converges to a unique cost-to-go function $\tilde{V} \in \mathcal{V}$. In order to prove the convergence to optimal policy, we need to show that $V^* = \tilde{V}$.

Let us define the restricted Bellman operator $\mathcal{T}\colon\ \mathcal{V} \to \mathcal{V}$ based on Equation (3.6).

$$\mathcal{T}V(s) \triangleq \mathbb{E}\left[\min_{s' \in S_{s,q}} \left(\delta(s, s') + cost(s', q) + \gamma V(s')\right)\right]$$

For a particular configuration s and query $\hat{q}$, the optimal action chosen for the next step would be

$$a^*_{s,q}(V) = \arg\min_{s' \in S_{s,q}} \left(\delta(s, s') + cost(s', q) + \gamma V(s')\right)$$

If $\mathcal{T}V(s) \geq \mathcal{T}U(s)$ for $U, V \in \mathcal{V}$, then

$$\begin{aligned}
0 \leq \mathcal{T}V(s) - \mathcal{T}U(s) &= \mathbb{E}\left[\delta(s, a^*_{s,q}(V)) + cost(a^*_{s,q}(V), q) + \gamma V(a^*_{s,q}(V))\right] \\
&\quad - \mathbb{E}\left[\delta(s, a^*_{s,q}(U)) + cost(a^*_{s,q}(U), q) + \gamma U(a^*_{s,q}(U))\right] \\
&\leq \mathbb{E}\left[\delta(s, a^*_{s,q}(U)) + cost(a^*_{s,q}(U), q) + \gamma V(a^*_{s,q}(U))\right] \\
&\quad - \mathbb{E}\left[\delta(s, a^*_{s,q}(U)) + cost(a^*_{s,q}(U), q) + \gamma U(a^*_{s,q}(U))\right] \\
&= \gamma\mathbb{E}\left[\gamma V(a^*_{s,q}(U)) - U(a^*_{s,q}(U))\right] \\
&\leq \gamma\mathbb{E}\left[\|V - U\|\right] \\
&= \gamma\|V - U\|.
\end{aligned}$$

This result states that if $\mathcal{T}V(s) \geq \mathcal{T}U(s)$, then $\mathcal{T}V(s) - \mathcal{T}U(s) \leq \gamma\,|V(s) - U(s)|$. If we assume that $\mathcal{T}V(s) \leq \mathcal{T}U(s)$, then the same reasoning produces $\mathcal{T}V(s) - \mathcal{T}U(s) \geq -\gamma|V(s) - U(s)|$. This means that we have $|\mathcal{T}V(s) - \mathcal{T}U(s)| \leq \gamma|V(s) - U(s)||$ for all configuration $s \in S$. From the definition of the sup-norm, we write

$$\begin{aligned}
\sup_{s \in S} \|\mathcal{M}v(s) - \mathcal{M}u(s)\| &= \|\mathcal{M}v - \mathcal{M}u\| \\
&\leq \gamma\|v - u\|.
\end{aligned}$$

This means that if $0 \leq \gamma < 1$ then $\mathcal{M}$ is a contraction mapping. By[Powell, 2007;Proposition 3.10.2], there exists a unique V^* such that $\mathcal{T}V^* = V^*$, and an arbitrary V^0, the

Algorithm 14 Recursive least squares estimation.

1: **procedure** RLSE($\hat{\epsilon}^t, \bar{B}^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t$)
2: $\gamma^t \leftarrow 1 + (\boldsymbol{\eta}^t)^T \bar{B}^{t-1} \boldsymbol{\eta}^t$
3: $\bar{B}^t \leftarrow \bar{B}^{t-1} - \frac{1}{\gamma^t}(\bar{B}^{t-1}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T\bar{B}^{t-1})$
4: $\boldsymbol{\zeta}^t \leftarrow \boldsymbol{\zeta}^{t-1} - \frac{1}{\gamma^t}\bar{B}^{t-1}\boldsymbol{\eta}^t\hat{\epsilon}^t$
5: **return** $B^t, \boldsymbol{\zeta}^t$.
6: **end procedure**

sequence V^n generated by $V^{n+1} = \mathcal{T}V^n$ converges the fixed point of the space V^*. By the convergence of LSPI [Lagoudakis and Parr, 2003a], $V^* = \tilde{V}$. From Proposition 2, the optimal cost-to-go function V^* also satisfying $\mathcal{M}V^* = V^*$. Hence $V^* = \tilde{V}$ and the assertion follows. □

3.4.4 Learning the Cost Model

In this section, we do not assume the availability of any form of cost model. The cost model is learned through iterative projection and estimation. We assume that there exists a feature mapping $\boldsymbol{\eta}$: $S \times Q \rightarrow \mathbb{R}^n$ such that $cost(s, q) \approx \boldsymbol{\zeta}^T\boldsymbol{\eta}(s, q)$ for some vector $\boldsymbol{\zeta} \in \mathbb{R}^n$. Changing the configuration from s' to s'' can be considered as executing a special query $q(s', s'')$. Therefore, we approximate $\delta(s', s'') = cost(s', q(s', s'')) \approx \boldsymbol{\zeta}^T\boldsymbol{\eta}(s', q(s', s''))$. The vector $\boldsymbol{\zeta}$ can be updated iteratively using the well-known *recursive least squares estimation* (RLSE) as shown in Algorithm 14. Here, $\boldsymbol{\eta}^t = \eta(s_{t-1}, \hat{q}_t)$ and $\hat{\epsilon}^t = (\boldsymbol{\zeta}^{t-1})^T\boldsymbol{\eta}^t - cost(s_{t-1}, \hat{q}_t)$ is the prediction error. Combining RLSE with Algorithm 13 brings the new algorithm with recursive least squares estimation of cost model which is shown in Algorithm 15. It follows the same workflow as Figure 3.2 where the model parameters now refer to $\boldsymbol{\theta}$ and $\boldsymbol{\zeta}$.

Algorithm 15 Least squares policy iteration with recursive least squares estimation.

1: Initialise the initial configuration s_0.
2: Initialise $\boldsymbol{\theta}^0 = \boldsymbol{\theta} = \mathbf{0}$ and $B^0 = \epsilon I$.
3: Initialise $\boldsymbol{\zeta}^0 = \mathbf{0}$ and $\bar{B}^0 = \epsilon I$.
4: **for** t=1,2,3,... **do**
5: Let $\hat{q}_t$ be the just received query.
6:

$$s_t \leftarrow \underset{s \in S_{s_{t-1}, \hat{q}_t}}{\arg\min} (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s_{t-1}, q(s_{t-1}, s)) + (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s, \hat{q}_t) + \gamma \boldsymbol{\theta}^T \boldsymbol{\phi}(s) \tag{3.8}$$

7: Change the configuration to s_t.
8: Execute query $\hat{q}_t$.
9: $\hat{C}^t \leftarrow \delta(s_{t-1}, s_t) + cost(s_t, \hat{q}_t)$.
10: $\hat{\epsilon}^t \leftarrow (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s_{t-1}, \hat{q}_t) - cost(s_{t-1}, \hat{q}_t)$
11: $(B^t, \boldsymbol{\theta}^t) \leftarrow LSTD(\hat{C}^t, B^{t-1}, \boldsymbol{\theta}^{t-1}, \boldsymbol{\phi}(s_{t-1}), \boldsymbol{\phi}(s_t))$
12: $(\bar{B}^t, \boldsymbol{\zeta}^t) \leftarrow RLSE(\hat{\epsilon}^t, \bar{B}^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t)$
13: **if** $(\boldsymbol{\theta}^t)$ converges **then**
14: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^t$.
15: **end if**
16: **end for**

3.5 Automated Database Tuning with Regularised Cost-Model Learning

In the results that we will present in Section 3.7.3, we observe a higher variance of Algorithm 15 for index tuning than that of the state-of-art WFIT algorithm [Schnaitter and Polyzotis, 2012]. This high variance is caused mainly due to the unstable estimation of the cost model. As Algorithm 15 decides the policy depending on the estimated cost model, any error in the cost model causes instability in its outcome.

The process of cost-model estimation by accumulating information of incoming queries is analogous to approximating a function online from its incoming samples. Here, the function is the per-stage cost model $C\colon S \times S \times \widetilde{Q} \to \mathbb{R}$. Here, $\widetilde{Q}$ is the extended set of queries given by $Q \cup \{q(s, s') \mid s, s' \in S\}$. We obtain this $\widetilde{Q}$ by considering configuration updates as special queries, as explained in Section 3.4.3. Now, the per-stage cost function can be defined as

$$C(s_{t-1}, s_t, \hat{q}_t) = cost(s_{t-1}, q(s_{t-1}, s_t)) + cost(s_t, \hat{q}_t)$$

This equation shows that if we consider changing the configuration from s to s' as executing a special query $q(s, s')$, approximating the function $cost\colon S \times \widetilde{Q} \to \mathbb{R}$ in turn approximates the per-stage cost.

As explained in the previous section, we approximate *cost* online using linear projection to the feature space of the observed state and query. At each step we obtain some vector $\boldsymbol{\zeta}$ such that $cost(s, q) \approx \boldsymbol{\zeta}^T \boldsymbol{\eta}(s, q)$. Here, $\boldsymbol{\eta}(s, q)$ is the feature vector corresponding to state s and query q. In order to obtain the optimal approximation, we initialise with an arbitrary ζ and then recursively improve our estimate of $\boldsymbol{\zeta}$ using recursive least squares estimation (RLSE) algorithm [Young, 2011]. But the issues with RLSE are:

i. It tries to minimise the square error per step

$$\hat{\epsilon}_t^2 = \left((\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta^t} - cost(s_{t-1}, \hat{q}_t)\right)^2$$

which is highly sensitive to outliers. If RLSE faces some query or configuration update which is very different from the previously observed queries, the estimation of $\boldsymbol{\zeta}$ can change drastically.

ii. The algorithm becomes unstable if the components of $\boldsymbol{\eta}(s, q)$ are highly correlated. This may happen when the algorithm passes through a series of related queries.

As the reinforcement learning algorithm uses the estimated cost model to decide policies and to evaluate them, error or instability in the estimated cost model at any step affects its performance. Specifically, large deviations arise in the estimated cost model due to the queries which are far from the previously learned distribution. This costs the learning algorithm some time to adapt. It also affects the policy and evaluation of the present state and action. We thus propose to use a *regularised* cost-model estimator instead of RLSE, which is less sensitive to outliers and relatively stable, so as to improve the performance of Algorithm 15 and decrease its variance.

3.5.1 Regularised Cost-Model Estimator

In order to avoid the effect of outliers, we can penalise high variance of $\boldsymbol{\zeta}$ by adding a regularisation term with the squared prediction error of RLSE. Thus at time step t, the new estimator will try to find

$$\boldsymbol{\zeta}^t = \arg\min_{\zeta} P^t \quad \text{given} \;\; \hat{\epsilon}_t, \overline{B}^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t \tag{3.9}$$

such that:

$$\begin{aligned} P^t &\triangleq \hat{\epsilon}_t^2 + \lambda \|\boldsymbol{\zeta}^{t-1}\|_2^2 \\ &= \left(\langle \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta^t} \rangle - cost(s_{t-1}, \hat{q}_t)\right)^2 + \lambda \langle \boldsymbol{\zeta}^{t-1}, \boldsymbol{\zeta}^{t-1} \rangle. \end{aligned}$$

Here, $\lambda \in \mathbb{R}^+$ is the regularisation parameter. Square of L_2-norm, $\|\boldsymbol{\zeta}\|_2^2$, is the regularisation function. $\boldsymbol{\eta^t} \triangleq \boldsymbol{\eta}(s_{t-1}, \hat{q}_t)$ is the feature vector of state s_{t-1} and query $\hat{q}_t$. We call this squared error the *loss function* L_t which is defined at time $t > 0$ for a given choice of $\boldsymbol{\zeta}^t$. Thus,

$$L_t(\boldsymbol{\zeta}^t) \triangleq \left(\langle \boldsymbol{\zeta}^t, \boldsymbol{\eta^t} \rangle - cost(s_{t-1}, \hat{q}_t)\right)^2.$$

The dual of this problem can be considered as picking up such a vector $\boldsymbol{\zeta}^t$ inside an n-dimensional Euclidean ball $\mathbb{B}_\lambda^n$ of radius $s(\lambda)$ that minimises the error $\hat{\epsilon}_t^2$. From an optimisation point of view, we choose $\boldsymbol{\zeta}^t$ inside $\mathbb{B}_\lambda^n \triangleq \{\boldsymbol{\zeta} \mid \|\boldsymbol{\zeta}\|_2^2 \leq s(\lambda) \text{ and } \boldsymbol{\zeta} \in \mathbb{R}^n\}$ rather than searching for it in the whole space of $\mathbb{R}^n$. This estimator penalises any drastic change in the cost model due to some outlier query. If some query tries to pull $\boldsymbol{\zeta}^t$ out of $\mathbb{B}_\lambda^n$, this estimator regularises $\boldsymbol{\zeta}^t$ at the boundary. It also induces sparsity in the components of estimating vector $\boldsymbol{\zeta}$ that eliminates the instability due to highly correlated queries.

Algorithm 16 Regularised cost-model estimation.

1: Initialise $\boldsymbol{\zeta}^0 = \mathbf{0}$ and $R^0 = \varepsilon I$.
2: **for** t=1,2,3,... **do**
3: $\quad \hat{\epsilon}_t \leftarrow (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}^t - cost(s_{t-1}, \hat{q}_t)$
4: $\quad \gamma^t \leftarrow \lambda + (\boldsymbol{\eta}^t)^T R^{t-1} \boldsymbol{\eta}^t$
5: $\quad R^t \leftarrow R^{t-1} - \frac{1}{\gamma^t}(R^{t-1}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T R^{t-1})$
6: $\quad \boldsymbol{\zeta}^t \leftarrow \boldsymbol{\zeta}^{t-1} - \frac{1}{\gamma^t} R^{t-1}\boldsymbol{\eta}^t \hat{\epsilon}_t$
7: $\quad$ **return** $R^t, \boldsymbol{\zeta}^t$
8: **end for**

The online penalised cost-model estimation algorithm obtained from this formulation is shown in Algorithm 16. Generally, the optimal values of ε and λ are decided using a cross-validation procedure. In Section 3.6.4, we derive the optimal value of ε and a probable estimation for λ for the index tuning problem. This will decide optimal values of the hyper-parameters for a given set of workload with theoretical performance bounds.

3.5.2 Performance Bound

Now, we depict this online cost-model estimation task as a game between a decision maker and an adversary [Rockafellar, 2015]. In database tuning, the decision maker is our cost-model estimation algorithm, and the adversary is the workload providing an uncertain sequence of queries. Then, we can formulate the game as Algorithm 17.

Algorithm 17 Cost-model Estimation Game.

1: Initialise $\boldsymbol{\zeta}^0 = \mathbf{0}$.
2: **for** t=1,2,3,..., T **do**
3: Algorithm 16 picks $\boldsymbol{\zeta}^t \in \mathbb{B}^n_\lambda$ according to Equation (3.9)
4: Adversary picks $(\boldsymbol{\eta}_t, c_t)$
5: Algorithm suffers loss $L_t(\boldsymbol{\zeta}^t)$
6: **end for**

We can define the regret of this game after time step T as,

$$Reg_T \coloneqq \sum_{t=1}^{T} L_t(\boldsymbol{\zeta}^t) - \sum_{t=1}^{T} L_t(\boldsymbol{\zeta}^{\mathrm{OPT}}) \tag{3.10}$$

where $\boldsymbol{\zeta}^{\mathrm{OPT}}$ is the optimal solution picked up by an offline expert that minimises the cumulative loss after time step T. Reg_T is the difference between cumulative sum of errors up to time T obtained using Algorithm 16 and the optimal offline algorithm. This regret term captures deviation of the cost-model estimated by the Algorithm 17 from the computable optimal cost model.

As the loss function $L(\boldsymbol{\zeta})$ is the square of the error between estimated and actual values of cost at time t, it is a convex function over the set of $\boldsymbol{\zeta}$. According to the analysis given in [Rockafellar, 2015], we canonically describe our estimation model as a scheme to develop a Legendre potential function $\boldsymbol{\Phi}\colon \mathbb{B}^n_\lambda \to \mathbb{R}^{\geq 0}$ that evolves with time t for the given workload. The initial value of potential is given by

$$\boldsymbol{\Phi}_0(\boldsymbol{\zeta}^{\mathrm{OPT}}) \triangleq \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2$$

and its value at time t is updated as

$$\boldsymbol{\Phi}_t(\boldsymbol{\zeta}^{\mathrm{OPT}}) \triangleq \boldsymbol{\Phi}_{t-1}(\boldsymbol{\zeta}^{\mathrm{OPT}}) + \frac{1}{\lambda} L_t(\boldsymbol{\zeta}^t).$$

Now, we can reformulate Equation (3.9) as

$$\boldsymbol{\zeta}^t = \underset{\boldsymbol{\zeta} \in \mathbb{B}_\lambda^n}{\arg\min} \left[D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^{t-1}) + \frac{1}{\lambda} \left(\boldsymbol{\nabla} L_{t-1}(\boldsymbol{\zeta}_{t-1}) \right)^T \boldsymbol{\zeta}^{t-1} \right] \tag{3.11}$$

Here, $D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^{t-1})$ is the Bregman divergence [Nielsen and Bhatia, 2013] between $\boldsymbol{\zeta}^{\mathrm{OPT}}$ and $\boldsymbol{\zeta}^{t-1}$ along the potential field $\boldsymbol{\Phi}_0(\boldsymbol{\zeta}^{\mathrm{OPT}})$. This term in Equation (3.11) inclines Algorithm 16 to choose such a $\boldsymbol{\zeta}$ which is nearest to optimal $\boldsymbol{\zeta}^{\mathrm{OPT}}$ on the $\|\boldsymbol{\zeta}\|^2$ manifold. $(\boldsymbol{\nabla} L_{t-1}(\boldsymbol{\zeta}_{t-1}))^T \boldsymbol{\zeta}^{t-1}$ is the change of the loss function in the direction of $\boldsymbol{\zeta}^{t-1}$. Minimisation of this term is equivalent to selection of such a $\boldsymbol{\zeta}^t$ that minimises the corresponding loss.

Thus, the $\boldsymbol{\zeta}^t$ picked up by the Algorithm is the one that minimises a linear combination of these two terms weighted by λ. From this formulation we can obtain the following lemma for the regret bound.

Lemma 3. *After time step T, the upper bound of the regret of Algorithm 16 can be given by*

$$Reg_T \leq \lambda \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{1}{\lambda} \sum_{t=1}^{T} \hat{\epsilon}_t{}^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t. \tag{3.12}$$

Proof. Applying Theorem 1 of [Warmuth and Jagota, 1997] on Equation (3.11) we get the inequalities,

$$\begin{aligned} Reg_T &\leq \lambda \left[D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^0) - D_{\boldsymbol{\Phi}_T}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^{T+1}) + \sum_{t=1}^{T} D_{\boldsymbol{\Phi}_t}(\boldsymbol{\zeta}^t, \boldsymbol{\zeta}^{t+1}) \right] \\ &\leq \lambda \left[D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^0) + \sum_{t=1}^{T} D_{\boldsymbol{\Phi}_t}(\boldsymbol{\zeta}^t, \boldsymbol{\zeta}^{t+1}) \right]. \end{aligned}$$

From the definition of the Legendre potential we get:

$$\begin{aligned}
\boldsymbol{\Phi}_t(\boldsymbol{\zeta}) &= \boldsymbol{\Phi}_{t-1}(\boldsymbol{\zeta}) + \frac{1}{\lambda} L_t(\boldsymbol{\zeta}) \\
&= \|\boldsymbol{\zeta}\|^2 + \sum_{t=1}^{T} \left(\langle \boldsymbol{\zeta}, \boldsymbol{\eta^t} \rangle - cost(s_{t-1}, \hat{q}_t) \right)^2 \\
&= \boldsymbol{\zeta}^T \left(I + \frac{1}{\lambda} \sum_{t=1}^{T} \boldsymbol{\eta}^t (\boldsymbol{\eta}^t)^T \right) \boldsymbol{\zeta} - \boldsymbol{\zeta}^T \left(\frac{1}{\lambda} \sum_{t=1}^{T} cost(s_{t-1}, \hat{q}_t) \boldsymbol{\eta}^t \right) + \sum_{t=1}^{T} cost(s_{t-1}, \hat{q}_t)^2 \\
&= \boldsymbol{\zeta}^T (R^T)^{-1} \boldsymbol{\zeta} - \boldsymbol{\zeta}^T b_T + C_T
\end{aligned}$$

where $b_T = \sum_{t=1}^{T} c^t \boldsymbol{\eta}^t$ and $C_T = \sum_{t=1}^{T} cost(s_{t-1}, \hat{q}_t))^2$. Thus, the dual of the potential is

$$\boldsymbol{\Phi}_t^*(\boldsymbol{\zeta}) = \boldsymbol{\zeta}^T R^T \boldsymbol{\zeta} - 2\boldsymbol{\zeta}^T R^T b_T + (b_T)^T R^T b_T$$

Following this, we get by using the definition of $\boldsymbol{\Phi}_0$ and properties of Bregman divergence,

$$\begin{aligned}
D_{\boldsymbol{\Phi}_0}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^0) &= D_{\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2}(\boldsymbol{\zeta}^{\mathrm{OPT}}, \boldsymbol{\zeta}^0) \\
&= \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2
\end{aligned}$$

and

$$\begin{aligned}
D_{\boldsymbol{\Phi}_t}(\boldsymbol{\zeta}^t, \boldsymbol{\zeta}^{t+1}) &= D_{\boldsymbol{\Phi}_t^*} \left(\boldsymbol{\nabla} \boldsymbol{\Phi}_t(\boldsymbol{\zeta}^{t+1}), \boldsymbol{\nabla} \boldsymbol{\Phi}_t(\boldsymbol{\zeta}^t) \right) \\
&= D_{\boldsymbol{\Phi}_t^*} \left(\mathbf{0}, \boldsymbol{\nabla} \boldsymbol{\Phi}_t(\boldsymbol{\zeta}^t) \right) \\
&= D_{\boldsymbol{\Phi}_t^*} \left(\mathbf{0}, \frac{1}{\lambda} \boldsymbol{\nabla} L_t(\boldsymbol{\zeta}^t) \right) \\
&= \frac{1}{\lambda^2} \left(\boldsymbol{\nabla} L_t(\boldsymbol{\zeta}^t) \right)^T R^t \left(\boldsymbol{\nabla} L_t(\boldsymbol{\zeta}^t) \right) \\
&= \frac{1}{\lambda^2} \left(\langle \boldsymbol{\zeta}, \boldsymbol{\eta^t} \rangle - cost(s_{t-1}, \hat{q}_t) \right)^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t \\
&= \frac{1}{\lambda^2} \hat{\epsilon_t}^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t.
\end{aligned}$$

By substituting these results in the aforementioned inequality, we get

$$Reg_T \leq \lambda \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{1}{\lambda} \sum_{t=1}^{T} \hat{\epsilon_t}^2 (\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t. \qquad \square$$

Lemma 4. *If $R^0 \in \mathbb{R}^{n \times n}$ and invertible,*

$$(\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t = 1 - \frac{\det(R^t)}{\det(R^{t-1})} \quad \forall t = 1, 2, \ldots, T \tag{3.13}$$

Proof. From [Lai and Wei, 1982], we get if there exists an invertible matrix $B \in \mathbb{R}^{n \times n}$ such that $A = B + \boldsymbol{x}\boldsymbol{x}^T$, where $\boldsymbol{x} \in \mathbb{R}^n$, then

$$\boldsymbol{x}^T A^{-1} \boldsymbol{x} = 1 - \frac{\det(B)}{\det(A)} \tag{3.14}$$

We define in Algorithm 16, $R^0 = \varepsilon I$ with $\varepsilon \in (0, 1)$. Thus, R^0 it is invertible. Since $(R^t)^{-1} = (R^{t-1})^{-1} + \boldsymbol{\eta}^t (\boldsymbol{\eta}^t)^T$, by the Sherman–Morrison formula, all R^t's are invertible for $t \geq 0$. Thus, simply replacing A by $(R^t)^{-1}$ and B by $(R^{t-1})^{-1}$ in Equation (3.14), we obtain

$$(\boldsymbol{\eta}^t)^T R^t \boldsymbol{\eta}^t = 1 - \frac{\det((R^{t-1})^{-1})}{\det((R^t)^{-1})} = 1 - \frac{\det(R^t)}{\det(R^{t-1})}$$

since, $\det((R^t)^{-1}) = \frac{1}{\det(R^t)}$. $\square$

Using Lemmas 3 and 4, we finally derive the regret bound for the regularised cost-model estimator in the following theorem.

Theorem 7. *If we consider the error as a bounded function such that $0 \leq \hat{\epsilon_t}^2 \leq E_{\max}$ and $\|\boldsymbol{\eta}^t\|_\infty \leq \delta$,*

$$Reg_T \leq \lambda \|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda} \left[n \ln \left(1 + \frac{\varepsilon \delta^2 T}{n} \right) - (n-1) \ln(\varepsilon) \right] \tag{3.15}$$

where $R^0 = \varepsilon I$, and $\varepsilon \in (0, 1)$.

Proof. Let us assume the squared error has an upper bound $E_{\max}$ for a given workload. Under this assumption, we get from Equations (3.12) and (3.13),

$$\begin{aligned}
Reg_T &\leq \lambda\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\sum_{t=1}^{T}\left(1 - \frac{\det(R^t)}{\det(R^{t-1})}\right) \\
&\leq \lambda\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 - \frac{E_{\max}}{\lambda}\sum_{t=1}^{T}\ln\left(\frac{\det(R^t)}{\det(R^{t-1})}\right) \\
&= \lambda\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\ln\left(\frac{\det(R^0)}{\det(R^T)}\right) \\
&= \lambda\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[\ln(\varepsilon) - \ln(\det(R^T))\right] \\
&= \lambda\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[\ln(\varepsilon) + \ln\left(\det\left(\frac{1}{\varepsilon}I + \sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T\right)\right)\right] \\
&= \lambda\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[\sum_{k=1}^{n}\ln\left(1+\varepsilon\lambda_k\right) - (n-1)\ln(\varepsilon)\right].
\end{aligned}$$

Because

$$\det\left(\frac{1}{\varepsilon}I + \sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T\right) = \varepsilon^{-n}\det\left(I + \varepsilon\sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T\right) \qquad = \varepsilon^{-n}\prod_{k=1}^{n}(1+\varepsilon\lambda_k)$$

where $\lambda_1, \ldots, \lambda_n$ are eigenvalues of the matrix $\sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T$. As the eigenvalues of $\sum_{t=1}^{T}\boldsymbol{\eta}^t(\boldsymbol{\eta}^t)^T$ are equal to the eigenvalues of its Gram matrix $G_{ij} = (\boldsymbol{\eta}^i)^T\boldsymbol{\eta}^j$, we can write

$$\sum_{k=1}^{n}\lambda_k = Trace(G) = \sum_{t=1}^{T}(\boldsymbol{\eta}^t)^T\boldsymbol{\eta}^t \leq \delta^2 T$$

where $\|\boldsymbol{\eta}^t\|_\infty \leq \delta$, that is, the maximum value of any component of $\boldsymbol{\eta}$ is bounded by δ. In the above inequality, the equality holds if and only if $\lambda_1 = \lambda_2 = \ldots = \lambda_n = \frac{\delta^2 T}{n}$. By applying this condition, we get the regret bound as

$$Reg_T \leq \lambda\|\boldsymbol{\zeta}^{\mathrm{OPT}}\|^2 + \frac{E_{\max}}{\lambda}\left[n\ln\left(1 + \frac{\varepsilon\delta^2 T}{n}\right) - (n-1)\ln(\varepsilon)\right]. \qquad \square$$

This theorem shows that our estimation of the cost model using Algorithm 16 is always upper bounded by a constant value depending on the optimal solution added with a term that increases with time logarithmically. This shows that the regret, which is the cumulative deviation of the cost model computed by Algorithm 16 with respect to the optimal one, increases logarithmically with time. This implies that the error of estimation of the cost function at each time step is considerably small, stable, and does not blow up with time.

3.6 Case Study: Adaptive Index Tuning

In this section, we consider the design of instances of Algorithms 13 and 15 for the case of configurations differing in their secondary indexes. Configuration changes correspond to creating and deleting indexes. We show how the search space can be reduced in this case. We design feature mappings $\boldsymbol{\phi}$ and $\boldsymbol{\eta}$ for index tuning. We also show tighter regret bounds, and also in turn, how it leads to optimal parameter selection.

3.6.1 Reducing the Search Space

Let I be the set of indexes can be created. Each configuration $s \in S$ is a subset of the power set 2^I. For each $s \in S$, let $|s|$ be the number of indexes contained in s. For example, 7 attributes in the schema of R yield a total of 13699 indexes and a total of 2^{13699} possible configurations. Such a large search space invalidates a naive brute-force search for **P1**.

For any query $\hat{q}$, we let $r(\hat{q})$ be a function that returns a set of recommended indexes. This function may be already provided by the database system (e.g., as with IBM DB2), or it can be implemented externally [Agrawal et al., 2000; Bruno and Chaudhuri, 2007a]. Let $d(\hat{q}) \subseteq I$ be the set of indexes being modified (update,

insertion or deletion) by $\hat{q}$. We define the reduced search space as

$$S_{s,\hat{q}} \triangleq \{s' \in S \mid (s - d(\hat{q})) \subseteq s' \subseteq (s \cup r(\hat{q}))\}. \tag{3.16}$$

Deleting indexes in $d(\hat{q})$ will reduce the index maintenance overhead and creating indexes in $r(q)$ will reduce the query execution cost. Note that the definition of $S_{s,\hat{q}}$ here is a subset of the one defined in section 3.4.2 which deals with general configurations. For indexes, we can further reduce the size of searching set because deleting an index not in $d(q)$ or creating an index not in $r(q)$ for the current configuration s does not reduce the cost of query execution.

Notice that for tree-structured indexes (e.g., B$^+$-tree) we could further consider the *prefix closure* of indexes for optimisation. For any configuration $s \subseteq 2^I$, define the prefix closure of s as

$$\langle s \rangle = \{i \in I \mid i \text{ is a prefix of an index } j \text{ for some } j \in s\}. \tag{3.17}$$

Thus in (3.16), we use $\langle r(\hat{q}) \rangle$ to replace $r(\hat{q})$ for better approximation. The intuition is that in case of $i \notin s$ but $i \subseteq \langle s \rangle$ we can leverage the prefix index to answer the query.

3.6.2 Defining the Feature Mapping ϕ

Let V be the cost-to-go function following a policy. As aforementioned, both Algorithms 13 and 15 rely on a proper feature mapping $\boldsymbol{\phi}$ towards an appropriate approximation $V(s) \approx \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for some vector $\boldsymbol{\theta}$. The challenge lies in how to define $\boldsymbol{\phi}$ under the scenario of index tuning. We explain its definition as following. For each $s, s' \in S$, define

$$\phi_{s'}(s) = \begin{cases} 1, & \text{if } s' \subseteq s \\ -1, & \text{otherwise.} \end{cases}$$

Let $\boldsymbol{\phi} = (\phi_{s'})_{s' \in S}$. Notice that $\phi_\emptyset$ is an intercept term since $\phi_\emptyset(s) = 1$ for all $s \in S$. The following proposition shows the effectiveness of $\boldsymbol{\phi}$ for capturing the values of the cost-to-function V.

Proposition 2. *There exists an unique* $\boldsymbol{\theta} = (\theta_{s'})_{s' \in S}$ *such that*

$$V(s) = \sum_{s' \in S} \theta_{s'} \phi_{s'}(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s). \tag{3.18}$$

Proof. Suppose $S = \{s^1, s^2, \ldots, s^{|S|}\}$. Notice that we use superscripts to denote the ordering of elements in S.

Let $\boldsymbol{V} = (V(s))_{s \in S}^T$ and M be a $|S| \times |S|$ matrix such that

$$M_{i,j} = \phi_{s^j}(s^i).$$

Let $\boldsymbol{\theta}$ be a $|S|$-dimension column vector such that $M\boldsymbol{\theta} = \boldsymbol{V}$. If M is invertible then $\boldsymbol{\theta} = M^{-1}V$ and thus (3.18) holds. We now show that M is invertible. Let $\boldsymbol{\psi}$ be a $|S| \times |S|$ matrix such that

$$\boldsymbol{\psi}_{i,j} = M_{i,j} + 1.$$

We claim that $\boldsymbol{\psi}$ is invertible and its inverse is the matrix $\boldsymbol{\tau}$ such that

$$\boldsymbol{\tau}_{i,j} = (-1)^{|s^i| - |s^j|} \boldsymbol{\psi}_{i,j}.$$

To see this, consider

$$\begin{aligned}(\boldsymbol{\tau}\boldsymbol{\psi})_{i,j} &= \sum_{1 \le k \le |S|} (-1)^{|s^i| - |s^k|} \boldsymbol{\psi}_{i,k} \boldsymbol{\psi}_{k,j} \\ &= \sum_{s_j \subseteq s_k \subseteq s_i} (-1)^{|s^i| - |s^k|}.\end{aligned}$$

Therefore $(\boldsymbol{\tau}\boldsymbol{\psi})_{i,j} = 1$ if and only if $i = j$. By the Sherman-Morrison formula, M is also invertible. □

However, for any configuration s, $\boldsymbol{\theta}(s)$ is a $|2^I|$ dimensional vector. To reduce the dimensionality, based on (3.18), the cost-to-go function can be approximated by

$$V(s) \approx \sum_{s' \in S, |s'| \leq N} \theta_{s'} \phi_s(s) \tag{3.19}$$

for some integer N. Here we assume that the collaborative benefit among indexes could be negligible if the number of indexes exceeds N. In particular when $N = 1$, we have

$$V(s) \approx \theta_0 + \sum_{i \in I} \theta_i \phi_i(s). \tag{3.20}$$

and

$$\phi_i(s) = \begin{cases} 1, & \text{if } i \in s \\ -1, & \text{otherwise.} \end{cases}$$

where we ignore all the collaborative benefits among indexes in a configuration. This is reasonable since any index in a database management system is often of individual contribution for answering queries [Ramakrishnan et al., 2003]. Therefore, we derive $\boldsymbol{\phi}$ from (3.20) as

$$\boldsymbol{\phi}(s) = (1, (\phi_i(s))_{i \in I}^T)^T.$$

By using this feature mapping $\boldsymbol{\phi} \in \mathbb{R}^{|I|+1}$, we can approximate the cost-to-go function $V(s) \approx \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for some vector $\boldsymbol{\theta} \in \mathbb{R}^{|I|+1}$.

3.6.3 Defining the Feature Mapping η

A good feature mapping for approximating the function δ and the function *cost* must take into account both the benefit from the current configuration and the maintenance overhead of the configuration.

To capture the difference between the index set recommended by the database system and the index set of the current configuration, we define a function

$$\boldsymbol{\beta}(s, \hat{q}) = (1, (\boldsymbol{\beta}_i(s, \hat{q}))_{i \in I}^T)^T,$$

where $\boldsymbol{\beta}$ is a $|I| + 1$ dimensional real-valued vector with components

$$\boldsymbol{\beta}_i(s, \hat{q}) = \begin{cases} 0, & i \notin r(\hat{q}) \\ 1, & i \in r(\hat{q}) \text{ and } i \in s \\ -1, & i \in r(\hat{q}) \text{ and } i \notin s. \end{cases} \tag{3.21}$$

If the execution of query $\hat{q}$ cannot benefit from index i then $\boldsymbol{\beta}_i(s, \hat{q})$ always equals zero; otherwise, $\boldsymbol{\beta}_i(s, \hat{q})$ equals 1 or -1 depending on whether s contains i or not.

For tree-structured indexes, we could further consider the prefix closure of indexes as defined in (3.17) for optimisation. Thus in (3.21), we use $\langle s \rangle$ and $\langle r(\hat{q}) \rangle$ to replace s and $r(\hat{q})$, respectively, for better approximation.

On the other hand, to capture whether a query (update, insertion or deletion) modifies any index in the current configuration, we define a function

$$\boldsymbol{\alpha}(s, \hat{q}) = (\alpha_i(s, \hat{q}))_{i \in I}$$

where $\boldsymbol{\alpha}$ is a $|I|$ dimensional real-valued vector with components

$$\alpha_i(s, \hat{q}) = \begin{cases} 1, & \text{if } i \in s \text{ and } \hat{q} \text{ modify } i \\ 0, & \text{otherwise.} \end{cases}$$

Notice that if $\hat{q}$ is a selection query, $\boldsymbol{\alpha}$ trivially returns $\mathbf{0}$.

By combining $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$, we get the feature mapping

$$\boldsymbol{\eta} = (\boldsymbol{\beta}^T, \boldsymbol{\alpha}^T)^T$$

such that $\boldsymbol{\eta} \in \{0, \pm 1\}^{2|I|+1}$. Thus, $n \leq |I| + 1$. We use this feature mapping $\boldsymbol{\eta}$ to approximate the cost function of configuration update δ, and the cost function of query execution *cost* as mentioned in Section 3.4.4.

3.6.4 Performance Bounds for Regularised COREIL

rCOREIL applies Algorithm 16 for cost-model estimation, while COREIL uses RLSE for this. If we follow Algorithm 15, on line 13 rCOREIL calls the regularised cost-model estimator with arguments $\hat{\epsilon}^t, R^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t$ instead of RLSE. Following Theorem 7 and the construction of the feature map in Section 3.6.3, Proposition 3 gives a tighter regret bound for the cost-model estimation of rCOREIL.

Proposition 3. *If we consider the error as a bounded function such that* $0 \leq {\hat{\epsilon}_t}^2 \leq E_{\max}$*, and the horizon* T *and the number of indexes* n *satisfies* $0 < n^2 - n < T$*, then*

$$Reg_T^{rCOREIL} \leq \lambda \|\boldsymbol{\zeta}^{OPT}\|^2 + \frac{E_{\max}}{\lambda} \left[n \ln \left(1 + \frac{T}{n} \right) + n \ln \left(\frac{T}{n} \right) \right] \tag{3.22}$$

and the optimal value for ε *is given by:*

$$\varepsilon^* = \frac{n^2 - n}{T}.$$

Proof. From Section 3.6.3, $\|\boldsymbol{\eta}^t\|_\infty \leq 1$. Equation (3.15) transforms into

$$Reg_T^{rCOREIL} \leq \lambda \|\boldsymbol{\zeta}^{OPT}\|^2 + \frac{E_{\max}}{\lambda} \left[n \ln \left(1 + \frac{\epsilon T}{n} \right) - (n-1) \ln(\varepsilon) \right].$$

Now, we determine the optimal value of ε by minimising the RHS of above inequality as this will impose tighter limit on the bound. Thus,

$$\left[\frac{\partial(\text{RHS})}{\partial\varepsilon}\right]_{\varepsilon^*=0} = 0.$$

By solving this, we get $\varepsilon^* = \frac{n^2-n}{T}$. Substituting this value in the previous inequality gives us the regret bound for regularised COREIL algorithm as

$$Reg_T^{rCOREIL} \leq \lambda\|\boldsymbol{\zeta}^{OPT}\|^2 + \frac{E_{\max}}{\lambda}\left[n\ln\left(1+\frac{T}{n}\right) + n\ln\left(\frac{T}{n}\right)\right]. \qquad \square$$

Similarly, we can also find out the optimal value of λ that will make the upper bound tightest.

Corollary 1. *If the value of optimal solution $\boldsymbol{\zeta}^{OPT}$ can be predicted beforehand, the optimal value of λ is given by*

$$\lambda^* = \frac{E_{\max}}{\|\boldsymbol{\zeta}^{OPT}\|^2}\left[n\ln\left(1+\frac{T}{n}\right) + n\ln\left(\frac{T}{n}\right)\right],$$

where the stopping time T satisfies $n^2 - n < T$.

Proof. As an optimal λ will minimise the RHS of Equation (3.22), we get it by setting the partial derivative of the RHS with respect to λ as zero. This simply gives us, $\lambda^* = \frac{E_{\max}}{\|\boldsymbol{\zeta}^{OPT}\|^2}\left[n\ln\left(1+\frac{T}{n}\right) + n\ln\left(\frac{T}{n}\right)\right]$. $\square$

Substituting the optimal value of λ in Equation (3.22) for a given T and $\boldsymbol{\zeta}^{OPT}$, we get

$$Reg_T^{rCOREIL} \leq \|\boldsymbol{\zeta}^{OPT}\|^2 + 2E_{\max}\ \ln\left(1+\frac{T}{n}\right).$$

This shows the guarantee on the logarithmic growth of regret for cost-model estimation by rCOREIL once the parameters are properly set.

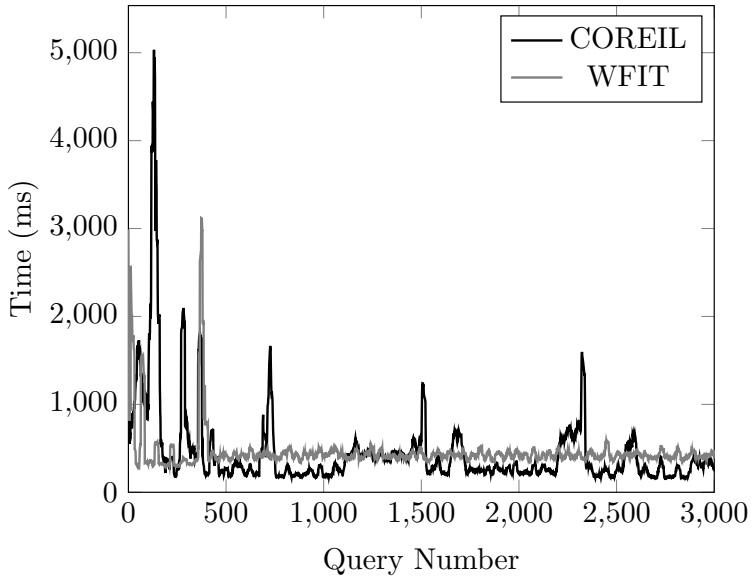


Figure 3.3: Evolution of the efficiency (total time per query) of the two systems from the beginning of the workload (smoothed by averaging over a moving window of size 20)

3.7 Performance Evaluation

In this section, we present an empirical evaluation of COREIL and rCOREIL through two sets of experiments. In the first set of experiments, we implement a prototype of COREIL in Java. We compare its performance with that of the state-of-the-art WFIT algorithm [Schnaitter and Polyzotis, 2012] (briefly described in Section 3.7.2). In the results, we can see that COREIL shows competitive performance with WFIT but has higher variance. This validates the efficiency of the reinforcement learning approach to solve the index tuning problem *on the fly*. This shows that, even without any assumption of a pre-determined cost model, it is possible to perform at the level of the state-of-the-art. In the second set of experiments, we evaluate the performance of rCOREIL with respect to COREIL. The results show enhancements in performance by rCOREIL as reasoned in Section 3.5. This validates the claim in Section 3.5 that the higher variance of COREIL is due to suboptimal use of the RLSE algorithm. It also establishes the fact that if we serve the learning algorithm with an enhanced estimation of cost-model, it improves the performance substantially. In these experiments, we also

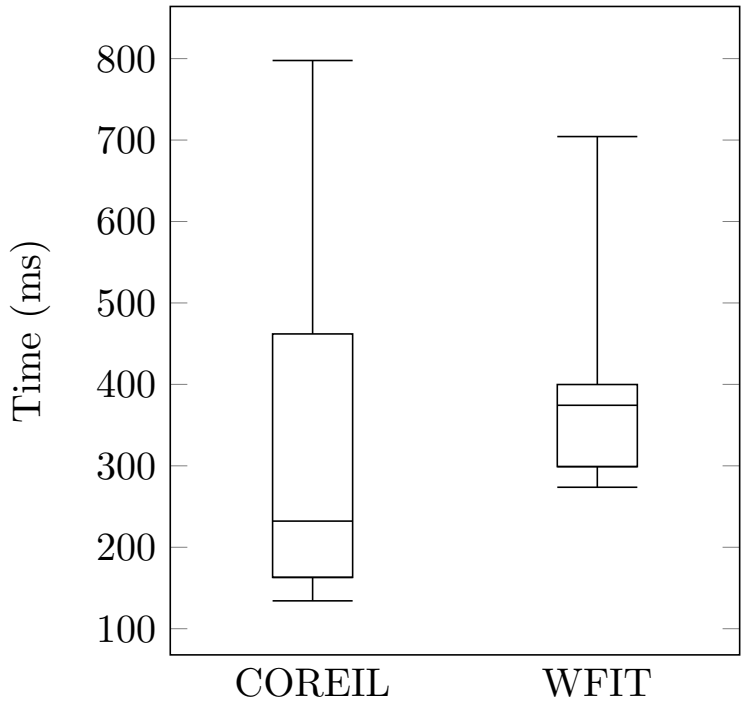


Figure 3.4: Box chart of the efficiency (total time per query) of the two systems. We show in both cases the 9th and 91th percentiles (whiskers), first and third quartiles (box) and median (horizontal rule).

check the sensitivity of rCOREIL with respect to the parameter λ and cross-validate the optimal value for the given workload.

3.7.1 Dataset and Workload

The dataset and workload is conforming to the TPC-C specification [Raab, 1993] and generated by the OLTP-Bench tool [Difallah et al., 2013]. The 5 types of transactions in TPC-C are distributed as NewOrder 45%, Payment 43%, OrderStatus 4%, Delivery 4% and StockLevel 4%. Each of these transactions are associated with $3 \sim 5$ SQL statements (query/update). The scale factor used throughout the experiments is 2. We do not leverage any repetition or periodicity of the workload in our approach; still for robustness there may be up to 10% of repetition of queries. Note that [Schnaitter and Polyzotis, 2012] additionally uses the dataset NREF in its experiments. However, this dataset and workload are not publicly available.

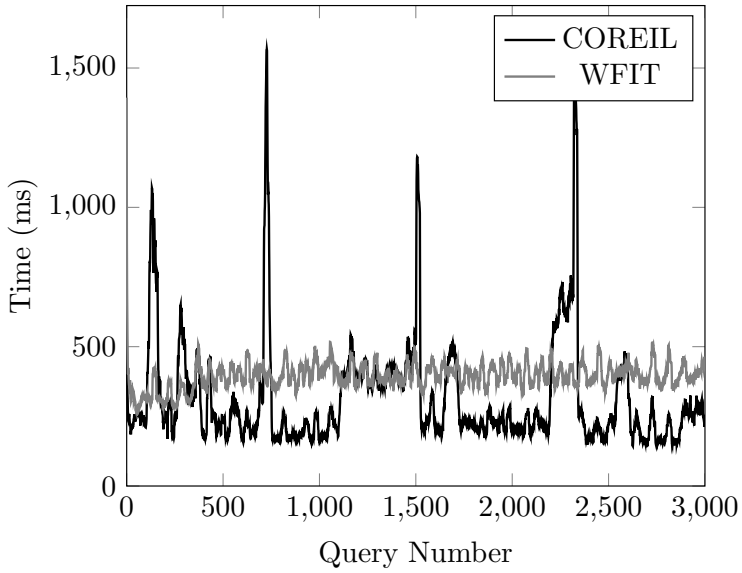


Figure 3.5: Evolution of the overhead (time of the optimisation itself) of the two systems from the beginning of the workload (smoothed by averaging over a moving window of size 20)

3.7.2 WFIT: Brief Description

WFIT is proposed in [Schnaitter and Polyzotis, 2012] as a method of semi-automatic index tuning. This algorithm keeps the database administrator in the loop by generating recommendations. These recommendations are generated through a feedback loop originating from the administrator's preferences. This process is based on the Work Function Algorithm [Borodin and El-Yaniv, 1998]. In order to determine the change of configuration, WFIT considers all the queries observed in the past. Then it solves a deterministic problem of minimising the total processing cost. However, while doing so, it assumes the existence of a pre-determined cost model served by the database system or administrator. Due to use of a pre-defined cost model for all the datasets and workloads it faces the problems discussed in the Introduction. Results documented in the following sections will show the importance of a reinforcement learning approach to make the process generic and cost-model oblivious.

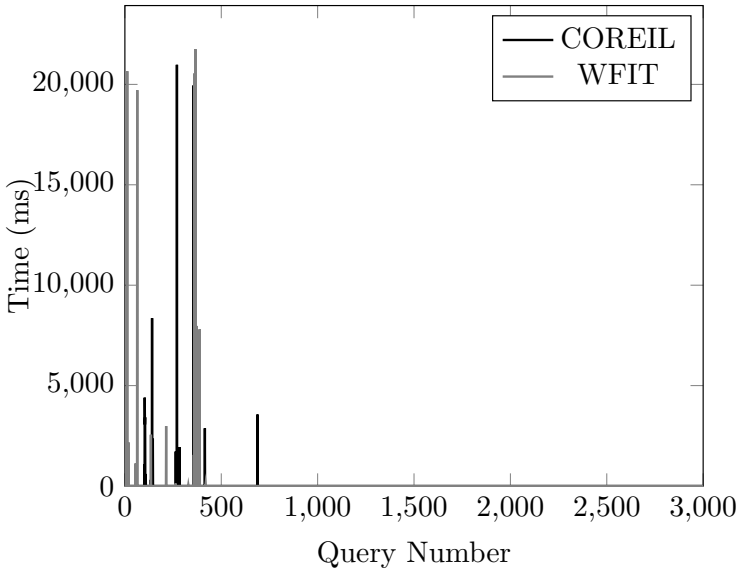


Figure 3.6: Evolution of the time taken by configuration change (index creation and destruction) of the two systems from the beginning of the workload; no configuration change happens past query number 700. All values except the vertical lines shown in the figure are zero.

3.7.3 COREIL: Experiments and Results

Experimental Set-up

We conduct all the experiments on a server running IBM DB2 10.5. The server is equipped with Intel i7-2600 Quad-Core @ 3.40 GHz and 4 GB RAM. We measure wall-clock times for execution of all components. Specially, for execution of workload queries or index creating/dropping, we measure the response time of processing corresponding SQL statement in DB2. Additionally, WFIT uses the what-if optimiser of DB2 to evaluate the cost. In this setup, each query is executed only once and all the queries were generated from one execution history.

Efficiency

Figure 3.3 shows the total cost of processing TPC-C queries for online index tuning of COREIL and WFIT. Total cost consists of the overhead of corresponding tuning algorithm, cost of configuration change and that of query execution. Results show that,

after convergence, COREIL has lower processing cost most of the time. But COREIL converges slower than WFIT, which is expected since it does not rely on the what-if optimiser to guide the index creations. [4] With respect to the whole execution set, the average processing cost of COREIL (451 ms) is competitive to that of WFIT (452 ms). However, if we calculate the average processing cost of the 500^{th} query forwards, the average performance of COREIL (357 ms) outperforms that of WFIT (423 ms). To obtain further insight from these data, we study the distribution of the processing time per query, as shown in Figure 3.4. As can be seen, although COREIL exhibits larger variance in the processing cost, its median is significantly lower that that of WFIT. All these results confirms that COREIL has better efficiency than WFIT under a long term execution.

Figures 3.5 and 3.6 show analysis of the overhead of corresponding tuning algorithm and cost of configuration change respectively. By comparing Figure 3.3 with Figure 3.5, we can see that the overhead of the tuning algorithm dominates the total cost and the overhead of COREIL is significantly lower than that of WFIT. In addition, WFIT tends to make costlier configuration changes than COREIL, which is reflected in a higher time for configuration change. This would be discussed further in the micro-analysis. Note that both methods converge rather quickly and no configuration change happens beyond the 700th query.

A possible reason for the comparatively smaller overhead of COREIL with respect to WFIT, in addition to not relying on a possibly costly what-if optimiser, is the MDP structure. In MDPs, all the history of the system is assumed to be summarised in the present state and the cost-function. Thus, COREIL has to do less book-keeping than WFIT.

[4] By convergence we mean the first stable patch in Figure 3.3 after the series of high spikes, around the 500^{th} query. The convergence point is qualitatively chosen by observing characteristics of the curve.

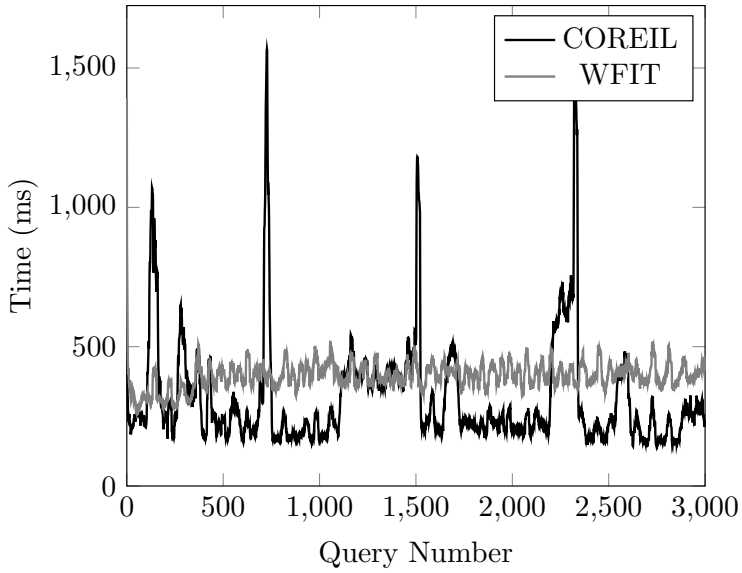


Figure 3.7: Evolution of the effectiveness (query execution time in the DBMS alone) of the two systems from the beginning of the workload (smoothed by averaging over a moving window of size 20); logarithmic y-axis

Effectiveness

To verify the effectiveness of indexes created by the tuning algorithms, we extract the cost of query execution from the total cost. Figure 3.7 (note the logarithmic y-axis) indicates that the set of indexes created by COREIL shows competitive effectiveness with that created by WFIT, though WFIT is more effective in general and exhibits less variance after convergence. Again, this is to be expected since COREIL does not have access to any cost model for the queries. As previously noted, the total running time is lower for COREIL than WFIT, as overhead rather than query execution dominates running time for both systems.

We have also performed a micro-analysis to check whether the indexes created by the algorithms are reasonable. We observe that WFIT creates more indexes with longer compound attributes, whereas COREIL is more parsimonious in creating indexes. For instance, WFIT creates a 14-attribute index as shown below.

```
[S_W_ID, S_I_ID, S_DIST_10, S_DIST_09, S_DIST_08, S_DIST_07,
S_DIST_06, S_DIST_05, S_DIST_04, S_DIST_03, S_DIST_02,
```

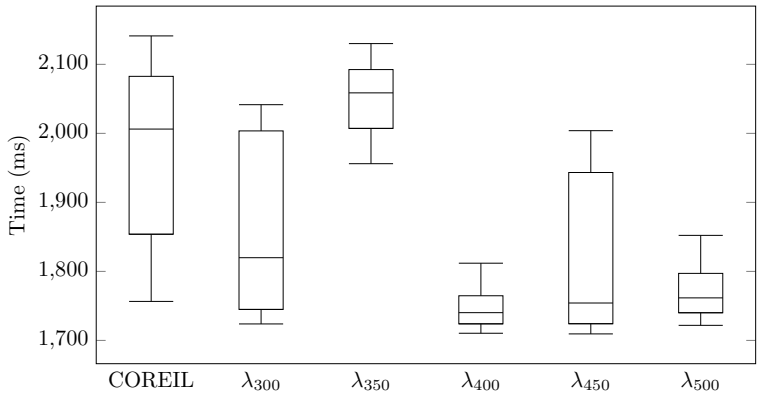


Figure 3.8: Box chart of the efficiency (total time per query) of COREIL and its improved version with different values of λ. We show in both cases the 9th and 91st percentile (whiskers), first and third quartiles (box) and median (horizontal rule).

```
S_DIST_01, S_DATA, S_QUANTITY]
```

The reason of WFIT creating such a complex index is probably due to multiple queries with the following pattern.

```
SELECT S_QUANTITY, S_DATA, S_DIST_01, S_DIST_02, S_DIST_03,
S_DIST_04, S_DIST_05, S_DIST_06, S_DIST_07, S_DIST_08,
S_DIST_09, S_DIST_10
FROM STOCK
WHERE S_I_ID = 69082 AND S_W_ID = 1;
```

In contrast, COREIL tends to create shorter compound-attribute indexes. For example, COREIL created an index `[S_I_ID, S_W_ID]` which is definitely beneficial to answer the query above and is competitive in performance compared with the one created by WFIT.

3.7.4 rCOREIL: Experiments and Results

Experimental Set-up

We run COREIL and rCOREIL, with a set of λ values $300, 350, 400, 450$, and 500. The previous set of experiments have already established competitive performance of

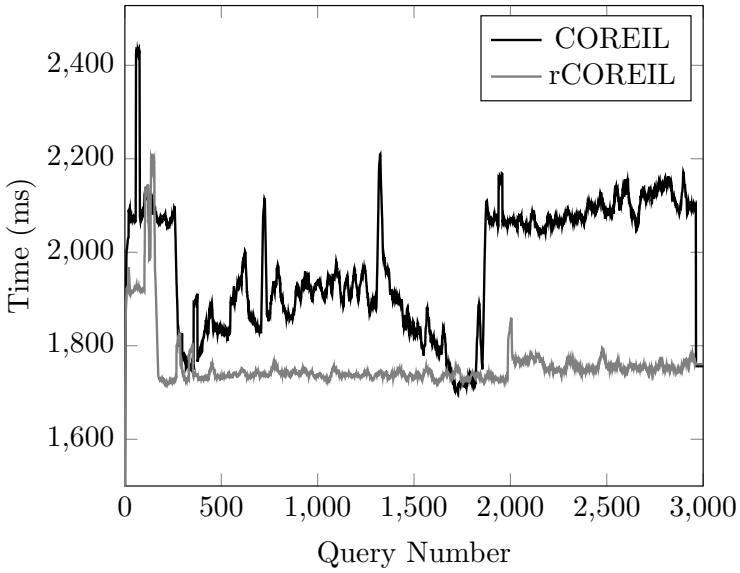


Figure 3.9: Evolution of the efficiency (total time per query) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20)

COREIL with WFIT. In this set we evaluate the basic idea of rCOREIL: providing regularised estimation of cost-model enhances the performance of COREIL and also stabilises it. We conduct all the experiments on a server running IBM DB2 10.5 with scale factor and time measure, mentioned in the previous set of experiments. But here the server is installed on a 64 bit Windows virtual box with dual-core 2-GB hard disk. It operates in an Ubuntu machine with Intel i7-2600 Quad-Core @ 3.40 GHz and 4 GB RAM. This eventually makes both version of algorithms slower in comparison to the previous physical machine installation.

Efficiency

As the offline optimal outcome for this workload is unavailable beforehand, we set an expected range of λ as $[300, 600]$ depending on the other parameters like the number of queries and the size of state space. Figure 3.8 shows efficiency of COREIL and rCOREIL with different values of λ. As promised by Algorithm 16, variations of rCOREIL are always showing lesser median and variance of total cost. We can also observe from the boxplot, the efficiency is maximum as well as the variance is minimum

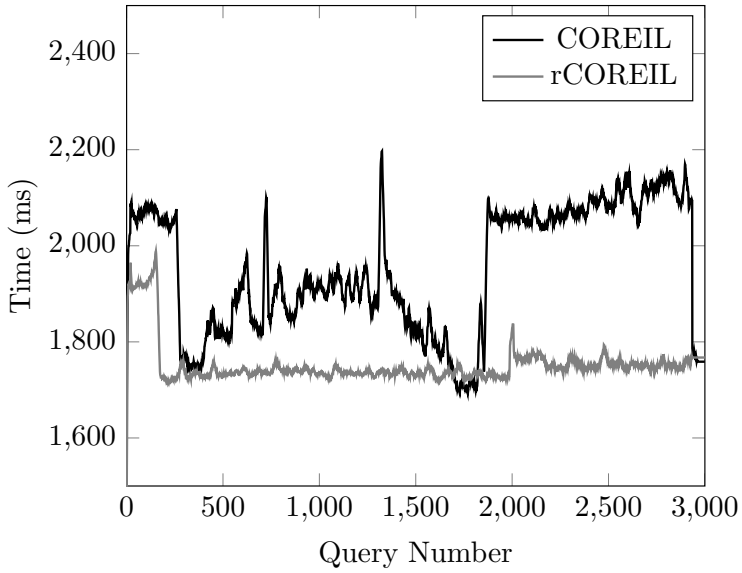


Figure 3.10: Evolution of the overhead (time of the optimisation itself) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20)

for $\lambda = 400$. As efficiency is the final measure that controls runtime performance of the algorithm, we have considered this as optimal value of λ for further analysis. This process is analogous to cross-validation of parameter λ, where the proved bounds help us to set a range of values for searching it instead of going through an arbitrary large range of values. Though here we are validating depending upon the result obtained from the whole run of 3,000 queries in the workload, the optimal λ would typically be set, in a realistic scenario, after running first 500 queries of the workload with different parameter values and then choosing the optimal one. Figure 3.9 shows that rCOREIL with $\lambda = 400$ outperforms COREIL. With respect to the whole execution set, the average processing cost of rCOREIL is 1758 ms which is significantly less than that of COREIL (1975 ms). Also the standard deviation of rCOREIL is 90ms which is half of that of COREIL, 180ms. This enhanced performance and low variance establishes the claim that if we serve the learning algorithm with a better estimation of cost-model it will improve.

Figures 3.10 and 3.11 show analysis of the overhead of corresponding tuning algorithms and cost of configuration change respectively. In this set of experiments also,

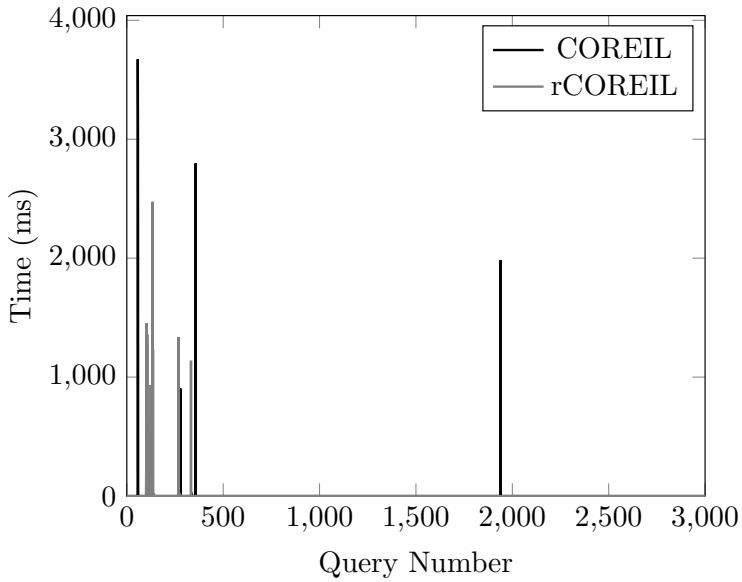


Figure 3.11: Evolution of the time taken by configuration change (index creation and destruction) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload; no configuration change happens past query #2000. All values except the vertical lines shown in the figure are zero.

we can see that the overhead of the tuning algorithms dominates their total cost. Here, the overhead of rCOREIL for each query is on an average 207ms lower than that of COREIL. This is more than 10% improvement over the average overhead of COREIL. In addition, rCOREIL (mean: 644ms) also makes cheaper configuration changes than COREIL (mean: 858ms). rCOREIL also converges faster than COREIL as the last configuration update made by rCOREIL occurs at the 335^{th}query but the last two updates for COREIL occur at the 358^{th} and 1940^{th} queries respectively. If we look closely, the 358^{th} and 1940^{th} queries in this particular experiment are:

```
SELECT COUNT(DISTINCT (S_I_ID)) AS STOCK_COUNT
FROM ORDER_LINE, STOCK
WHERE OL_W_ID = 2 AND OL_D_ID = 10 AND OL_O_ID < 3509
AND OL_O_ID >= 3509 - 20 AND S_W_ID = 2
AND S_I_ID = OL_I_ID AND S_QUANTITY < 20;
```

and

```
SELECT COUNT(DISTINCT (S_I_ID)) AS STOCK_COUNT
```

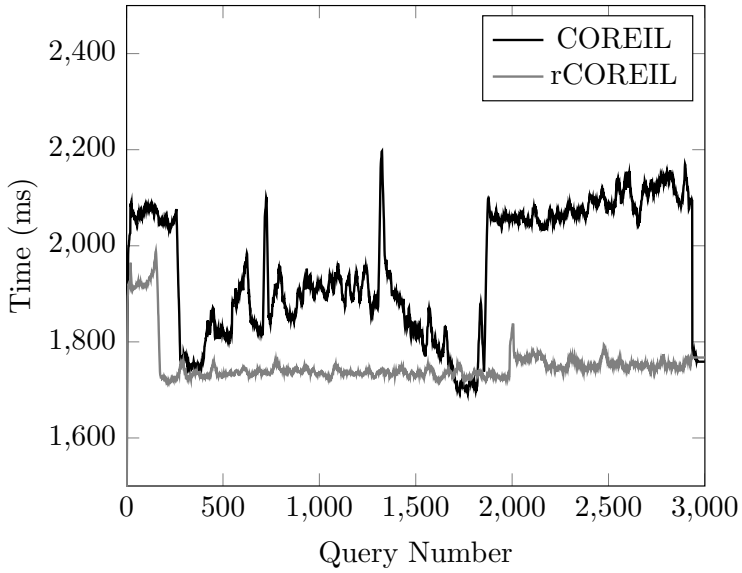


Figure 3.12: Evolution of the effectiveness (query execution time in the DBMS alone) of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20); logarithmic y-axis

```
FROM ORDER_LINE, STOCK
WHERE OL_W_ID = 1 AND OL_D_ID = 8 AND OL_O_ID < 3438
AND OL_O_ID >= 3438 - 20 AND S_W_ID = 1
AND S_I_ID = OL_I_ID AND S_QUANTITY < 11;
```

In reaction to this, COREIL creates indexes `[ORDER_LINE.OL_D_ID,ORDER_LINE.OL_W_ID]` and `[STOCK.S_W_ID, STOCK.S_QUANTITY]` respectively. It turns out that such indexes are not of much use for most other queries (only 6 out of 3000 queries benefit of one of these indexes). COREIL makes configuration updates to tune the indexes for such queries, while the regularised cost model of rCOREIL does not make configuration updates due to rare and complex events, because it regularises any big change due to such an outlier. Instead, rCOREIL has a slightly higher the overhead to find out the optimal indexes. For example, in the window consisting of 10 queries after the 359^{th} query average overhead of rCOREIL increases from 1724ms to 1748ms.

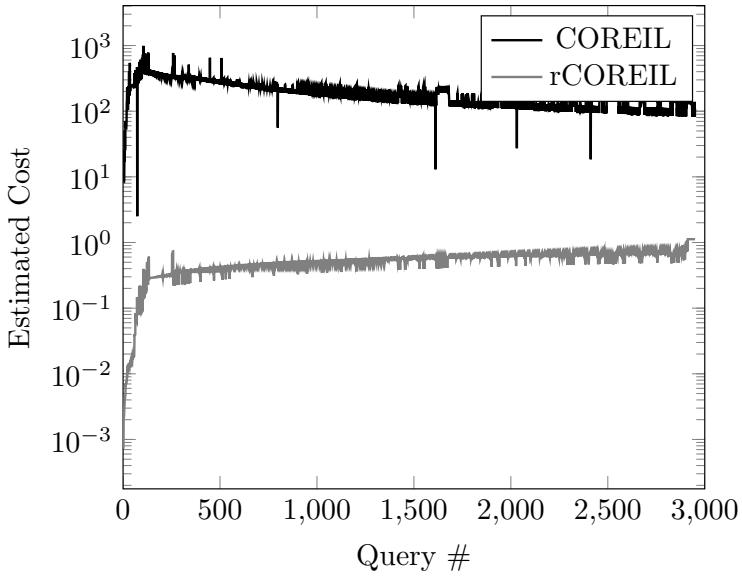


Figure 3.13: Evolution of the estimated costs of COREIL and rCOREIL with $\lambda = 400$ from the beginning of the workload (smoothed by averaging over a moving window of size 20); logarithmic y-axis

Effectiveness

Like Section 3.7.3, here also we extract the cost of query execution to verify the effectiveness of indexes created by the tuning algorithms. Figure 3.12 indicates that the set of indexes created by rCOREIL are significantly more effective than those created by COREIL. We can see the average query execution time of rCOREIL is less than that of COREIL almost by a factor of 10.

At a micro-analysis level, we observe rCOREIL creates only one index with two combined attributes, all other indexes being single-attribute. On the other hand, COREIL creates only one index with a single attribute whereas all other indexes have two attributes. This observation shows that though COREIL creates parsimonious and efficient indexes, rCOREIL shows even better specificity and effectiveness in doing so.

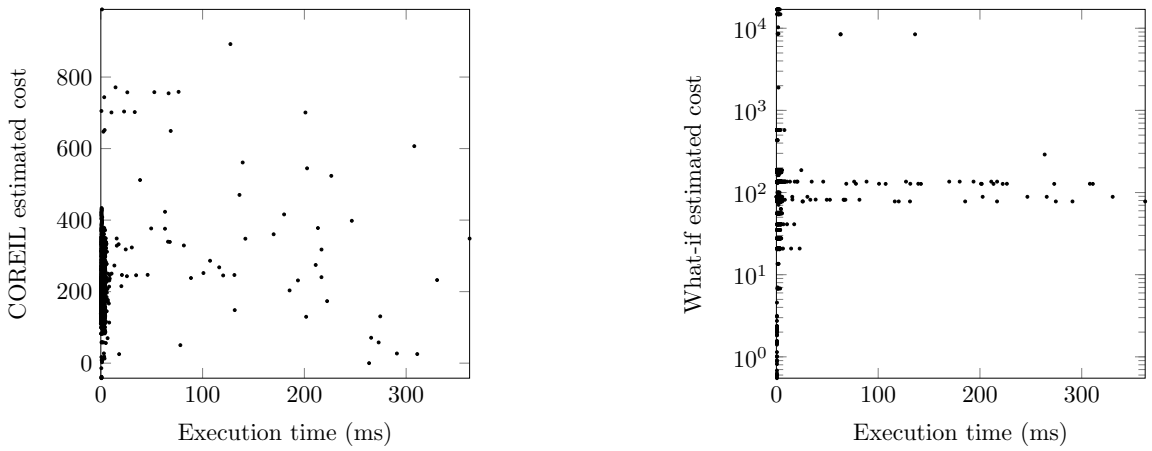


Figure 3.14: Scatter plot of the estimated cost by COREIL and the what-if optimiser vs execution time. Left shows correlation between cost estimated by COREIL and actual execution time (in ms). Right shows (on a log y-axis) correlation between the cost estimated by the what-if optimiser and the actual execution time (in ms) in the same run.

3.7.5 Analysis of Cost Estimator

In order to examine the quality of the three cost estimators used by WFIT, COREIL, and rCOREIL to predict the actual cost of query executions or configuration updates, we observe the actual execution time, the estimated cost, and that returned by the what-if optimiser during every run of experiments for COREIL and rCOREIL, respectively. The scatter plot of Figure 3.14 shows that the what-if cost has significantly less correlation (0.013) with the actual execution time than COREIL (0.1539) Again, the scatter plot of Figure 3.15 shows the regularised cost estimated by rCOREIL has significantly higher positive correlation (0.1558) than that predicted by the what-if optimiser. This proves that the execution time estimated by COREIL and rCOREIL are significantly more reliable than the ones estimated by what-if optimiser. It can also been observed that rCOREIL provides better estimations: visually, there are many more points at the middle of Figure 3.15 (left) with positive inclination.

Finally, Figure 3.13 shows that the regularised cost model estimator of rCOREIL gives a more stable estimation of the cost model than that of COREIL, as the cost

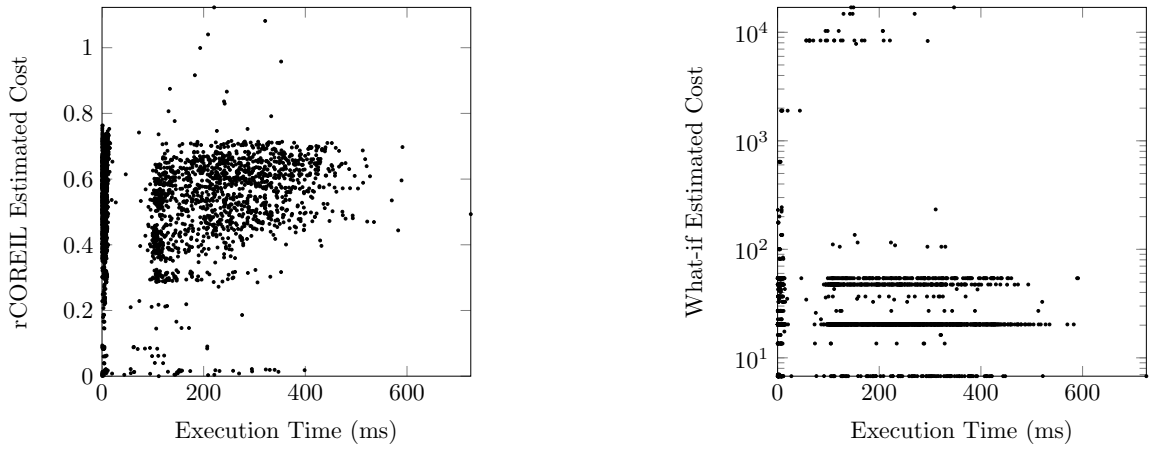


Figure 3.15: Scatter plot of the estimated cost by rCOREIL and the what-if optimiser vs execution time. Left shows correlation between cost estimated by rCOREIL and actual execution time (in ms). Right shows (on a log y-axis) correlation between the cost estimated by the what-if optimiser and the actual execution time (in ms) in the same run.

model estimated by COREIL (averaged over 20 queries) shows higher variance and also sensitivity to changes in types of queries.

3.8 Conclusion

Cost-model oblivious adaptive database tuning. We have presented a cost-model oblivious solution to the problem of performance tuning. We first formalised the problem as a Markov decision process. Then we devised and presented a solution, which addresses both issues of the curse of dimensionality and of over-fitting. We instantiated the problem to the case of index tuning. For this case, we implemented and evaluated the COREIL and rCOREIL algorithms, with and without regularisation, respectively. Experiments show competitive performance with respect to the state-of-the-art WFIT algorithm, despite our approach being cost-model oblivious. We also show that as our cost-model estimation becomes crisp and stable the performance of learner improves significantly. Beyond the material presented in this paper, we continue studying the universality and robustness of the COREIL and rCOREIL approaches.

Specially for rCOREIL, it is an interesting problem to determine the optimal regularisation parameter on the go or to adapt it with the dynamics of workload. Though now this process causes us only a one-time up-front cost, following the flavour of our approach we would like to perform it online. One possible method is to run COREIL for the first 500 queries and to calculate the costs for different set of regularisation parameter values simultaneously for that period. Following that, we can choose the parameter value that causes minimum average estimation of the cost function.

We are now running further empirical performance evaluation tests with other datasets such as TPC-E, TPC-H and dedicated benchmarks for online index tuning [Jimenez et al., 2011]. For completeness from an engineering perspective, we are considering concurrent access, which was ignored in the algorithm and experiments presented in this paper for the sake of simplicity. We are also going to look at the favourable case of predictable workload such as periodic transactions. Furthermore, we are extending the solution to other aspects of database configuration, including partitioning and replication. For each of these aspects, we need to devise specific and non-trivial heuristics that help curb the combinatorial explosion of the configuration space as well as specific intelligent initialisation techniques.

Finally, note that a critical assumption in our approach is that queries arrive sequentially and that nothing is known ahead of time about the workload. Both assumptions do not held in a number of realistic settings: queries can be submitted concurrently to the database, and a workload may often be predictable, such as when it consists of similar transactions, repeated on different data items.

Cost-model oblivious online algorithm for MDPs. Designing COREIL and rCOREIL actually develops a general approach to design online algorithms to solve MDPs. In the proposed scheme, an algorithm has to use functional approximation technique for resolving curse of dimensionality, estimates a cost model to learn a function, and uses actor-critic approach to exploration-exploitation is shown in Figure 3.16. Figure 3.16 shows the arrangement of these blocks that leads to an online algorithm

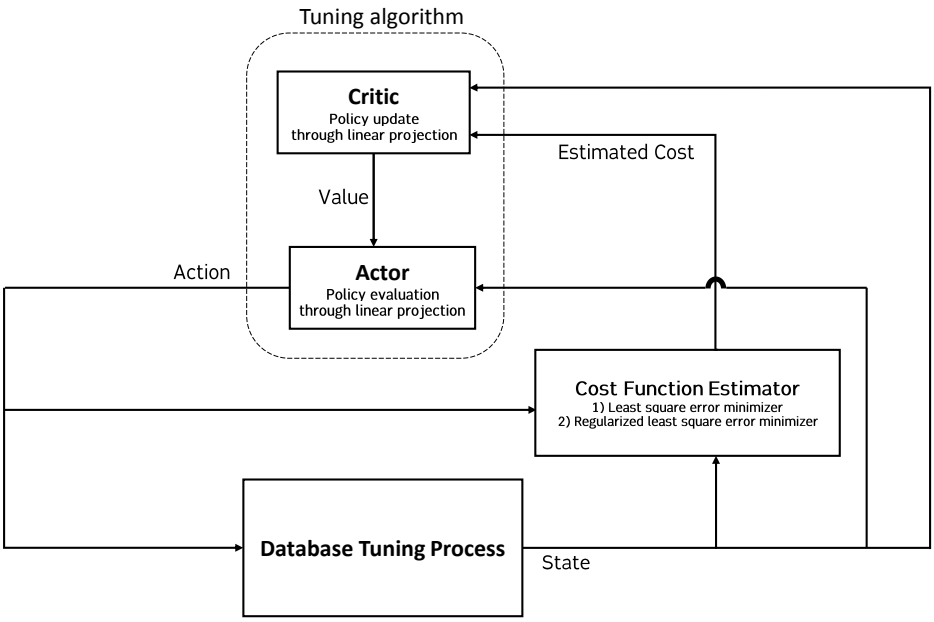


Figure 3.16: Designing online cost-model oblivious MDP solving algorithm with functional approximation technique.

with fast computational time, guaranteed convergence, and cost-model learning capability. Here, the environment is the database tuning process with unknown query distribution. We choose the LSTD for the policy evaluation block. The restricted search space approximation of Bellman equation for the policy update block. The cost function estimation uses recursive least square estimation and recursive regularised least square estimation techniques. Design of feature maps is instantiated for index tuning problems such that they take values like ± 1, or 0. Assignment of such values to different components is application motivated while the values themselves are chosen to induce sparsity during computation.

Though we have chosen such blocks according to the application that we are dealing with, and our intention to achieve theoretical guarantees of convergence, speed, and accuracy of estimation, the blocks can be varied widely. The cost function estimators can be replaced by other online regression, and functional regression blocks. Even probabilistic estimators like Gaussian process regression or online bandit optimisation

techniques can be used to estimate the cost function. In case of the policy evaluation block, online versions of several algorithms like LSTD(λ) or Batch-iFDD can be used. The policy update block can be replaced by several gradient based techniques, such as natural actor-critic or advantage actor-critic. Even following the present developments in reinforcement learning algorithms, the actor-critic block can be replaced by a deep reinforcement learning algorithm for applications with sufficient amount of offline data. We only instantiate a specific variant of this algorithm design technique for COREIL and rCOREIL that indicates that using a stable cost-estimator with reasonable regret bound, and a feature mapping with compatible convergence properties would lead to a well-behaved, fast, and converging online algorithm for cost-model oblivious MDPs.

Chapter 4

Learning with Unknown Transitions: Live Migration of Virtual Machines

Life is pleasant. Death is peaceful. It's the transition that's troublesome.

-Isaac Asimov[1]

In this chapter, we discuss design of algorithms using functional approximation methods for Markov decision processes with unknown transition functions, and develop it in the context of energy- and performance-efficient live migration of virtual machines in Clouds.

Cloud providers leverage live migration of virtual machines [Clark et al., 2005] to reduce energy consumption and allocate resources efficiently in data centers. Each migration decision depends on three questions. The questions are *when* to move a virtual machine, *which* virtual machine to move, and *where*, i.e. to which physical host to move it? Dynamic, uncertain, and heterogeneous workloads running on virtual machines make such decisions difficult. Knowledge-based and heuristics-based algorithms are commonly used to tackle this problem. Knowledge-based algorithms, such as MaxWeight scheduling algorithms [Maguluri et al., 2012], are dependent on the specifics and the dynamics of the targeted Cloud architectures and applications.

[1]Quoted in "Digital video transition analysis and detection" by Wei Jyh Heng and King N. Ngan, 2002.

Heuristics-based algorithms, such as MMT algorithms [Beloglazov and Buyya, 2012; Beloglazov et al., 2012], suffer from high variance and poor convergence because of their greedy approach. We propose an online reinforcement learning algorithm called Megh [Basu et al., 2017c]. Megh does not require prior knowledge. It learns the dynamics of the workload as-it-goes. Megh models the problem of energy- and performance-efficient resource management during live migration as a Markov decision process and solves it. While several learning algorithms are proposed to solve this problem, these algorithms remain confined to the academic realm as they face the curse of dimensionality. They are either not scalable in real-time, as it is the case of MadVM [Han et al., 2016], or need an elaborate offline training, as it is the case of Q-learning. Megh overcomes these deficiencies. Megh uses a novel dimensionality reduction scheme to project the combinatorially explosive state-action space to a polynomial dimensional space with a sparse basis. Megh has the capacity to learn uncertain dynamics and the ability to work in real-time. Megh is both scalable and robust. We implement Megh using the CloudSim toolkit [Calheiros et al., 2011] and empirically evaluate its performance with the PlanetLab [Park and Pai, 2006], and the Google Cluster [Reiss et al., 2011] workloads. Experiments validate that Megh is more cost-effective, incurs smaller execution overhead and is more scalable than MadVM and MMT. An empirical sensitivity analysis explicates the choice of parameters in experiments.

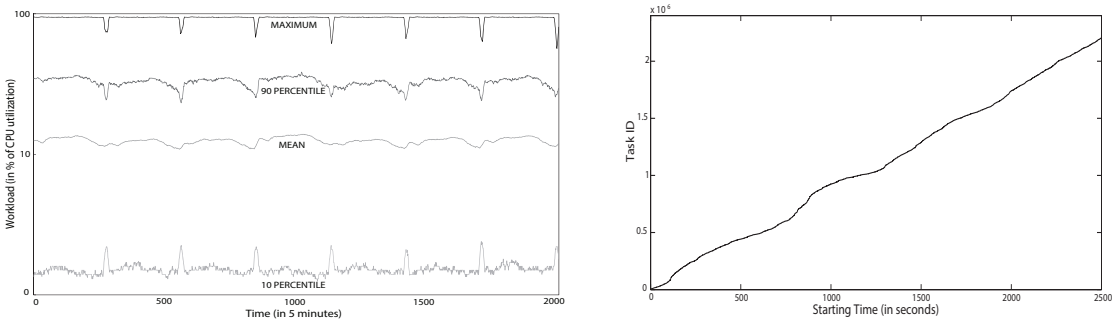
4.1 Introduction

Infrastructure as a Service (IaaS) environments of Cloud computing leverage virtualisation technology [Barham et al., 2003] to provide a shared platform of resources accessible at any time and from anywhere through the Internet. Cloud providers allocate Virtual Machine instances (VM) on a cluster of Physical Machines (PM). VMs allow users to share physical resources concurrently. Therefore, VMs enhance utilisation of resources and increase return on investment for Cloud providers.

Making such an optimal allocation of resources is challenging not only in general-purpose IaaS Clouds [Li et al., 2017] but also in Clouds with specialised features like scientific computing [Iosup et al., 2011] or online transaction. A large number of users accessing the Cloud, the diversity of applications, and the heterogeneity of hardware yield significant variations in performance. Furthermore, the uncertain dynamics of workloads creates abrupt and unpredictable changes in resource utilisation. Thus, dynamic allocation of VMs in Clouds is indispensable. In order to avoid disruption due to dynamic allocation, [Clark et al., 2005] and [Nelson et al., 2005] proposed the idea of a live migration scheme. During live migration, pages from the memory of the migrating VM are copied to the destination machine while it keeps on running on its present host. If properly carried out, live migration causes minimal downtime and minimal noticeable effect from the user end. Live migration raises three questions to the Cloud administrator: *which* VM to move, *where*, i.e. to which physical host to move, and *when* to move?

These resource management decisions during live migration drastically affect the energy consumption of the Cloud data centres. As energy consumption contributes almost 75% of the operation cost of a data center [Belady, 2007], from the Cloud provider side it is the most important metric for live migration. Migration events may also cause significant deterioration of the Quality of Service (QoS) promised by the Cloud providers and can violate the Service Level Agreements (SLAs) [Wieder et al., 2011]. These agreements also define monetary penalties for the Cloud providers when violated. In this work, we develop cost models for the SLA violations and the energy consumption during a live migration and aggregate them to construct an operation cost.

Energy- and performance-efficient resource management in Cloud data centres is difficult as the workloads running on the corresponding VMs are uncertain, dynamic and heterogeneous. Figures 4.1(a) and 4.1(b) reasserts this nature of the workloads in Cloud data centers. Knowledge-based and heuristics-based algorithms are applied to solve the resource management problem. Knowledge-based algorithms, such as



(a) PlanetLab Workload (b) Google Cluster workload

Figure 4.1: Dynamics of PlanetLab workloads and starting times of tasks in Google Cluster. The y-axis shows the %of CPU usage by the user and the x-axis shows time discretised in the unit of 5 minutes. Lines from up to down show maximum, 90 percentile, mean, and, 10 percentile of all the workloads at any instance.

MaxWeight scheduling algorithms [Maguluri et al., 2012], or [Tseng et al., 2017] for video streaming data centers, are oblivious to the specifics and the dynamics of Cloud architectures and applications that do not belong to their knowledge-base. Heuristics like dynamic consolidation algorithms [Beloglazov and Buyya, 2012; Beloglazov et al., 2012] do not use such specific knowledge base. They save the power by greedily accumulating a majority of VMs on a smaller number of servers. Heuristics-based algorithms improve the performance by taking cost-effective VM migration decisions from under- or over-utilised servers. These heuristics may become unstable while tackling uncertain dynamics and may make suboptimal decisions due to their myopic and greedy nature.

The shortcomings of knowledge-based and heuristics-based algorithms has motivated us to look into reinforcement learning [Sutton and Barto, 1998]. Reinforcement learning is a framework of machine learning as discussed in Chapters 1 and 2. In reinforcement learning, an agent operating in an uncertain environment tries to take optimal decisions by learning more about the dynamics of its surroundings as-it-goes. If we consider the Cloud administrator *system* as a learning agent and the user workloads operating on the Cloud with corresponding resource distribution as the uncertain environment, our problem manifests as a reinforcement learning problem. Here, the

system tries to take *optimal* live migration decisions as-it-goes by learning the dynamics of the workload and adapting accordingly. A *policy* or a sequence of decisions made by reinforcement learning is optimal if it does live migration and resource management of data center with minimum operation cost. Reinforcement learning tries and computes such an optimal policy by predicting as-it-goes the optimal decisions based on immediate costs. As the number of ways the VMs can be allocated to the hosts or PMs is combinatorially large, it creates a huge state space and also makes reinforcement learning intractable. This problem of exploding state space is called curse of dimensionality in reinforcement learning. Curse of dimensionality restricts the applicability of recently proposed learning algorithms in real-life scenarios. These algorithms are either not scalable in real-time, as it is the case of MadVM, or need an elaborate offline training, as it is the case of Q-learning. We propose an online reinforcement learning algorithm, called Megh, to solve this problem as-it-goes. Megh projects the state space into a smaller vector space and learns the dynamics of the workloads without assuming any model or prior knowledge. Megh is a robust algorithm to learn the uncertainty and diversity of workloads as-it-goes. At each step, the sparsity of the projected space is leveraged to act effectively without creating any significant overhead in the course of live migration. The data structure exploiting this sparsity makes Megh time-efficient and therefore, a contending real-time solution for energy- and performance-efficient live migration.

We evaluate the performance of Megh by simulating it using the CloudSim toolkit [Calheiros et al., 2011] over workload data extracted from PlanetLab [Park and Pai, 2006] and Google Cluster [Reiss et al., 2011]. We compare Megh with state-of-the-art dynamic consolidation based Minimum Migration Time (MMT) algorithms: THR-MMT, IQR-MMT, MAD-MMT, LR-MMT, and LRR-MMT [Beloglazov et al., 2012; Beloglazov and Buyya, 2012]. We also test the performance of Megh against MadVM [Han et al., 2016], which is the most recent reinforcement learning based algorithm for dynamic resource management in a data center. Experiments prove the efficiency of Megh as it significantly reduces the total operation cost and the number of VM mi-

grations occurring over a period of time with respect to the competing algorithms. Unlike MadVM suffering from the curse of dimensionality, Megh takes significantly smaller execution time than MMT heuristics even for large data center configurations. The results validate the robustness, efficiency and real-time execution of Megh to cost-effectively decide live migrations under uncertain workload dynamics. A comparative scalability analysis also demonstrates Megh's better scalability than THR-MMT. A sensitivity analysis empirically explicate our choices of parameters controlling the exploration-exploitation trade-off of Megh.

Our Contribution. We propose an online reinforcement learning algorithm, Megh, to solve the problem of energy- and performance-efficient live VM migration where the workload dynamics are not known a priori. We develop a sparse projection scheme that approximates the value function uniquely (Theorem 1). While the projection scheme reduces the complexity of Megh and practically resolves the curse of dimensionality, Megh asymptotically converges to the optimal policy (Theorem 2). The projection scheme and the proposed online transition operator update induce two significant improvements in Megh's performance. Firstly, Megh is oblivious to the training phase. Megh learns the workload dynamics on-the-go while optimizing the decisions simultaneously. Secondly, each iteration of Megh incurs small execution time proportional to the number of VM migrations happening at that iteration. We experimentally verify these outcomes for the system and the cost model discussed in Section 4.3, and the workload traces from PlanetLab and Google Cluster. Comparative performance evaluation validates that Megh reduces 14% and 8% operational cost with respect to THR-MMT and MadVM (in testing phase) respectively, while Megh incurs 95% and 25% execution time in comparison with THR-MMT and MadVM.

Structure of the Chapter. The rest of this chapter is organised as follows. In Section 4.2, we review the related work. In Section 4.3, we depict the system model and build up the mathematical formulation to calculate costs of energy consumption and SLA violation. We introduce the problem of cost-optimal live migration as a reinforce-

ment learning problem and formulate it mathematically in Section 4.4. Following that in Section 4.5, we propose an algorithm Megh to solve it in real-time. In Section 4.6, we elaborate the detailed experimental set-up and also evaluate the performance of Megh. We discuss the future research directions and conclude the paper in Section 4.7.

4.2 Literature Review and Contextualisation

While Megh tries to perform *energy and performance efficient live VM migrations for resource management*, the form of the problem it solves and the way it solves are based on *reinforcement learning*. Here, we review the related works in these two areas.

4.2.1 Dynamic VM Consolidation

A profitable strategy for Cloud vendors is the dynamic consolidation of underutilised virtual machines to fewer physical servers to save hardware, to reduce energy consumption [Nathuji and Schwan, 2007], and to eliminate hotspots [Wood et al., 2007]. Due to the dynamic nature of Cloud workloads, there have been many studies in the field to investigate an optimal dynamic VM provisioning plan. One key requirement of dynamic VM consolidation is to pack VMs tightly while preserving SLAs. [Mann, 2015] recently presented an extensive survey of the problem models and optimisation algorithms. [Wang et al., 2011] consider the dynamic network bandwidth demand for real workloads and model the VM consolidation into a Stochastic Bin Packing problem. [Song et al., 2014] similarly applied a variant of the relaxed on-line bin packing model, which was shown to work well on a small-scale cluster. [Maguluri et al., 2012] further modelled VM consolidation using a stochastic model where jobs arrive according to a stochastic process, and described MaxWeight algorithms, a family of frame-based non-preemptive VM configuration policies to improve overall throughput. Compared to existing models and algorithms, Megh makes no *a priori* assumption on the workload arriving pattern or load distribution, which may be adapted to various scenarios

while requiring a small number of migration requests and thus having little impact on running workloads.

In the existing literature, the Minimum Migration Time (MMT) family of algorithms [Beloglazov et al., 2012; Beloglazov and Buyya, 2012] function without any assumption on the workload model like Megh and perform in real-time. Due to this general structure and online mode of operation, we have compared Megh's performance with them. These algorithms are heuristics designed for energy and performance efficient dynamic consolidation of VMs in Clouds. They start migrating a VM when its utilisation crosses a certain threshold. The threshold can be fixed (for THR-MMT) or determined adaptively (for IQR-MMT, MAD-MMT, LR-MMT and LRR-MMT) from the summary statistics of workloads' history. The VM is migrated to a different host such that the migration time is minimum. These methods are greedy heuristics that suffer from high variation and instability like other heuristic-based algorithms, while Megh, being a learning algorithm, does not.

4.2.2 Reinforcement Learning Algorithms for VM Migration

Reinforcement learning [Sutton and Barto, 1998] is a framework of machine learning. In reinforcement learning, an agent aims at taking optimal decisions by developing an understanding of the constantly evolving environment around it. As mentioned in 2.2, *Markov decision process* (MDP) [Puterman, 2009] is a formulation for modelling and solving reinforcement learning problems. MDPs assume that it is sufficient to remember the present state of the system to decide the next decision or action, while rewards of state-action pairs carry the relevant information of system's history. The agent tries to fix a policy or a sequence of decisions that will maximise the cumulative sum of rewards acquired.

[Farahnakian et al., 2014] and [Masoumzadeh and Hlavacs, 2013] apply Q-learning algorithm [Watkins and Dayan, 1992] for energy–efficient resource management in Clouds. [Rao et al., 2009] uses it for automatic reconfiguration of resource sharing VMs. Q-learning is an offline algorithm. We have to go through computationally expensive

training periods of a few hundred iterations before using it in an online setup like the one addressed. But there is no reliable guarantee on the optimality of Q-learning for online learning setup for any approximated value function [Baird et al., 1995]. The general efficient VM migration problem may consist of cases where the algorithm encounters a significant variance in the real-life workload than the training one due to change in user base or their applications. Under such conditions, Q-learning has a high probability to break down or perform sub-optimally.We have done a comparative performance analysis with respect to Q-learning. We omit an elaborate description of that in this article due to Q-learning's dependence on offline training and presence of a recent, on-line approach called MadVM.

MadVM [Han et al., 2016] models the energy-efficient dynamic resource management of VMs as an approximate MDP. This algorithm assumes no prior knowledge of workload and uses value iteration [Bellman, 1957b] algorithm to solve the problem. At each step, MadVM tries to select decisions that simultaneously maximise the expected cumulative rewards of each of the VMs. This algorithm is indirect as it does not try to optimise directly over a policy space but rather rely exclusively on value function approximation, that hopefully returns a near-optimal policy. Due to the combinatorially large state space of the problem, MadVM also faces the curse of dimensionality of reinforcement learning approaches. This leads to a key state selection procedure to connect the policy space and the value functions. This procedure for dimensionality reduction, however, is computationally expensive. MadVM tries to simultaneously optimise the utility functions of each of the VMs. Simultaneous optimisation requires bookkeeping of transition functions and evaluation of key states for each of them. This computational burden makes MadVM poorly scalable for real-time applications.

Furthermore, MadVM is a critic-only algorithm whereas Q-learning is an actor-only algorithm. Actor-only algorithms suffer from high variance due to its sensitivity to the estimates of the gradient. Critic-only algorithms are stable but usually needs a discretised version of the state-action space. Discretisation may lead to suboptimal results. Megh relies on the *actor-critic* [Grondman et al., 2012a] approach of algorithm design.

The actor tries to evaluate the policy as an incremental functional approximation problem. The critic leverages this estimated policy for approximating and updating value function using samples collected as-it-goes. This feedback ensures better convergence property and stability. In the proposed approach, we use such an off-policy actor-critic variant of least-square policy iteration (LSPI) algorithm [Lagoudakis and Parr, 2003b] as the skeleton. We utilise the projection based dimensionality reduction techniques and sparsity-based improved data structures described in Section 4.5 to construct our real-time learner Megh.

4.3 A Cloud Data Centre: System and Cost Models

In the following subsections, we describe the system model of a data centre used by Megh. Following that, we formulate cost models for energy consumption and SLA violation.

4.3.1 System Model

In IaaS environments Cloud providers serve the users with virtualised computing resources over the Internet. In order to model such a system, we consider a data centre consisting of M heterogeneous physical machines (PMs) or hosts. Each of these PMs is characterised on the basis of the number of CPUs, the number of cores, the amount of RAM and the network bandwidth. Here, the performance of a CPU is defined in Millions Instructions Per Second (MIPS). In our paper, we consider the CPUs belonging to the same PM as a single-core CPU with the cumulative MIPS performance of all the cores. Independent users submit requests for provisioning of computing resources to the Cloud. In turn, N users are assigned to N heterogeneous VMs hosted by M PMs. Each of the VMs is allocated CPU performance, memory size, RAM, and network bandwidth as per the users' requirements. We assume no *a priori* knowledge of the applications, workload dynamics and the provisioning time of VMs. This allows

us to deal with both general-purpose and specialised setting of mixed workloads with uncertain dynamics that utilise the resources of a PM concurrently.

The proposed reinforcement learning algorithm, Megh, is implemented as a part of the global resource manager of the Cloud. This global manager acts as an interface between users' workloads and requirements, and the virtualisation layer. The Virtual Machine Managers (VMMs) operating at each of the physical nodes act as the continuous monitoring systems. They send the workload dynamics of each VM and the resources utilised by them to the global manager. The global manager acts as the learning agent in Megh. The global manager accumulates the information and allocates the resources such that the energy consumption as well as the SLA violation will be minimised. Following this, the decision is sent to VMMs as a resource map. VMs are migrated and consolidated accordingly. Megh may migrate the VMs allocated in an underloaded PM to another PM with potential capacity and put the first PM down to sleep. Similarly, if a PM gets overloaded, some of the VMs operating on it are migrated to another PM such that the expenditure for energy consumption and SLA violation remains minimal.

Following previous works on energy–efficient live migration of Clouds [Beloglazov and Buyya, 2012; Beloglazov et al., 2012; Han et al., 2016], we consider CPU utilisation data as the key metric of characterising the workloads. We are aware of the importance of bandwidth and memory as resources, and research works [Lago et al., 2017; Yu et al., 2017] accounting available bandwidth and network traffic as principal decision variables for VM migration. One can build cost models for these resources and add them as additional modules in the cost calculation without modifying Megh algorithmically.

4.3.2 Energy Consumption Cost

Energy consumption cost of the Cloud data center can be considered as a function of time $C_p : \mathbb{N} \to \mathbb{R}^{\geq 0}$, such that

$$C_{\mathrm{p}}(t) = c_{\mathrm{p}} \int_0^t P(\tau) d\tau, \qquad \forall t \geq 0. \tag{4.1}$$

Here, $c_{\mathrm{p}} \in \mathbb{R}^+$ denotes the cost of consuming 1 Watt of power for 1 second. It is a fixed constant according to the place where the data center is built up, whereas $P : \mathbb{N} \to \mathbb{R}^{\geq 0}$ is the function representing the amount of power (in Watts) consumed by the data center at time τ (in seconds). This function does not only depend on the workload dynamics of VMs but also on the CPU performance, memory size, disk storage and cooling system of the PMs installed in the data center [Minas and Ellison, 2009]. Following the works by [Beloglazov et al., 2012], we leverage the power consumption data provided by the SPECpower_ssj® 2008 benchmark [Huppler et al., 2012; SPECpower Committee, 2014] rather than moving our focus to precisely modelling $P(\theta)$. This is a certified industry-standard benchmark to evaluate the power and performance characteristics of server-class computer equipments. SPECpower_ssj® 2008 is tested on a wide variety of operating systems and hardware architectures to remove extensive dependence on data center infrastructure for power–performance characteristics calculations. This benchmark [SPECpower Committee, 2014] provides energy consumption level y for a collection of servers with different CPU architectures under a workload of $x\%$ working on its CPU, as shown later in Table 4.1. Now, if we assume that the Cloud management system extracts the workload dynamics at a certain time granularity, say $\tau > 0$, we can model the cost of energy consumption up to time t as

$$C_{\mathrm{p}}(T) = c_{\mathrm{p}} \sum_{k=0}^{T} \sum_{i=1}^{M} y_i(k\tau)\tau, \quad \forall \;\; T \geq 0 \tag{4.2}$$

where, $T \triangleq \lceil \frac{t}{\tau} \rceil$ represents the discretised version of time t, $y_i(k\tau) \in \mathbb{R}^{\geq 0}$ is the power consumed by the i^{th} PM at time $k\tau \in (0, t)$, and M denotes the total number of PMs operating in the data center.

4.3.3 SLA Violation Cost

Though energy consumption covers the major part of the Cloud provider's expenditure, Quality of Service (QoS) provided by the Cloud is a concern from the user's side. Specifically, QoS is negotiated using a legal agreement between the user and the Cloud provider, called Service Level Agreement (SLA). SLAs provided by companies like Amazon, Microsoft and Google confirm that service providers promise to pay users certain monetary penalties if the QoS degrades below certain levels. We also observe that QoS is defined as the uptime percentage of the user. *Uptime* is the percentage of total access time for which the user can utilise the Cloud services without any interruption. *Downtime* is the percentage of total access time for which the user cannot utilise the Cloud services due to the interruption. Some of the Cloud providers do not consider any continuous downtime below 5 minutes as a degradation of QoS to provide the system privilege. In this paper, we consider the exact downtime without such bias. Thus, *SLA violation cost* at time t for a Cloud with M PMs and N VMs can be expressed as,

$$C_{\mathrm{v}}(t) = \sum_{j=1}^{N} c_{\mathrm{v}}^{j}(t), \qquad \forall t \geq 0 \tag{4.3}$$

Here, $c_{\mathrm{v}}^{j}(t)$ is the SLA violation cost for VM j at time t. We define $c_{\mathrm{v}}^{j}(t)$ as

$$c_{\mathrm{v}}^{j}(t) = \begin{cases} cv_1, & \text{if user's downtime percentage up to } t \in (0.05\%, 0.10\%] \\ cv_2, & \text{if user's downtime percentage up to } t > 0.10\% \\ 0, & \text{otherwise} \end{cases}$$

The system model considers each VM is used to virtually assign computing resources to each of the users. Thus, the user would be paid a certain penalty if the service is down for more than certain threshold of total usage.

As we allocate and manage the resources by migrating the VMs from one machine to another, we face two scenarios of QoS degradation. In the first case, when one or multiple VMs are allocated to a PM, it faces a sudden rise of workload. The PM gets overloaded. Overloading occurs when VMs try to use more resources than the capacity of the host PM. Overloading provokes migration of VMs from that host to another. Due to discretised time of observations by the global learning agent, and the inherent delay of the host system to react and adapt to the scenario, some time is lost before the migration decision is made and executed. During this period, the VMs working on that host remain suspended or their performance degrades substantially. This phenomenon introduces a downtime in each of the VMs working on that host. This window of time is termed as the overloading time. In this paper, we denote *overloading time* of host PM i at time t as $T_{\mathrm{o}_{it}}$. $T_{\mathrm{o}_{it}}$ represents the total time during which the host i has experienced the utilisation of greater than $\beta\%$ leading to overloading. The active time $T_{\mathrm{a}_{it}}$ of the PM i is defined as the total time for which it is serving the users. Thus, we define the *percentage of overloading time* as the fraction of the active time for which the host is overloaded, i.e.

$$O^i(t) \triangleq \frac{T_{\mathrm{o}_{it}}}{T_{\mathrm{a}_{it}}}. \tag{4.4}$$

In the second case, the downtime is caused by the live migration process itself. Though the live migration transfers a VM from a host PM to another destination PM without suspending the running application, it still causes a downtime. The *migration time* is defined as the time required to copy all the pages of a VM from its present host memory to the destination memory for a given network bandwidth. M_{jt} denotes the amount of memory used by VM j right before initiating the migration at time t. B_{jt} denotes the available bandwidth of the network. The expected migration time of

VM j is defined as

$$TM_{jt} \triangleq \frac{M_{jt}}{B_{jt}}.$$

Thus, the downtime of VM j during live migration is estimated as the time for which its estimated CPU utilisation $\widehat{u_j}(t)$ will be less than a certain threshold. This threshold is introduced as a given $\alpha\% > 0$ of the workload $u_j(t)$ that is demanded from the VM by the user. Thus, we estimate the live migration downtime of VM j at time t as

$$T_{\mathrm{d}_{jt}} \triangleq \int_{t}^{t+TM_{jt}} \mathbb{1}\left(\widehat{u_j}(\tau) < \alpha u_j(\tau)\right) d\tau,$$

where $u_j(t)$ is the CPU utilisation by VM j at that time t. $\mathbb{1}$ is the indicator function defined as

$$\mathbb{1}\left(\widehat{u_j}(t) < \alpha u_j(t)\right) \triangleq \begin{cases} 1, & \widehat{u_j}(t) < \alpha u_j(t) \\ 0, & \text{otherwise} \end{cases}, \qquad \forall t \geq 0.$$

If $T_{\mathrm{r}_{jt}}$ is the total active time requested by the VM j till the time t, we estimate the *percentage of live migration downtime* of VM j as the ratio of the estimated migration time of VM j and the total active time.

$$D^j(t) \triangleq \frac{T_{\mathrm{d}_{jt}}}{T_{\mathrm{r}_{jt}}}. \tag{4.5}$$

Thus, the total downtime percentage for VM j up to time t is defined as the sum of its downtime due to live migration and the overloading time of the PMs, which got overloaded while the VM was operating on it. Equations (4.4) and (4.5) provide us a mathematical model to calculate the SLA violation cost for each of the VMs. Though we develop and use the aforementioned cost model for SLA violation, it can be replaced with other cost models considering varying market prices and various subtle factors [Alsarhan et al., 2018] without further modifying Megh.

4.4 Live Virtual Machine Migration as a Learning Problem

In this section, we formulate the problem of energy– and performance–efficient resource management during live migration of VMs as a reinforcement learning problem.

Let us consider a Cloud data center with M PMs. Each of the PMs has homogeneous CPU capacity h. Each of the VMs is assigned to each of the users on the basis of their requests. Thus, the maximum number of users that the Cloud can handle is the maximum number of VMs it can allocate. Though the workloads and requirements of users may differ, the maximum CPU capacity that can be allocated to a VM is a constant, say v. Under the worst case scenario, when each of the VMs will ask for maximum CPU capacity, the maximum number of VMs n that can be allocated to a single PM is $\lfloor \frac{h}{v} \rfloor$. Furthermore, the total number of VMs N that can be allocated to the data center at any instance is Mn. The VMs are accessed by a large volume of users with diverse requirements and applications, and the dynamics of these workloads are also uncertain. This may cause a sudden change in workloads of one or multiple VMs and consequently overloading of hosts. Then one of the VMs working on the overloaded host has to be migrated to another destination PM such that cost for energy consumption and SLA violation remains minimal. While doing so the system has to decide which VM to move to which destination host and when to start moving, so that the penalty will be minimum ensuring maximum profit of Cloud provider and also maximum QoS for users.

[Dertouzos and Mok, 1989] proves that optimal scheduling of tasks in a multiprocessor system is impossible in the absence of any prior knowledge of the deadline and the request distribution. [Sha et al., 2004] states that resource allocation among even soft real-time tasks under fully stochastic environment is analytically intractable. Thus, online allocation of tasks in a data center with unknown job request distribution and unknown job durations is intractable, and learning the stochastic nature of workload is essential for taking optimal decisions.

We model the process of live migration with uncertain workloads as a Markov Decision Process [Puterman, 2009]. In this model, the *state space* S is Cartesian product of the set of all configurations of the VMs on the PMs $\mathcal{C}$, and the workloads $W(t)$ operating on the VMs at time t. At a given time t, $W(t)$ is a real-valued vector with N components, where each component represents the CPU usage of a VM at that instance. Since $W(t)$ varies continuously and stochastically, it makes the state space infinite dimensional and introduces uncertainty in state transitions. The *action space* A corresponds to migration of any of the VMs from one PM to another depending on the operating workloads. Each action is represented by a pair (j, k), where j is the migrating VM, and k is the destination PM. In order to capture the uncertainty of workloads, we define *transition function* $f : S \times A \to \mathcal{P}(S)$, where $\mathcal{P}$ is a probability measure over state space. Given the present state and an action, f returns the probability to reach another state. But in our problem, it is not known *a priori* and has to be learned. *The cost* of changing a configuration s_{t-1} of VMs to another configuration s_t is given by

$$C(s_{t-1}, s_t) = \triangle C_p(s_{t-1}, s_t) + \triangle C_v(s_{t-1}, s_t) \quad \forall \ \ t \in \{\tau, 2\tau, \ldots\}. \tag{4.6}$$

$\triangle C_p(s_{t-1}, s_t)$ and $\triangle C_v(s_{t-1}, s_t)$ are the costs of energy consumption and SLA violation in the interval $(t-1, t]$. Here, C_p and C_v are defined by Equations (4.2) and (4.3) respectively. We observe $\triangle C_p(s_{t-1}, s_t)$ is always positive as the system will always consume some energy whether any migration happens or not. We also observe that $\triangle C_v(s_{t-1}, s_t) \geq 0$. The equality holds if and only if there is no SLA violation in that interval.

This formulation reduces the problem to finding the sequence of configurations that minimises the sum of future per-stage costs. Unlike MadVM that assumes an average cost structure and computationally considers the effect of a migration is limited to a fixed future time horizon, we assume an infinite horizon formulation of MDP [Sutton and Barto, 1998]. Infinite horizon means an action will affect all the future states

and actions of the system. This formulation makes the cumulative sum of future per-stage costs infinite. In order to circumvent this problem a *discount factor* $\gamma \in [0, 1)$ is introduced. Mathematically, γ makes the cumulative sum of per-stage costs convergent. Physically, γ let the effect of a past action decay with each passing instance. The discount factor inclines the system to give more importance to immediate costs than to costs distant in the future, which follows a practical intuition. Now, the problem translates into finding the sequence of configurations that minimises a *discounted cumulative cost*. Under Markov assumption, a configuration change depends on its present state only. Given the current configuration and workloads, i.e the current state s_t, a policy $\pi\colon S \rightarrow A$ determines the next decision a_t. We define the *cost-to-go* function V^π for a policy π as

$$V^\pi(s) \triangleq \mathbb{E}_f \left[\sum_{t=1}^{\infty} \gamma^{t-1} C(s_{t-1}, s_t) \right] \tag{4.7}$$

such that the initial state $s_0 = s$, and s_t is the state reached from state s_{t-1} through an action $\pi(s_{t-1})$). The value of $V^\pi(s)$ represents the expected cumulative cost for following the policy π from the current configuration s. Thus, $V^\pi(s)$ allows us to optimise the long-term effect of migration decisions, unlike greedy MMT algorithms that try to minimise the present cost only. Let Π be the set of all policies for the given set of VMs on the cluster of PMs. Now, the problem can be phrased as computing an optimal policy π^* that minimises the expected cumulative cost.

$$\pi^* \triangleq \underset{\pi \in \Pi}{\arg\min}\ V^\pi(s_0) \tag{4.8}$$

4.5 Megh: Learn to Migrate As-you-go

Depending on the cost model developed in Section 4.3 and the problem formulation in Section 4.4, we propose an online actor-critic algorithm, Megh. Megh answers three basic questions of the VM migration problem: *when* to start migrating the VM, *which* VM to migrate, and *where* i.e, to which PM to migrate it.

Megh answers these questions by solving the minimisation problem of Equation (4.8). This equation shows that optimal decision making is analogous to computing the optimal function π^* that minimises the cost-to-go function. We can perceive this as a sequential functional approximation problem over the space Π. In order to do so, we begin with an initial guess of the policy π_0. Following that, we gain more information about the configuration of VMs and also the dynamics of workloads running on them. We use this information to improve the policy such that it keeps the current estimation of cost-to-go function minimum. In reinforcement learning literature, this strategy is known as *policy iteration* [Sutton and Barto, 1998] (Algoirthm 7).

If transition function f i.e, the stochastic nature of workload and its effect on migration, is known *a priori*, we can apply Bellman's dynamic programming technique [Bellman and Kalaba, 1965] to update the estimate of cost-to-go function at every time t. The update equation is known as Bellman equation. In this problem, we express is as

$$V^{\overline{\pi}_t}(s) = \mathbb{E}_f \left[C(s, s') + \gamma V^{\overline{\pi}_{t-1}}(s') \right]. \tag{4.9}$$

Thus, the updated policy would be $\pi_t = \arg\min_{\overline{\pi}_t \in \Pi} V^{\overline{\pi}_t}(s)$. The algorithm terminates when there is no or very small change in the policy. Policy iteration has strong optimality and convergence properties [Powell, 2007].

In the VM migration problem policy iteration suffers from two main issues. Firstly, to update the cost-to-go function in Equation (4.9) and to find the optimal policy, we have to search through the whole state-action space. The state space consists of all possible configurations of VMs on all the PMs and is combinatorially large. As computation of an estimate of the cost-to-go function involves searching through the state space S, high dimensionality of S makes the policy update expensive and almost impossible to perform in real-time. This exponential blow-up in computation due to the huge state space is called the *curse of dimensionality* [Powell, 2007]. Secondly, the expectation in Equation (4.9) is not computable as the stochastic nature of workload, its correlation with VM configurations and their transitions are not known *a priori*. In order to conserve the robustness and universality of Megh, we cannot restrict this

workload dynamics to a specific model. Indeed that would narrow down the applications and the hardware architectures the algorithm can deal with. Megh solves both the issues.

In order to solve the curse of dimensionality, Megh projects the state-action space to a $d = N \times M$ dimensional space X. X is spanned with d basis vectors $\{\phi_{jk}\}_{j=0,k=0}^{N,M}$. Each of the basis ϕ_{jk} corresponds to an action (j, k) such that the jk^{th} component of it is one, and all other elements are zero. All the actions or configuration changes in the Cloud is represented using these basis vectors or linear combinations of them. The basic rationale behind this projection is that during transition from a state to another only a part of the state space, which is one action away from the present state, is reachable. Instead of searching over the whole state space in each and every step, it is logical to search in a subspace X that contains all the states s' reachable from s by actions ϕ_{jk} or linear combinations of them. Thus, the combinatorially explosive state-action space of VM configurations is projected to a polynomial dimensional vector space with a sparse basis. The basis is called sparse because they have the dimension $d = N \times M$ but has only one non-zero component that matters for computation. Hence Megh approximates the cost-to-go function as $V(s_{t+1}) = \theta^T \phi_{a_t}$, where $a_t = \pi_t(s_t)$ is the action taken at time t. This enable Megh to update the cost-to-go function effectively in real-time. We prove that for the basis function that we have constructed, we would obtain a unique projection to approximate the value function at a given time.

Theorem 8. *Given the basis vectors $\{\phi_{jk}\}_{j=0,k=0}^{N,M}$ spanning the state-action space $S \times A$, there exists a unique projection vector $\boldsymbol{\theta} \in \mathbb{R}^{MN}$ that expresses the value function as $V(s) = \theta^T \phi_{\pi(s)}$.*

Proof. As we project the state-action space $S \times A$ to the space X spanned by d dimensional basis vectors $\{\phi_j\}_{j=0}^{d}$, we reduce our search space from whole state-action space to a subspace S^t. S^t is the set of all the states reachable through one migration action from the present state $s_t \in S$. Suppose $S^t = \{s^1, s^2, \ldots, s^d\}$. Note that we use superscripts to denote the ordering of elements in S^t.

Thus, at each time-step t, we update the value functions of the only reachable states in S^t. Let $\boldsymbol{V} = (V(s))_{s \in S^t}^T$ and Ψ be a $d \times d$ matrix such that

$$\Psi_{i,j} = \phi_{\pi(s^t)}[j] \qquad \forall j = 1, \ldots, d$$

where, s^i is the state reachable from s^t using action $\pi(s^t)$. Let $\boldsymbol{\theta}$ be a $|S|$-dimension column vector such that $\Psi\theta = \boldsymbol{V}$. If Ψ is invertible, $\theta = \Psi^{-1}\boldsymbol{V}$ and Theorem 8 holds.

We claim that Ψ is invertible and its inverse is the matrix Ω such that,

$$\Omega_{i,j} = (-1)^{|s^i|-|s^j|}\Psi_{i,j}.$$

In order to establish this construction, let us consider the i, j^{th} element of the matrix obtained by multiplying Ψ and Ω. If $s_i \sim s_j$ means state s_j is reachable from state s_i through one of the d migration actions, then

$$\begin{aligned}(\Omega\Psi)_{i,j} &= \sum_{1 \leq k \leq |S^t|} (-1)^{|s^i|-|s^k|}\Psi_{i,k}\Psi_{k,j} \\ &= \sum_{s_j \sim s_k \sim s_i} (-1)^{|s^i|-|s^k|}.\end{aligned}$$

Therefore $(\Omega\Psi)_{i,j} = 1$ if and only if $i = j$. Thus, Ψ is invertible and there exists a unique projection for given basis vectors. □

Still the expectation of the cost-to-go function is not computable due to lack of prior knowledge of workload dynamics and how it affects the VM configurations and their transitions. In order to capture this notion, we create a stochastic matrix $T \in \mathbb{R}^{d \times d}$. T accumulates the possibility of using an action to move to another configuration from the present one depending on the nature of workload and the changes caused by them. In this work, we begin with $T_0 = \frac{1}{\delta}\mathbb{I}_d$, where δ is a large positive number and $\mathbb{I}_d$ is an identity matrix of order d. Here, we have considered δ as d. It implies that initially, there is no bias and the system can migrate any of the VMs to any of the PMs equally probably. As the system extracts information of the workload and VM configurations

Algorithm 18 Megh

function MEGH(S, A, γ, ϵ, $Temp_0$)
  Initialise $\delta \leftarrow d, B_0 \leftarrow \frac{1}{\delta}\mathbb{I}_{d\times d}, \phi_0 \leftarrow \mathbf{0}_d$,
  $\theta_0 \leftarrow \mathbf{0}_d, \pi(s_0) \leftarrow \mathbf{0}_d, z_0 \leftarrow \mathbf{0}_d, C_0 \leftarrow 0$
  while $t \geq 1$ **do**
    $a_t \leftarrow \arg\max_{a\in A} \quad \pi_t(s_t)$
    Take action a_t.
    Observe state s_{t+1}.
    $C_{t+1} \leftarrow$ Calculate cost using Equation (4.6).
    $B_{t+1} = T_{t+1}^{-1}$ update using Equation (4.10).
    $z_{t+1} \leftarrow z_t + \phi_{a_t} C_{t+1}$
    $\theta_{t+1} \leftarrow B_{t+1} z_{t+1}$
    $\pi(s_{t+1}) \leftarrow PolicyCalculator(\phi_{a_t}, \theta_{t+1})$
  end while
end function

at each time step t, it decides an action a_t according to the policy π_t. Using this information, we update the operator T as

$$T_{t+1} = T_t + \phi_{a_t}\left[\phi_{a_t} - \gamma\phi_{\pi_t(s_{t+1})}\right]^T. \tag{4.10}$$

$\phi_{\pi_t(s_{t+1})}$ represents the probable action at time $t+1$, if the policy π_t is followed at the next time instance. Thus, Equation (4.10) captures the effect of present state and action and its influence in future action with a discount γ.

In Megh, we plug in these two schemes of polynomial size projection space X and incremental update of the operator T to Least-Square Policy Iteration algorithm [Lagoudakis and Parr, 2003b]. Megh first tries to find out an estimation of cost-to-go function by least-square estimation in the actor format and then to update the policy such that it maximises the estimate in the critic format. The pseudo-code of Megh is depicted in Algorithm 18.

Theorem 9. *If for a Markovian policy $\pi \in \Pi$, there exists a real-valued projection vector $\theta \in \mathbb{R}^d$ and the basis vectors $\{\phi_{jk}\}_{j=0,k=0}^{N,M}$ such that $V_\pi(s) = \theta^T\phi_{\pi(s)}$ for any configuration s, Algorithm 18 will converge to an optimal policy π^*. π^* is the fixed point in Π with respect to the Bellman operator defined by Equation* (4.12).

Proof. Let us denote the set of all possible value functions V^π obtained using policy $\pi \in \Pi$ as $\mathcal{V}$. Without loss of generality, we can assume $\mathcal{V}\colon\ S \to \mathbb{R}$ be a set of bounded, real-valued functions. Then $\mathcal{V}$ is a Banach space with the sup-norm $\|V\| = \|V\|_\infty = \sup|V(s)|$ for any $V \in \mathcal{V}$. Now, we narrow down our problem of Equation (4.8) by including the smaller reachable state space. If $s_t \in S_{t-1}$ i.e, s_t is reachable from s_{t-1} through one of feasible migrations,

$$\underset{\pi\in\Pi}{\arg\min} \quad \mathbb{E}_f \left[\sum_{t=1}^{\infty} \gamma^{t-1} C(s_{t-1}, s_t)\right] \tag{4.11}$$

In this narrower space, Algorithm 18 is analogous to LSPI over the reduced search space X. For this new problem given by Equation (4.11), Algorithm 18 converges to a unique cost-to-go function, say $\tilde{V} \in \mathcal{V}$. We need to show that the cost-to-go function estimated by Algorithm 18 is the optimal one i.e, $V^* = \tilde{V}$.

Let us define the process of updating policy as a modified Bellman operator $\mathcal{T}\colon\ \mathcal{V} \to \mathcal{V}$. Using Equation (4.9), we define $\mathcal{T}$ as

$$\mathcal{T}V(s) = \min_{s'\in S_s} \mathbb{E}_f \left[C(s, s') + \gamma V(s')\right]. \tag{4.12}$$

For a given state $s \in S$, let

$$a_s^*(V) = \underset{s'\in S_s}{\arg\min} \left(C(s, s') + \gamma V(s')\right).$$

If $s^*(v)$ is the state obtained by following optimal policy π^* from value function v and state s, and $\mathcal{T}V(s) \geq \mathcal{T}U(s)$ for $V, U \in \mathcal{V}$, then

$$\begin{aligned}
0 \leq \mathcal{T}V(s) - \mathcal{T}U(s) &= \mathbb{E}\left[C(s, s^*(V)) + \gamma V(s^*(V))\right] - \mathbb{E}\left[C(s, s^*(U)) + \gamma U(s^*(U))\right] \\
&\leq \mathbb{E}\left[C(s, s^*(U)) + \gamma V(s^*(U))\right] - \mathbb{E}\left[C(s, s^*(U)) + \gamma u(s^*(U))\right] \\
&= \gamma \mathbb{E}\left[V(s^*(U)) - U(s^*(U))\right] \\
&\leq \gamma \mathbb{E}\left[\|V - U\|\right]
\end{aligned}$$

$$= \gamma \|V - U\|.$$

This result states that if $\mathcal{T}V(s) \geq \mathcal{T}U(s)$, then $\mathcal{T}V(s) - \mathcal{T}U(s) \leq \gamma|V(s) - U(s)|$. If we assume that $\mathcal{T}V(s) \leq \mathcal{T}U(s)$, the same reasoning produces $\mathcal{T}V(s) - \mathcal{T}U(s) \geq -\gamma|V(s) - U(s)|$. Thus we can conclude, $|\mathcal{T}V(s) - \mathcal{T}U(s)| \leq \gamma|V(s) - U(s)|$ for all configuration $s \in S$. From the definition of our norm, we can write

$$\begin{aligned} \sup_{s \in S} |\mathcal{T}V(s) - \mathcal{T}U(s)| &= \|\mathcal{T}V - \mathcal{T}U\| \\ &\leq \gamma \|V - U\|. \end{aligned}$$

This means for $0 \leq \gamma < 1$, $\mathcal{T}V(s)$ is a contraction mapping. Following [Lagoudakis and Parr, 2003b;Proposition 3.10.2], there exists a unique V^* such that $\mathcal{T}V^* = V^*$. Thus, for an arbitrary initial value function $V^0 \in \mathcal{V}$, the sequence V^t generated by $V^{t+1} = \mathcal{T}V^t$ converges to V^*. By the property of convergence of LSPI [Lagoudakis and Parr, 2003b], $V^* = \tilde{V}$. As the cost function C is a positive and monotonically increasing function, the optimal cost-to-go function V^* also satisfies $\mathcal{T}V^* = V^*$. Hence, $V^* = \tilde{V}$, and the property of convergence of LSPI is preserved in Algorithm 1. □

Algorithm 19

```
1: function PolicyCalculator(φ_{a_t}, θ_{t+1})
2:     Temp_{t+1} ← Temp_t exp(−ε)
3:     for all i = 1, ..., d do
4:         Q(s_{t+1}, a_i) ← φ_{a_i}^T θ_{t+1}
5:     end for
6:     MIN_Q ← min_a Q(s_{t+1})
7:     for all i = 1, ..., d do
8:         π(s_{t+1})_i ← exp[ (−Q(s_{t+1}, a_i) + MIN_Q) / Temp_{t+1} ]
9:     end for
10: end function
```

Instead of greedily choosing the action with maximum $V^{\pi_t}(s_{t+1})$, we have used Boltzmann exploration as the on-policy algorithm. The pseudocode is illustrated in Algorithm 19. This technique compares the goodness of an action with respect to the

others and allows the algorithm explore more. We refer to the discussion in Section 2.2 for detailed discussion. Here, we have started with an initial temperature value $Temp_0$ and decay it consequently with a factor $\exp(-\epsilon)$. Initially, the large *Temp* means rather than choosing the maximum greedily it is trying to explore more. As *Temp* decreases with time, *PolicyCalculator* becomes the greedy selection of the maximum.

Managing the Complexity Bottleneck

Algorithm 18 has space complexity of $O(d^2)$ and time complexity of $O(d^3)$. Though the algorithm is computationally cheaper and faster than the actual combinatorially explosive problem scenario, still it can be slow enough for a real-time system operating over a large number of VMs and PMs. The space complexity bottleneck is storing the $d \times d$ matrix B. The time complexity bottleneck is computing the inverse of the operator T to update B at each time-step, as shown in Line 9 in Algorithm 18. If we use the Gauss-Jordan elimination process [Atkinson, 2008] provided by linear algebra packages [Anderson et al., 1999], inversion of T costs time complexity of $O(d^3)$. In order to compute the inverse incrementally at every step, we use Sherman-Morrison Formula [Sherman and Morrison, 1949] on Equation (4.10) given by,

$$B_{t+1} = B_t - \frac{B_t \phi_{a_t} \left[\phi_{a_t} - \gamma \phi_{\pi_t(s_{t+1})}\right]^T B_t}{1 + \left[\phi_{a_t} - \gamma \phi_{\pi_t(s_{t+1})}\right]^T B_t \phi_{a_t}}. \tag{4.13}$$

Thus, the time complexity of every step is reduced to $O(d^2)$.

We reduce the complexity further by leveraging the sparsity of the basis vectors ϕ_{a_i}'s. Since all the zero entries are redundant in the calculation of product, we store only the non-zero entries of the matrix B and vector ϕ_{a_i} as a triplet (row number, column number, value). This reduces the initial storing size to $O(d)$. Because during initialisation we start with a diagonal matrix of order d, and d basis vectors each with single non-zero entry. The storing size increases at each step as per the number of migrations happened during the interval. Thus the multiplication in Equation (4.13) turns into choosing the non-zero terms in B_t according to the 1 entries in ϕ_{a_i}'s involved

Table 4.1: Power Consumption of servers in Watts for different level of workload [Huppler et al., 2012; SPECpower Committee, 2014]

Server Type	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	93.7	97	101	105	110	116	121	125	129	133	135

in the calculation and then adding or subtracting them. It reduces the time complexity of Line 9 in Algorithm 18 to $O(\#m)$, where $\#m$ is the number of migrations per step. The aforementioned use of online update and inversion technique, and also leveraging the sparsity of the basis vector reduces both the space and time complexity of Megh substantially. These techniques provide Megh the speed-up to be a real-time system while keeping its structure and learn-as-you-go strategy intact.

4.6 Performance Evaluation

4.6.1 Experimental Setup

We perform experiments using the CloudSim toolkit [Calheiros et al., 2011] as the simulation platform. CloudSim uses CPU utilisation as the key metric to characterise the workloads. We follow this characterisation throughout our experiments. In the power model, we use the standard price of the local power providers, 0.18675, USD/kWh to calculate the energy consumption cost. We assume that the user has to pay 1.2 USD per hour for using a VM instance. Though it is a bit costlier than reality, it does not harm the analysis. Following the model mentioned in Section 4.3.3, we also assume that Cloud providers would pay back 16.7% and 33.3% of user's money depending on whether the performance degradation is less than or greater than 0.10%. We consider $\beta = 70\%$ as the overloading threshold of the PMs and $\alpha = 30\%$ for the minimum CPU usage threshold by VMs during migration. The experiments are conducted on a server with two AMD Opteron(TM) Processor 6272 CPUs. Each CPU has eight cores, 128 GB memory and clock rate of 2.1GHz. Each core has two threads.

MMT algorithms are tested using the code embedded with the CloudSim toolkit, whereas Megh and MadVM are implemented and embedded in the CloudSim framework using Java. For both of them, the value of γ is set to 0.5. $\gamma = 0.5$ imposes 50:50 importance of both new and old information. $Temp_0$ and ϵ are set to 3 and 0.01 respectively for the experiments in Section 4.6.3 and 4.6.4.We explicate such choice of parameters in Section 4.6.5. At each time-step, we allow a maximum 2% of VMs to be migrated by Megh.

4.6.2 Dataset and Workload

PlanetLab Dataset

CloudSim contains workloads extracted from the CoMoN project which was a monitoring infrastructure for PlanetLab [Park and Pai, 2006]. Each of the workloads consists of CPU utilisation data extracted at a regular interval of 5 minutes for a span of 7 days. Figure 4.1(a) shows the statistical nature of the workload and depicts inherent uncertainty in its dynamics. The workloads are working on a set of 800 heterogeneous physical machines (PMs). Half of these PMs are HP ProLiant ML110 G4 servers and the other half are HP ProLiant ML110 G5 servers. The power consumption characteristics of these two servers is obtained from SPECbenchmark and is shown in Table 4.1. Though they follow different energy consumption models, each of them has a dual-core processor with 4GB RAM and are provided with 1 Gbps network bandwidth. There are a total of 1052 applications are running on this system. Each of the applications are allocated on a VM with 1 vcpu, 0.5-2.5GB RAM, 0.5-2.5 MIPS and 100 Mbps bandwidth.

Google Cluster Dataset

The Google Cluster trace represents dynamic tasks running on Google's Hadoop MapReduce clusters with 12,500 heterogeneous machines [Li et al., 2017]. The trace contains continuous information of 29 days with event records and sampled resource

Table 4.2: Performance Evaluation for PlanetLab

Algorithms	THR-MMT	IQR-MMT	MAD-MMT	LR-MMT	LRR-MMT	Megh
Total cost (USD)	1347	1504	1367	1392	1392	1155
#VM migrations	325299	444624	331304	324079	324079	2309
#Active hosts	666	684	682	692	692	203
Execution time (ms)	2016	3077	2226	1924	2080	1426

usage at an interval of 5 minutes. We select 500 machines as physical machines and the tasks scheduled on those machines as virtual machine workloads. We create 2000 virtual machines with each running an individual task to completion and switching to another. Unlike PlanetLab where all of the workloads are together varying intensely, the Google Cluster trace has tasks with varying durations, starting times, and obfuscated resource usages as shown in Figure 4.1(b).

PlanetLab is a huge geo-distributed computing platform consisting of hundreds of sites and more than one thousand nodes [Park and Pai, 2006]. It is hosted by organisations across the world. Users can access the computing resources by deploying applications to a subset of the nodes in the form of VMs. The trace is collected from PlanetLab to track the CPU usage of each VM's workload. The result represents the typical workload running in an enterprise Cloud environment. While the PlanetLab trace is mainly related to academic and other organisational computation tasks, the Google Cluster trace records the events in Google's Hadoop MapReduce clusters. Google's trace shows the characteristics of workloads running in the publicly available Cloud systems [Li et al., 2017]. Evaluating Megh with the traces from both the community and the industry validates its universality and robustness.

4.6.3 Comparative Performance Analysis

Megh vs MMT algorithms

Table 4.2 depicts the performance of Megh and the MMT algorithms on a week-long trace of PlanetLab. Table 4.3 summarises the performance of the aforementioned algorithms for the Google Cluster dataset. Total cost of operation of the data center (in USD) obtained by adding the power consumption cost and SLA violation cost,

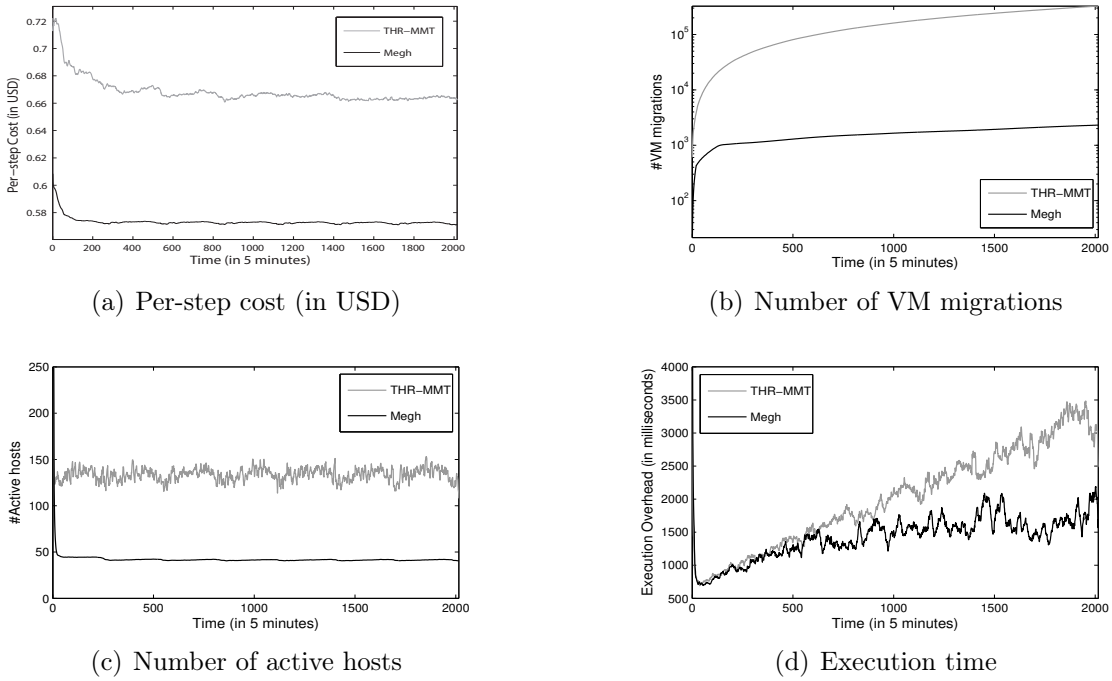
Table 4.3: Performance Evaluation for Google Cluster

Algortihm	THR-MMT	IQR-MMT	MAD-MMT	LR-MMT	LRR-MMT	Megh
Total cost (USD)	706	708	708	710	710	688
#VM migrations	299352	262185	266706	233172	233172	3104
#Active host	82	72	73	59	59	194
Execution time (ms)	2887	4030	4000	3889	3923	1945

the number of VM migrations, average number of active hosts and execution time (in milliseconds) of each iteration of the algorithms are used as the performance measures of the algorithms. As THR-MMT performs the best among the MMT algorithms, we show a comparison of Megh with THR-MMT in Figures 4.2 and 4.3.

We observe from Tables 4.2 and 4.3 after 7 days of operation Megh reduces the expenditure by 14.25% for PlanetLab and 2.5% for Google Cluster with respect to that of THR-MMT. Figures 4.2(a) and 4.3(a) show the per-step operation cost for Megh not only converges faster than the contending algorithms but also has less variance for both PlanetLab and Google. Here, the per-step operation cost includes both the energy consumption cost and the SLA violation cost in the 5 minutes interval between two observations. Due to the learn-as-you-go policy, Megh takes around 100 time-steps before reaching the almost stable cost per-step. We do not observe such a fast convergence for THR-MMT. Being a greedy heuristics, THR-MMT still faces high variance and instability even after initial convergence. These observations validate robustness and stability of Megh for optimal resource management for a diverse set of workloads with respect to other heuristics.

In order to measure the performance of the system and its QoS, we use the number of VM migration as another metric. In our experiments, we consider that during the course of migration the CPU capacity allocated to a VM on the destination node is same as that of the present host. This means that each migration may cause some SLA violation. Therefore, it is crucial to minimise the number of VM migrations. The total number of VM migrations for THR-MMT is almost 140 times and 97 times more than that of Megh for PlanetLab and Google respectively. Figures 4.2(b) and 4.3(b) report the evolution of the cumulative number of VM migrations over the span of 7



(a) Per-step cost (in USD)

(b) Number of VM migrations

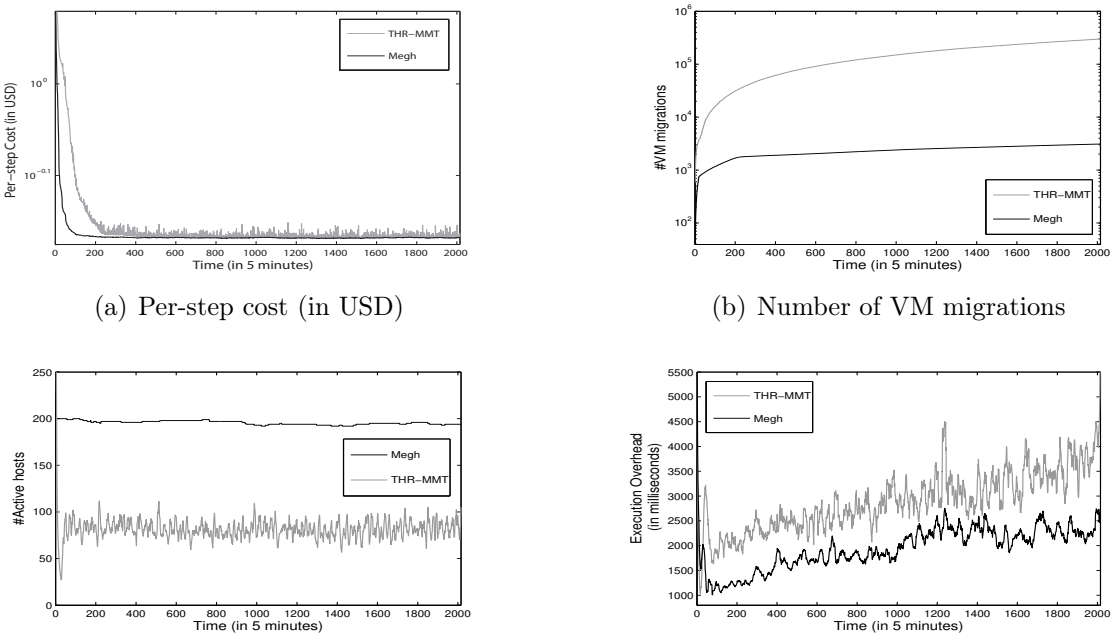
(c) Number of active hosts

(d) Execution time

Figure 4.2: Performance of Megh and THR-MMT algorithms for PlanetLab dataset

days. As the total number of VM migrations up to an instance for Megh is much less than that of the THR-MMT, it shows that at any instance Megh performs significantly better.

Decreasing the number of active hosts also decreases the power consumption. Thus, the number of active hosts is also used as a performance metric for resource management algorithms. Though reducing the number of active hosts is the approach taken by VM consolidation algorithms, it may prove not to be a perfect metric. Because keeping a larger number of hosts at very low utilisation level may cause less power consumption than keeping a few hosts at very high utilisation level. We observe this dilemma from Figures 4.2(c) and 4.3(c). For PlanetLab, Megh keeps fewer hosts active than other MMT algorithms, whereas for Google it keeps more active VMs while incurring the least per-step cost for both datasets. While the results establish Megh's effectiveness to solve the live migration decisions with less expenditure and better QoS, Megh has to fulfil another criterion to be a real-time system: a small execution time. From Figures 4.2(d) and 4.3(d), we observe Megh is running faster than that of the



(a) Per-step cost (in USD)

(b) Number of VM migrations

(c) Number of active hosts

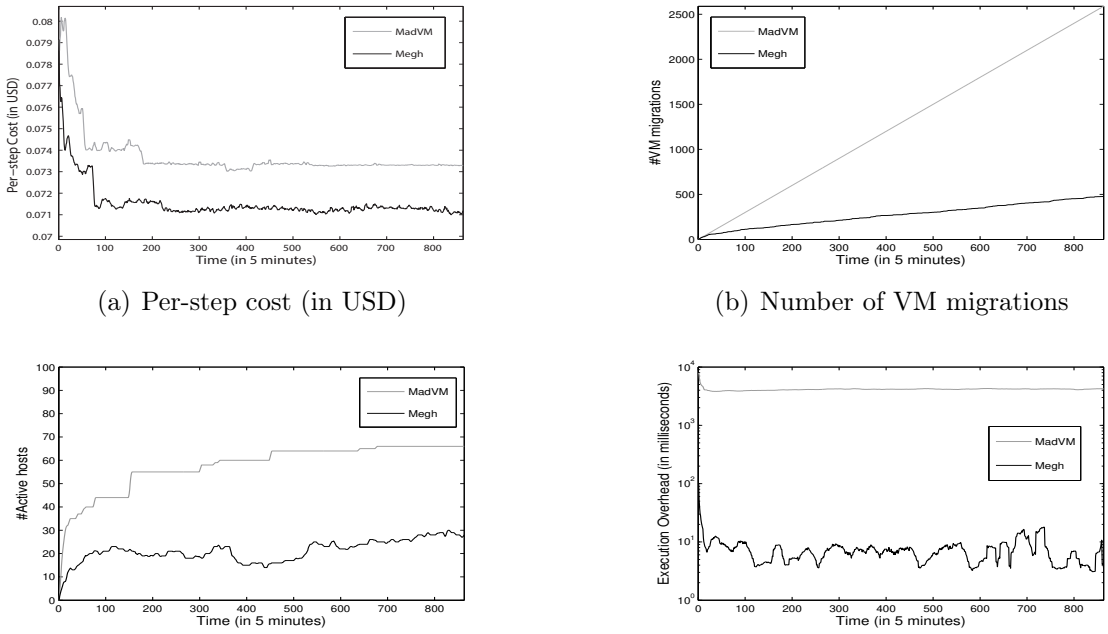
(d) Execution time

Figure 4.3: Performance of Megh and THR-MMT algorithms for Google Cluster dataset.

heuristic based online algorithms. As shown in Tables 4.2 and 4.3, Megh speeds up the decision making by 1.41 and 1.48 times with respect to THR-MMT for PlanetLab and Google respectively. Since migration time of a VM is in the order of a few seconds, speed up of Megh with respect to the state-of-the-art can help the system to make decisions and to execute them with significantly less overhead or downtime to the process of migration. This, in turn, improves the QoS of the system too. This empirically proves the efficiency of Megh not only as an effective learning algorithm but also as an eligible real-time resource management system in Clouds.

Megh vs MadVM

MadVM fails to scale-up for the complete PlanetLab or Google Cluster in our experimental facilities. Thus, in order to compare the performance of Megh with MadVM, we have chosen two random sets of 150 workloads running on 100 PMs for 3 days from PlanetLab and Google Cluster traces. In the beginning, all these workloads are



(a) Per-step cost (in USD)

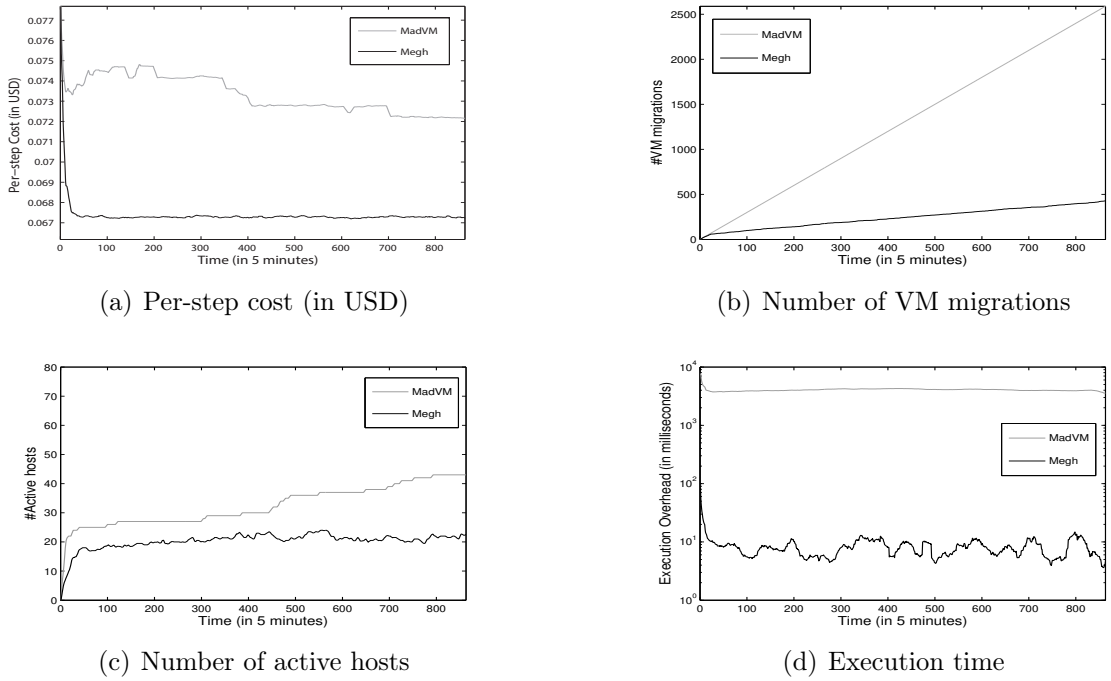
(b) Number of VM migrations

(c) Number of active hosts

(d) Execution time

Figure 4.4: Performance of Megh and MadVM for a dataset of 100 PMs and 150 VMs extracted from PlanetLab trace.

allocated uniformly at random to each of the PMs, such that there is no initial bias for the learning and the robustness of both the algorithms can be tested. The 50:50 ratio of two type of servers is still maintained. From Figures 4.4(a) and 4.5(a), we observe that Megh incurs less cost (4.3% and 8.8%) than MadVM at every time step. Figures 4.4(b) and 4.5(b) show Megh causes significantly less number (5.5 and 6.1 times) of migrations than MadVM. Figures 4.4(c) and 4.5(c) depict at every time step MadVM (average ∼58 and 34) keeps more hosts active than Megh (average ∼21 and 20). But the main factor where MadVM stumbles is the execution time. MadVM takes on an average 4143ms and 4057ms to execute a single iteration for a system of 100 PMs and 150 VMs, which is almost the same as the migration time of a VM of 0.5 GB RAM in the PlanetLab set-up. As the reinforcement learning algorithms face the curse of dimensionality and have a huge transition matrix for bookkeeping at each time step, it makes reinforcement learning algorithms slower for a real-time system. Though authors of MadVM tries to handle such scenario, Figures 4.4(d) and 4.5(d)



(a) Per-step cost (in USD)

(b) Number of VM migrations

(c) Number of active hosts

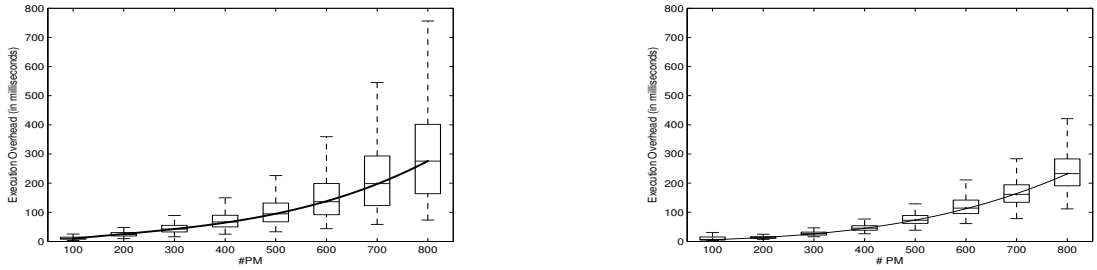
(d) Execution time

Figure 4.5: Performance of Megh and MadVM for a dataset of 100 PMs and 150 VMs extracted from Google Cluster trace.

depict its inability to scale in real-time for large data centers. Since Megh leverages the sparsity-based projection technique (Theorem 8), along with the specialised data structure (Section 4.5), it takes the same migration decisions in approximately 7ms and 8ms respectively for PlanetLab and Google datasets. The experiments validate that though Megh uses the reinforcement learning framework for learning the workload dynamics and making migration decisions, it is significantly more efficient and faster than the latest state-of-art reinforcement learning algorithm for dynamic VM management.

4.6.4 Scalability Analysis

Scalability is an important issue that an algorithm has to achieve in order to perform for a large-scale Cloud data center. We show a comparative analysis of scalability of Megh and THR-MMT in Figures 4.6(a) and 4.6(b). In order to conduct such experiments, we randomly choose m and n number of PMs and VMs from the PlanetLab



(a) THR-MMT (b) Megh

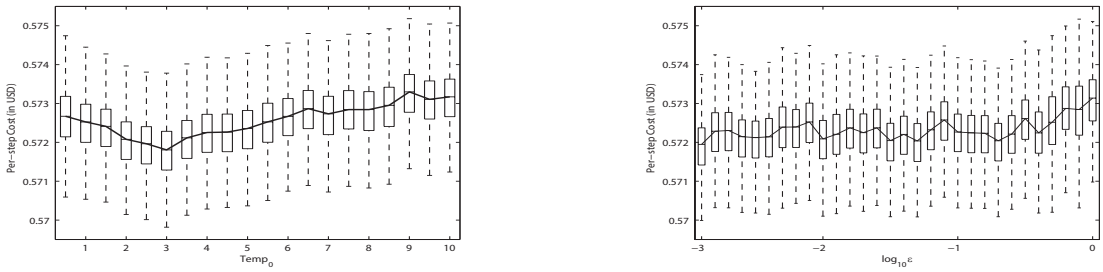
Figure 4.6: Scalability analysis of THR-MMT (left) and Megh (right).

data. Here, both m and n take values in $\{100, 200, 300, 400, 500, 600, 700, 800\}$. For each value of m and n, we conduct 25 experiments with 25 randomly chosen set of PMs and VMs. We observe from Figures 4.6(a) and 4.6(b) as the number of PMs and VMs increase, the execution time per-step increases for both THR-MMT and Megh. With the increase of number of PMs and VMs, the decision making algorithm has to choose among larger set of actions and has to face an increased uncertainty in workload dynamics. Thus, this increase in execution time is intuitive and natural. For Megh the rise in execution time is much smaller than that of THR-MMT. This significant difference in per-step execution time shows that Megh scales up better than THR-MMT. This scalability establishes Megh more effective as a real-time decision maker for large-scale Clouds.

As MadVM is not scalable after 100∼150 PMs, we cannot conduct such a comparative study with it.

4.6.5 Parameter Sensitivity

$Temp_0$ and ϵ are used as parameters to tune the exploration-exploitation trade-off of Megh. We test and analyse Megh's performance on different values of the parameters. We vary $Temp_0$ from 0.5 to 10 with a granularity of 0.5 while keeping $\epsilon = 0.001$. We run experiments on 30 distinct values of ϵ, which belong to the interval $[10^{-3}, 10^{0}]$ and are at a logarithmic (base 10) distance of 0.1. In this case, $Temp_0$ is fixed to 1.



(a) $Temp_0$-sensitivity

(b) ϵ-sensitivity

Figure 4.7: Sensitivity of per-step cost (in USD) on $Temp_0$ and ϵ.

For each value of $Temp_0$ and ϵ, Megh is tested 25 times on the PlanetLab dataset described in Section 4.6.2.

Figures 4.7(a) and 4.7(b) show boxplots of per-step cost (in USD) of Megh for each of the values of the parameter. These boxplots depict the median and 90 percentile distribution of the per-step cost. We observe that the median cost decreases first as the $Temp_0$ increases but the cost rises as $Temp_0$ becomes greater than 3. Though for ϵ this change in per-step cost is a bit sporadic, we empirically observe that the variance and the median both reach a local minimum at $\epsilon = 0.001$.

Since use of $Temp$ in Algorithm 19 allows Megh to explore more rather than direct exploitation, increase in $Temp_0$ would increase the initial exploration. We observe till $Temp_0 = 3$ this increase in exploration is decreasing the median cost. Because increased exploration stops agent from getting stuck at local minima and take decisions more globally. After that point, we see the adverse effect of too much exploration. As $Temp_0$ increases after 3, the algorithm cannot benefit enough from exploitation. Thus, the curve instantiate the exploration-exploitation trade-off in case of Megh.

ϵ controls decay of $Temp_0$ with time. As $Temp_0$ decays, the exploratory nature turns dormant and exploitative nature begins to dominate. Thus, increase in ϵ would cause faster decay of $Temp$. Though we expect to observe similar nature as that of the variation of $Temp_0$, here we find out a bit of sporadic nature where it is hard to detect a single tipping point for exploration-exploitation trade-off. Hence, we make our choice empirically from observation.

4.7 Conclusion

This work addresses the problem of energy– and performance–efficient resource management during live migration of VMs in a Cloud data center. Uncertain dynamics and diversity of workloads as well as the heterogeneous Cloud hardware demand for a generic algorithm to solve the efficient VM migration problem under uncertainty. Reinforcement learning provides a general framework to learn as-you-go to take decisions under uncertainty. Thus, we propose a reinforcement learning algorithm, Megh, that works irrespective of application and hardware heterogeneity while learning the uncertain dynamics. State-of-the-art reinforcement learning algorithms encounter curse of dimensionality and unavailability of a model for workload's uncertainty. These issues make such algorithms not scalable in real-time and asks for extensive training respectively. Megh dissolves both of the issues in real-time. In order to overcome the curse of dimensionality, Megh projects the combinatorially explosive state-action space to a polynomial dimensional space with sparse basis. Megh updates the transition operator incrementally without using any prior knowledge of workload dynamics. Through this update, Megh learns the uncertainty and dynamics of workload as-it-goes. We leverage a data structure based on the sparsity of the basis for fast and scalable real-time updates and learning. Megh incurs the smallest cost and the least execution overhead with respect to its contenders both on PlanetLab and Google Cluster workloads. This validates Megh's claim as a cost-effective, time-efficient and robust algorithm. The comparative scalability analysis of Megh and THR-MMT demonstrates that Megh has better scalability than the competing algorithm. We explicate our choices of parameters controlling the exploration-exploitation trade-off through a sensitivity analysis of Megh.

We are currently investigating the opportunity to take advantage of additional knowledge about the workload, such as periodicity, and also to leverage knowledge of the network topology like fat-trees [Leiserson, 1985]. Finally, following previous works on energy efficient live migration of Clouds, we have considered only CPU utilisation data. We are aware of important of bandwidth as a resource and works [Lago et al.,

2017; Yu et al., 2017] accounting available bandwidth and network traffic as principal decision variables for VM migration. We are confident that network and memory sharing can be seamlessly accommodated without modifying our solution algorithmically. We are studying the necessary extensions of the cost model to such settings in order to apply Megh.

Part II

An Information Geometric Approach to Learning with Incomplete Information

Chapter 5

BelMan: An Information Geometric Approach to Multi-armed Bandits

Exploration is in our nature. We began as wanderers, and we are wanderers still.

— *Carl Sagan, Cosmos, 1980.*

In this chapter, we propose a generic Bayesian information-geometric approach to the exploration–exploitation trade-off in stochastic multi-armed bandit problems. The learning problem is set in the statistical manifold of joint distributions representing the uncertainty on beliefs and rewards of the arms, which we refer to as *belief-reward distributions*. At each time step, the belief-reward distributions of the arms are summarised by their barycentre In this manifold. We refer to this barycentre as the *pseudobelief-reward*. We implement the approach as an algorithm that we call BelMan. BelMan alternates between the projection of the pseudobelief-reward distribution onto the belief-reward distributions to choose the arm to play, and the projection of the resulting belief-reward distributions onto the pseudobelief-reward distribution. Additionally, BelMan introduces a mechanism that infuses an exploitative bias by gradually concentrating on higher rewards. We refer to this as the *focal distribution*. Incorporation of these information geometric constructions makes BelMan uniformly applicable to exploration–exploitation, pure exploration and two-phase reinforcement

learning. We instantiate BelMan to the classes of Bernoulli and exponential bandits, respectively. Comparative performance evaluation with state-of-the-art algorithms shows that BelMan is not only competitive but also outperforms other approaches in challenging setups such as those involving many arms and continuous rewards.

5.1 Introduction

The Multi-armed bandit problem [Robbins, 1952] is a sequential decision-making problem [DeGroot, 2005] in which a gambler plays a set of arms to obtain a sequence of rewards. In the *stochastic bandit* problem [Bubeck et al., 2012], the rewards are yielded from corresponding reward distributions on arms. These reward distributions belong to the same family of distributions but differ in the parameters. These parameters are unknown to the gambler. In the classical setting, the gambler devises a policy, choosing a sequence of arm draws, that maximises the *expected cumulative reward* [Robbins, 1952]. In an equivalent formulation, the gambler devises a policy that minimises the *expected cumulative regret* [Lai and Robbins, 1985], that is the expected cumulative deficit of reward caused by the gambler not always playing the optimal arm. In order to achieve this goal, the gambler must learn at the same time the parameters of the reward distributions of arms. Thus, solving the stochastic bandit problem consists in devising strategies that combine both the accumulation of information to reduce the uncertainty of decision making, *exploration*, and the accumulation of rewards, *exploitation* [Macready and Wolpert, 1998]. Hereby, we refer to this stochastic bandit problem as the *exploration–exploitation bandit* problem to highlight the trade-off. If a policy relies on independent phases of exploration and exploitation, it necessarily yields a suboptimal regret bound [Garivier et al., 2016a]. The gambler has to adaptively balance and intertwine exploration and exploitation [Auer et al., 2002]. In a variant of the stochastic bandit problem, called the *pure exploration bandit* problem [Bubeck et al., 2009], the goal of the gambler is solely to accumulate information about the arms. In another variant of the stochastic bandit problem, the gambler interacts with the ban-

dit in two consecutive phases of pure exploration and exploration-exploitation. Putta and Tulabandhula proposed an MDP variant of this problem and named it the *two-phase reinforcement learning* problem [Putta and Tulabandhula, 2017a]. Following this nomenclature, we refer to the bandit variant of two-phase reinforcement learning as the *two-phase bandit problem*.

Although frequentist algorithms with optimism in the face of uncertainty such as UCB [Auer et al., 2002] and KL-UCB [Garivier and Cappé, 2011] work considerably well for the exploration–exploitation bandit problem, their frequentist nature prevents effective assimilation of a priori knowledge about the reward distributions of the arms [Kawale et al., 2015]. Beside this, the Bayesian formulation of uncertainty as the probability distributions allows us to construct a space of uncertainties, and to investigate further the geometrical properties of the space. Bayesian algorithms for the exploration–exploitation problem, such as Thompson sampling [Thompson, 1933] and Bayes-UCB [Kaufmann et al., 2012a], leverage a prior distribution that summarises a priori knowledge. However, as argued in [Kaufmann and Kalyanakrishnan, 2013], there is a need for Bayesian algorithms that also cater for pure exploration. Neither Thompson sampling nor Bayes-UCB are able to do so.

Our contribution. We propose a unified Bayesian approach to address the exploration-exploitation, pure exploration, and two-phase bandit problems. We address these problems from the perspective of information representation, accumulation, and balanced induction of bias. Following Bayesian algorithms [Thompson, 1933], we maintain a parametrised *belief* distribution for each arm representing the uncertainty on the parameter of its reward distribution. Extending this representation, we use a joint distribution to express the uncertainty on both the belief and the reward distributions of each arm. We refer to these joint distributions as the *belief-reward distributions* of the arms. We set the learning problem in the statistical manifold [Amari and Nagaoka, 2007] of the belief-reward distributions, which we call the *belief-reward manifold*. The

belief-reward manifold provides a representation for controlling pure exploration and exploration–exploitation, and to design a unifying algorithmic framework.

As we argue in Chapter 2 and in the preceding discussion, we find out that learning through exploration is a fundamental component of the three variants of the bandit problem. Once the distributions are learnt completely, rest of the problem turns into an optimisation problem. Due to this inherent requirement of exploration in bandit problems, we begin with the construction facilitating exploration. Exploration requires a collective representation of the accumulated knowledge about the arm. From an information-geometric point of view [Barbaresco, 2013; Agueh and Carlier, 2011], the barycentre of the belief-reward distributions in the belief-reward manifolds serves as a succinct summary. We refer to this barycentre as the *pseudobelief-reward*. We prove that the pseudobelief-reward is a unique representation in the manifold. Though pseudobelief-reward facilitates the accumulation of knowledge, it is essential for the exploration–exploitation bandit problem to also incorporate a mechanism that gradually concentrates on higher rewards [Macready and Wolpert, 1998]. We introduce a distribution that induces such an increasing exploitative bias. We refer to this distribution as the *focal distribution*. We incorporate it into the definition of the pseudobelief-reward distribution to construct the *pseudobelief-focal-reward distribution*. This pushes the summarised representation towards the arms having higher expected rewards. We implement the focal distribution using an exponential function of the form $\exp(x/\tau(n))$, where x is the reward, and a parameter $\tau(n)$ dependent on time n and is referred to as *exposure*. Exposure controls the exploration–exploitation trade-off.

In Section 5.4, we apply these information geometric constructions to develop the BelMan algorithm. BelMan alternates information (I-) and reverse information (rI-) projections [Csiszár, 1984] between belief-reward distributions of the arms and the pseudobelief-focal-reward distribution. I-projection of the pseudobelief-focal-reward onto belief-rewards selects an arm. As it is played and a reward is collected, BelMan updates the belief-reward distribution of the corresponding arm by rI-projection of

the updated belief-reward distributions onto the pseudobelief-focal-reward. We prove the law of convergence of the pseudobelief-focal-reward distribution for BelMan, and that BelMan asymptotically converges to the choice of the optimal arm under some assumptions. BelMan can be tuned, using the exposure, to support in continuum from pure exploration to exploration–exploitation and two-phase bandit problems [Basu et al., 2018c].

We instantiate BelMan for distributions of the exponential family [Brown, 1986]. These distributions lead to analytical forms that allow to derive well-defined and unique I- and rI-projections as well as to devise an effective and fast computation. In Section 5.5, we empirically evaluate the performance of BelMan on different sets of arms and parameters for Bernoulli and exponential distributions, thus showing its applicability to both discrete and continuous rewards. We experimentally and comparatively evaluate BelMan with state-of-the-art algorithms: UCB [Auer et al., 2002], KL-UCB, KL-UCB-Exp [Garivier and Cappé, 2011], Bayes-UCB [Kaufmann et al., 2012a], Thompson Sampling [Thompson, 1933], and Gittins Index [Gittins, 1979], in these different settings. Results demonstrate that BelMan is not only competitive but also outperforms existing algorithms for challenging setups such as those involving many arms and continuous rewards. For the two-phase bandit problem, results show that BelMan spontaneously adapts with the explored information, and in turn, escalate efficiency.

5.2 Revisiting the Multi-armed Bandit Literature

Though we have described the exploration–exploitation bandit problem, and pure exploration bandit problems in Section 2.1 of Chapter 2, we revisit the existing literature of the bandit problems briefly to contextualise the algorithm.

Exploration–exploitation bandit problem. In the exploration–exploitation bandits, the agent searches for a policy that maximises the *cumulative reward* $S(\mathcal{A}, T)$ for a given time horizzon $T \in \mathbb{R}$. As defined in Equation 2.1, cumulative reward is the

expected sum of rewards accumulated by the agent $\mathcal{A}$ till time T. A policy[1] is *asymptotically consistent* [Robbins, 1952] if it asymptotically tends to choose the arm with maximum expected reward $\mu^* \in \mathbb{R}$, i.e.,

$$\lim_{T\to\infty} \frac{S(\mathcal{A}, T)}{T} = \mu^*. \tag{5.1}$$

The *cumulative regret* $R(\mathcal{A}, T)$ [Lai and Robbins, 1985] is the expected deficit of reward that the gambler faces as she plays the present sequence of arms instead of the optimal arm a^*. Following Equation 2.1, we express the cumulative regret for a K-arm stochastic bandit as

$$R(\mathcal{A}, T) \triangleq \mu^* \times T - \sum_{a=1}^{K} \left[\mu^a \times \mathbb{E}[n_a(T)]\right],$$

where $\mu^a \in \mathbb{R}$ is the expected rewards of arm a, and $n_a(T)$ is the number of times an arm a is played till time T. [Lai and Robbins, 1985] proved that for all asymptotically consistent algorithms satisfying $R(\mathcal{A}, T) = o(T^c)$ for a $c \in [0, 1)$, the cumulative regret increases in $\Theta(\log T)$ i.e. logarithmically with time T. Such algorithms are called *asymptotically efficient*. We refer to the discussion in Section 2.1.1 for further discussion of the optimality of bandit algorithm and regret analysis.

Based on the Lai-Robbins bound, [Auer et al., 2002] extensively studied the upper confidence bound (UCB) family of algorithms. These algorithms operate on the philosophy of optimism in face of uncertainty. They compute the upper confidence bounds of each of the arm's distributions in a frequentist way and choose the one with the maximum upper confidence bound optimistically expecting that one to be the arm with maximum expected reward. Later on, UCB family of algorithms was analysed and improved to propose algorithms such as KL-UCB [Garivier and Cappé, 2011] and DMED [Honda and Takemura, 2011].

[1]Here, the terms agent and policy are often used interchangeably as they stand synonymously in the context of decision making in bandits.

Frequentist approaches implicitly assume a 'true' parametrization $\theta_a^{\text{true}} \in \mathbb{R}$ for each of the K reward distributions $f_a(X)$. [2] In contrast, Bayesians model the uncertainty on the parameters using another probability distribution $B(\theta_1, \ldots, \theta_k)$ [DeGroot, 2005; Scott, 2010]. B is called the *belief distribution*. Bayesian algorithms begin with a prior belief distribution $B_0(\theta_1, \ldots, \theta_k)$ over the parameters. They iteratively compute a posterior distribution that minimises the Bayesian regret $\text{BR}(\mathcal{A}, T, B_0)$ for the given prior belief and time horizon T. [3] [Gittins, 1979] proposed an algorithm that computes at each step a set of indices for the arms. This is called the Gittins index algorithm. Gittins index is proven to be optimal for discounted Bayesian bandits with Bernoulli rewards. Explicit computation of the indices is not always tractable and they do not provide clear insights into what they look like and how they change as sampling proceeds [Nino-Mora, 2011]. Thus, researchers developed approximation algorithms [Lai, 1988] and sequential sampling schemes like Thompson sampling [Thompson, 1933] for the Bayesian bandits. At any iteration, Thompson sampling samples k parameter values from the belief distributions and chooses the arm that has maximum expected reward for them. [Kaufmann et al., 2012a] also proposed a Bayesian analogue of the UCB algorithm. Unlike the original, it uses belief distributions to keep track of arm uncertainty and update them using Bayes' theorem, computes UCBs for each arm using the belief distributions, and chooses the arm accordingly.

Pure exploration bandit problem. In this variant of the bandit problem, the agent aims to learn the reward distributions of the arms. In a parametric setup, it is analogous to learning the parameters of the reward distributions upto a given accuracy. In a non-parametric setup or even in the frequentist approaches, it is formulated as learning the moments of the reward distributions such as expectations, and variances, upto a given accuracy. Bubeck et al. formulated this notion as minimisation of the simple regret rather than cumulative regret [Bubeck et al., 2009]. *Simple regret* at

[2] Here, X represents the random variable that corresponds to the reward, and $a \in \{1, \ldots, K\}$ indicates to an arm.

[3] TBayesian regret $\text{BR}(\mathcal{A}, T, B_0)$ is defined by Equation 2.6 in Definition 4.

time n is the expected difference between the maximum achievable reward X_{A^*} and the sampled reward X_{A_n}. Bubeck et al. proved that, for Bernoulli bandits, if an algorithm solving exploration–exploitation bandit achieves an upper bound on regret, it cannot reduce the expected simple regret by more than a fixed lower bound. This establishes the fundamental difference between exploration–exploitation bandits and pure exploration bandits. Audibert and Bubeck identified the pure exploration problem as *best arm identification* problem. This approach led to the Successive Rejects algorithm under fixed budget constraints [Audibert and Bubeck, 2010], and LUCB family of algorithms [Kaufmann and Kalyanakrishnan, 2013]. Existing frequentist algorithms [Audibert and Bubeck, 2010; Bubeck et al., 2013; Kaufmann and Kalyanakrishnan, 2013] do not provide an intuitive framework to unify both the pure exploration and the exploration–exploitation scenarios. Still these algorithms validate exploration as the most fundamental component of the bandit problems, and the need of effective learning through exploration for efficient decision making in bandits.

Two-phase reinforcement learning. Two-phase reinforcement learning problems append the exploration–exploitation problem after the pure exploration problem. The agent gets an initial phase of pure exploration for a given window. In this phase, the agent collects more information about the underlying reward distributions. Following this, the agent goes through the exploration–exploitation phase. In this phase, it solves the exploration–exploitation problem and focuses on maximising the cumulative reward. This setup is perceivable as an initial online model building or 'training' phase followed by an online problem solving or 'testing' phase. This problem setup often emerges in applications [Faheem and Senellart, 2015], where the decision maker explores for an initial phase to create a knowledge base and another phase to take decisions by leveraging this pre-built knowledge base. In applications, this way of beginning the exploration–exploitation is called a warm start. Thus, two-phase reinforcement learning builds a middle ground between commonly used model-free and model-dependent approaches [Sutton and Barto, 1998] in MDPs.

Formally, this knowledge-base is a prior distribution built from the agent's experience. Since Bayesian methods naturally accommodate and leverage prior distributions, Bayesian formulation provide the scope to approach this problem without any modification. Putta and Tulabandhula approached this problem by amalgamating two sampling techniques, Posterior Sampling for Pure Exploration (PSPE), and Posterior Sampling for Reinforcement Learning (PSRL) [Osband et al., 2013], for episodic fixed horizon MDPs [Dann and Brunskill, 2015]. PSPE uses Bayesian update to create a posterior distribution for the reward distribution of a policy. Then, PSPE samples from the distribution in order to evaluate the policies. These two steps are performed iteratively for the initial pure exploration phase. PSRL [Osband et al., 2013] is an extension of Thompson sampling for episodic MDPs. Unlike Thompson sampling, they also use Markov chain Monte Carlo method for creating the posteriors corresponding to each of the policies. Though the amalgamation of these two methods for the two phase problems in episodic MDPs perform reasonably, they lack a reasonable unified structure attacking the problem and a natural cause to pipeline them.

We use this formulation of two-phase reinforcement learning problems to test the power of unification of the proposed framework, and its stability of transition from the pure exploration bandits to the exploration–exploitation bandits. The variant of the two-phase reinforcement learning for bandits is investigated in Section 5.5. We call it two-phase bandit problem for further reference.

5.3 Bandits: Problem Formulation

Though we have formulated the finite-arm stochastic bandit problem in Section 2.1.1, we illustrate the formulation, the notations, and the assumptions required for further discussion.

We consider a finite number, $K > 1$, of independent arms. An arm a corresponds to a reward distribution $f_a(X)$. We assume that the reward distributions belong to the same parametric family of probability distributions, such as Bernoulli, Gaussian,

and so on. Thus, each distribution $f_a(X)$ is equivalent to the distribution $f_{\theta_a}(X)$ with parameter $\theta_a \in \Theta$.[4] For example, for Bernoulli bandits, the reward distribution of arm a is $f_{\theta_a}(X) \triangleq {\theta_a}^X (1-\theta_a)^{(1-X)}$ where the reward X is either 0 or 1. We assume that the parametric family of reward distributions is known to the algorithm but the 'true' parametrisation is unknown. For example, the algorithm knows that the reward distributions belong to Bernoulli family but not the true values of the parameters $[\theta_a^{true}]_{a\in\{1,\dots,K\}}$. Following the bandit literature [Lai and Robbins, 1985], we assume the expectations of the reward distributions $\mu_a(\theta_a) \triangleq \int X f_{\theta_a}(X)$ are well-defined and finite for all the K arms. We also assume that there exists an optimal arm a^* with $\mu^* = \max_a \mu_a$. Thus, the optimal policy OPT is playing the arm a^* from the beginning to the time T.

The agent sequentially chooses an arm $A_t \in \{1,\dots,K\}$ at each time step $t \in [T]$ that generates a sequence of rewards $[X_t]_{t=1}^T$. Drawing an arm is termed as an *action* and the set of all actions i.e, $\mathcal{A} = \{1,2,\dots,K\}$ is called the *action space*. The algorithm computes a *policy* that sequentially draws a set of arms depending on her previous actions, observations and intended goal. The algorithm does not know the 'true' parameters of the arms $\{\theta_a^{\text{true}}\}_{a=1}^K$ a priori. Thus, the algorithm try to estimate the parameters in order to take more informed decisions. The uncertainty over the estimated parameters $\{\theta_a\}_{a=1}^K$ is represented using a probability distribution $B(\theta_1,\dots,\theta_K)$. Following the Bayesian terminology, call $B(\theta_1,\dots,\theta_k)$ the *belief distribution*. In the Bayesian approach, the algorithm starts with a prior belief distribution $B_0(\theta_1,\dots,\theta_k)$ [Jaynes, 1968]. The actions taken and rewards obtained by time t create the history of the bandit process, $\mathcal{H}_t \triangleq [(A_1,X_1),\dots,(A_{t-1},X_{t-1})]$. This history $\mathcal{H}_t$ is used to sequentially update the belief distribution over the parameters as $B_t(\theta_1,\dots,\theta_K) \triangleq \mathbb{P}(\theta_1,\dots,\theta_K \mid \mathcal{H}_t, B_0)$. We define the space consisting of all such distributions over the estimated parameters $\{\theta_j\}_{j=1}^k$ as the *belief space* $\mathcal{B} \triangleq \{B(\theta_1,\dots,\theta_K)\}$.

[4] $\Theta \subset \mathbb{R}^d$ is the space of parameters. The parameters can be scalar, or a d-dimensional real-valued vector.

Following the bandit literature, we assume the arms to be independent. Thus, the belief distribution over the parameters is decomposed as the product of belief distributions over each of the parameters.

Assumption 1 (Independence of Arms)**.** *At any time $t \in [T]$, the parameters $\{\theta_a\}_{a=1}^K$ are drawn independently from K belief distributions $\{b_t^a(.)\}_{a=1}^K$, such that*

$$B_t(\theta_1, \ldots, \theta_K) = \prod_{a=1}^{K} b_t^a(\theta_a) \triangleq \prod_{a=1}^{K} \mathbb{P}(\theta_a \mid \mathcal{H}_t). \tag{5.2}$$

Though Assumption 1 is followed throughout this paper, it is not essential to develop the framework BelMan relies on. However, it is assumed to make calculations easier. We assume the algorithm to perform Bayesian update of belief distributions.

Assumption 2 (Bayesian Evolution)**.** *When conditioned over $\{\theta_a\}_{a=1}^K$ and the choice of arm A_t, the sequence of rewards $[X_1, \ldots, X_t]$ is jointly independent. Thus, the Bayesian update at time t is given by*

$$b_{t+1}^a(\theta_{A_t}) \propto f_{\theta_{A_t}}(X_t) b_t^{A_t}(\theta_{A_t}) \tag{5.3}$$

if the arm A_t is drawn and a reward X_t is obtained. For all other arms, the belief remains unchanged at time t.

Furthermore, the reward distributions and the belief distributions are assumed to be smooth with respect to the corresponding parameters. Mathematically, the reward distribution of arm a i.e. $f_{\theta_a}(X) : \mathbb{R} \to [0, 1]$ is smoothly differentiable and a bijection with respect to the parameter $\theta_a \in \Theta$. Similarly, the belief distribution of arm a i.e. $b_{\eta_a}(\theta_a) : \Theta \to [0, 1]$ is smoothly differentiable and a bijection with respect to the parameter $\eta_a \in \mathbb{R}^{d'}$. η_a can be a real-valued scalar or vector depending on the belief distribution. These assumptions provide us a glitch-free condition to develop the information geometric methodology in the following section.

5.4 Methodology

In this section, we formulate the bandit problem in terms of belief-reward distributions and define the belief-reward manifold. Following this, we propose an alternating information projection scheme, BelMan, on the belief-reward manifold. In this context, we construct pseudobelief-reward and focal distributions. Finally, we instantiate BelMan for the exponential family of reward distributions.

5.4.1 A Primer on Information Geometry

Before delving into the construction of belief-reward manifolds, we provide a brief overview of manifold and statistical manifold.

Manifold

A *manifold* [Lang, 2006] is a space which is locally Euclidean. We get a neighbourhood $\mathcal{N}_P$ around every point P in a manifold that is topologically equivalent to the open unit ball in an Euclidean space, $\mathbb{R}^d$. d is called the *dimension* of the manifold. Thus, a manifold gives us a structure to locally transform the tools of Euclidean space, like distance, calculus etc, to a general space and vice versa. In order to accomplish this, we associate a d-dimensional vector, $\boldsymbol{\theta}_P$, of real-valued parameters to each point P of the space. Parameters associated with each point are called the *local co-ordinates* of the manifold. Equivalently, they can be thought of as a collection of d-mappings from each point P of a neighbourhood $\mathcal{N}_P$ to the Euclidean space $\mathbb{R}^d$. In order to use the mathematical tools of Euclidean spaces consistently, the manifold is divided into a collection of neighbourhoods. Each of the neighbourhoods has local co-ordinate mappings which are continuous and have continuous inverse mappings. Thus, the manifold would be topologically compact and it would be possible to move from one neighbourhood to the other continuously. Each of these neighbourhoods are called *chart* and the collection of charts is called an *atlas*. If each of such co-ordinate maps and their inverses are differentiable, we call the manifold *smooth*.

Statistical Manifold

A *statistical manifold* [Amari and Nagaoka, 2007] is a smooth manifold $\mathcal{M}$ where each point represents a probability distribution. The statistical structure of these manifolds provide them with a Riemannian metric G, called Fisher information metric, and also a pair of torsion-free connections ∇ and ∇^* [Lauritzen, 1987]. The metric and the connections together define the notion of distance and movements on the statistical manifold. This structure allows us to introduce tools of Riemannian geometry and to leverage them for statistical operations. A stochastic process evolves to provide a set of probability distributions in the statistical manifold that follow a certain dynamics. If we use a parametric family of distributions with d-parameters to represent them, we call this set of distributions a statistical or parametric model. Thus, a *statistical model* is formally defined as a subspace of the statistical manifold consisting of the points emerged from an underlying process and represented by a d-dimensional local co-ordinates. In order to maintain consistency and avoid singularity in calculation, we assume that the manifolds of our interest are defined over a well-defined support. We formalise this assumption as follows,

Assumption 3 (Restriction to support set)**.** *The statistical manifold $\mathcal{M}$ is a collection of probability distributions P defined on a fixed support set $Supp(P) \triangleq \{e : P(e) > 0\}$ and $Supp(P) \neq \emptyset$.*

Following this, we define the support of the statistical manifold $\mathcal{M}$ as $Supp(\mathcal{M}) \triangleq \cup_{P \in \mathcal{M}} \{Supp(P) \mid Supp(P) \neq \emptyset\}$. Further mathematical formulations and properties of manifolds and statistical manifolds are described in the appendix.

KL-divergence on the Manifold

Kullback-Liebler divergence (or KL-divergence) [Kullback, 1997] is a pre-metric measure of dissimilarity between two probability distributions.

Definition 6 (KL-Divergence)**.** If there exist two probability measures P and Q defined over a support set $Supp(P)$, and P is absolutely continuous with respect to Q [5], we define the KL-divergence between them as

$$D_{\mathrm{KL}}\left(P\|Q\right) \triangleq \int_{Supp(P)} \log \frac{\mathrm{d}P}{\mathrm{d}Q} \mathrm{d}P = \mathbb{E}_P \left[\log \frac{\mathrm{d}P}{\mathrm{d}Q}\right].$$

$\frac{\mathrm{d}P}{\mathrm{d}Q}$ is the Radon-Nikodym derivative of P with respect to Q [Durrett, 2010].

KL-divergence is the pre-metric between two distributions P and Q in a probability space because

i. KL-divergence is non-negative. $D_{\mathrm{KL}}\left(P\|Q\right) \geq 0$ for all P and Q. The equality holds if $P(e) = Q(e)$ for all $e \in Supp(P) = Supp(Q)$.

ii. KL-divergence is not symmetric. $D_{\mathrm{KL}}\left(P\|Q\right) \neq D_{\mathrm{KL}}\left(Q\|P\right)$ for all $P \neq Q$.

Since $D_{\mathrm{KL}}\left(P\|Q\right)$ represents the expected information lost if P is encoded using Q, it is also called *relative entropy*. Depending on the applications, P acts as the representative of 'true' underlying distribution obtained from observations or data or natural law, and Q represents the model or approximation of P. For two probability density functions $P(e)$ and $Q(e)$ defined over the same support set E, the KL-divergence can be rewritten as

$$D_{\mathrm{KL}}\left(P\|Q\right) = \int_{e\in E} P(e) \log \frac{P(e)}{Q(e)} de = H(P,Q) - h(P). \tag{5.4}$$

Here, $h(P) \triangleq -\int_{e\in E} P(e)\log P(e)de$ is entropy of P and $H(P,Q) \triangleq -\int_{e\in E} P(e)\log Q(e)de$ is the cross-entropy between P and Q. Thus, KL-divergence is representable as the difference between cross-entropy of P and Q, and its self-entropy. Since entropy is the measure of uncertainty of a distribution, KL-divergence acts as the difference between the uncertainty of the distribution itself and as much uncertainty can be captured using the model Q. From an information-theoretic perspective, KL-divergence emerges

[5] P is absolutely continuous with respect to Q if $Q(e) \neq 0$ for any e in the support set of P.

as the natural divergence function on the statistical manifold when we analyse the dynamics of the entropy function on the manifold. Except that, any general α-divergence function [Lauritzen, 1987] on the statistical manifold is a convex combination of ± 1-divergences. Mathematically, for $\alpha \in (-1, +1)$,

$$\begin{aligned} D^{(\alpha)}(P\|Q) &\triangleq \frac{1+\alpha}{2} D^{(+1)}(P\|Q) + \frac{1-\alpha}{2} D^{(-1)}(P\|Q) \\ &= \frac{1+\alpha}{2} D_{\mathrm{KL}}\left(Q\|P\right) + \frac{1-\alpha}{2} D_{\mathrm{KL}}\left(P\|Q\right). \end{aligned} \tag{5.5}$$

From a manifold perspective, the divergence function for the ± 1-connections and a convex mixture of D_{KL} divergences form the general notion of movement on a statistical manifold [Eguchi, 1992; Matumoto, 1993].

Exponential Family

Use of KL-divergence as a divergence measure on the statistical manifolds and also the issue of representation of a random variable using sufficient statistics provoked the study of the exponential family of distributions. Interesting properties of exponential family distributions, such as existence of finite representation of sufficient statistics, convenient mathematical form, and existence of moments, provided them a central stage in the field of mathematical statistics [Darmois, 1935; Koopman, 1936].

The *exponential family* [Brown, 1986] is a class of probability distributions which is defined by a set of *natural parameters* $\omega(\theta)$ and a *sufficient statistics* $T(X)$ of the random variable X as follows:

$$f_\theta(X) \triangleq g(X) \exp\left(\langle \omega(\theta), T(x) \rangle - A(\theta)\right).$$

Here, $g(X)$ is the *base measure* on reward X and $A(\theta)$ is called the *log-partition function*. The exponential family includes the majority of the distributions found in the bandit literature such as Bernoulli, beta, Gaussian, Poisson, exponential, and chi-squared. For $T(X) = X$, the log-partition function is logarithm of the Laplace transform of the base measure.

Example 3. *Bernoulli distribution with probability of success* $\theta \in (0,1)$ *is defined as*

$$f_\theta(X) \triangleq \text{Ber}(\theta) = \theta^X (1-\theta)^{(1-X)} \\ = \exp\left(X \log\left(\frac{\theta}{1-\theta}\right) + \log(1-\theta)\right)$$

for $X \in \{0,1\}$. *Here, the base measure* $g(x)$ *is* 1. *The sufficient statistics is* $T(X) = X$. *The natural parameter is* $\omega(\theta) = \log\left(\frac{\theta}{1-\theta}\right)$. *The log-partition function is* $A(\theta) = -\log(1-\theta) = \log(1+\exp(\omega))$.

We choose the exponential family to instantiate our framework not only because of its wide range and applicability but also due to its well behaving Bayesian and information geometric properties. From a sampling and uncertainty representation point of view, the exponential family is useful because of its finite representation of sufficient statistics. Specifically, sufficient statistics of exponential family can represent any arbitrary number of independent identically distributed samples using a finite number of variables [Koopman, 1936]. This keeps the uncertainty representation tractable for exponential family distributions.

From a Bayesian point of view, the useful property of the exponential family is the existence of *conjugate distributions* which also belong to this family [Brown, 1986]. Two parametric distributions $f_{\boldsymbol{\theta}}(x)$ and $b_{\boldsymbol{\eta}}(\boldsymbol{\theta})$ are conjugate if the posterior distribution $\mathbb{P}(\boldsymbol{\theta}|x)$ formed by multiplying them has the same form as $b_{\boldsymbol{\eta}}(\boldsymbol{\theta})$. Mathematically, the conjugate distribution of the distribution of Equation 5.4.1 is given by $b_{\boldsymbol{\eta}}(\boldsymbol{\theta}) \triangleq \mathbb{P}(\boldsymbol{\theta}|\boldsymbol{\eta}, v) = f(\boldsymbol{\eta}, v)\exp(\langle \boldsymbol{\eta}, \boldsymbol{\theta}\rangle - vA(\boldsymbol{\theta})) = f(\boldsymbol{\eta}, v)g(\boldsymbol{\theta})^v \exp(\langle \boldsymbol{\eta}, \boldsymbol{\theta}\rangle)$. Here, η is the parameter of the conjugate prior and $v > 0$ corresponds to the effective number of observations that the prior contributes. Thus, if the reward distribution belongs to the exponential family, the belief distribution is represented as: $b_{\boldsymbol{\eta}}(\theta) \triangleq h(\theta)\exp\left(\langle \eta, T(\theta)\rangle - A(\eta)\right)$ with the natural parameters $\eta \in \mathbb{R}^{d'}$.

From information geometric point of view, exponential family distributions are flat with respect to KL-divergence [Amari and Nagaoka, 2007]. Thus, both information and reverse information projections [Csiszár, 1984] that we would use in BelMan are

well-defined and unique. Thus, at each iteration, we obtain an optimal and unambiguous computation of the decision variables of BelMan. [Amari and Nagaoka, 2007] also stated that the necessary and sufficient condition for a parametric probability distribution to have an efficient estimator is that the distribution belongs to the exponential family and has an expectation parametrisation. Thus, working with exponential family distributions implicitly supports the well-defined nature and possibility of getting an efficient estimation.

5.4.2 Belief-reward Manifold

As we have described the components of information geometry that we are going to use, now we construct the statistical manifold structure for the stochastic bandits. There are two layers of uncertainty involved in this present formulation of bandits. The first layer involves the reward which is unknown due to the stochastic behaviour of the reward generation of each arm. Thus, reward is modelled as a random variable X, and for each of the arms, a reward distribution $f_{\theta_a}(X)$ is assumed to represent the corresponding uncertainty. This layer of uncertainty is inherent to the stochastic bandit problem. This lead us to work with the expected value of accumulated reward than the random variable itself. In Section 2.1, we discuss this reasoning in the context of existing bandit literature. The second layer of uncertainty involves the parameters of the reward distributions. Since the reward distributions of the arms are not completely known, the algorithm tries to estimate the reward distributions, and in turn, the parameters to gain more information about the available choices. Thus, the parameters of the reward distributions θ_a's operate also as a set of random variables. The corresponding uncertainty arises due to the lack of information about them, and the policy that the algorithm computes to play the arms. This uncertainty is captured by a set of belief distributions $b_{\eta_a}(\theta_a)$ over each of the parameters of the arms. These two random variables and their corresponding distributions are estimated and leveraged to represent the uncertainty throughout a bandit process. Hence, amalgamating

both of them into a single mathematical structure is the first step to represent the uncertainty of learning and reward generation in a bandit process.

We propose to use the joint distributions $\mathbb{P}(X, \theta)$ on reward X and parameter θ in order to represent the uncertainties of partial information about the reward distributions along with the stochastic nature of reward. We call $\mathbb{P}(X, \theta)$'s the *belief-reward distributions*.

Definition 7 (Belief-Reward Distribution)**.** The joint distribution $\mathbb{P}_t^a(X, \theta_a)$ on reward X and parameter θ_a for the arm a at time t is defined as the *belief-reward distribution*.

$$\mathbb{P}_t^a(X, \theta_a) \triangleq \frac{b_t^a(\theta_a) f_{\theta_a}(X)}{\int\limits_{X \in \mathbb{R}} \int\limits_{\theta_a \in \Theta} b_t^a(\theta_a) f_{\theta_a}(X) \mathrm{d}\theta_\mathrm{a} \mathrm{dX}} = \frac{1}{Z} b_n^j(\theta_a) f_{\theta_a}(X). \tag{5.6}$$

Here, $Z \triangleq \int\limits_{X \in \mathbb{R}} \int\limits_{\theta_a \in \Theta} b_t^a(\theta_a) f_{\theta_a}(X) \mathrm{d}\theta_\mathrm{a} \mathrm{dX}$ is the corresponding normalisation factor.

If $f_\cdot(X)$ is a smooth function of θ_a for all a, the space of all reward distributions constructs a smooth statistical manifold [Amari and Nagaoka, 2007], $\mathcal{R}$. We call $\mathcal{R}$ the *reward manifold*. If belief B over the reward distribution parameters is a smooth function of its parameters, the belief space $\mathcal{B}$ constructs another statistical manifold. We call $\mathcal{B}$ the *belief manifold* of the multi-armed bandit process. Assumption 1 implies that the belief manifold $\mathcal{B}$ is a product of K statistical manifolds $\mathcal{B}^a \triangleq \{b^a(\theta_a)\} = \{b_{\eta_a}(\theta_a)\}$. Here, $\mathcal{B}^a$ is the statistical manifold of belief distributions for arm a. If the parametrisation η_a of the belief distributions is smooth and bijective, $\mathcal{B}^a$'s can be represented by a single manifold $\mathcal{B}_\theta$.

Lemma 5 (Belief-Reward Manifold)**.** *The set of belief-reward distributions* $\mathbb{P}(X, \theta_a)$ *for all* a *constructs a manifold* $\mathcal{B}_\theta\mathcal{R}$*, such that* $\mathcal{B}_\theta\mathcal{R} = \mathcal{B}_\theta \times \mathcal{R}$*. We refer to it as the* belief-reward manifold.

The Bayesian belief update after each iteration is a movement on the belief manifold from a point b_n^j to another point b_{n+1}^j *with maximum information gain* from the obtained reward. Such movement from prior to posterior belief-reward distribution maximises the gain in KL-divergence [Kullback, 1997] after incorporating the new

reward. Thus, the belief-reward distributions of the played arms evolve to create a set of trajectories on the belief-reward manifold. The goal of pure exploration is to control such trajectories collectively such that after a long enough time each of the belief-rewards accumulate enough information to resemble the 'true' reward distributions well enough. The goal of exploration–exploitation is to gain enough information about the 'true' reward distributions while increasing the cumulative reward in the path, i.e, by inducing a bias towards playing the arms with higher expected rewards.

From a manifold perspective, a convex mixture of D_{KL} divergences form the general notion of movement on the belief-reward manifold. Thus, KL-divergence between two belief-reward distributions is an effective and natural quantifier of movement, and also of information accumulation during Bayesian update. Hence, for updating the beliefs in an optimal manner, and to decrease the uncertainty, we have to represent the observations using a knowledge-base, and to minimise the KL-divergence between the knowledge-base and other distributions respectively. If $\mathcal{P}$ are the candidate belief-reward distributions of the arms formed by accumulation of actions and rewards, and $\mathcal{Q}$ is the set of their summarised representation inducing exploration–exploitation, the alternating minimisation scheme looks for the most succinct representation $\mathcal{Q}$ of the knowledge and the exploitation bias while choosing such arms whose belief-reward distributions resemble their true reward distributions as much as possible.

5.4.3 Pseudobelief: Summarising the Explored Knowledge

As we argue that exploration is the fundamental phenomenon in all the variants of bandit problems, we hereby leverage the belief-reward manifold structure to develop pseudobelief-reward distribution to facilitate exploration and learning.

In order to control the exploration, the algorithm has to represent a summary of the collective knowledge on the belief-reward distributions of the arms. Since belief-reward distribution of each arm is a point on the belief-reward manifold, geometrically their barycenter on the belief-reward manifold represents a valid summary of the uncertainty over all the arms [Agueh and Carlier, 2011]. Since the belief-reward manifold

is a statistical manifold, we obtain from information geometry that this barycenter is the point on the manifold that minimises the sum of KL-divergences from the belief-rewards of all the arms [Barbaresco, 2013; Amari and Nagaoka, 2007]. We refer to this minimising belief-reward distribution as the pseudobelief-reward distribution of all the arms.

Definition 8 (Pseudobelief-Reward Distribution)**.** A *pseudobelief-reward distribution* $\bar{\mathbb{P}}_t(X,\theta)$ is a point in the belief-reward manifold that minimises the sum of KL-divergences from the belief-reward distributions $\mathbb{P}_t^a(X,\theta_a)$ of all the arms.

$$\bar{\mathbb{P}}_t(X,\theta) \triangleq \underset{\mathbb{P}\in\mathcal{B}_\theta\mathcal{R}}{\arg\min} \sum_{a=1}^{K} D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta_a)\|\mathbb{P}(X,\theta)\right). \tag{5.7}$$

In Theorem 10, we prove existence and uniqueness of the pseudobelief-reward for given belief-reward distributions. This proves the pseudobelief-reward to be an unambiguous representative of collective knowledge. We also prove in Corollary 2 that the pseudobelief-reward distribution $\bar{\mathbb{P}}_t$ is the projection of the average belief-reward distribution $\hat{\mathbb{P}}_t(x,\theta) = \sum_a \mathbb{P}_t^a(X,\theta_a)$ on the belief-reward manifold. This result validates the claim of pseudobelief-reward as the summary of the belief-rewards of all the arms.

Properties of Pseudobelief–reward: Existence, Uniqueness and Consistency

In order to establish pseudobelief–reward as a valid knowledge-base for all the arms, we have to prove that it exists uniquely and its parameters can be consistently estimated. We prove these propertied in Theorem 10, Corollary 2, and Theorem 11. The proofs require two assumptions. Firstly, the belief–reward manifold can be covered by a single chart. Let us express the belief-reward manifold as a triad of the space and the connections $(\mathcal{B}_\theta\mathcal{R}, \nabla, \nabla^*)$. If η and η' are two coordinate mappings of the belief-reward manifolds that yields the ∇- and ∇^*-affine coordinates, then they cover the whole space i.e. $Domain(\eta) = Domain(\eta') = \mathcal{B}_\theta\mathcal{R}$. This implies that every belief–reward distribution in the belief-reward manifold is a bijective function of parameters. Secondly, there exist unique ∇-geodesics between any two points of the belief–reward

manifold. This implies that there exists a unique ∇-geodesic (and dually ∇^*-geodesic) connecting any two points P and Q in the belief-reward manifold, and all of it lies in $\mathcal{B}_\theta\mathcal{R}$. Thus, for any two coordinate mappings, $\eta(\mathcal{B}_\theta\mathcal{R})$ and $\eta'(\mathcal{B}_\theta\mathcal{R})$ are convex subsets of an Euclidean space $\mathbb{R}^{d'}$. This implies that the belief-reward manifold is ∇-convex, and a convex divergence function between any two belief–reward distributions is uniquely exist. Instead of having such generic requirements, we represent our proofs in form of the exponential family distributions due to ease of presentation and our limited interest.

In Theorem 10, we prove that the pseudobelief-reward distribution at time t is described by the expectation coordinate which is the average of expectation coordinates of the belief-reward distributions of the arms at that time. Let us denote the expectation parameter of each of the belief-reward distributions of the arms arms at time t by μ_t^a, and the expectation parameter for the pseudobelief-reward at time t by $\hat{\mu}_t$.

Theorem 10. *For a given set of belief-reward distributions $\{\mathbb{P}_t^a\}_{a=1}^K$ defined on the same support set, having a finite expectation, and belonging to the exponential family of distributions, $\bar{\mathbb{P}}_t$ is uniquely defined such that the expectation parameter satisfies $\hat{\mu}_t = \frac{1}{K}\sum_{a=1}^K \mu_t^a(\theta)$.*

Proof. For belief–reward distributions $\mathbb{P}_t^a = \frac{1}{Z_t^a} f_\theta(X) b_{\eta_t^a}(\theta)$ and $\mathbb{P} = \frac{1}{Z} f_\theta(X) b_\eta(\theta)$, the KL-divergence is defined as

$$
\begin{aligned}
D_{\mathrm{KL}}\left(\mathbb{P}_t^a \| \mathbb{P}\right) &= \int_\theta \int_X \mathbb{P}_t^a(X,\theta) \log \frac{\mathbb{P}_t^a(X,\theta)}{\mathbb{P}(X,\theta)} \mathrm{dX}\mathrm{d}\theta \\
&= \int_\theta \int_X f_\theta(X) b_{\eta_t^a}^a(\theta) \log \frac{b_{\eta_t^a}^a(\theta)}{b_\eta(\theta)} \mathrm{dX}\mathrm{d}\theta \\
&= \int_\theta b_{\eta_t^a}^a(\theta) \log \frac{b_{\eta_t^a}^a(\theta)}{b_\eta(\theta)} \left[\int_X f_\theta(X) \mathrm{dX}\right] \mathrm{d}\theta \\
&= \int_\theta b_{\eta_t^a}^a(\theta) \log \frac{b_{\eta_t^a}^a(\theta)}{b_\eta(\theta)} \mathrm{d}\theta \\
&= \mathbb{E}_{b_t^a}\left[\langle \eta_t^a, \omega(\theta)\rangle - \Psi(\eta_t^a) - \langle \eta, \omega(\theta)\rangle + \Psi(\eta)\right] \\
&= \langle \eta_t^a - \eta, \mu_t^a \rangle - \Psi(\eta_t^a) + \Psi(\eta).
\end{aligned}
$$

Thus, the objective function that $\bar{\mathbb{P}}$ minimises is given by

$$F(\mathbb{P}) \triangleq \frac{1}{K}\sum_{a=1}^{K} D_{\mathrm{KL}}\left(\mathbb{P}_t^a \| \mathbb{P}\right) = \frac{1}{K}\sum_{a=1}^{K}\langle \eta_t^a - \eta, \mu_t^a\rangle - \frac{1}{K}\sum_{a=1}^{K}\Psi(\eta_t^a) + \Psi(\eta). \tag{5.8}$$

Since the exponential family distributions are dually flat [Amari and Nagaoka, 2007], we get a unique expectation parametrisation μ_t^a of the belief distributions for a given natural parametrisation η_t^a. The expectation parameter is defined as $\mu_t^a \triangleq \mathbb{E}_{b_t^a}[\omega(\theta)] = \nabla_\eta \Psi(\eta_t^a)$. μ_t^a dually expresses a natural parametrisation. Mathematically, $\eta_t^a = \nabla_{\mu_t^a}(\langle \eta_t^a, \mu_t^a\rangle - \Psi(\eta_t^a)) = \nabla_{\mu_t^a}\Phi(\mu_t^a)$. $\Psi(\eta_t^a)$ and $\Phi(\mu_t^a)$ are log-normalisers under two parametrisations, and are convex conjugate to each other. If we define $\hat{\mu}_t \triangleq \frac{1}{K}\sum_{a=1}^{K}\mu_t^a$, we get a unique natural parameter $\hat{\eta}_t$ as the dual of $\hat{\mu}_t$. This allows us to rewrite Equation 5.8 as

$$\begin{aligned} F(\mathbb{P}) &= [\langle \hat{\eta}_t - \eta, \hat{\mu}_t\rangle - \Psi(\hat{\eta}_t) + \Psi(\eta)] + \frac{1}{K}\sum_{a=1}^{K}[\langle \eta_t^a, \mu_t^a\rangle - \Psi(\eta_t^a) - \langle \hat{\eta}_t, \hat{\mu}_t\rangle + \Psi(\hat{\eta}_t)] \\ &= D_{\mathrm{KL}}\left(\mathbb{P}_{\hat{\mu}_t} \| \mathbb{P}\right) + \frac{1}{K}\sum_{a=1}^{K}\Phi(\mu_t^a) - \Phi(\hat{\mu}_t) \\ &\geq \frac{1}{K}\sum_{a=1}^{K}\Phi(\mu_t^a) - \Phi(\hat{\mu}_t). \end{aligned}$$

Since $D_{\mathrm{KL}}\left(\mathbb{P}_{\hat{\mu}_t} \| \mathbb{P}\right) = 0$ for $\mathbb{P} = \mathbb{P}_{\hat{\mu}_t}$, $F(\mathbb{P})$ reaches unique minimum $F(\mathbb{P}_{\hat{\mu}_t})$ for the belief–reward distribution with expectation parameter $\hat{\mu_t} \triangleq \frac{1}{K}\sum_{a=1}^{K}\mu_t^a$. Thus, for a given set of belief–reward distributions the pseudobelief–reward distribution $\bar{\mathbb{P}}_t(X, \theta) \triangleq \mathbb{P}_{\hat{\mu}_t}(X, \theta)$ is a unique distribution in belief–reward manifold. □

Hereby, we establish as a unique summariser of all the belief–reward distributions. Using this uniqueness proof, we show that the pseudobelief–reward distribution $\bar{\mathbb{P}}$ is projection of the average belief–reward distribution $\hat{\mathbb{P}} \triangleq \frac{1}{K}\sum_{a=1}^{K}\mathbb{P}^a$ on the belief–reward manifold.

Corollary 2. *The pseudobelief-reward distribution $\bar{\mathbb{P}}_t(X,\theta)$ is the unique point on the belief-reward manifold that has minimum KL-divergence from the distribution $\hat{\mathbb{P}}_t(X,\theta) \triangleq \frac{1}{K}\sum_{a=1}^{K}\mathbb{P}_t^a(X,\theta)$.*

Proof. KL-divergence from $\hat{\mathbb{P}}_t(X,\theta)$ to a belief–reward distribution $\mathbb{P}(X,\theta)$ is

$$\begin{aligned} D_{\mathrm{KL}}\left(\hat{\mathbb{P}}_t \| \mathbb{P}\right) &= D_{\mathrm{KL}}\left(\hat{\mathbb{P}}_t \| \bar{\mathbb{P}}_t\right) + \langle \hat{\eta}_t - \eta, \hat{\mu}_t \rangle - \Psi(\hat{\eta}_t) + \Psi(\eta) \\ &= D_{\mathrm{KL}}\left(\hat{\mathbb{P}}_t \| \bar{\mathbb{P}}_t\right) + D_{\mathrm{KL}}\left(\bar{\mathbb{P}}_t \| \mathbb{P}\right). \end{aligned}$$

Here, $\bar{\mathbb{P}}_t$ is the pseudobelief distribution with $\hat{\eta}_t$ and $\hat{\mu}_t$ as defined in Theorem 10. Since $\hat{\mathbb{P}}_t$ is a mixture of belief–reward distributions, it does not belong to the belief–reward manifold. Thus, $\hat{\mathbb{P}}_t \neq \bar{\mathbb{P}}_t$ and $D_{\mathrm{KL}}\left(\hat{\mathbb{P}}_t \| \bar{\mathbb{P}}_t\right) > 0$. Hence, $D_{\mathrm{KL}}\left(\hat{\mathbb{P}}_t \| \mathbb{P}\right)$ attends unique minimum for $\mathbb{P} = \bar{\mathbb{P}}_t$. □

5.4.4 Focal Distribution: Inducing Exploitative Bias

Creating a succinct pseudobelief-reward is essential for both pure exploration and exploration–exploitation but not sufficient for maximising the cumulative reward in case of exploration–exploitation. If a reward distribution having such increasing bias towards higher rewards is amalgamated with the pseudobelief-reward, the resulting belief-reward distribution provides a representation in the belief-reward manifold to balance the exploration–exploitation. Such a reward distribution is called the *focal distribution*. The product of the pseudobelief-reward and the focal distribution jointly represents the summary of explored knowledge and exploitation bias using a single belief-reward distribution. We refer to this as the *pseudobelief-focal-reward distribution-reward distribution* In this paper, we use $\exp\left(\frac{X}{\tau(t)}\right)$ with a time dependent and controllable parameter $\tau(t)$ as the reward distribution inducing increasing exploitation bias.

Definition 9 (Focal Distribution)**.** A *focal distribution* is a reward distribution of the form $L_t(X) \propto \exp\left(\frac{X}{\tau(t)}\right)$, where $\tau(t)$ is a decreasing function of $t \geq 1$. We term $\tau(t)$ the *exposure* of the focal distribution.

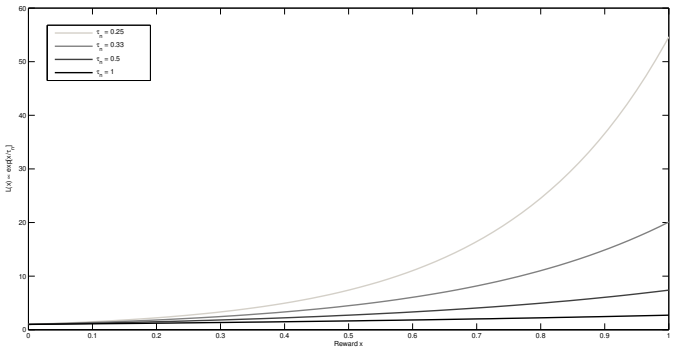


Figure 5.1: Evolution of the focal distribution over $X \in [0, 1]$ for $\tau(t) = 1, 0.5, 0.33$ and 0.25.

The focal distribution gradually concentrates on higher rewards as the exposure $\tau(n)$ decreases with time. We see this feature in Figure 5.1. Thus, it constrains using KL-divergence to choose distributions with higher rewards and induces the exploitive bias. Following the upper bounds of regret obtained in [Garivier and Cappé, 2011], we set the focal distribution to $\tau(t) \triangleq [\log(t) + C \times \log(\log(t))]^{-1}$ where C is a constant[6] in the exploration–exploitation bandit problem. As the exposure $\tau(t)$ decreases with t, the focal distribution gets more concentrated on higher reward values. For the pure exploration bandits, we set the exposure $\tau(t) = \infty$ to remove any bias towards higher reward values i.e, exploitation.

Pseudobelief-focal-reward distribution

We amalgamate the pseudobelief-reward and the focal distribution to jointly represent the summary of explored knowledge and exploitation bias using a single belief-reward distribution. Thus, the product distribution of the pseudobelief-reward distribution and the focal distribution constructs the *pseudobelief-focal-reward distribution*.

$$\bar{\mathbb{Q}}_t(X, \theta) \triangleq \frac{1}{\bar{Z}_t} \bar{\mathbb{P}}(X, \theta) \exp\left(\frac{X}{\tau(t)}\right)$$

[6]We choose the value $C = 15$ in the experimental evaluation.

Algorithm 20 BelMan

Input: Time horizon T, Number of arms K, Prior belief B_0.
for $t = 1$ **to** T **do**
    /* *I-projection* */
    Draw arm a_t such that

$$A_t = \underset{a\in\{1,\dots,K\}}{\arg\min}\, D_{\mathrm{KL}}\left(\mathbb{P}^a_{t-1}(X,\theta)\|\bar{\mathbb{Q}}_{t-1}(X,\theta)\right). \tag{5.9}$$

    /* *Accumulation of observables* */
    Sample a reward X_t out of $f_{\theta_{A_t}}$.
    Update the belief-reward distribution of A_t to $\mathbb{P}^{A_t}_t(X,\theta)$ using Bayes' theorem.
    /* *Reverse I-projection* */
    Update the pseudobelief-reward distribution to

$$\bar{\mathbb{Q}}_t(X,\theta) = \underset{\bar{\mathbb{Q}}\in\mathcal{B}_\theta\mathcal{R}}{\arg\min} \sum_{a=1}^{K} D_{\mathrm{KL}}\left(\mathbb{P}^a_t(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right). \tag{5.10}$$

end for

Here, $\bar{Z}_t = \int_{X\in\mathbb{R}}\int_{\theta\in\Theta}\bar{\mathbb{P}}(X,\theta)\exp\left(\frac{X}{\tau(t)}\right)\mathrm{d}\theta\ \mathrm{dX}$ is the normalisation factor. We use the pseudobelief-focal-reward distribution as the representative of explored knowledge and exploitation bias in our algorithm. Following Equation (5.7), we compute the pseudobelief-focal-reward distribution as

$$\bar{\mathbb{Q}}_t(X,\theta) \triangleq \underset{\bar{\mathbb{Q}}}{\arg\min} \sum_{a=1}^{K} D_{\mathrm{KL}}\left(\mathbb{P}^a_t(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right).$$

Now, we incorporate this in our framework of Algorithm 20 to alternatively minimise the KL-divergence between belief-reward distributions over parameters of the arms and the pseudobelief-reward distribution.

5.4.5 BelMan: An Alternating Projection Scheme

A bandit algorithm performs three operations in each step– chooses an arm, samples from the reward distribution of the chosen arm and incorporate the sampled reward to update the knowledge-base. BelMan (Algorithm 20) performs the first and the last

operations by alternately minimising the KL-divergence $D_{\mathrm{KL}}(.\|.)$ [Kullback, 1997] between the belief-reward distributions of the arms and the pseudobelief-focal-reward distribution-reward distribution. BelMan chooses to play the arm whose belief-reward incurs minimum KL-divergence with respect to the pseudobelief-focal-reward distribution. Following that, BelMan uses the reward collected from the played arm to do Bayesian update of the belief-reward and to update the pseudobelief-focal-reward distribution-reward distribution to the point minimising the sum of KL-divergences from the belief-rewards of all the arms. [Csiszár, 1984] geometrically formulated such minimisation of KL-divergence with respect to a participating distribution as a projection to the set of the other distributions. For a given t, the belief-reward distributions of all the arms $\mathbb{P}_t^a(X,\theta)$ form a set $\mathcal{P} \subset \mathcal{B}_\theta\mathcal{R}$ and the pseudobelief-focal-reward distribution-reward distributions $\bar{\mathbb{Q}}_t(X,\theta)$ constitute another set $\mathcal{Q} \subset \mathcal{B}_\theta\mathcal{R}$.

The algorithm is initially provided (Line 1) with a prior belief distribution $B_0(\boldsymbol{\theta}) \triangleq \prod_{a=1}^{K} b_{\eta_a}(\theta_a)$. This allows the initial construction of the belief-reward distributions of the arms and also the pseudobelief-focal-reward distribution. After this step, the algorithm has to choose an arm that maximises information or reward gain and then to incorporate this observation to create a better representation of knowledge and exploitation bias. BelMan does this by alternating information projection between the belief-rewards of the arms and the pseudobelief-focal-reward in the belief-reward manifold. Both I- and rI-projections are valid and well-defined if the KL-divergence between any two distributions in $\mathcal{P}$ and $\mathcal{Q}$ is defined and finite.

Assumption 4 (Absence of Singularities)**.** *If the distribution families $\mathcal{P}$ and $\mathcal{Q}$ are defined over the sets $Supp(\mathcal{P}) \triangleq \{e : P(e) > 0, \forall P \in \mathcal{P}\}$ and $Supp(\mathcal{Q}) \triangleq \{e : Q(e) > 0, \forall Q \in \mathcal{Q}\}$ respectively, none of the supports are empty and $Supp(\mathcal{P}) \subseteq Supp(\mathcal{Q})$.*

Definition 10 (I-Projection)**.** The *information projection* (or *I-projection*) of a distribution $\bar{\mathbb{Q}} \in \mathcal{Q}$ onto a non-empty, closed, convex set $\mathcal{P}$ of probability distributions, $\mathbb{P}^j$'s, defined on a fixed support set is defined by the probability distribution $\mathbb{P}^* \in \mathcal{P}$ that has minimum KL-divergence to $\mathcal{Q}$: $\mathbb{P}^* \triangleq \arg\min_{\mathbb{P}^a \in \mathcal{P}} D_{\mathrm{KL}}(\mathbb{P}^a \| \bar{\mathbb{Q}})$.

BelMan decides which arm to pull by an I-projection of the pseudobelief-focal-reward distribution onto the beliefs-rewards of each of the arms (Lines 3–4). This operation amounts to computing

$$\begin{aligned} A_t &\triangleq \arg\min_a D_{\mathrm{KL}}\left(\mathbb{P}^a_{t-1}(X,\theta)\|\bar{\mathbb{Q}}_{t-1}(X,\theta)\right) \\ &= \arg\max_a \left(\mathbb{E}_{\mathbb{P}^a_{t-1}(X,\theta)}\left[\frac{X}{\tau(t)}\right] - D_{\mathrm{KL}}\left(b^a_{t-1}(\theta)\|b_{\bar{\eta}_{t-1}}(\theta)\right)\right). \end{aligned}$$

The first term symbolises the expected reward of arm a. Maximising this term alone is analogous to greedily exploiting the present information about the arms. The second term quantifies the amount of uncertainty that can be decreased if arm a is chosen on the basis of the present pseudobelief. The exposure $\tau(t)$ of the focal distribution keeps a weighted balance between exploration and exploitation. Decreasing $\tau(t)$ decreases the exploration with time which is quite an intended property of an exploration–exploitation algorithm.

Since $D_{\mathrm{KL}}\left(\mathbb{P}^a_{t-1}\|\bar{\mathbb{Q}}_{t-1}\right) = H(\mathbb{P}^a_{t-1},\bar{\mathbb{Q}}_{t-1}) - h(\mathbb{P}^a_{t-1})$, we observe that the I-projection $\mathbb{P}^{A_t}_{t-1}$ is the distribution in $\mathcal{P}$ that maximises the entropy $h(\mathbb{P}^a_{t-1})$, while minimising the cross entropy $H(\mathbb{P}^a_{t-1},\bar{\mathbb{Q}}_{t-1})$. Thus, as per the discussion in Section 5.4.1, $\mathbb{P}^{A_t}_{t-1}$ is the distribution in $\mathcal{P}$ that is most similar to $\bar{\mathbb{Q}}_{t-1}$. This also implies that the I-projection $\mathbb{P}^{A_t}_{t-1}$ captures at least the first moment of the fixed distribution $\bar{\mathbb{Q}}_{t-1}$.

In the last part (Lines 8–9), the updated beliefs are used to obtain the pseudobelief-focal-reward distribution using rI-projection. Following Theorem 10, rI-projection would lead to a unique pseudobelief-focal-reward distribution for a given set of belief-rewards and exposure $\tau(t)$. Here, BelMan is inducing the exploitative bias. It keeps the pseudobelief-focal-reward distribution away from the 'actual' barycentre of the belief-reward distributions and pushes it towards the arms with higher expected reward. Increasing exploitative bias eventually merges the pseudobelief-focal-reward distribution to the distribution of the arm having the highest expected reward.

Following that (Line 5–7), the agent plays the chosen arm a_t and samples a reward X_t. This observation is incorporated in the belief of the arm using Bayes' rule of Equation (5.3).

Definition 11 (rI-Projection)**.** The *reverse information projection* (or *rI-projection*) of a distribution $\mathbb{P}^a \in \mathcal{P}$ onto $\mathcal{Q}$, which is also a non-empty, closed, convex set of probability distributions on a fixed support set, is defined by the distribution $\bar{\mathbb{Q}}^* \in \mathcal{Q}$ that has minimum KL-divergence from $\mathbb{P}^a$: $\bar{\mathbb{Q}}^* \triangleq \arg\min_{\bar{\mathbb{Q}} \in \mathcal{Q}} D_{\mathrm{KL}}(\mathbb{P}^a \| \bar{\mathbb{Q}})$.

The rI-projection finds the distribution $\bar{\mathbb{Q}}^*$ from a space of candidate distributions $\mathcal{Q}$ that encodes maximum information of the distribution $\mathbb{P}^a$. If the set of candidate distributions is engendered by a statistical model, the rI-projection of the empirical distribution formed from samples to the model is equivalent to finding the *maximum likelihood estimate*. Since rI-projection aims to maximise the complete likelihood rather than finding a distribution with similar entropy, $\bar{\mathbb{Q}}^*$ also captures higher moments of the fixed distribution $\mathbb{P}^a$. Thus, it is computationally more demanding but more informative than I-projection.

Due to the underlying minimisation operation, if we begin from $\mathbb{P}_0 \in \mathcal{P}$ and $\bar{\mathbb{Q}}_0 \in \mathcal{Q}$ and alternately perform I-projection and reverse I-projection, it will lead to two distributions $\mathbb{P}^{best}$ and $\bar{\mathbb{Q}}^{best}$ for which the KL-divergence between sets $\mathcal{P}$ and $\mathcal{Q}$ is minimum [Csiszár, 1984].

Law of Convergence for the Pseudobelief-reward Distribution

We are simultaneously approximating the belief–reward parameters as well as the pseudobelief–reward parameters. If we look into the belief update step (Equation 5.3), we observe that the belief distribution of each arm $b^a_{\eta^a_t}(\theta)$ is updated by incorporating independent and identically distributed (i.i.d.) samples obtained from the reward distribution of arm a. Let us assume that BelMan has played total T times and any arm a for n^a_T times. BelMan performs naïve Bayesian updates with i.i.d. samples X_t such that at time t

$$b^a_{t+1}(\theta_{A_t}) \propto f_{\theta_{A_t}}(X_t) b^{A_t}_t(\theta_{A_t}).$$

The belief distributions of all the other arms which are not played at time t remain unchanged. This leads the belief distributions to follow central limit theorem [Durrett, 2010]. This implies that if $\tilde{\mu}^a_{n^a_T}$ is the estimate of the expectation parameters of the belief distribution of arm a constructed from the sampled rewards $\{X^a_t\}_{t=1}^{n^a_T}$, then the scaled difference between the estimated expectation parameters and true values of the expectation parameters $\sqrt{n^a_T}(\tilde{\mu}^a_{n^a_T} - \mu^{true}_a)$ converges in distribution to a centered normal random vector in $\mathcal{N}(0, Cov^a)$. Here, $\mathcal{N}(0, Cov^a)$ denotes a normal distribution with zero mean and covariance matrix Cov^a. In Theorem 11, we show that the estimator of the mean parameters of pseudobelief is also consistent with these estimators and satisfies central limit theorem.

Theorem 11 (Central Limit Theorem)**.** *If $\tilde{\bar{\mu}}_T \triangleq \frac{1}{K}\sum_{a=1}^K \tilde{\mu}^a_{n^a_T}$ is estimator of the expectation parameters of the pseudobelief distribution at time T, the scaled error in estimation of the expectation parameter of the pseudobelief-reward distribution, $\sqrt{T}(\tilde{\bar{\mu}}_T - \bar{\mu}^{true})$, converges in distribution to a normal random vector sampled from a normal distribution $\mathcal{N}(0, \bar{Cov})$ with zero mean and covariance matrix $\bar{Cov}$. Additionally, the covariance matrix of the limiting distribution of estimation error of pseudobelief-reward distribution $\bar{Cov}$ is a linear combination of that of the covariance matrices corresponding to the arms i.e. $\sum_{a=1}^K \lambda_a Cov^a$ with weights $\lambda_a \in \mathbb{R}^+$. The weight λ^as are such that they asymptotically tend to $\frac{T}{K^2 n^a_T}$ as $T \to \infty$.*

Proof. The characteristic function for $\sqrt{T}(\tilde{\bar{\mu}}_T - \bar{\mu}^{true})$ is

$$
\begin{aligned}
\mathbf{\Phi}_{\sqrt{T}(\tilde{\bar{\mu}}_T - \bar{\mu})}(z) &= \mathbb{E}\left[\exp(\iota\langle z, \sqrt{T}(\tilde{\bar{\mu}}_T - \bar{\mu}^{true})\rangle)\right] \\
&= \mathbb{E}\left[\exp(\iota\langle z, \frac{\sqrt{T}}{K}\sum_{a=1}^K(\tilde{\mu}^a_{n^a_T} - \mu^{true}_a)\rangle)\right] \\
&= \prod_{a=1}^K \mathbb{E}\left[\exp(\iota\langle z, \frac{\sqrt{T}}{K}(\tilde{\mu}^a_{n^a_T} - \mu^{true}_a)\rangle)\right] \\
&= \prod_{a=1}^K \mathbb{E}\left[\exp(\iota\langle \frac{\sqrt{T}}{K\sqrt{n^a_T}} z, \sqrt{n^a_T}(\tilde{\mu}^a_{n^a_T} - \mu^{true}_a)\rangle)\right]
\end{aligned}
$$

$$= \prod_{a=1}^{K} \mathbf{\Phi}_{\sqrt{n_T^a}(\tilde{\mu}^a_{n_T^a} - \mu_a^{true})} \left(\frac{\sqrt{T}}{K\sqrt{n_T^a}} z \right)$$

Since each of the $\sqrt{n_T^a}(\tilde{\mu}^a_{n_T^a} - \mu_a^{true})$ converges in distribution to a random vector that follows the normal distribution $\mathcal{N}(0, \Sigma^j)$ with zero means and the covariance matrix $\lim_{T\to\infty} \sum_{a=1}^{K} \left(\frac{\sqrt{T}}{K\sqrt{n_T^a}} \right)^2 Cov^a = \sum_{a=1}^{K} \lambda_a Cov^a \triangleq \bar{Cov}$. □

Theorem 11 shows that the parameters of pseudobelief can be constantly estimated and their estimation would depend on the accuracy of the estimators of individual arms with a weight on the number of draws on the corresponding arms. Thus, the uncertainty in the estimation of the parameter is more influenced by the arm that is least drawn and less influenced by the arm most drawn. In order to decrease the uncertainty corresponding to pseudobelief, we have to draw the arms less explored.

Asymptotic Consistency of BelMan

As we have already proved the mathematical constructions corresponding to BelMan exist and are statistically consistent, we now prove that BelMan leverages this structure to find out the optimal arm and to play it asymptotically. We show that if the belief-reward functions are bounded from both above and below, and the exposure function grows fast enough to satisfy $\frac{1}{\tau(T)} = \Omega\left(\frac{\max\left|\log \mathbb{P}_T^a\right|}{\sqrt{V(\mathbb{P}_T^1)}} \right)$,[7] BelMan converges to choose the optimal arm for bandit with finite expected rewards and finite arms. We describe this result of asymptotic consistency of BelMan in Theorem 12.

Theorem 12 (Asymptotic Consistency)**.** *If all the arms have finite expected rewards* $|\mu_a| < \pm\infty$ *and finite variances* $V(\mathbb{P}_T^a) < \infty$*, there exists at least an optimal arm with expected reward* $\mu^* \triangleq \max_a \mu(\theta_a)$*,* $|\log\ \mathbb{P}_t^a(X, \theta)| \leq C_t$ *with high probability for* $+\infty > C_t \geq 0$*, and the exposure satisfies* $\frac{1}{\tau(t)} \geq \sqrt{2\log 2}\frac{C_t}{\sqrt{V(\mathbb{P}_t^1)}}$ *with high probability* 1 *for all* $t \in [T]$*, then with high probability*

$$\lim_{T\to\infty} \frac{S(T)}{T} = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} (X_{A_t}) \right] = \mu^*. \tag{5.11}$$

[7] $\sqrt{V(\mathbb{P}_T^1)}$ is variance of the belief-reward distribution at time of the optimal arm at time T.

Proof. Without loss of generality, let us consider that there exists at least one optimal arm and it is identified as the arm $a = 1$. At the I-projection step, we choose the arm that has minimum KL-divergence $D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right)$ from the pseudobelief–focal distribution. Thus, we have to prove that for large t, $D_{\mathrm{KL}}\left(\mathbb{P}_t^1(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right) - D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right)$ is non-positive for any $a \neq 1$. We begin as follows,

$$
\begin{aligned}
&D_{\mathrm{KL}}\left(\mathbb{P}_t^1(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right) - D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right) \\
&= \underbrace{\int_X \int_\theta \mathbb{P}_t^1(X,\theta)\log \mathbb{P}_t^1(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX} - \int_X \int_\theta \mathbb{P}_t^a(X,\theta)\log \mathbb{P}_t^a(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX}}_{\mathbf{T1}} \\
&\quad + \underbrace{\int_X \int_\theta \left[\mathbb{P}_t^a(X,\theta) - \mathbb{P}_t^1(X,\theta)\right]\log \bar{\mathbb{Q}}(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX}}_{\mathbf{T2}}
\end{aligned}
$$

The first term **T1** is the difference in entropy in two of the arms.

$$
\begin{aligned}
\mathbf{T1} &= \int_X \int_\theta \mathbb{P}_t^1(X,\theta)\log \mathbb{P}_t^1(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX} - \int_X \int_\theta \mathbb{P}_t^a(X,\theta)\log \mathbb{P}_t^a(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX} \\
&= \int_X \int_\theta \left[\mathbb{P}_t^a(X,\theta) - \mathbb{P}_t^1(X,\theta)\right]\log \mathbb{P}_t^1(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX}\ -\ D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta)\|\mathbb{P}_t^1(X,\theta)\right) \\
&\underset{(a)}{\leq} \int_X \int_\theta \left[\mathbb{P}_t^a(X,\theta) - \mathbb{P}_t^1(X,\theta)\right]\log \mathbb{P}_t^1(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX} \\
&\underset{(b)}{\leq} \int_X \int_\theta \left|\left[\mathbb{P}_t^a(X,\theta) - \mathbb{P}_t^1(X,\theta)\right]\log \mathbb{P}_t^1(X,\theta)\right|\ \mathrm{d}\theta\ \mathrm{dX} \\
&\underset{(c)}{\leq} \sup_{X,\theta}\left|\log \mathbb{P}_t^1(X,\theta)\right| \int_X \int_\theta \left|\mathbb{P}_t^a(X,\theta) - \mathbb{P}_t^1(X,\theta)\right|\ \mathrm{d}\theta\ \mathrm{dX} \\
&\underset{(d)}{\leq} C_t\sqrt{\frac{\log 2}{2}\ D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta)\|\mathbb{P}_t^1(X,\theta)\right)}
\end{aligned}
$$

The inequality (a) is due to the non-negativity of KL-divergence. Inequality (b) is derived from the monotonicity of integrals. This means that if $f \leq g$ for all $w \in W$ then $\int_{w\in W} f(w)\ \mathrm{dw}\ \leq \int_{w\in W}\ g(w)\ \mathrm{dw}$. Boundedness of the logarithmic density function of the pseudobelief-reward as stated in the assumption results to inequality (c). Inequality (d) is obtained from Pinsker's inequality [Cover and Thomas, 2012].

Similarly, we get for the second term **T2**:

$$\begin{aligned}
\mathbf{T2} &= \int_X \int_\theta \left[\mathbb{P}_t^a(X,\theta) - \mathbb{P}_t^1(X,\theta)\right] \log \bar{\mathbb{Q}}(X,\theta)\ \mathrm{d}\theta\ \mathrm{dX}) \\
&= \int_X \int_\theta \left[\mathbb{P}_t^a(X,\theta) - \mathbb{P}_t^1(X,\theta)\right] \log \left(\prod_a \mathbb{P}_t^a(X,\theta)^{\lambda_t^a}\right)\ \mathrm{d}\theta\ \mathrm{dX} \\
&- \frac{1}{\tau(t)} \mathbb{E}_{\mathbb{P}_t^1(X,\theta)-\mathbb{P}_t^a(X,\theta)}\left[X\right] + \log \bar{Z}_t \times \mathbb{E}_{\mathbb{P}_t^1(X,\theta)-\mathbb{P}_t^a(X,\theta)}\left[1\right] \\
&\underset{(e)}{\leq} C_t \sqrt{\frac{\log 2}{2}}\ \sqrt{D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta) \| \mathbb{P}_t^1(X,\theta)\right)} - \frac{\Delta_t^a}{\tau(t)}.
\end{aligned}$$

Here, $\Delta_t^a \triangleq \mu_t^1 - \mu_t^a$, which means the difference between the expected reward of the optimal arm and the suboptimal arm a. Inequality (e) is obtained by applying AM-GM inequality, inequalities (a), (b), (c), and (d) in sequence. Thus,

$$\mathbf{T1} + \mathbf{T2} \leq C_t \sqrt{2\log 2} \sqrt{D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta) \| \mathbb{P}_t^1(X,\theta)\right)} - \frac{\Delta_a}{\tau(t)}.$$

The RHS is non-positive if

$$\frac{1}{\tau(t)} \geq C_t\ \sqrt{2\log 2}\ \frac{\sqrt{D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta) \| \mathbb{P}_t^1(X,\theta)\right)}}{\Delta_t^a}.$$

Now, we get from the transportation lemma in [Boucheron et al., 2013;Section 4.10], we get $\frac{\sqrt{D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta) \| \mathbb{P}_t^1(X,\theta)\right)}}{\Delta_t^a} \geq \frac{1}{\sqrt{V(\mathbb{P}_t^1)}}$. Thus, the RHS transforms to $\frac{C_t\ \sqrt{2\log 2}}{\sqrt{V(\mathbb{P}_t^1)}}$. Hence for any exposure function that satisfies the growth rate $\frac{1}{\tau(t)} = \Omega\left(\frac{\max|\log \mathbb{P}_t^a|}{\sqrt{V(\mathbb{P}_t^1)}}\right)$, BelMan would asymptotically choose the optimal arm. This proves that BelMan is asymptotically consistent for finite-arm stochastic bandit problems. □

This lower bound on $\frac{1}{\tau(T)}$ being inversely proportional to Δ^a indicates that we have to induce higher exploitative bias to reach higher rewards if the difference between the optimal and the suboptimal arm is minute. It is intuitive as the algorithm would need more samples to distinguish between the optimal and the suboptimal similar to it.

We intuitively validate this claim. We can show the KL-divergence between belief-reward of arm j and the pseudobelief-focal-reward is $D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta)\|\bar{\mathbb{Q}}(X,\theta)\right) = (1-\lambda_a)h(b_t^a) - \frac{1}{\tau(t)}\mu_t^a$, where λ_a's are computed as per Theorem 11. As $t \to \infty$, the entropy of belief on each arm reduces to a constant dependent on its internal entropy. Thus, when $\frac{1}{\tau(t)}$ exceeds the entropy term for a large t, BelMan greedily chooses the arm with highest expected reward. Hence, BelMan is asymptotically consistent.

5.4.6 BelMan for Exponential Family Distributions

BelMan is applicable to any belief-reward distribution for which KL-divergence is computable and is finite. Additionally, if we assume the reward distributions to be in the exponential family of distributions, the belief distributions, being conjugate to the reward distributions, also belong to the exponential family [Brown, 1986]. This makes the belief-reward distributions flat with respect to KL-divergence. Thus, both I-and rI-projections in BelMan are well-defined and unique for exponential family reward distributions. Furthermore, if we identify the belief-reward distributions with expectation parameters, we obtain the pseudobelief as an affine sum of them. This allows us to compute pseudobelief-reward distribution directly instead of computing its dependence on each belief-reward separately. The exponential family includes the majority of the distributions found in the bandit literature such as Bernoulli, beta, Gaussian, Poisson, exponential, and χ^2. We choose the exponential family to instantiate our framework not only because of its wide range and applicability but also due to the aforementioned Bayesian and information geometric properties.

If the reward distribution belongs to the exponential family, it can be represented as

$$f_\theta(X) \triangleq g(X)\exp\left(\langle\omega(\theta), T(X)\rangle - A(\theta)\right).$$

Here, $g(X)$ is the *base measure* on reward X and $A(\theta)$ is called the *log-partition function*. Since the conjugate distribution of a distribution in exponential family also belongs to the exponential family [Brown, 1986], the belief distribution also belongs

to the exponential family, and is represented as the conjugate of reward distribution. Thus, we express the belief distribution as

$$b_\eta(\theta) \triangleq g(\theta) \exp\left(\langle \eta, T(\theta) \rangle - A(\eta)\right). \tag{5.12}$$

Here, η is the *natural parameters*, $T(\theta)$ the *sufficient statistics*, $g(\theta)$ is the *base measure* and $A(\eta)$ is the *log-partition function*. Thus, the posterior distribution $\mathbb{P}(\theta|X = x)$ formed by multiplying the belief and reward distributions has the same exponential form as $b_\eta(\theta)$.

Since exponential family distributions are flat with respect to KL-divergence [Amari and Nagaoka, 2007], both I-and rI-projections in BelMan are well-defined and unique. Thus, at each iteration, we obtain an optimal and unambiguous choice of the arm and pseudobelief respectively. Thus, working with exponential family distributions implicitly supports the well-defined nature and possibility of getting an efficient estimation. Theorem 10 and 11 validate these properties for BelMan.

Bernoulli Bandits

In case of Bernoulli bandits, we assume that drawing an arm returns the rewards 1 and 0 with probability θ and $1 - \theta$ respectively. Thus, the reward distribution of an arm a is $f_{\theta_a}(X) \triangleq \text{Ber}(X; \theta_a)$. Following the Bayesian approach, we choose the conjugate beta prior to begin with. Thus, we keep the prior belief over each arm as a beta distribution with shape parameters $\{\alpha^a\}_{a=1}^K$ and $\{\beta^a\}_{a=1}^K$. After t-iterations the prior over the probability of success of the a^{th} arm is

$$b_t^a(\theta_a) \triangleq \text{Beta}(\theta_a; \alpha_t^a, \beta_t^a) = \frac{1}{B(\alpha_t^a, \beta_t^a)} \theta_a^{\alpha_t^a - 1} (1 - \theta_a)^{\beta_t^a - 1},$$

for $\alpha_t^a, \beta_t^a \in \mathbb{N}$ and $\theta_a \in (0, 1)$. Here, α_t^a and β_t^a are the number of successes and failures, respectively, for the arm a till iteration t.

We begin with both α_0^a and β_0^a to be 1 for all arms. This amounts to the uniform distribution over 0 and 1. This initialization allows us to choose all the arms with

equal probability and without any initial bias. We update this belief eventually as we further draw the arms and compute it using BelMan. Under this setting of beta prior and Bernoulli reward, we compute the targeted KL-divergence of BelMan as

$$\begin{aligned}
&\sum_{a=1}^{K} D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta)\|\bar{\mathbb{Q}}_{t-1}(X,\theta)\right) \\
&= \sum_{a=1}^{K}[-\frac{1}{\tau(t)}\frac{\alpha_t^a}{n_t^a} - \log\left(B\left(\alpha_t^a,\beta_t^a\right)\right) + (\alpha_t^a - \bar{\alpha}_{t-1})\Psi(\alpha_t^a) + (\beta_t^a - \bar{\beta}_{t-1})\Psi(\beta_t^a) \\
&- (n_t^a - \bar{n}_{t-1})\Psi(n_t^a)] + K\log\left(\frac{\bar{\alpha}_{t-1}\exp(\frac{1}{\tau(t)}) + \bar{\beta}_{t-1}}{\bar{n}_{t-1}}\right) + K\log\left(B\left(\bar{\alpha}_{t-1},\bar{\beta}_{t-1}\right)\right).
\end{aligned}$$

Here, $n_t^a = \alpha_t^a + \beta_t^a$ is the total number of times an arm a is played till time t, $\bar{n} = \bar{\alpha} + \bar{\beta}$ and Ψ is the digamma function [Bernardo, 1976] defined as the derivative of the logarithm of gamma function, i.e. $\frac{d}{da}(\log\Gamma(a))$.

In Line 4 of Algorithm 20, we first perform the I-projection to decide which arm A_t to draw to minimize the KL-divergence. Following this, we update the pseudobelief using I-projection in Line 9 of Algorithm 20. In order to perform this update, we find out such $\bar{\alpha}$ and $\bar{\beta}$ that minimize the objective and update the pseudobelief accordingly. The presence of pseudobelief offers BelMan a chance to explore the less successful arms to minimize the entropy, while the Focal distribution creates the scope of exploiting the present information of the best arm.

Exponential Bandits

The *exponential distribution* is another member of the exponential family. For a given positive *rate parameter* $\theta_a \in \mathbb{R}^+$, the reward distribution of arm a of exponential bandit is $f_{\theta_a}(X) \triangleq \theta_a \exp(-\theta_a X)$ for reward $X \in [0,\infty)$. Following the structure of Sections 5.4.6 and the previous Bernoulli case, we obtain the gamma distribution, another member of the exponential family, as the conjugate prior. After time t, the belief distribution corresponding to a^{th} arm is expressed as

$$b_t^a(\theta_a) \triangleq \mathrm{Gamma}\left(\theta_a;\alpha_t^a,\beta_t^a\right) = \frac{(\beta_t^a)^{\alpha_t^a}}{\Gamma(\alpha_t^a)}{\theta_a}^{\alpha_t^a-1}\exp(-\theta_a\beta_t^a),$$

for both shape and rate parameters $\alpha_t^a, \beta_t^a > 0$. Here, α_t^a and β_t^a are, respectively, the number of times the arm a is played and sum of the rewards obtained by playing the arm till iteration t. As we update using Equation (5.3), we get gamma distributions with parameters $\alpha_{t+1}^a = \alpha_t^a + 1$, and $\beta_{t+1}^a = \beta_t^a + X_t$ if the arm a is played and a reward X_t is obtained. Under this specific setting of gamma prior and exponential reward, we compute the targeted KL-divergence of BelMan as

$$\begin{aligned}
&\sum_{a=1}^{K} D_{\mathrm{KL}}\left(\mathbb{P}_t^a(X,\theta) \| \bar{\mathbb{Q}}(X,\theta)\right) \\
&= \sum_{a=1}^{K} [-\frac{1}{\tau(t)}\frac{\alpha_t^a}{\beta_t^a} - \log\left(\Gamma\left(\alpha_t^a\right)\right) + (\alpha_t^a - \bar{\alpha}_{t-1})\Psi(\alpha_t^a) - \frac{\alpha_t^a}{\beta_t^a}(\beta_t^a - \bar{\beta}_{t-1}) \\
&+ \bar{\alpha}_{t-1}\log\beta_t^a] + K\log\bar{Z}_t + K\log\left(\Gamma\left(\bar{\alpha}_{t-1}\right)\right) - K\bar{\alpha}_{t-1}\log\bar{\beta}_{t-1}.
\end{aligned}$$

We incorporate this analytical form in Algorithm 20 and update it as mentioned in the Bernoulli case.

5.5 Empirical Performance Analysis

In this section, we experimentally verify BelMan's performance for exploration-exploitation bandits, and two-phase bandits respectively. We also comparatively analyse its performance with respect to the state-of-the-art algorithms, and empirically prove it to achieve logarithmic regret bound. We use the pymaBandits library [Cappé et al., 2012] for implementation of all the algorithms except ours, and run it on MATLAB 2014a.

5.5.1 Exploration–exploitation Bandit

We evaluate the performance of BelMan for two exponential family distributions – Bernoulli and exponential. They stand for discrete and continuous rewards respectively. We plot the evolution of the mean and the 75 percentile of cumulative regret and number of suboptimal draws. For each instance, we run experiments for 25 runs each consisting of 1000 iterations.

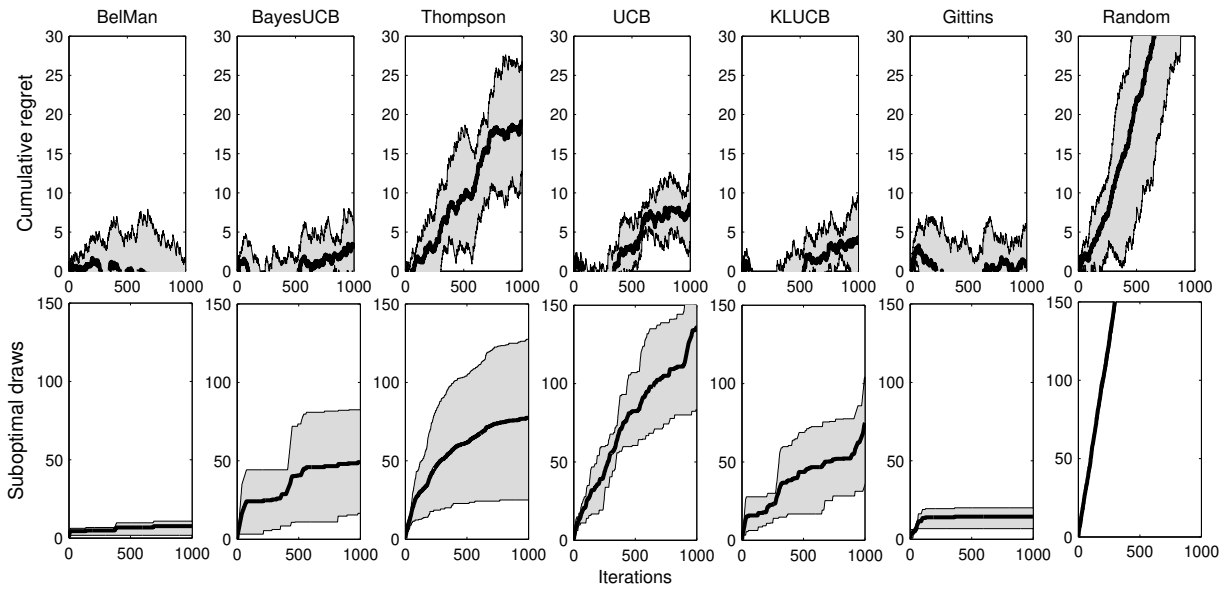


Figure 5.2: Evolution of cumulative regret (top), and number of suboptimal draws (bottom) for 500 iterations for 2-arm Bernoulli bandit with means $\{0.8, 0.9\}$. The dark black line shows the average. The grey area shows 75 percentile.

We also experimented on another 2-arm bandit scenario with means 0.45 and 0.55. Figures 5.5 depicts the evolution of cumulative regret and suboptimal draws for BelMan and the other competing algorithms. Similar to Figure 5.5, we observe the cumulative regret of BelMan grows at first linearly and then it transits to a state of slow growth. Except showing this ideal behaviour, BelMan performs competitively with the contending algorithms. This shows its efficiency as a candidate solution to the exploration–exploitation bandit.

Figure 5.6 shows performance for 10-arm Bernoulli bandit. For this setup, BelMan outperforms other algorithms. We also observe though the number of arms increases from Figure 5.5 to Figure 5.6 that performance of all algorithms is comparatively better in the first case. This is explainable from the fact that hardness of minimising cumulative regret increases as the number of arms increases. Beside that, as more arms with identical or almost identical distributions appear, the algorithm requires more exploration to separate them and to determine which one is optimal. The difference in performance between Figure 5.5 and 5.2 indicates this.

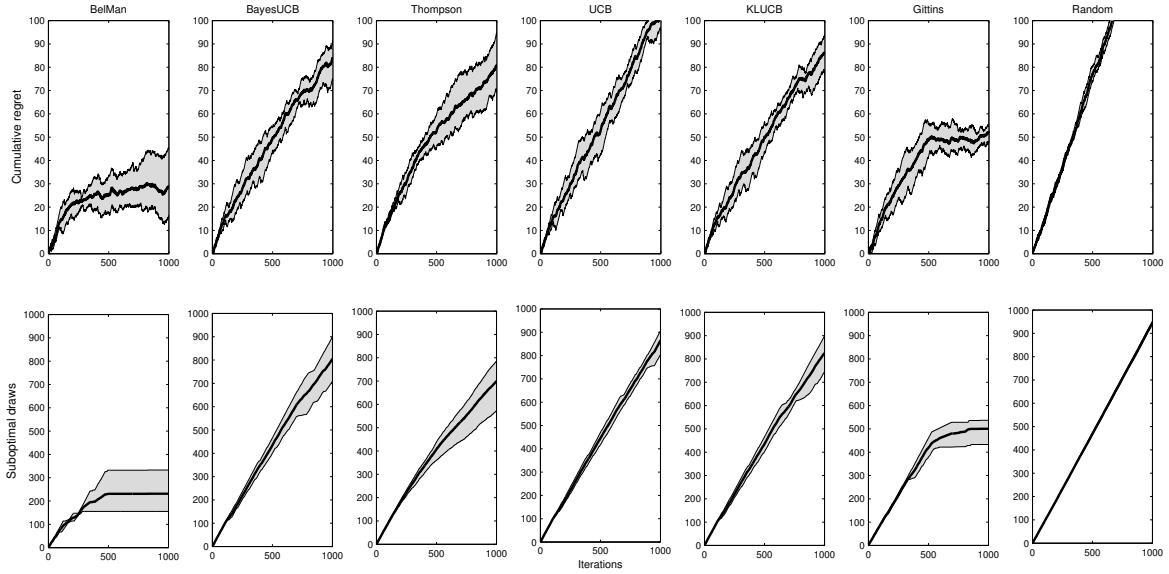


Figure 5.3: Evolution of cumulative regret (top), and number of suboptimal draws (bottom) for 1000 iterations for 20-arm Bernoulli bandit.

We finally tested BelMan on an exponential bandit consisting of 5-arms with expected rewards $\{0.2, 0.25,\ 0.33, 0.5, 1.0\}$. We compare performance of BelMan with state-of-the-art frequentist method tailored for exponential distribution of rewards, called KL-UCBExp [Garivier and Cappé, 2011]. We also compare it with Thompson sampling, UCBtuned and uniform sampling method (Random). The results are shown in Figure 5.7 and 5.8. Since the formulation is oblivious to boundedness of the distribution, we choose to validate also on unbounded rewards. In Figure 5.7, it outperforms all the other algorithms. In Figure 5.8, though KL-UCBexp performs the best, performance of BelMan is still competitive with it.

These results validate BelMan's claim as a generic solution to a wide range of bandit problems. We compare the performance of BelMan with frequentist methods like UCB [Auer et al., 2002] and KL-UCB [Garivier and Cappé, 2011], and Bayesian methods like Thompson sampling [Thompson, 1933] and Bayes-UCB [Kaufmann et al., 2012a]. For Bernoulli bandits, we also compare with Gittins index [Gittins, 1979] which is the optimal algorithm for Markovian finite arm independent bandits with discounted

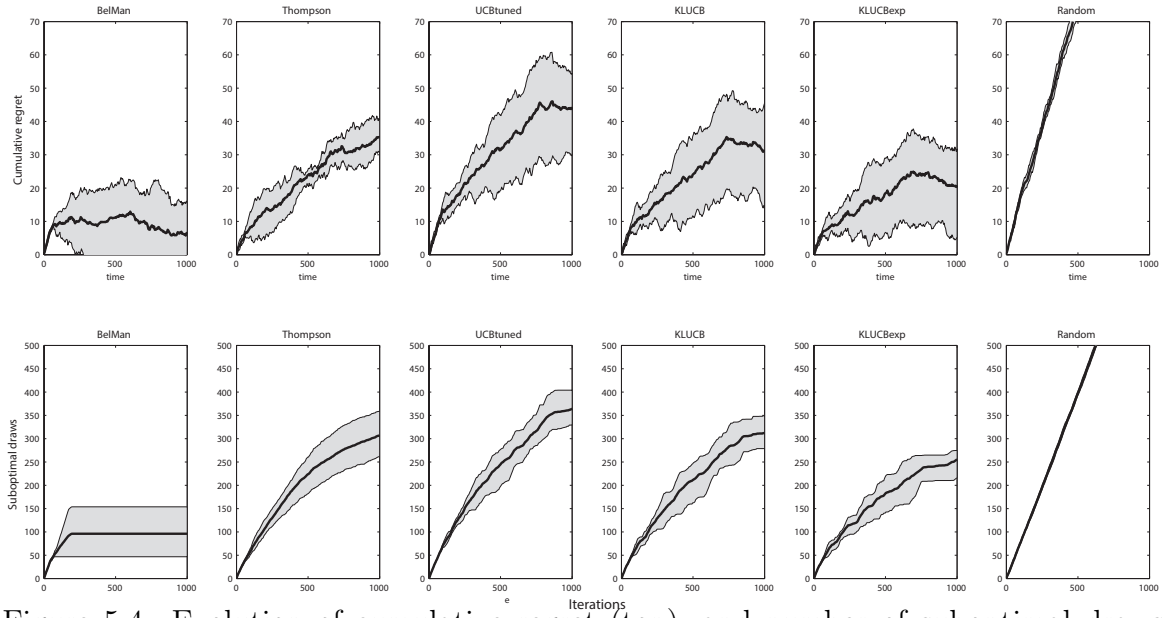


Figure 5.4: Evolution of cumulative regret (top), and number of suboptimal draws (bottom) for 1000 iterations for 5-arm bounded exponential bandit with parameters $\{1, 2, 3, 4, 5\}$.

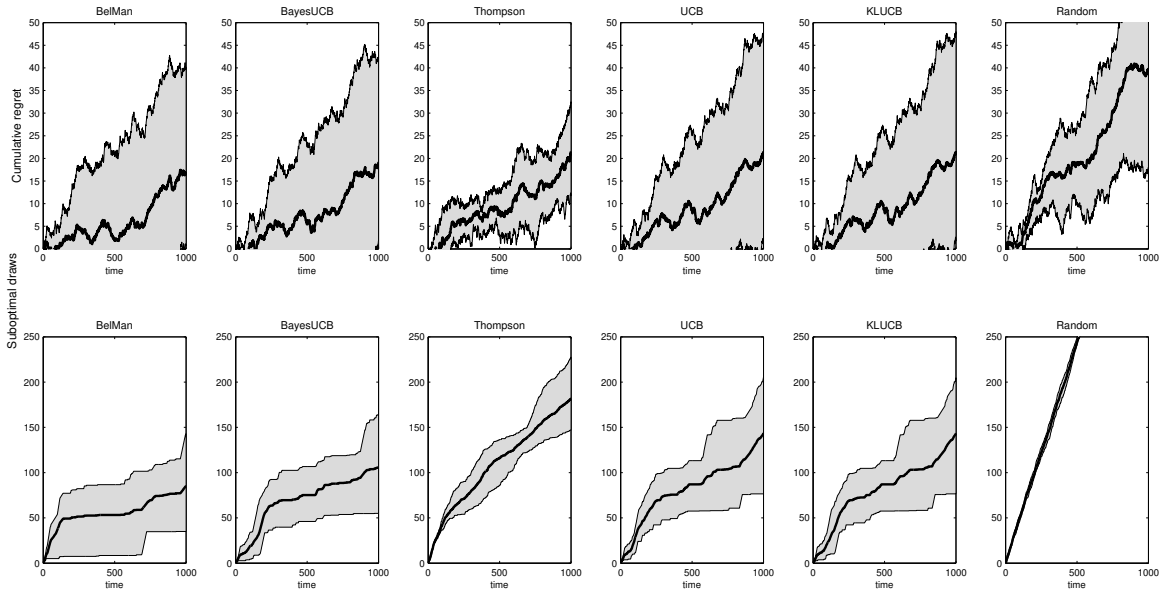


Figure 5.5: Evolution of cumulative regret (top), and number of suboptimal draws (bottom) for 500 iterations for 2-arm Bernoulli bandit with means 0.45 and 0.55. The dark line shows the average over 25 runs. The grey area shows 75 percentile.

rewards. Though we are not interested in the discounted case, but the algorithm is indeed transferable to the finite horizon setting with slight manipulation. Though it is often computationally intractable, we use it as the optimal baseline for Bernoulli bandits. We also plot performance of the uniform sampling method (*Random*), as a naïve baseline.

For the 2-arm Bernoulli bandit of Figure 5.2, left ($\theta_1 = 0.8, \theta_2 = 0.9$), we observe that at the very beginning the cumulative regret of BelMan grows linearly and then transitions to a state of slow growth. This initial linear growth of suboptimal draws followed by a logarithmic growth is an intended property of any optimal bandit algorithm as can be seen in the performance of competing algorithms and also pointed out by [Garivier et al., 2016b]: an initial phase dominated by exploration and a second phase dominated by exploitation. The phase change indicates the ability of the algorithm to reduce uncertainty after a certain number of iterations and to find a trade-off between exploration and exploitation. BelMan performs comparatively well with respect to the contending algorithms, achieving the phase of exploitation faster than others, with significantly less variance. Figure 5.3 depicts similar features of BelMan for 20-arm Bernoulli bandits (with means 0.25, 0.22, 0.2, 0.17, 0.17, 0.2, 0.13, 0.13, 0.1, 0.07, 0.07, 0.05, 0.05, 0.05, 0.02, 0.02, 0.02, 0.01, 0.01, and 0.01). Since more arms ask for more exploration and more suboptimal draws, all algorithms show higher regret values. On all experiments performed, BelMan outperforms the competing approaches.

We also simulated BelMan on exponential bandits: 5 arms with expected rewards $\{0.2, 0.25, 0.33, 0.5, 1.0\}$. Figure 5.4 shows that BelMan performs more efficiently than state-of-the-art methods for exponential reward distributions- Thompson sampling, UCBtuned [Auer et al., 2002], KL-UCB, and KL-UCB-exp, a method tailored for exponential distribution of rewards [Garivier and Cappé, 2011]. This demonstrates BelMan's broad applicability and efficient performance in complex scenarios.

We have also run the experiments 50 times with horizon 50 000 for the 20 arm Bernoulli bandit to verify the asymptotic behaviour of BelMan. Figure 5.9 shows that BelMan's regret gradually becomes linear with respect to the logarithmic axis. Fig-

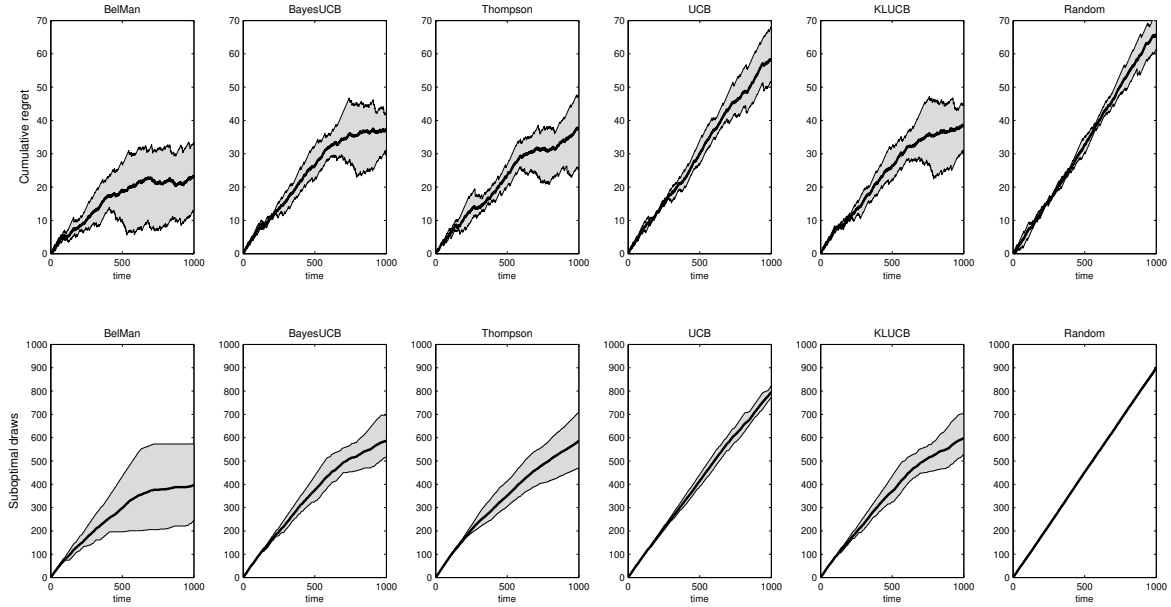


Figure 5.6: Evolution of cumulative regret (top), and number of suboptimal draws (bottom) for 500 iterations for 10-arm Bernoulli bandit with means $\{0.1, 0.05, 0.05, 0.05, 0.02, 0.02, 0.02, 0.01, 0.01, 0.01\}$. The dark black line shows the average. The grey area shows 75 percentile.

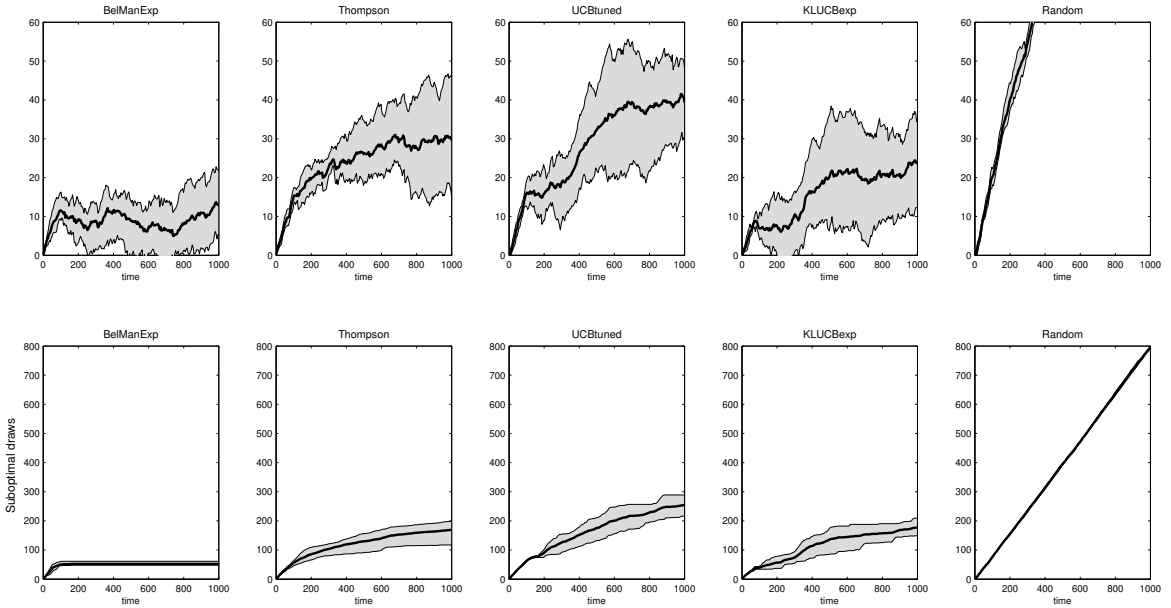


Figure 5.7: Evolution of cumulative regret (top), and number of suboptimal draws (bottom) for 1000 iterations for 5-arm unbounded exponential bandit with parameters $\{0.2, 0.25, 0.33, 0.5, 1.0\}$.

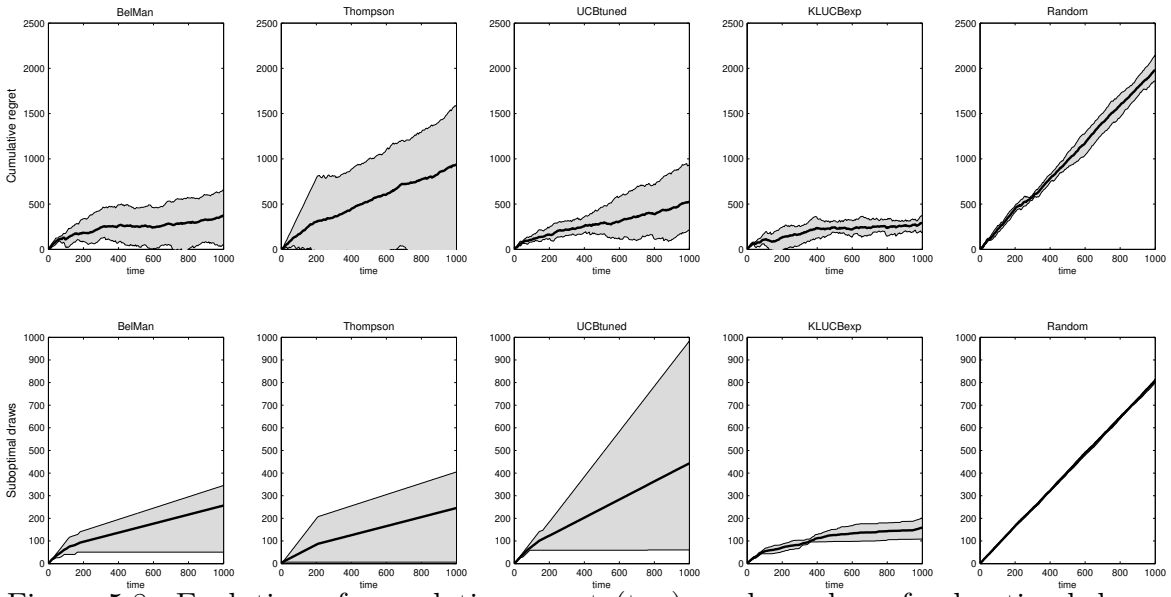


Figure 5.8: Evolution of cumulative regret (top), and number of suboptimal draws (bottom) for 1000 iterations for 5-arm unbounded exponential bandit with parameters $\{1, 2, 3, 4, 5\}$.

ure 5.9 empirically validates BelMan to achieve logarithmic regret like the competitors which are theoretically proven to reach logarithmic regret.

5.5.2 Two-phase Bandit

In this experiment, we simulate a two-phase setup, as in [Putta and Tulabandhula, 2017b]: the agent first does pure exploration for a fixed number of iterations, then move to exploration–exploitation. This is possible since BelMan supports both modes and can transparently switch. The setting is that of the 20-arm Bernoulli bandit. The two-phase algorithm is exactly BelMan (Algorithm 20) with $\tau(n) = \infty$ for an initial phase of length T_{EXP} followed by the decreasing function of n as indicated previously. Thus, BelMan gives us a single algorithm for three setups of bandit problems – pure exploration, exploration–exploitation, and two-phase learning. We only have to choose a different $\tau(n)$ depending on the problem addressed. This supports BelMan's claim as a generalised, unified framework for bandit problems.

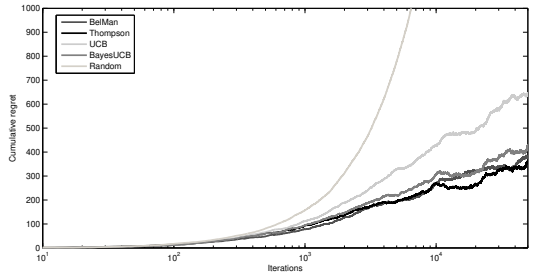


Figure 5.9: Evolution of (mean) regret for exploration–exploitation 20-arm Bernoulli bandits with horizon=50,000.

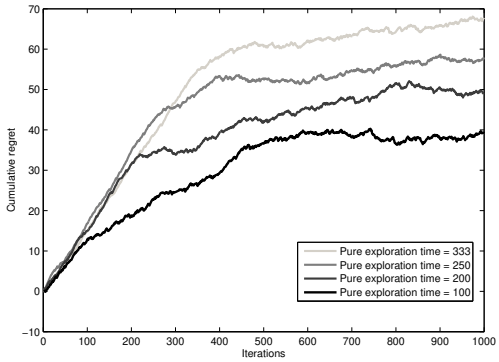


Figure 5.10: Evolution of (mean) cumulative regret for two-phase 20-arm Bernoulli bandits.

We observe a sharp phase transition in Figure 5.10. While the pure exploration version acts in the designated window length, it explores almost uniformly to gain more information about the reward distributions. We know for such pure exploration the cumulative regret grows linearly with iterations. Following this, the growth of cumulative regret decreases and becomes sublinear. If we also compare it with the initial growth in cumulative regret and suboptimal draws of BelMan in Figure 5.3, we observe that the regret for the exploration–exploitation phase is less than that of regular BelMan exploration–exploitation. Also, with increase in the window length the phase transition becomes sharper as the growth in regret becomes very small. In brief, there are three major lessons of this experiment. First, Bayesian methods provide an inherent advantage in leveraging a priori knowledge (here, from the first phase). Second, a pure exploration phase helps in improving the performance during the exploration–exploitation phase. Third, we can leverage the exposure to control the exploration–exploitation trade-off.

5.6 Conclusion

BelMan implements a generic Bayesian information geometric approach for stochastic multi-armed bandit problems. It operates in a statistical manifold of joint distributions of beliefs and rewards. Their barycentre, the pseudobelief-reward, forms the basis of the exploration component of the algorithm. The algorithm is further ex-

tended by composing the pseudobelief-reward distribution with a reward distribution that gradually concentrates on higher rewards by means of a time-dependent function, the exposure. In short, BelMan addresses the issue of the adaptive balance of explorationexploitation from the perspective of information representation, accumulation, and balanced induction of bias. Consequently, BelMan can be uniformly tuned to support pure exploration, exploration–exploitation, and two-phase reinforcement learning problems. We prove the law of convergence of the pseudobelief-focal-reward distribution for BelMan. BelMan is asymptotically consistent. The proof of consistency indicates that a growth of exposure transforms the exploration–exploitation trade-off into pure exploitation after accumulating sufficiently enough samples. BelMan, when instantiated to rewards modelled by any distribution of the exponential family, conveniently leads to analytical forms that allow to derive a well-defined and unique projection as well as to devise an effective and fast computation. We empirically evaluate the performance of BelMan for Bernoulli and exponential distributions. The results of the experiments validate that BelMan asymptotically achieves logarithmic regret. The results show that, for the two-phase reinforcement learning problem, BelMan not only spontaneously adapts with but also leverages explored information to escalate efficiency. The comparative results show that BelMan is not only competitive but also outperforms existing algorithms for challenging setups such as those involving many arms and continuous rewards.

We are investigating the analytical asymptotic efficiency and stability of BelMan. We are also investigating how BelMan can be extended to other settings such as dependent arms, non-parametric distributions and continuous arms.

Chapter 6

QBelMan: An Information Geometric Approach to Queueing Bandits

Experiment escorts us last-
his pungent company
will not allow an Axiom
an Opportunity.

— *Emily Dickinson, Experiment escorts us last, 1770.*

We introduce the information-geometric approach to multi-armed bandits in this chapter. This formulation allows us to develop an algorithm, BelMan [Basu et al., 2018c], for finite-arm stochastic multi-armed bandit problems. As we have developed the theory and the algorithm design technique, we want to instantiate it in a real-life application, and to investigate its efficiency and effectiveness for the corresponding problem.

In this chapter, we instantiate BelMan for the problem of scheduling jobs in a multiple-server multiple-queue system with known arrival rates and unknown service

rates. The goal of the agent is to choose such a server for the given system such that the total queue length, i.e. the number of jobs waiting in the queue, will be as low as possible. We formulate this problem as a finite-arm stochastic bandit problem that aims to minimise the cumulative queue length. This problem is referred as the *queueing bandit* [Krishnasamy et al., 2016]. The unknown service rates introduce the exploration-exploitation problem. Thus, the *queueing bandit* problem differs from the standard optimisation problem discussed in the queueing theory literature.

We instantiate BelMan for queueing bandits with Bernoulli service rates and exponential service rates respectively. Experimental results instantiate that BelMan is performing more efficiently than the state-of-the-art algorithms in both of these variants of queueing bandit that we have investigated.

6.1 Introduction

We formulate the allocation problem in a multiple-server multiple-queue system with known arrival rate and unknown service rates as a finite-arm stochastic multi-armed bandit problem (Section 2.1). The queue-server pair and the allocation algorithms are analogous to the arms and the agent in the bandit problem respectively. This formulation of queueing is relevant for modelling a vast range of service systems, such as supply and demand in online platforms (e.g., Uber, Airbnb, etc.), order flow in financial markets (e.g., limit order books), packet flow in communication networks, and supply chains. In these systems, queueing is an essential part of the model and also the queue parameters are dynamic and unknown.

In classical queueing theory literature, both the arrival rates and the service rates are assumed to be accurately known and it leads to an analytical solution for queue performance. For the aforementioned systems, due to the lack of such knowledge of accurate parameters, the proposed multi-armed bandit formulation provides a model-free framework to explore the queue parameters and to exploit them on-the-go. We formulate this problem of queueing bandits in Section 6.3.

In order to resolve the exploration-exploitation trade-off in the bandit formulation, we adapt our information geometric algorithm, BelMan [Basu et al., 2018c], for multi-armed bandits. This approach not only solves the proposed formulation but also caters for an information geometric analysis of allocation in a queueing system. BelMan represents the uncertainty related to the queue parameter as a belief distribution and the aggregated knowledge about the queue as a belief-reward distribution. BelMan uses an I-projection [Csiszár, 1984] to project the belief-rewards of all the arms on their KL-divergence barycentre, referred to as the pseudobelief-reward [Basu et al., 2018c]. The pseudo belief-reward acts as the summarised knowledge-base of all the arms and evokes the server to explore the arms with most uncertain belief-reward distribution in the reverse I-projection step [Basu et al., 2018c]. An incremental exploitative bias is infused by means of a reward distribution that evolves and concentrates on higher rewards. The rate of the concentration of this bias, called exposure, allows us to control the exploration-exploitation trade-off.

In Section 6.5, we comparatively evaluate our approach with the state-of-the-art bandit algorithms, such as Thompson sampling, Q-ThS, and Q-UCB. Experiments validate the similitude of the behaviour of our approach with the proved bounds on queue regret. Experiments instantiate that BelMan performs more efficiently and effectively than the competing algorithms for both the cases of Bernoulli service rates, and exponential service rates. Specifically, under the conditions where Thompson sampling, and QThS are unable to achieve optimal queue regret bounds, BelMan stays stable and outperforms the state-of-the-art algorithm.

6.2 A Primer on Queueing and Bandits

This paper belongs to the junction of two fields of research- queueing theory and multi-armed bandits. In this section, we review and summarise the research literature in queueing theory and the previous endeavours to apply bandits to queues. This allows

us to contextualise our contribution in the existing literature. We refer to Section 2.1 for detailed literature on bandits.

6.2.1 Queueing Theory

The queueing theory literature [Cox and Smith, 1961] discusses about the statistical problem of arrival of customers and serving them. The general metric that most queueing mechanism minimise is the queue length or the average waiting time in the array. Researchers [Cohen and Boxma, 1985] studied queuing theory from kaleidoscopic approaches such as a branch of mathematics studying variants of birth-death process to a field of operations research to calculate the performance measures. Kendall proposed a notation $a/b/c : d/e/f$ to represent all these variants unambiguously. Here a denotes inter-arrival time distribution, b is the service time distribution, and c represents the number of servers. d denotes the maximum number of jobs that can be occupied in the system (waiting and in service) with an infinite number of waiting positions for default. e indicates the queueing discipline (First Come First Serve, Last Come First Serve, Round Robin etc.) with First Come First Serve (FCFS) as default, and f stands for the population size from which customers rush to the system with infinity as default.

In most of the cases, the processes of arrival and service are memoryless i.e, Markovian, and the corresponding time distributions are Poisson and exponential [Shortle et al., 2018]. This is represented by $M/M/K$. Though queueing theory proposes the optimal steady-state solution and detailed analysis of the transient phase, the literature assumes complete prior knowledge of the arrival and service rates. This is not always true for real-life applications. Though there are works on transient queues for which the arrival and service rate change through a pre-defined stochastic process, the pre-requisite knowledge of the stochastic process narrows down its scope. This scenario engenders a need of bandit algorithms in queueing literature that would learn the arrival and service rates on-the-go as well as minimise the queue length or waiting time in the queue efficiently.

6.2.2 Multi-armed Bandits in Queueing

Following the Gittins index formalism, [Niño-Mora, 2006, 2007; Buyukkoc, 1985; Van Mieghem, 1995; Jacko, 2010] have applied multi-armed bandits for the queueing and scheduling systems. These queueing studies focus on infinite-horizon costs, i.e., statistically steady-state behaviour, where the focus typically is on conditions for optimality of index policies. Further, the models do not typically consider user-dependent server statistics and provides little insight to the learning procedure.

We focus here on algorithms and analysis to optimize finite time regret. [Krishnasamy et al., 2016] proposed a finite-time analysis of queueing systems with known arrival rate and unknown service rates. They also leverage this analysis to design a variant of Thompson sampling, Q-ThS, which satisfies the asymptotic optimality condition. Still, their analysis is limited to the Bernoulli servers only i.e, the servers may or may not serve a customer with a certain probability. This is unrealistic because the systems have service rates in form of generalised distributions like exponential distributions. Our proposed algorithm, QBelMan, does not only achieve the asymptotic optimality for Bernoulli bandits but also shows similar performance for general service distributions. It also additionally provides an information theoretic insight of learning the queueing parameters and improving the performance accordingly that eventually makes it almost as optimal as the optimal queueing algorithm with full information.

6.3 Queueing Bandit: Problem Formulation

We consider a discrete-time queueing system with N queues and K servers. The queues are indexed by $n \in \{1, \ldots, N\}$ and the servers are indexed by $k \in \{1, \ldots, K\}$. Arrivals to the queue and service offered by the servers are assumed to be independent and identically distributed across time. The mean arrival rates are denoted by $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_N]$, where $\lambda_n \in \mathbb{R}^+$. The mean service rates are referred as $\boldsymbol{\mu} \in [\mu_{nk}]_{N \times K}$, where μ_{nk} is the service rate of server k for queue n. At a time, a server can serve the jobs coming from a queue only. We assume the queue to be stable which means

the maximum arrival rates of queues is bounded by maximum service rate of servers $\lambda_n < \max_{k \in [K]} \mu_{nk}$ for all queues $n \in [N]$.

Now, the problem is to choose such pairs of queues and servers at each time $t \in [T]$ such that the number of jobs waiting in queues is as low as possible. The number of jobs waiting in queues is called the *queue length* of the system. The agent, which is the scheduling algorithm in this case, tries to minimise this queue length for a given horizon $T > 0$. The arrival rates are known to the scheduling algorithm but the service rates are unknown to it. This creates the need to learn about the service distributions, and in turn, engenders the exploration-exploitation dilemma.

We assume that the first order Markov property holds for the scheduler. Hence, the scheduling decision at time t is based on the observations available at time $t-1$. If $a_t = [(n_1, k_1), \ldots, (n_{\min(N,K)}, k_{\min(N,K)})]$ is the set of queue-server matchings chosen by the scheduler at time t, the service provided by the chosen servers at time t is the set of samples obtained from the queueing distributions corresponding servers and queues in the matching. Thus, the reward obtained from the system at time t is the value of the service provided $X_{a_t} = S(t)$, and the queue-server pairs are the corresponding arms. If the number of arrivals to the queues at time t is $A(t)$, the queue length at time t is defined as

$$Q(t) \triangleq Q(t-1) + A(t) - S(t),$$

where $Q : [T] \to \mathbb{R}^{\geq 0}$, $A : [T] \to \mathbb{R}^{\geq 0}$, and $S : [T] \to \mathbb{R}^{\geq 0}$. Our goal is to investigate how bandit algorithms or their variants can be used to minimise this queue length over a finite horizon T. [Krishnasamy et al., 2016] proposed this formulation to study the effect of queueing behaviour on bandit algorithms. They termed this problem formulation as *queueing bandit* that we follow in this chapter.

Following the bandit literature, [Krishnasamy et al., 2016] proposed to use *queue regret* as the performance measure of a queueing bandit algorithm. Queue regret is defined as the difference in the queue length if a bandit algorithm is used instead of an optimal matching algorithm with full information about the arrival and service rates. The optimal matching includes queue-server pairs such that each queue with a

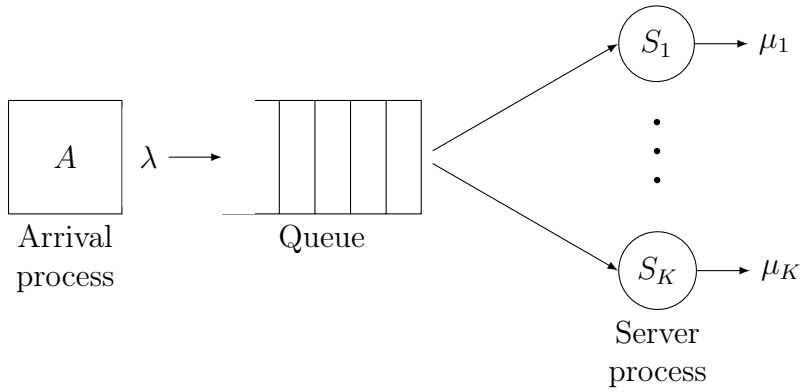


Figure 6.1: A queuing system $(A/S/K)$ with arrival rate λ and service rates $\mu_1, \ldots, \mu_K$.

corresponding server such that the service rate of the corresponding server is maximum for that queue. In order to assure the existence of an optimal matching, we assume that there exists a unique optimal server k_n^* for each queue n such that $\mu_{nk_n^*} = \max_k \mu_{nk}$, and for any two queue n and n', the optimal servers are not the same i.e. $k_n^* \neq k_{n'}^*$. Thus, the *optimal algorithm* OPT knows all the arrival and service rates, and allocates the queue to servers with the best service rate. We define the queue length corresponding to this algorithm as the optimal queue length $Q^{\mathrm{OPT}}(t)$. Hence, we define the queue regret of a queueing bandit algorithm $\mathcal{A}$ as

$$\Psi(t) \triangleq \mathbb{E}\left[Q(t) - Q^{\mathrm{OPT}}(t)\right]. \tag{6.1}$$

In order to keep the bandit structure, we assume that both the queue length $Q(t)$ of algorithm $\mathcal{A}$ and that of the optimal algorithm $Q^{\mathrm{OPT}}(t)$ starts with the same stationary state distribution $\nu(\boldsymbol{\lambda}, \boldsymbol{\mu})$. This initial state distribution $\nu(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is the steady state distribution of the system induced by the optimal policy. Under these assumptions, the goal of the algorithm $\mathcal{A}$ is to minimise the queue regret $\Psi(T)$ for a given time horizon T.

We illustrate such a queueing system with single queue and K servers in Figure 6.1. It is a $A/S/K$ queueing system with arrival process A, arrival rate λ, service process S, and service rates $\mu_1, \ldots, \mu_K$ respectively. In case of single queues, the queueing bandit become simpler and almost synonymous to the finite-arm stochastic bandit

problem. Here, we assume existence of the optimal server with maximum service rate $\mu^* = \max_k \mu_k$. The optimal algorithm allocates the jobs coming from the queue to the optimal server. The scheduler schedules the jobs to different server that causes different service times, and in turn, estimation of service rates of them. The goal is to minimise the corresponding queue regret, or alternatively, to minimise the queue length. We assume the system to be stable which means $\lambda < \mu^*$. We define the difference between the maximum service rate and the arrival rate as the load of the system $\epsilon \triangleq \mu^* - \lambda > 0$. We show theoretical results on a single-queue system because as per queueing theory a single queue system is more efficient than a multiple queue system for job allocation [Shortle et al., 2018]. Additionally, for multiple queue system, we can show the queue regret of the system to be the sum of the queue regrets of each queue. Since the optimal servers are different for different queues and operate independently, optimising the queue regret of the system is equivalent to optimising the queue regrets of individual queues. Thus, it suffices to analyse the single queues for practical application purposes, Though for the sake of generality, we will investigate both of the single queue and multiple queue systems in the experiments.

6.3.1 $M/B/K$ Queueing Bandit

In the $M/B/K$ queueing bandit, we assume the arrival process to be Markovian, and the service process to be Bernoulli. For simplicity, we present the calculations and the model for a single queue system. Tha arrival process being Markovian means that the stochastic process describing the number of arrivals $A(t)$ has increments independent of time. This makes the distribution of $A(t)$ to be a Poisson distribution [Durrett, 2010] with mean arrival rate λ. We denote the Bernoulli distribution of the service time of server k to be $B_k(\mu_k)$. $B_k(\mu_k)$ implies that the server processes a job with probability $\mu_k \in (0, 1)$ and refuses to serve it with probability $1 - \mu_k$. As mentioned earlier, at each time a job in the queue can be served by at most one server. The problem is to determine which server will serve a specific job with minimum possible failure.

Let a_t be the server that is being scheduled at time t, and $S(n)$ be the service offered by the server a_t, i.e., $S(n) = X_{a_t} \in \{0, 1\}$. The queue length $Q(t))$ is interpreted as the number of jobs that the $(t-1)$-th departure leaves behind. If $A(t)$ is the number of customers that join the queue between time t and $t+1$, then

$$Q(t+1) = \begin{cases} Q(t) - X_{a_t} + A(t) & \text{if } Q(t) > 0 \\ A(t) & \text{if } Q(t) = 0 \end{cases}$$

This implies that if the server k is chosen, the distribution of queue length will be sampled from the distribution

$$\mathbb{P}(Q(t+1) \leq A(t)) = (1 - \frac{\lambda}{\mu_k}) \left[1 + \frac{\lambda}{\mu_k(1-\mu_k)} \sum_{i=1}^{A(t)} \frac{\lambda(1-\mu_k)}{\mu_k(1-\lambda)}^{i-1} \right].$$

Thus, each time the queue k is chosen the queue length increases by an amount sampled from this distribution, and the scheduling algorithm has to allocate one of them to a server. The goal is to perform the scheduling in such a way that the queue regret will be minimised.

6.3.2 $M/M/K$ Queueing Bandit

Though the $M/B/K$ queue provides a well-behaved scenario to analyse the theoretical properties, it is not practical in majority of real-life applications. A Bernoulli service process implies that either a job is served or not with a certain probability, which is not the case in real-life; rather, servers serve each job with a certain delay. Thus, the processing time of the jobs can be delayed but they are not going to remain unserviced.

This brings us to investigation of a more realistic and widely used $M/M/K$ model of queues. In the $M/M/K$ model, both the arrival and service processes are Markovian. This means that service delay of one job does not depend on the service delay of the previous job. Thus, we obtain the service time to form an exponential distribution with mean service rate μ_k for server k. This implies that the average delay in service

for each job is μ_k for server k. Thus, we similarly define the growth in queue as

$$Q\left(t+1\right) = \begin{cases} Q\left(t\right) - X_{a_t} + A\left(t\right) & \text{if } Q\left(t\right) > 0 \\ A(t) & \text{if } Q\left(t\right) = 0 \end{cases}$$

where X_{a_t} is sampled from an exponential distribution with parameter μ_{a_t}. This implies that if the server k is chosen, the distribution of queue length will be sampled from the distribution

$$\mathbb{P}\left(Q(t+1) \leq A(t)\right) = (1 - \frac{\lambda}{\mu_k}) \left[1 + \sum_{i=1}^{A(t)} (\frac{\lambda}{\mu_k})^i\right].$$

Thus, each time the queue k is chosen the queue length increases by an amount sampled from this distribution, and the scheduling algorithm has to allocate some of them to a server. The goal is to perform the scheduling in such a way that the queue regret will be minimised.

6.4 Methodology

In this section, we provide brief description of the state-of-the-art Q-ThS and Q-UCB algorithms. Following that, we discuss our algorithm QBelMan. We conclude this section by providing theoretical performance bounds of BelMan.

6.4.1 Q-ThS and Q-UCB: The state-of-the-art Algorithms

Q-UCB and Q-Thompson sampling (Q-ThS) [Krishnasamy et al., 2016] are variants of the classical bandit algorithms UCB [Auer et al., 2002] and Thompson sampling [Thompson, 1933] proposed for the queuing bandits with Markovian arrival process and unknown Bernoulli service process. They facilitate an initial exploration window followed by the UCB or Thompson sampling algorithm for exploration–exploitation trade-off. In the initial exploration window, the servers are chosen uniformly randomly. This

facilitates gain in information about the service rates of the servers. This window is assigned to be $\frac{3K\log^2 T}{T}$ for a given horizon T and number of servers K. Following this, the standard versions of UCB and Thompson sampling are used respectively for Q-UCB and Q-ThS. [Krishnasamy et al., 2016] also proved that the queue regret from Q-ThS is upper bounded by $O(\log^3 t)$ and lower bounded by $\Omega(\frac{\log t}{\log\log t})$ for the early stage when the queue regret is growing. They also proved that for the late stage where the queue regret decays, the queue regret of Q-ThS is upper bounded by $O(\frac{\log^3 t}{t})$ and lower bounded by $\Omega(\frac{1}{t})$. This transition from early stage to late stage depends on the load of the queueing system ϵ. The smaller is ϵ, the longer it takes the algorithm to converge to the decay in queue length for the late stage. The limitation of their algorithms is the setting of Bernoulli service distribution. Since $M/B/K$ queueing system is hardly available in real-life applications, and Q-UCB and Q-ThS are fine tuned for $M/B/K$ system but not for $M/M/K$ systems.

6.4.2 QBelMan: BelMan for Queueing Bandits

Following the same algorithm design technique, we extend BelMan to QBelMan for the online scheduling in the queueing systems. Here, each server is treated as an arm and the corresponding distribution of queue length as the reward distribution. QBelMan differs from BelMan by an initial phase of exploration analogous to the modification to Q-ThS from Thompson sampling, and to Q-UCB from UCB.

Similar to BelMan, QBelMan maintains belief distributions over the service rates of the service distributions of the servers. QBelMan constructs the belief-reward manifold using the joint distribution over the service rate and the service delay. It begins with an initial prior distribution, or initial belief distribution, over the unknown service parameters of the servers. QBelMan performs an initial phase of exploration to gain more information and to update the belief distributions. Following that, QBelMan performs the I-projection and reverse-I projections alternatively in order to select the server to send the job to, and to update the pseudobelief-focal-reward distribution

Algorithm 21 QBelMan

Input: Time horizon T, Number of arms K, Prior on parameters B_0.
for $t = 1$ **to** T **do**
  if $rand(0,1) \leq \frac{3K \log^2 T}{T}$ **then**
    Play an arm a_t uniformly randomly.
  else
    /* *I-projection* */
    Draw arm a_t such that

$$a_t = \arg\min_k D_{\mathrm{KL}} \left(\mathbb{P}^k_{t-1}(S,\theta) \| \bar{\mathbb{Q}}_{t-1}(S,\theta) \right). \tag{6.2}$$

    /* *Accumulation of observables* */
    Sample a service time $S(t)$ out of $f_{\mu_{a_t}}$.
    Update the belief-reward distribution of a_t to $\mathbb{P}^{a_t}_{t+1}(S,\theta)$ using Bayes' theorem.
    /* *Reverse I-projection* */
    Update the pseudobelief-reward distribution to

$$\bar{\mathbb{Q}}_t(S,\theta) = \arg\min_{\bar{\mathbb{Q}} \in \mathcal{B}_\theta \mathcal{R}} \sum_{k=1}^{K} D_{\mathrm{KL}} \left(\mathbb{P}^k_t(S,\theta) \| \bar{\mathbb{Q}}(S,\theta) \right). \tag{6.3}$$

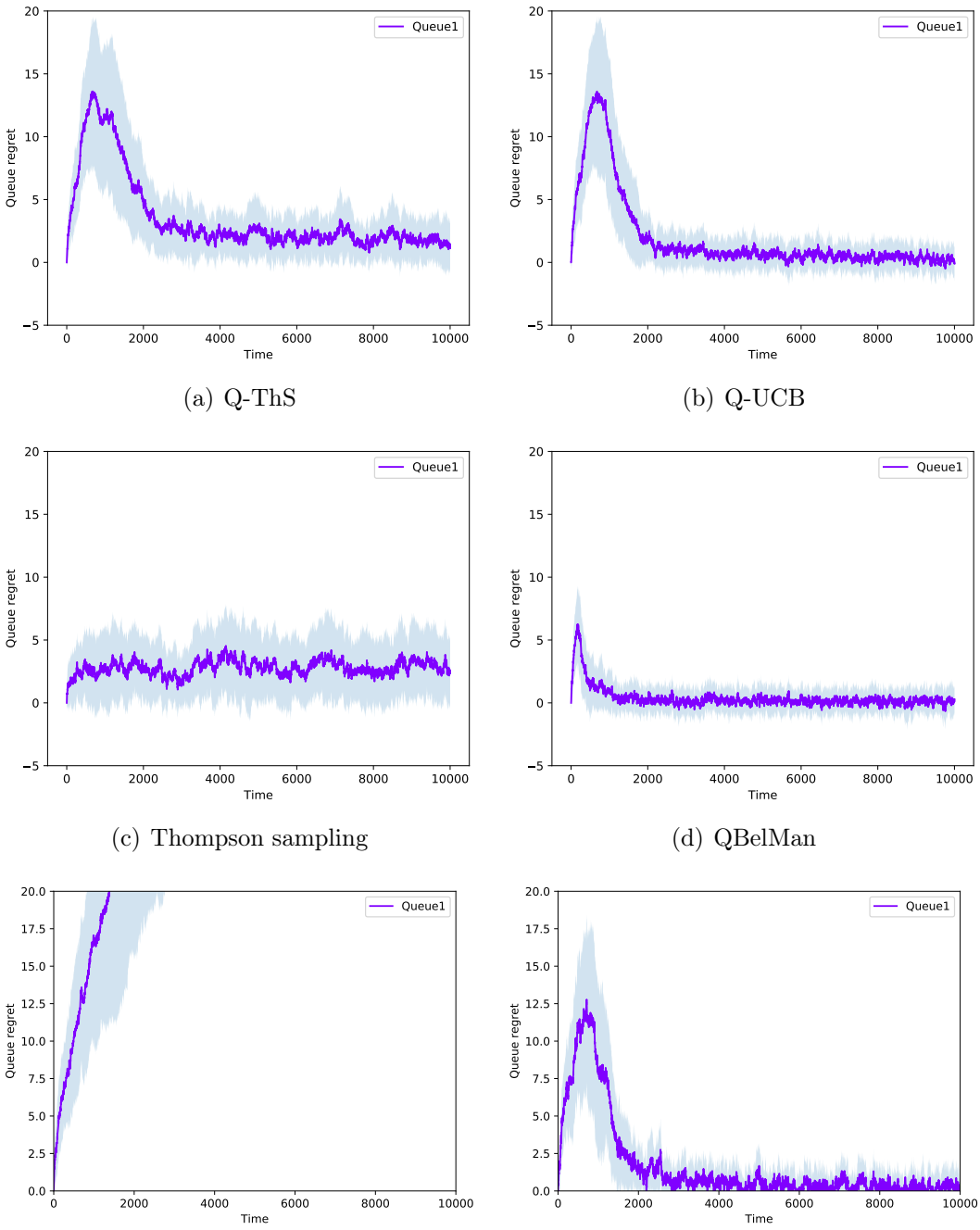
  end if
end for

over the service delay and the service rate. The pseudocode of QBelMan is elaborated in Algorithm 21.

Though the initial window of exploration accumulates more information, it also grows the queue regret initially but following that QBelMan comes back to the original BelMan. This design technique ensures initial information gain as well as retaining the asymptotic consistency of BelMan

6.5 Experimental Analysis

In this section, we experimentally verify BelMan's performance for exploration-exploitation bandits, and two-phase bandits respectively. We also comparatively analyse its performance with respect to the state-of-the-art algorithms QUCB, QThS, and, Thompson sampling. We empirically verify it to achieve logarithmic regret bound. We use Python



(a) Q-ThS

(b) Q-UCB

(c) Thompson sampling

(d) QBelMan

(e) Random exploration

(f) BelMan

Figure 6.2: Queue regret for 1 queue and 5 server setting with Poisson arrival and Bernoulli service distribution. The dark line in the middle shows the mean queue regret whereas the shaded area shows 33 percentiles of queue regret below and above it.

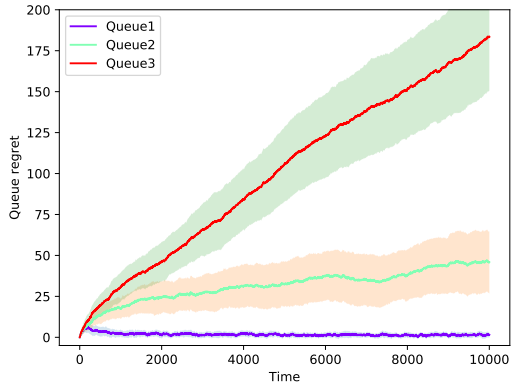
2.7 to develop a library [1] for solving queueing bandit problems using the aforementioned algorithms, BelMan, QBelMan, and random exploration for both the Bernoulli and exponential service distributions. We simulate the algorithms for each bandit algorithms for 10,000 iterations and 100 runs

Single queue $M/B/5$ bandit. We run the experiments for single queue and 5 servers setting with Poisson arrivals and Bernoulli service distributions. We set the arrival rate to be $\lambda = 0.35$. We set the service rates to be $[0.5, 0.33, 0.33, 0.33, 0.25]$. This makes the first server with service rate 0.5 to be the optimal choice. Since the load $\epsilon = 0.5 - 0.35 = 0.15 > 0$, the queueing system is stable. We illustrate the result of 100 simulations for 10000 time steps in Figure 6.2. The dark line in the middle shows the mean queue regret whereas the shaded area shows 33 percentiles of queue regret below and above it.

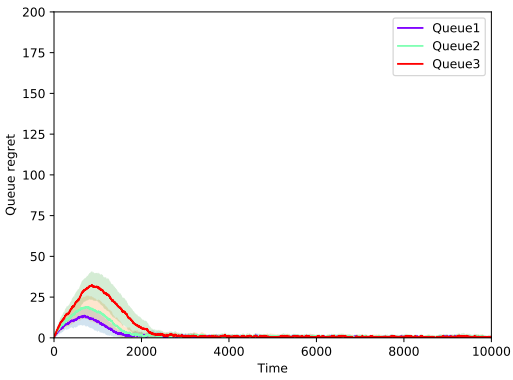
We comparatively evaluate the performance of Q-ThS, Q-UCB, Thompson Sampling, Q-BelMan, Random exploration, and BelMan respectively. Figure 6.2 illustrates the results. Figure 6.2 validates the shape of the queue regret to satisfy the theoretical bounds for the asymptotically optimal algorithms except random exploration. Random exploration does not satisfy the pattern as it leads to linear number of choice of suboptimal servers. Figure 6.2 shows that QBelMan performs efficiently than the competing algorithms with a significant decrease in queue regret. Interestingly, though the maximum queueing regret for Thompson sampling is less the other algorithms, it does not converge. Thompson sampling shows critically variance and higher regret than Q-UCB, BelMan, and Q-BelMan in the later stage.

Three queue $M/B/5$ bandit. Though we illustrate the results for single queue system, we extend the experiments to a queueing system with three queues and five servers. We perform the experiments with arrival rates 0.35 for each one of them. We

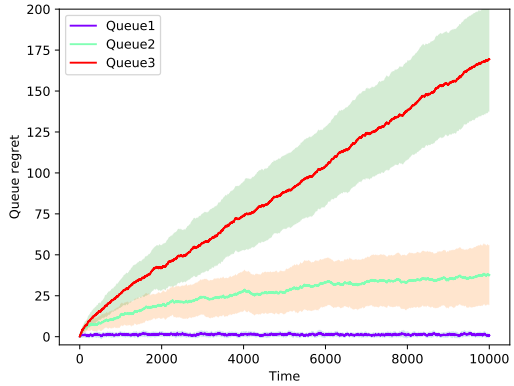
[1] The git repository: https://github.com/Debabrota-Basu/QBelMan



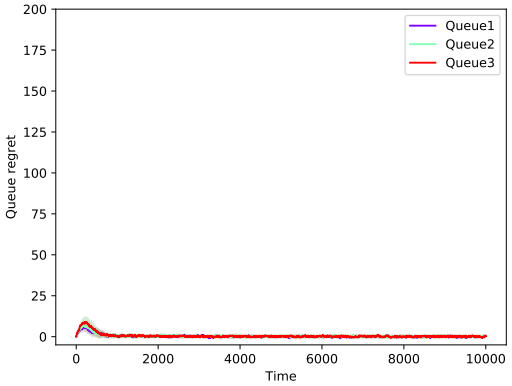
(a) Q-ThS



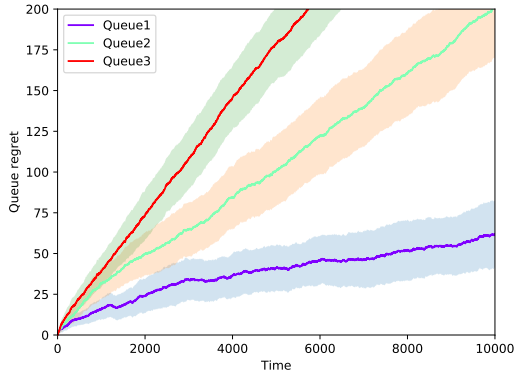
(b) Q-UCB



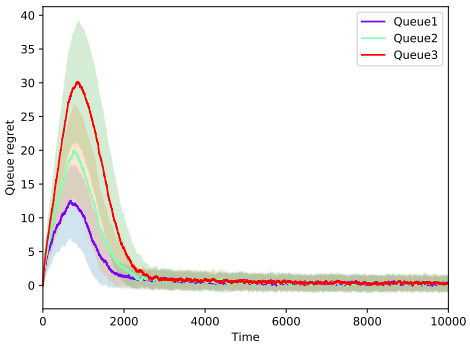
(c) Thompson sampling



(d) QBelMan



(e) Random exploration



(f) BelMan

Figure 6.3: Queue regret for 3 queue and 5 server setting with Poisson arrival and Bernoulli service distribution.

assign the service rates to be a 5×3 matrix

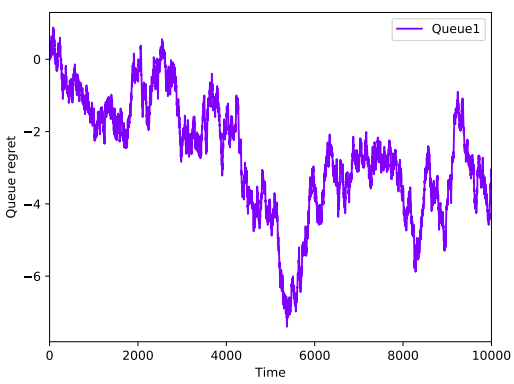
$$\boldsymbol{\mu} = [[0.5, 0.33, 0.33, 0.33, 0.25], [0.33, 0.5, 0.25, 0.33, 0.25], [0.25, 0.33, 0.5, 0.25, 0.25]].$$

This makes the optimal matching to be $\{(1,1),(2,2),(3,3)\}$. Since the loads are positive for every queue, the queueing system is stable.

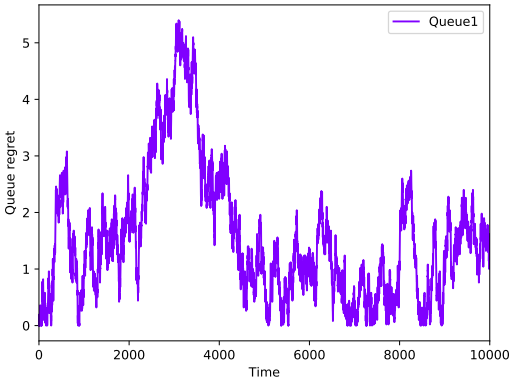
We comparatively evaluate the performance of Q-ThS, Q-UCB, Thompson Sampling, Q-BelMan, Random exploration, and BelMan respectively. Figure 6.3 shows that QBelMan performs significantly better than the competing algorithms with less queue regret and faster convergence rate. Additionally, we observe that as the service rates of suboptimal arms to be explored decreases the queue regret blows up for both Q-ThS, and Thompson sampling. This observation indicates that even the classical bandit algorithms of Thompson sampling family may not be stable under queueing setup.

Single queue $M/M/5$ bandit. We run the experiments for single queue and 5 servers setting with Poisson arrivals and Exponential service distributions. We set the arrival rate to be $\lambda = 0.5$. We set the service rates to be $[1, 0.9, 0.5, 0.5, 0.33]$. This makes the first server with service rate 1 to be the optimal choice. Since the load $\epsilon = 1 - 0.5 = 0.5 > 0$, the queueing system is stable.

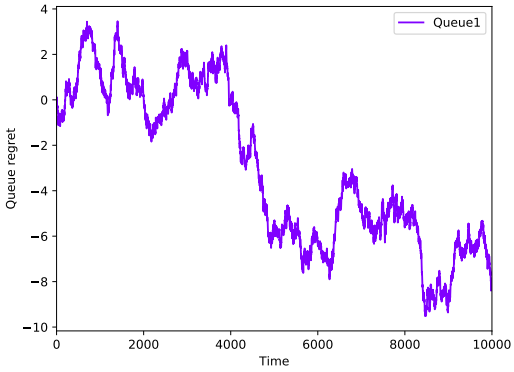
We comparatively evaluate the performance of Q-ThS, Thompson Sampling, Q-BelMan, Random exploration, and BelMan respectively. We cannot apply Q-UCB in this M/M setting because it is not generalisable to exponential service distributions. We illustrate the results in Figure 6.4. In Figure 6.4, we observe the pattern to be significantly different than that of the Bernoulli service distributions. The variance of the queue regret is quite high for all of the algorithms. Still QBelMan incurs less queue regret than other algorithms while random exploration incurs the highest queue regret. This shows not only generality of QBelMan as a framework that acts on eclectic service distribution families while perform efficiently with respect to the state-of-the-art algorithms.



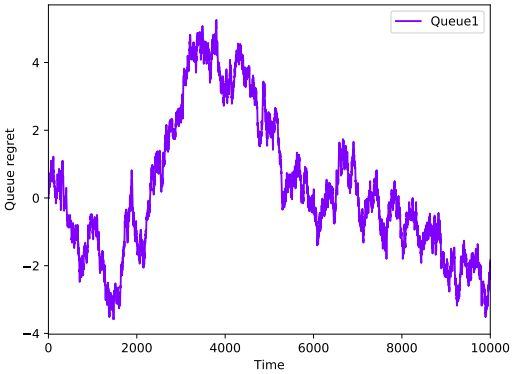
(a) Q-ThS



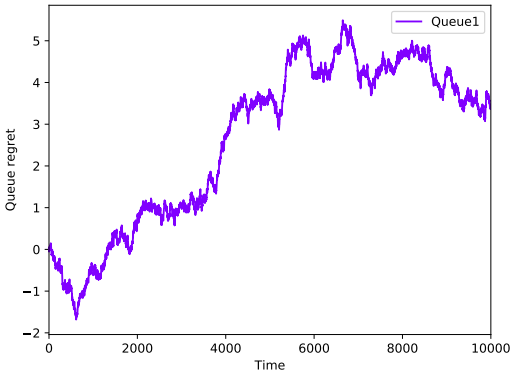
(b) Thompson sampling



(c) QBelMan



(d) BelMan



(e) Random exploration

Figure 6.4: Queue regret for 1 queue and 5 server setting with Poisson arrival and exponential service distribution.

6.6 Conclusion

In this chapter, we formulate the job scheduling in a multiple-queue multiple-server system with known arrival rates and unknown service rates as a finite-arm stochastic bandit problem. The difference between the queueing bandit and the traditional bandit setup is that the performance measure is queue regret that depends on the arrival process, and queueing behaviour. Following the algorithm design technique of [Krishnasamy et al., 2016], we propose a variant of BelMan with an initial exploration window to address the queueing bandit problem. As the existing literature is built around the $M/B/K$ systems, we investigate performance of BelMan both theoretically and experimentally for this type of systems. BelMan not only achieves the same order of growth in early stage and decay in late stage as the state-of-the-art algorithms but also performs significantly better than all of them. In order to instantiate the generality of BelMan, we evaluate it for $M/M/K$ queues. The experiments show that BelMan incurs less queue regret than the competing algorithms. This proves both efficiency and generality of BelMan as an algorithmic framework for bandits.

There is still no available bounds on the pattern of change of queue regret with time for $M/M/K$ systems. We are investigating similar bounds for $M/M/K$ queues. Additionally, queues are traditionally studied as MDPs. Specifically, in their transient state, multiple-queue multiple-server systems are scheduled as MDPs. We are exploring queueing systems to extend this information geometric frameworks from bandits to MDPs. A path towards this development that we are working on is discussed in Chapter 7.

Chapter 7

The Closure

We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time.

— *Thomas Eliot, Little Gidding, 1942.*

Every conclusion is a closure of an exploration–exploitation dilemma between continuing the research further and communicating it through a thesis. While facing the same dilemma in our investigation to the problem of learning to make decisions with incomplete information, we decide to draw a closure to this thesis in this chapter. Thus, we begin this chapter with a brief concluding discussion of the findings in the preceding chapters (Section 7.1). The conclusions are not the terminal states of the research directions explored in this thesis. Rather the important things are the perspectives developed through them and how they provide a direction to a unified perspective to reinforcement learning. We discuss this briefly in Section 7.2. In Section 7.3, we briefly mention the immediate future works that we are pursuing now.

7.1 Conclusions

In this thesis, we have discussed about the problem of learning to make decisions with incomplete information from a reinforcement learning formalism. We principally

deal with two problem settings- multi-armed bandits and Markov decision processes. Theoretically, we address three scenarios: Markov decision processes without a known reward function (Chapter 3), with an unknown transition function (Chapter 4), and multi-armed bandits with an unknown reward function (Chapters 5 and 6). In order to do so, we use the tools of online optimisation and functional approximation for the first two problems (Chapters 3 and 4). For the last one, we generalise our notion of functional approximation as that of estimating unknown distributions with an information geometric framework (Chapters 5 and 6). We apply the developed methodologies to address three real-life application problems, such as automated database tuning [Basu et al., 2015a,b, 2016], energy and performance efficient live migration of virtual machines [Basu et al., 2017c,b], and online scheduling in queuing systems [Basu et al., 2018a].

We begin this body of work with a primer on reinforcement learning in Chapter 2. Specifically, Chapter 2 discusses the problem formulation and state-of-the-art algorithms for multi-armed bandits and Markov decision processes. This chapter clarifies and justifies the problem models and assumptions pursued in the following chapters. Chapter 2 also posits the works of Chapters 3, 4, 5, and 6 in the corresponding state-of-the-art from a methodological perspective.

Chapter 3 proposes and validates that we can solve a Markov decision process with an unknown reward function by using a regularised function estimator with compatible stability guarantees in parallel with an online reinforcement learning algorithm. We instantiate these methodologies through two automated index tuning algorithms without cost models, COREIL and rCOREIL [Basu et al., 2015a,b, 2016]. These algorithms perform efficiently and effectively than the state-of-the-art algorithms for index tuning with a fixed cost model.

Chapter 4 proposes and validates that we can solve a Markov decision process with unknown transition function by projecting the observed transition function in a well-designed feature space with convergence guarantees in parallel with an online reinforcement learning algorithm. The proposed algorithm, Megh [Basu et al., 2017c,b],

experimentally outperforms the state-of-the art live virtual machine migration algorithms and even the reinforcement learning algorithms that assumes the correct model for the transition function is known.

Chapter 5 investigates the multi-armed bandits where the reward distributions are unknown, and have to be estimated by the agent. We use this scope to develop a generic information geometric framework to address the issues of exploration–exploitation trade-off [Basu et al., 2018d]. We leverage this to design an algorithm, BelMan [Basu et al., 2018c], that uniformly addresses the classical stochastic bandit problem, pure exploration bandit problem and two-phase reinforcement learning problem. We theoretically prove that BelMan is asymptotically consistent, and experimentally validate its efficiency for bandits with continuous rewards and multiple arms.

Chapter 6 instantiates the information geometric framework of BelMan for online scheduling of jobs in a multiple-queue multiple-server system with unknown arrival and service rates [Basu et al., 2018a]. We theoretically prove BelMan to be asymptotically optimal algorithm and experimentally illustrate that it outperforms the state-of-the-art algorithms for servers with Bernoulli distributions. These results validate applicability in addition with generality of BelMan as a framework. We also extended this methodology for queues with Markovian service process. The experimental results prove BelMan's applicability to real-life applications, while the state-of-the-art which is designed to serve only the Bernoulli service distributions.

7.2 Perspectives

Here, we discuss the perspectives that we obtain through development of an information geometric view towards the exploration–exploitation trade-off problem in reinforcement learning. This part proposes to construct the belief-reward manifold in order to investigate the information accumulation, processing and learning of an agent systematically and geometrically. We develop in this direction with the problem for-

mulation of multi-armed bandits. Multi-armed bandits provide us the initial ground to develop the theory without worrying about the planning components of MDP. The investigation of Chapters 5 and 6 builds a perspective towards exploration-0exploitation in different variants of bandits. We also reach a potential extension of this methodology to MDPs. We present this extension conceptually in [Basu et al., 2018d]. We elaborate the concept and the connection in this section.

7.2.1 An Information Geometric Approach to MDP

In Chapter 5, we present the information-geometric approach to multi-armed bandits. As discussed in Section 2.1, the environment $\mathcal{E}$ in multi-armed bandits is constituted by K reward distributions $f_{\theta_1^{true}}(X), \ldots, f_{\theta_K^{true}}(X)$ with unique parametrisations $\theta_1^{true}, \ldots, \theta_K^{true} \in \mathbb{R}^d$. The parameters $\theta_1^{true}, \ldots, \theta_K^{true}$ are not known to the agent $\mathcal{A}$. Thus, she begins with a set of distributions $\{b(\theta_1), \ldots, b(\theta_K)\}$ representing the uncertainty of the corresponding parameters. These distributions are called belief distributions as they probabilistically represent the belief of the agent about the corresponding parameter values. Since the true values of the parameters can be accurately estimated only after accumulating infinite number of samples from each of them, the agent always makes decisions on the basis of the belief distributions and not the true environment. Thus, decision making with incomplete information is actually decision making in a belief space which is sequentially updated by accumulating samples from the environment. Since the agent interacts with the belief space to get more information, learning the belief distributions corresponding to the arms play a central role in exploration, and thus, in bandit algorithms.

Still the beliefs do not suffice to represent the uncertainty regarding the interaction of the agent with the environment. Since reward generation from the arms itself is stochastic in nature, we have to consider the reward distributions also to model the uncertainty of interaction between the agent and the environment. This motivated us to construct the joint space of belief and reward distributions, which is called the belief-reward manifold. Belief-reward manifold provides us a framework to update the

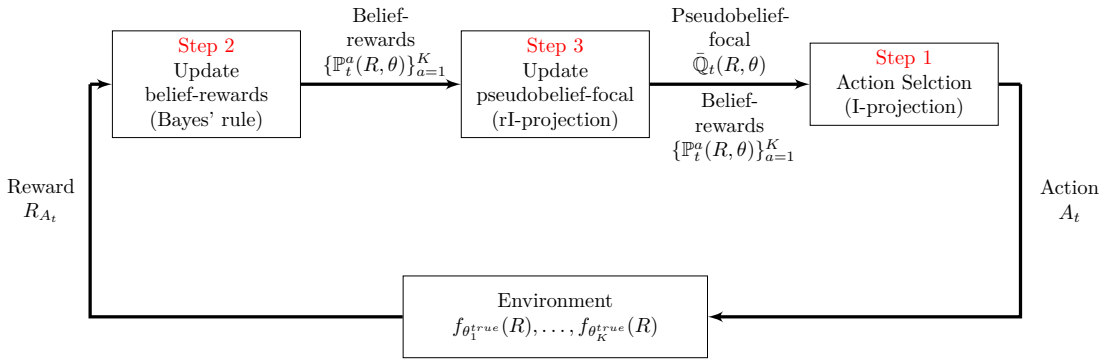


Figure 7.1: The block diagram of BelMan algorithm for multi-armed bandits

estimation of belief distributions, to construct a collective summary of knowledge as pseudobelief-reward, and to select the arm to play. In BelMan, we perform the update through Bayes' rule, and the later steps through alternating information projection. The block diagram of BelMan is shown in Figure 7.1 as it puts all the aforementioned components in a loop.

Figure 7.1 shows that the decision making block of BelMan interacts with all other blocks that can be represented as interaction with the belief-reward manifold. It also indicates the fact that better inference is needed to fulfil the goal of maximising the cumulative reward. As we discussed earlier in Section 2.1, they should go interactively. This is reflected in the objective function used for action selection in BelMan

$$\begin{aligned} A_t &\triangleq \arg\min_a D_{\mathrm{KL}}\left(\mathbb{P}_{t-1}^a(X,\theta)\|\bar{\mathbb{Q}}_{t-1}(X,\theta)\right) \\ &= \arg\max_a \left(\mathbb{E}_{\mathbb{P}_{t-1}^a(X,\theta)}\left[X\right] - \tau(t)\ D_{\mathrm{KL}}\left(b_{t-1}^a(\theta)\|b_{\bar{\eta}_{t-1}}(\theta)\right)\right) \end{aligned} \tag{7.1}$$

Equation (7.1) shows that the chosen action A_t does not maximise only the estimate of reward at each step but a regularised estimate of reward. The regularisation function includes the KL-divergence between the belief distribution of the arm a and the pseudobelief representing the collective knowledge over the arms. The regularisation factor is called the exposure, and is a decreasing time dependent function $\tau(t)$. Thus, the sequence of actions $\{A_t\}_{t=1}^T$ does not maximise the cumulative reward $\sum_{t=1}^T \mathbb{E}_{P^{A_t}}[X]$ but a weighted sum of rewards and KL-divergences $\sum_{t=1}^T \left[\mathbb{E}_{P^{A_t}}[X] - \tau(t) D_{\mathrm{KL}}\left(b^{A_t}(\theta)\|b_{\bar{\eta}_t}\right)\right]$.

This regularisation imposes an exploration bias to facilitate learning, and thus induces the exploration–exploitation trade-off.

Now, we extend this framework of decision making with incomplete information, and interaction between the agent and the belief of environment to the exploration–exploitation dilemma in MDPs (Section 2.2.9). Specifically, let us consider that we have an MDP $\mathcal{M} = \langle S, A, \mathcal{P}, R \rangle$. Thus, we model the environment $\mathcal{E}$ is a state-action space $S \times A$. The agent interacts with the state-action space and obtains the observations from it. Since the exact model of interaction between states and actions is not available to the agent, the agent tries to compute a randomised policy $\pi : S \times A \to [0, 1]$ [1] depending on her belief about the occurrence of a state-action pair (s, a). Thus, the randomised policy of the agent along with the transition function $\mathcal{P} : S \times A \times S \to [0, 1]$ induces a stationary distribution $B_\pi(s, a)$ over the state-action space, and a stationary distribution $\nu_\pi(s)$ over the state space. [2] A stationary distribution over the state space is defined as $\nu_\pi : S \to [0, 1]$ such that it satisfies the mass-balance equation [Puterman, 2009]

$$\nu_\pi(s') = \sum_{s \in S} \sum_{a \in A} [\mathcal{P}(s' \mid s, a) \times \pi(a \mid s) \times \nu_\pi(s)] \quad \forall \ \ s' \in S.$$

This implies that the distribution ν_π is unique for a given policy π [Szepesvári, 2010]. It also implies that the probability of occurrence of a state while using the given policy and the transition function is balanced and computable as a mass-balance equation. The stationary distribution $B_\pi : S \times A \to [0, 1]$ over the state-action space $S \times A$ is defined as

$$B_\pi(s, a) \triangleq \nu_\pi(s) \ \pi(s \mid a) \qquad \forall \ \ (s, a) \in S \times A.$$

[1] Alternatively, we can express the randomised policy as $\pi : S \to \Delta_A$, where Δ_A is the set of probability distributions over the action space.

[2] Here, we assume the MDP to be unichain. [Puterman, 2009] shows that an MDP is unichain under mild conditions, such as all stationary policies induce an irreducible and aperiodic Markov chain over state-action space.

This formulation of B_π allows us to define the average reward of a policy π as

$$\bar{V}(\pi) \triangleq \sum_{s\in S}\sum_{a\in A} B_\pi(s,a)\mathcal{R}(s,a) = \mathbb{E}_{(s,a)\sim B_\pi}[R(s,a)], \tag{7.2}$$

where $R(s,a) \in \mathbb{R}$ is the reward obtained by taking action a at state s. This shows that B_π is the distribution to be learned by the agent in order to control the average reward per-step, and also to maximise the total reward obtained through the MDP. Thus, in analogy with the bandits, we call B_π the belief of the agent for a given policy π. Equation (7.2) also shows that in order to maximise the average reward per-step the random variable that we deal with is $B_\pi(s,a)R(s,a)$ [3]. This is a random variable defined as a product of belief and reward which is evaluated for each state-action pair and a given policy. Hence, it motivates us to extend the belief-reward manifold framework to MDPs. We construct the belief-reward manifold $\mathcal{BR}$ as the set of all distributions over the product of belief and reward $B_\pi(s,a)R(s,a)$ for all policies $\pi \in \Pi$ [4]. Thus, the objective of maximising average reward in an MDP is equivalent to finding $\mu^* \in \mathcal{BR}$ which is $\arg\max_{\mu_\pi\in\mathcal{BR}} \bar{V}(\pi)$.

Since the reward function is often not controllable by the agent, thus it makes sense to incorporate it in learning but not in optimisation. Thus, solving the MDP is often equivalent to finding the optimal belief distribution $B^* \triangleq B_{\pi^*}$ [5] over state-action space such that $B^* = \arg\max_{B_\pi\in\mathcal{B}} \bar{V}(\pi)$. This equation instantiates the need of learning and optimisation in order to solve the MDP optimally. In order to do so and to balance the corresponding exploration–exploitation, we extend the Equation (7.1) of BelMan to MDPs such that at time t

$$B_t = \arg\max_{B\in\mathcal{B}} \ \left(\mathbb{E}_{B(s,a)}[R(s,a)] - \tau(t)\ D(B, B_{t-1})\right). \tag{7.3}$$

[3] Here, the reward obtained from a state-action pair $R(s,a)$ itself is a random variable.

[4] Π is the set of all feasible stationary policy inducing unichain MDP.

[5] We can define B^* as the belief distribution constructed by optimal policy π^* because each stationary policy induces a stationary belief distribution for a given MDP.

where D is an information-geometric divergence measure on the belief manifold, such as KL-divergence, Renyi divergence, Wassersetin distance, and so on, that incorporates the belief that the agent wants to compute now, and the belief that the agent had in last time step. The regularisation factor $\tau(t)$ can be a pre-defined constant or a time dependent function.

This approach takes us to the literature of intrinsically motivated reinforcement learning, and curiosity-driven reinforcement learning (Section 2.2.9). Here, the information-theoretic divergence plays as the intrinsic motivation to explore the environment, or curiosity to learn as they are called in literature. Specifically, we can visualise this extension of BelMan as a generalisation of information regularised MDPs (Section 2.2.9). If we design the function $D(B, B_{t-1})$ to be the KL-divergence between the belief to be computed and the belief of time $t-1$ i.e. $D(B, B_{t-1}) = D_{\mathrm{KL}}\left(B \| B_{t-1}\right)$, we obtain the relative entropy policy search (REPS) algorithms [Peters et al., 2010; Neu et al., 2017], and entropy guided mirror descent algorithms for policy search [Montgomery and Levine, 2016]. [Schulman et al., 2015; Mnih et al., 2016] designed the function $D(B, B_{t-1})$ to be the average [6] KL-divergence between the policy to be computed and the policy of time $t-1$ i.e. $D(\pi, \pi_{t-1}) = \mathbb{E}_{\nu_{\pi_t}}\left[D_{\mathrm{KL}}\left(\pi \| \pi_{t-1}\right)\right]$. If we plug it in Equation (7.3), we obtain the policy selection rule as

$$
\begin{aligned}
\pi_t &= \arg\max_{\pi \in \Pi} \quad \mathbb{E}_{s \sim \nu_{\pi_t}}\left[\mathbb{E}_{a \sim \pi}\left[R(s,a)\right] - \tau\ D_{\mathrm{KL}}\left(\pi(. \mid s) \| \pi_{t-1}(. \mid s)\right)\right] \\
\implies B_t &= \arg\max_{B_\pi \in \mathcal{B}} \quad \mathbb{E}_{B_\pi}\left[R(s,a)\right] - \tau\ \mathbb{E}_{B_\pi}\left[\log \frac{\pi(a \mid s)}{\pi_{t-1}(a \mid s)}\right].
\end{aligned}
$$

These results link this information geometric approach to the state-of-the-art literature. Hence, it indicates generality of the information-geometric approach to the MDP, and in general to the problem of decision making with incomplete information. This discussion also shows the possible avenues of investigation that this approach opens up, such as using different information geometric divergence function D, different representations of the knowledge-base beside B_{t-1}, and time dependent trade-off

[6] The expectation is taken over all possible states.

between exploration and exploitation using a time-dependent regularisation function $\tau(t)$. We would like to explore these information-geometric avenues and to exploit these perspectives in future research.

7.3 Future Work

As we have summarised our contributions and discussed the learned perspectives in the preceding sections, now we discuss the research problems that we are interested to pursue in follow-up work. We are pursuing presently two research works coming from these two avenues discussed in this thesis. One is to solve the MDP problem with unknown transition and reward functions using functional approximation methods. The other is to solve finite-horizon MDP problems with incomplete information using an extension of the information geometric approach proposed in BelMan.

In the first line of research using functional approximation techniques, we want to improve and merge the algorithms developed in rCOREIL and Megh that deal with MDPs with unknown reward and transition functions respectively. Such MDPs with unknown transition and reward functions are frequently encountered in real-life applications. We pick one such problem of optimising speed of ships under uncertain weather [Avgouleas, 2008; Motte et al., 1988] in order to instantiate the effectiveness and efficiency of the developed functional approximation methods. We propose in [Basu et al., 2017a] an algorithm deploying calculus of variation to solve the speed optimisation problem under full information about the weather and the ship. We are now extending it for the unknown ship models and unpredictable weather variations. We are also working on the fine tuning and further experimentation of the functional approximation techniques for this problem [Basu et al., 2018b].

The other line of research is inspired by the perspectives achieved from the information geometric works in bandits and the discussion in Section 7.2.1. We are now studying the information-geometric approach to MDPs, and how it unifies different approaches to solve exploration–exploitation problems in MDPs. This also indicates

towards a general algorithm design technique for MDPs using information-geometric divergences over the belief-reward manifold. We are now exploring different divergence functions and time-dependent regularisation factors that can provide better MDP solving algorithms than the existing ones with which the framework shares links. We are also working towards the proof of finite-time regret analysis of BelMan for multi-armed bandits that we would extend to prove a regret bound on performance of MDPs.

These two research works describe our present endeavour towards solving the problems of learning to make decisions with incomplete information efficiently, effectively, and in a more knowledgeable manner than where we began this thesis.

Bibliography

Agarwal, A., Foster, D. P., Hsu, D. J., Kakade, S. M., and Rakhlin, A. (2011). Stochastic convex optimization with bandit feedback. In *Advances in Neural Information Processing Systems*, pages 1035–1043.

Agrawal, S., Chaudhuri, S., and Narasayya, V. R. (2000). Automated selection of materialized views and indexes in sql databases. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, pages 496–505.

Agrawal, S. and Goyal, N. (2012). Analysis of Thompson sampling for the multi-armed bandit problem. In *COLT*, pages 39.1–39.26.

Agrawal, S., Narasayya, V., and Yang, B. (2004). Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*, pages 359–370.

Agueh, M. and Carlier, G. (2011). Barycenters in the wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924.

Alagiannis, I., Idreos, S., and Ailamaki, A. (2014). H2o: A hands-free adaptive store. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*.

Alsarhan, A., Itradat, A., Al-Dubai, A. Y., Zomaya, A. Y., and Min, G. (2018). Adaptive resource allocation and provisioning in multi-service cloud environments. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):31–42.

Amari, S.-I. and Nagaoka, H. (2007). *Methods of information geometry*, volume 191 of *Translations of mathematical monographs*. American Mathematical Society.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' guide*, volume 9. Siam.

Asadi, K. and Littman, M. L. (2016). An alternative softmax operator for reinforcement learning. *arXiv preprint arXiv:1612.05628*.

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

Audibert, J.-Y. and Bubeck, S. (2009). Minimax policies for adversarial and stochastic bandits. In *COLT*, pages 217–226.

Audibert, J.-Y. and Bubeck, S. (2010). Best arm identification in multi-armed bandits. In *COLT*, pages 41–53.

Audibert, J.-Y., Munos, R., and Szepesvári, C. (2009). Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19).

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2–3):235–256.

Auer, P. and Ortner, R. (2007). Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 49–56.

Avgouleas, K. (2008). *Optimal ship routing*. Thesis, Massachusetts Institute of Technology.

Baird, L. et al. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pages 30–37.

Baranes, A. and Oudeyer, P.-Y. (2009). Robust intrinsically motivated exploration and active learning. In *Development and Learning, 2009. ICDL 2009. IEEE 8th International Conference on*, pages 1–6. IEEE.

Barbaresco, F. (2013). Information geometry of covariance matrix: Cartan-siegel homogeneous bounded domains, mostow/berger fibration and frechet median. In *Matrix Information Geometry*, pages 199–255. Springer.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177.

Basu, D., Lin, Q., Chen, W., Vo, H. T., Yuan, Z., Senellart, P., and Bressan, S. (2015a). Cost-model oblivious database tuning with reinforcement learning. In *International Conference on Database and Expert Systems Applications*, pages 253–268. Springer.

Basu, D., Lin, Q., Chen, W., Vo, H. T., Yuan, Z., Senellart, P., and Bressan, S. (2016). Regularized cost-model oblivious database tuning with reinforcement learning. *Transactions on Large-Scale Data and Knowledge-Centered Systems*, 28:96–132.

Basu, D., Lin, Q., Yuan, Z., Senellart, P., and Bressan, S. (2015b). Apprentissage par renforcement pour optimiser les bases de donnéees indépendamment du modèle de coût. In *Proc. BDA*, Porquerolles, France. Conference without formal proceedings.

Basu, D., Pedrielli, G., Chen, W., Ng, S. H., Lee, L. H., and Bressan, S. (2017a). Sequential vessel speed optimization under dynamic weather conditions. In *International Maritime-Port Technology and Development Conference*.

Basu, D., Pedrielli, G., Senellart, P., and Bressan, S. (2018a). Belman in queue: An information geometric approach to queueing bandits with general distributions. Preprint.

Basu, D., Rahman, A., and Bressan, S. (2018b). Moq: A model oblivious learning algorithm to sequentially optimize speed of a vessel under uncertain weather. Submitting to IEEE Transactions on Intelligent Transportation Systems.

Basu, D., Senellart, P., and Bressan, S. (2018c). Belman: Bayesian bandits on the belief–reward manifold. *arXiv preprint arXiv:1805.01627*.

Basu, D., Senellart, P., and Bressan, S. (2018d). An information geometric analysis of explorationexploitation trade-off in reinforcement learning. Submitted to Workshop on Exploration in RL, ICML.

Basu, D., Wang, X., Hong, Y., Chen, H., and Bressan, S. (2017b). Learn-as-you-go with megh: Efficient live migration of virtual machines. Submitted to IEEE TPDS.

Basu, D., Wang, X., Hong, Y., Chen, H., and Bressan, S. (2017c). Learn-as-you-go with megh: Efficient live migration of virtual machines. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2608–2609. IEEE.

Bather, J. A. (1983). The minimax risk for the two-armed bandit problem. In Herkenrath, U., Kalin, D., and Vogel, W., editors, *Mathematical Learning Models — Theory and Algorithms*, pages 1–11, New York, NY. Springer New York.

Belady, C. L. (2007). In the data center, power and cooling costs more than the it equipment it supports.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

Bellman, R. (1956). A problem in the sequential design of experiments. *Sankhyā: The Indian Journal of Statistics (1933–1960)*, 16(3/4):221–229.

Bellman, R. (1957a). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

Bellman, R. (1957b). A markovian decision process. *Indiana University Mathematics Journal*, 6:679–684.

Bellman, R. and Kalaba, R. E. (1965). *Dynamic programming and modern control theory*, volume 81. Citeseer.

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5):755–768.

Beloglazov, A. and Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurr. Comput. : Pract. Exper.*, 24(13):1397–1420.

Benedikt, M., Bohannon, P., and Bruns, G. (2006). Data cleaning for decision support. In *Proceedings of the 1st International VLDB Workshop on Clean Databases (CleanDB'06)*.

Berlyne, D. E. (1966). Curiosity and exploration. *Science*, 153(3731):25–33.

Bernardo, J. M. (1976). Algorithm AS 103: Psi (digamma) function. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 25(3):315–317.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Bertsekas, D. P. (2013). *Abstract dynamic programming*. Athena Scientific Belmont, MA.

Bertsekas, D. P. (2017). Regular policies in abstract dynamic programming. *SIAM Journal on Optimization*, 27(3):1694–1727.

Bertsekas, D. P. and Shreve, S. (2004). *Stochastic optimal control: the discrete-time case*. Athena Scientific.

Bhargava, B. (1999). Concurrency control in database systems. *IEEE transactions on knowledge and data engineering*, 11(1):3–16.

Borodin, A. and El-Yaniv, R. (1998). *Online Computation and Competitive Analysis*. Cambridge University Press.

Boucheron, S., Lugosi, G., and Massart, P. (2013). *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57.

Brafman, R. I. and Tennenholtz, M. (2002). R-max: a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231.

Brown, L. D. (1986). *Fundamentals of Statistical Exponential Families: With Applications in Statistical Decision Theory*. Institute of Mathematical Statistics.

Bruno, N. and Chaudhuri, S. (2007a). An online approach to physical design tuning. In *Proceedings of the 23th IEEE International Conference on Data Engineering (ICDE'07)*, pages 826–835.

Bruno, N. and Chaudhuri, S. (2007b). Online autoadmin: (physical design tuning). In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD'07)*, pages 1067–1069.

Bruno, N. and Chaudhuri, S. (2008). Constrained physical design tuning. *Proceedings of the VLDB Endowment*, 1(1):4–15.

Bruno, N. and Chaudhuri, S. (2010). Interactive physical design tuning. In *Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE'10)*, pages 1161–1164.

Bruno, N. and Nehme, R. V. (2008). Configuration-parametric query optimization for physical design tuning. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, pages 941–952.

Bubeck, S., Cesa-Bianchi, N., et al. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122.

Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure exploration in multi-armed bandits problems. In *ALT*, pages 23–37. Springer.

Bubeck, S., Wang, T., and Viswanathan, N. (2013). Multiple identifications in multi-armed bandits. In *ICML*, pages 258–265.

Burnetas, A. N. and Katehakis, M. N. (1997). Optimal adaptive policies for markov decision processes. *Mathematics of Operations Research*, 22(1):222–255.

Bush, R. R. and Mosteller, F. (1953). A stochastic model with applications to learning. *The Annals of Mathematical Statistics*, pages 559–585.

Buyukkoc, C. (1985). c mu rule revisited. *Adv. Appl. Prob.*, 17(1):237–238.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Cappé, O., Garivier, A., and Kaufmann, É. (2012). pymaBandits. http://mloss.org/software/view/415/.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *AAAI*, volume 94, pages 1023–1028.

Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. (2017). Boltzmann exploration done right. In *Advances in Neural Information Processing Systems*, pages 6284–6293.

Chaudhuri, S. and Narasayya, V. (1998). Autoadmin: What-if index analysis utility. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 367–378.

Chen, L., Gupta, A., and Li, J. (2016). Pure exploration of multi-armed bandit under matroid constraints. In *COLT*, pages 647–669.

Chen, S., Lin, T., King, I., Lyu, M. R., and Chen, W. (2014). Combinatorial pure exploration of multi-armed bandits. In *NIPS*, pages 379–387.

Chentanez, N., Barto, A. G., and Singh, S. P. (2005). Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286.

Cohen, J. and Boxma, O. (1985). A survey of the evolution of queueing theory. *Statistica neerlandica*, 39(2):143–158.

Cover, T. M. and Thomas, J. A. (2012). *Elements of information theory*. John Wiley & Sons.

Cox, D. and Smith, W. (1961). *Queues*. Willey.

Csiszár, I. (1984). Sanov property, generalized I-projection and a conditional limit theorem. *The Annals of Probability*, 12(3):768–793.

Csiszár, I. and Tusnády, G. (1984). Information geometry and alternating minimization procedures. *Statistics and decisions*, Supplement issue No. 1:205–237.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017). Distributional reinforcement learning with quantile regression. *arXiv preprint arXiv:1710.10044*.

Dann, C. and Brunskill, E. (2015). Sample complexity of episodic fixed-horizon reinforcement learning. In *NIPS*, pages 2818–2826.

Darmois, G. (1935). Sur les lois de probabilites a estimation exhaustive. *C. R. Acad. Sci. Paris*, 200:1265–1266.

Das, S. and Kamenica, E. (2005). Two-sided bandits and the dating market. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 5, pages 19–24.

DeGroot, M. H. (2005). *Optimal statistical decisions*, volume 82 of *Wiley Classics Library*. John Wiley & Sons.

Dertouzos, M. L. and Mok, A. K. (1989). Multiprocessor online scheduling of hard-real-time tasks. *IEEE Transactions on software engineering*, 15(12):1497–1506.

Difallah, D. E., Pavlo, A., Curino, C., and Cudre-Mauroux, P. (2013). Oltp-bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment*, 7(4):277–288.

Durrett, R. (2010). *Probability: theory and examples*. Cambridge University Press.

Dvijotham, K. and Todorov, E. (2012). Linearly solvable optimal control. *Reinforcement learning and approximate dynamic programming for feedback control*, 17:119–141.

Eguchi, S. (1992). Geometry of minimum contrast. *Hiroshima Mathematical Journal*, 22(3):631–647.

Faheem, M. and Senellart, P. (2015). Adaptive web crawling through structure-based link classification. In *Proc. ICADL*, pages 39–51, Seoul, South Korea.

Farahnakian, F., Liljeberg, P., and Plosila, J. (2014). Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 500–507.

Filippi, S., Cappé, O., and Garivier, A. (2010). Optimism in reinforcement learning and kullback-leibler divergence. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 115–122. IEEE.

Fox, R., Pakman, A., and Tishby, N. (2015). Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*.

Gabillon, V., Ghavamzadeh, M., and Lazaric, A. (2012). Best arm identification: A unified approach to fixed budget and fixed confidence. In *NIPS*, pages 3212–3220.

Galichet, N., Sebag, M., and Teytaud, O. (2013). Exploration vs exploitation vs safety: Risk-aware multi-armed bandits. In *Asian Conference on Machine Learning*, pages 245–260.

Garivier, A. and Cappé, O. (2011). The KL-UCB algorithm for bounded stochastic bandits and beyond. In *COLT*, pages 359–376.

Garivier, A., Lattimore, T., and Kaufmann, É. (2016a). On explore-then-commit strategies. In *Advances in Neural Information Processing Systems 29*, pages 784–792. Curran Associates, Inc.

Garivier, A., Ménard, P., and Stoltz, G. (2016b). Explore first, exploit next: The true shape of regret in bandit problems. *arXiv preprint arXiv:1602.07182*.

Gelfand, I., Fomin, S., and Silverman, R. (2000). *Calculus of Variations*. Dover Books on Mathematics. Dover Publications.

Gittins, J. C. (1979). Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):148–177.

Gopalan, A. and Mannor, S. (2015). Thompson sampling for learning parameterized markov decision processes. In *Conference on Learning Theory*, pages 861–898.

Gordon, G. J. (1999). Approximate solutions to markov decision processes. Technical report, Carnegie-Mellon Univeristry, Pittsburgh, PA.

Gouriten, G., Maniu, S., and Senellart, P. (2014). Scalable, generic, and adaptive systems for focused crawling. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media (HT'14)*, pages 35–45.

Grondman, I., Busoniu, L., Lopes, G., and Babuska, R. (2012a). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(6):1291–1307.

Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. (2012b). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307.

Hager, W. (1989). Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239.

Hammer, M. and Niamir, B. (1979). A heuristic approach to attribute partitioning. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data (SIGMOD'79)*, pages 93–101.

Han, Z., Tan, H., Chen, G., Wang, R., Chen, Y., and Lau, F. C. M. (2016). Dynamic virtual machine management via approximate markov decision process. In *35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9.

Hazan, E. et al. (2016). Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325.

Honda, J. and Takemura, A. (2011). An asymptotically optimal policy for finite support models in the multiarmed bandit problem. *Machine Learning*, 85(3):361–391.

Howard, R. A. and Matheson, J. E. (1972). Risk-sensitive markov decision processes. *Management science*, 18(7):356–369.

Huo, X. and Fu, F. (2017). Risk-aware multi-armed bandit problem with application to portfolio selection. *Royal Society open science*, 4(11):171377.

Huppler, K., Lange, K.-D., and Beckett, J. (2012). Spec: Enabling efficiency measurement. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 257–258.

Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. (2011). Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710.

Jacko, P. (2010). Restless bandits approach to the job scheduling problem and its extensions. In Piunovskiy, A. B., editor, *Modern Trends in Stochastic Controlled Processes: Theory and Applications*. Luniver Press.

Jaynes, E. T. (1968). Prior probabilities. *IEEE Transactions on Systems Science and Cybernetics*, 4:227–241.

Jimenez, I., LeFevre, J., Polyzotis, N., Sanchez, H., and Schnaitter, K. (2011). Benchmarking online index-tuning algorithms. *IEEE Data Engineering Bulletin*, 34:28–35.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285.

Kappen, H. J. (2005). Linear theory for control of nonlinear stochastic systems. *Physical review letters*, 95(20):200201.

Kappen, H. J., Gómez, V., and Opper, M. (2012). Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182.

Kaufmann, É., Cappé, O., and Garivier, A. (2012a). On Bayesian upper confidence bounds for bandit problems. In *AISTATS*, pages 592–600.

Kaufmann, É. and Kalyanakrishnan, S. (2013). Information complexity in bandit subset selection. In *COLT*, pages 228–251.

Kaufmann, É., Korda, N., and Munos, R. (2012b). Thompson sampling: An asymptotically optimal finite-time analysis. In *ALT*, pages 199–213. Springer.

Kawale, J., Bui, H. H., Kveton, B., Tran-Thanh, L., and Chawla, S. (2015). Efficient Thompson sampling for online matrix-factorization recommendation. In *NIPS*, pages 1297–1305.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232.

Kimura, H. and Kobayashi, S. (1998). An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. In *ICML*, pages 278–286.

Konidaris, G. and Barto, A. (2006). An adaptive robot motivational system. In *International Conference on Simulation of Adaptive Behavior*, pages 346–356. Springer.

Koopman, B. O. (1936). On distributions admitting a sufficient statistic. *Transactions of the American Mathematical society*, 39(3):399–409.

Krishnasamy, S., Sen, R., Johari, R., and Shakkottai, S. (2016). Regret of queueing bandits. In *Advances in Neural Information Processing Systems*, pages 1669–1677.

Kullback, S. (1997). *Information theory and statistics*. Courier Corporation.

Lago, D., Madeira, E., and Medhi, D. (2017). Energy-aware virtual machine scheduling on heterogeneous bandwidths data centers. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1.

Lagoudakis, M. G. and Parr, R. (2003a). Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149.

Lagoudakis, M. G. and Parr, R. (2003b). Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149.

Lai, T. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6(1):4–22.

Lai, T. L. (1988). Asymptotic solutions of bandit problems. In Fleming, W. and Lions, P.-L., editors, *Stochastic differential systems, stochastic control theory and applications*, pages 275–292. Springer.

Lai, T. L. and Wei, C. Z. (1982). Least squares estimates in stochastic regression models with applications to identification and control of dynamic systems. *The Annals of Statistics*, pages 154–166.

Lang, S. (2006). *Introduction to differentiable manifolds*. Springer Science & Business Media.

Langford, J., Strehl, A., and Wortman, J. (2008). Exploration scavenging. In *Proceedings of the 25th international conference on Machine learning*, pages 528–535. ACM.

Lattimore, T. (2015). Optimally confident ucb: Improved regret for finite-armed bandits. *arXiv preprint arXiv:1507.07880*.

Lattimore, T. and Hutter, M. (2014). Near-optimal pac bounds for discounted mdps. *Theoretical Computer Science*, 558:125–143.

Lauritzen, S. L. (1987). Statistical manifolds. *Differential Geometry in Statistical Inference*, 10:163–216.

LeFevre, F., Sankaranarayanan, J., Hacigumus, H., Tatemura, J., Polyzotis, N., and Carey, M. J. (2014). Exploiting opportunistic physical design in large-scale data analytics. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD'14)*.

Leiserson, C. E. (1985). Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM.

Li, L. and Gruenwald, L. (2013). Self-managing online partitioner for databases (smopd): A vertical database partitioning system with a fully automatic online approach. In *Proceedings of the 17th International Database Engineering and Applications Symposium (IDEAS'13)*, pages 168–173.

Li, P., Guo, S., Miyazaki, T., Liao, X., Jin, H., Zomaya, A. Y., and Wang, K. (2017). Traffic-aware geo-distributed big data analytics with predictable job completion time. *IEEE Transactions on Parallel and Distributed Systems*, 28(6):1785–1796.

Lightstone, S. and Bhattacharjee, B. (2004). Automated design of multidimensional clustering tables for relational databases. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, pages 1170–1181.

Littman, M. L. (1996). *Algorithms for sequential decision making*. PhD thesis, Brown University Providence, RI.

Lohman, G. M. (2014). Is query optimization a "solved" problem? http://wp.sigmod.org/?p=1075.

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*, pages 206–214.

Luhring, M., Sattler, K.-U., Schmidt, K., and Schallehn, E. (2007). Autonomous management of soft indexes. In *Proceedings of the 2nd International Workshop on Self-Managing Data Bases (SMDB'07)*, pages 450–458.

Macready, W. G. and Wolpert, D. H. (1998). Bandit problems and the exploration/exploitation tradeoff. *IEEE Transactions on evolutionary computation*, 2(1):2–22.

Maguluri, S. T., Srikant, R., and Ying, L. (2012). Stochastic models of load balancing and scheduling in cloud computing clusters. In *INFOCOM, 2012 Proceedings IEEE*, pages 702–710.

Maillard, O.-A. (2013). Robust risk-averse stochastic multi-armed bandits. In *International Conference on Algorithmic Learning Theory*, pages 218–233. Springer.

Malek, A., Abbasi-Yadkori, Y., and Bartlett, P. (2014). Linear programming for large-scale markov decision problems. In *International Conference on Machine Learning*, pages 496–504.

Malik, T., Wang, X., Dash, D., Chaudhary, A., Ailamaki, A., and Burns, R. (2009). Adaptive physical design for curated archives. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management (SSDBM'09)*, pages 148–166.

Mann, Z. Á. (2015). Allocation of virtual machines in cloud data centers: A survey of problem models and optimization algorithms. *ACM Computing Surveys (CSUR)*, 48(1):11.

Mannor, S. and Tsitsiklis, J. N. (2004). The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5(Jun):623–648.

Masoumzadeh, S. S. and Hlavacs, H. (2013). Integrating vm selection criteria in distributed dynamic vm consolidation using fuzzy q-learning. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 332–338.

Matumoto, T. (1993). Any statistical manifold has a contrast function—on the c^3-functions taking the minimum at the diagonal of the product manifold. *Hiroshima Mathematical Journal*, 23(2):327–332.

Ménard, P. and Garivier, A. (2017). A minimax and asymptotically optimal algorithm for stochastic bandits. In *International Conference on Algorithmic Learning Theory*, pages 223–237.

Meuleau, N. and Bourgine, P. (1999). Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35(2):117–154.

Minas, L. and Ellison, B. (2009). *Energy efficiency for information technology: How to reduce power consumption in servers and data centers.* Intel Press.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Montgomery, W. H. and Levine, S. (2016). Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016.

Mosteller, F. et al. (1956). Stochastic learning models. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 5: Contributions to Econometrics, Industrial Research, and Psychometry.* The Regents of the University of California.

Motte, R., Burns, R. S., and Calvert, S. (1988). An overview of current methods used in weather routeing. *Journal of Navigation*, 41:101–114.

Nathuji, R. and Schwan, K. (2007). Virtualpower: coordinated power management in virtualized enterprise systems. In *Proce. SOSP*, pages 265–278.

Nehme, R. and Bruno, N. (2011). Automated partitioning design in parallel database systems. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD'11)*, pages 1137–1148.

Nelson, M., Lim, B.-H., and Hutchins, G. (2005). Fast transparent migration for virtual machines. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, pages 25–25. USENIX Association.

Neu, G. and Gómez, V. (2017). Fast rates for online learning in linearly solvable markov decision processes. *arXiv preprint arXiv:1702.06341.*

Neu, G., Jonsson, A., and Gómez, V. (2017). A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798.*

Nguyen, M.-Q. and Bourgine, P. (2014). *Multi-armed bandit problem and its applications in intelligent tutoring systems.* Master's Thesis, École Polytechnique.

Nielsen, F. and Bhatia, R. (2013). *Matrix information geometry.* Springer.

Niño-Mora, J. (2006). Marginal productivity index policies for scheduling a multiclass delay-/loss-sensitive queue. *Queueing Systems*, 54(4):281–312.

Niño-Mora, J. (2007). Dynamic priority allocation via restless bandit marginal productivity indices. *Top*, 15(2):161–198.

Nino-Mora, J. (2011). Computing a classic index for finite-horizon bandits. *INFORMS Journal on Computing*, 23(2):254–267.

North, D. W. (1968). A tutorial introduction to decision theory. *Systems Science and Cybernetics, IEEE Transactions on*, 4(3):200–210.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. (2016). Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*.

Oneto, L., Anguita, D., and Ridella, S. (2016). Pac-bayesian analysis of distribution dependent priors: Tighter risk bounds and stability analysis. *Pattern Recognition Letters*, 80:200–207.

Osband, I., Russo, D., and Van Roy, B. (2013). (More) efficient reinforcement learning via posterior sampling. In *NIPS*, pages 3003–3011.

Osband, I. and Van Roy, B. (2016). Why is posterior sampling better than optimism for reinforcement learning? *arXiv preprint arXiv:1607.00215*.

Osband, I. and Van Roy, B. (2017). On optimistic versus randomized exploration in reinforcement learning. *arXiv preprint arXiv:1706.04241*.

Oudeyer, P.-Y. and Kaplan, F. (2009). What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 1:1–6.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(6).

Papadomanolakis, S., Dash, D., and Ailamaki, A. (2007a). Efficient use of the query optimizer for automated physical design. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, pages 1093–1104.

Papadomanolakis, S., Dash, D., and Ailamaki, A. (2007b). Efficient use of the query optimizer for automated physical design. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, pages 1093–1104.

Park, K. and Pai, V. S. (2006). Comon: a mostly-scalable monitoring system for planetlab. *ACM SIGOPS Operating Systems Review*, 40(1):65–74.

Pavlo, A., Curino, C., and Zdonik, S. (2012). Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*, pages 61–72.

Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, volume 703. John Wiley & Sons.

Press, W. H. (2009). Bandit solutions provide unified ethical models for randomized clinical trials and comparative effectiveness research. *Proceedings of the National Academy of Sciences*, pages pnas–0912378106.

Puterman, M. L. (2009). *Markov decision processes: discrete stochastic dynamic programming*, volume 414. John Wiley & Sons.

Putta, S. R. and Tulabandhula, T. (2017a). Efficient reinforcement learning via initial pure exploration. *CoRR*, abs/1706.02237.

Putta, S. R. and Tulabandhula, T. (2017b). Pure exploration in episodic fixed-horizon Markov decision processes. In *AAMAS*, pages 1703–1704.

Raab, F. (1993). TPC-C - the standard benchmark for online transaction processing (OLTP). In Gray, J., editor, *The Benchmark Handbook*. Morgan Kaufmann.

Ramakrishnan, R., Gehrke, J., and Gehrke, J. (2003). *Database management systems*, volume 3. McGraw-Hill New York.

Rao, J., Bu, X., Xu, C.-Z., Wang, L., and Yin, G. (2009). Vconf: A reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 137–146, New York, NY, USA. ACM.

Rao, J., Zhang, C., Megiddo, N., and Lohman, G. (2002). Automating physical database design in a parallel database. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD'02)*, pages 558–569.

Rasin, A. and Zdonik, S. (2013). An automatic physical design tool for clustered column-stores. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*, pages 203–214.

Reiss, C., Wilkes, J., and Hellerstein, J. L. (2011). Google cluster-usage traces: format + schema. *Google Inc., White Paper*, pages 1–14.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535.

Rockafellar, R. T. (2015). *Convex analysis*. Princeton university press.

Rösch, P., Dannecker, L., Färber, F., and Hackenbroich, G. (2012). A storage advisor for hybrid-store databases. *Proceedings of the VLDB Endowment*, 5(12):1748–1758.

Rummery, G. A. and Niranjan, M. (1994). On-line q-learning using connectionist systems. *Technical Report*.

Russo, D. and Van Roy, B. (2016). An information-theoretic analysis of thompson sampling. *The Journal of Machine Learning Research*, 17(1):2442–2471.

Ruszczyński, A. (2010). Risk-averse dynamic programming for markov decision processes. *Mathematical programming*, 125(2):235–261.

Sani, A., Lazaric, A., and Munos, R. (2012). Risk-aversion in multi-armed bandits. In *Advances in Neural Information Processing Systems*, pages 3275–3283.

Sattler, K.-U., Geist, I., and Schallehn, E. (2003). Quiet: Continuous query-driven index tuning. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, pages 1129–1132.

Schmidhuber, J. (1991a). Adaptive curiosity and adaptive confidence. Technical report, Technical Report FKI-149-91, Institut f ur Informatik, Technische Universität Munchen.

Schmidhuber, J. (1991b). Curious model-building control systems. In *IEEE International Joint Conference on Neural Networks*, volume 2, pages 1458–1463.

Schnaitter, K., Abiteboul, S., Milo, T., and Polyzotis, N. (2006). Colt: Continuous on-line tuning. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*, pages 793–795.

Schnaitter, K., Abiteboul, S., Milo, T., and Polyzotis, N. (2007). On-line index selection for shifting workloads. In *Proceedings of the 2nd International Workshop on Self-Managing Data Bases (SMDB'07)*, pages 459–468.

Schnaitter, K. and Polyzotis, N. (2012). Semi-automatic index tuning: Keeping dbas in the loop. *Proceedings of the VLDB Endowment*, 5(5):478–489.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Scott, S. L. (2010). A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658.

Seldin, Y., Cesa-Bianchi, N., Auer, P., Laviolette, F., and Shawe-Taylor, J. (2012). Pac-bayes-bernstein inequality for martingales and its application to multiarmed bandits. In *Proceedings of the Workshop on On-line Trading of Exploration and Exploitation 2*, pages 98–111.

Sha, L., Abdelzaher, T., Årzén, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., and Mok, A. K. (2004). Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2-3):101–155.

Sherman, J. and Morrison, W. J. (1949). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Annals of Mathematical Statistics*, 20:317.

Shortle, J. F., Thompson, J. M., Gross, D., and Harris, C. M. (2018). *Fundamentals of queueing theory*, volume 399. John Wiley & Sons.

Silver, D. (2015). Lecture notes on reinforcement learning. COMPM050 Reinforcement Learning Course in University College London.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484.

Şimşek, Ö. and Barto, A. G. (2006). An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd international conference on Machine learning*, pages 833–840. ACM.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308.

Song, W., Xiao, Z., Chen, Q., and Luo, H. (2014). Adaptive resource provisioning for the cloud using online bin packing. *Computers, IEEE Transactions on*, 63(11):2647–2660.

SPECpower Committee (2014). SPEC power and performance benchmark methodology.

Still, S. and Precup, D. (2012). An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131(3):139–148.

Stillger, M., Lohman, G. M., Markl, V., and Kandil, M. (2001). Leo-db2's learning optimizer. In *VLDB*, volume 1, pages 19–28.

Strehl, A. L., Li, L., and Littman, M. L. (2009). Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444.

Strens, M. (2000). A bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, pages 943–950.

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 28. MIT press, Cambridge.

Szepesvári, C. (2010). *Algorithms for reinforcement learning*. Morgans & Claypool.

Szita, I. and Lőrincz, A. (2008). The many faces of optimism: a unifying approach. In *Proceedings of the 25th international conference on Machine learning*, pages 1048–1055. ACM.

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

Thrun, S. B. (1992). Efficient exploration in reinforcement learning. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA.

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

Todorov, E. (2008). General duality between optimal control and estimation. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4286–4292. IEEE.

Tseng, H. W., Yang, T. T., Yang, K. C., and Chen, P. S. (2017). An energy efficient vm management scheme with power-law characteristic in video streaming data centers. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1.

Van Mieghem, J. A. (1995). Dynamic scheduling with convex delay costs: The generalized c| mu rule. *The Annals of Applied Probability*, pages 809–833.

Wang, M., Meng, X., and Zhang, L. (2011). Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75.

Warmuth, M. K. and Jagota, A. K. (1997). Continuous and discrete-time nonlinear gradient descent: Relative loss bounds and convergence. In *Electronic proceedings of the 5th International Symposium on Artificial Intelligence and Mathematics*. Citeseer.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

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

White, D. J. (1993). *Markov decision processes*. John Wiley & Sons New York, NY.

Wieder, P., Butler, J. M., Theilmann, W., and Yahyapour, R. (2011). *Service level agreements for cloud computing*. Springer Science & Business Media.

Wiering, M. and Schmidhuber, J. (1998). Fast online q (λ). *Machine Learning*, 33(1):105–115.

Williams, R. J. (1992). Simple gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Williams, R. J. (1993). Tight performance bounds on greedy policies based on imperfect value functions. Technical report, College of Computer Science, Northeastern University.

Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. (2007). Black-box and gray-box strategies for virtual machine migration. In *Proc. NSDI*, pages 11–13.

Xu, C.-Z., Rao, J., and Bu, X. (2012). Url: A unified reinforcement learning approach for autonomic cloud management. *Journal of Parallel and Distributed Computing*, 72(2):95 – 105.

Young, P. (2011). Recursive least squares estimation. In *Recursive Estimation and Time-Series Analysis*, pages 29–46. Springer Berlin Heidelberg.

Yu, R., Xue, G., Zhang, X., and Li, D. (2017). Survivable and bandwidth-guaranteed embedding of virtual clusters in cloud data centers. In *Computer Communications, IEEE INFOCOM 2017-The 36th Annual IEEE International Conference on*. IEEE.

Ziebart, B. D. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438.

Zilio, D. C., Rao, J., Lightstone, S., Lohman, G., Storm, A., Garcia-Arellano, C., and Fadden, S. (2004a). Db2 design advisor: Integrated automatic physical database design. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, pages 1087–1097.

Zilio, D. C., Zuzarte, C., Lightstone, S., Ma, W., Lohman, G. M., Cochrane, R., Pirahesh, H., Colby, L. S., Gryz, J., Alton, E., Liang, D., and Valentin, G. (2004b). Recommending materialized views and indexes with ibm db2 design advisor. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC'04)*, pages 180–188.