THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Decision Making In An Uncertain World

Erli Wang

B.Sc.

A thesis submitted for the degree of Doctor of Philosophy at

The University of Queensland in 2019

School of Mathematics and Physics

Abstract

Decision making is widespread in many domains, including aircraft collision avoidance, autonomous driving and space exploration. Except for highly engineered environments, uncertainties are unavoidable and can not be ignored. It challenges the decision maker to figure out rational actions and solving such problems in large and complex scenarios are even harder. This thesis provides new perspectives to alleviate the difficulty of solving large-scale decision making problems under uncertainty.

The first decision making model considered is the stochastic Multi-Armed Bandit (MAB). It is an important model for studying the exploration-exploitation tradeoff. In this problem, a gambler has to repeatedly choose between a number of slot machines (arms) to maximize the total payout. The outcome of playing arms is stochastic and the total number of plays is fixed. To solve large-scale MABs, we introduce a method, called Cross-Entropy-based Multi-Armed Bandit (CEMAB), adopting the Cross-Entropy method as a noisy optimizer. Various MAB testing cases are used to compare the performance of CEMAB and different competitors. The results show that CEMAB is promising when the size of the arm space is large.

Making principled decisions in the presence of uncertainty is often facilitated by using the powerful framework of Partially Observable Markov Decision Processes (POMDPs). However, precisely because of its generality, solving this problem exactly is computationally intractable. Recently, approximate POMDP solvers have shown to be able to compute good decision strategies, but handling POMDPs with large action spaces remains difficult. We propose a sampling method called Quantile-Based Action Selector (QBASE) that can scale up to very large problems. We employ several scalable robotics scenarios with up to 100,000 actions to evaluate the performance of the proposed technique. Based on numerical experiments, QBASE performs significantly better than POMCP, a state-of-the-art solver, when the size of action space is large ($>$100).

Finding the best performance of POMDP solvers involves parameter optimization. It is usually done by searching for the good performing settings off-line, which unavoidably adds an extra burden to users. We extend QBASE to identify parameters automatically within the

planning time, called Adaptive Parameter Sampling (APS-QBASE). Two tasks with up to one million possible actions are used in numerical experiments. We find that APS-QBASE can achieve a higher policy quality than several on-line POMDP approaches with different action selectors, including QBASE and an enhancement of POMCP for handing large action spaces. A sensitivity study of APS-QBASE suggests that the proposed method can significantly reduce the difficulty of setting good parameters.

Declaration by author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my higher degree by research candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the policy and procedures of The University of Queensland, the thesis be made available for research and study in accordance with the Copyright Act 1968 unless a period of embargo has been approved by the Dean of the Graduate School.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate I have obtained copyright permission from the copyright holder to reproduce material in this thesis and have sought permission from co-authors for any jointly authored works included in the thesis.

Publications included in this thesis

1. **Erli Wang**, Hanna Kurniawati, and Dirk P. Kroese, CEMAB: A Cross-Entropy-based Method for Large-Scale Multi-Armed Bandits, *Australasian Conference on Artificial Life and Computational Intelligence*, 353-365, 2017

2. **Erli Wang**, Hanna Kurniawati, and Dirk P. Kroese, An On-Line Planner for POMDPs with Large Discrete Action Space: A Quantile-Based Approach, *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 273-277, 2018. (An expanded paper, "QBASE: an On-line Planner for Large Action POMDPs", is in progress for submission to the Artificial Intelligence Journal.)

Submitted manuscripts included in this thesis

No manuscripts submitted for publication.

Other publications during candidature

No other publications.

Contributions by others to the thesis

The thesis is based on my publication under supervision of Prof. Dirk P. Kroese and Dr. Hanna Kurniawati.

Statement of parts of the thesis submitted to qualify for the award of another degree

No works submitted towards another degree have been included in this thesis.

Research involving human or animal subjects

No animal or human subjects were involved in this research.

Acknowledgments

When I recall the past few years, my life as a Ph.D. student has, overall, been smooth and enjoyable. This Ph.D. work would not have been achieved without the insightful and endless supports from many people. I gratefully acknowledge them in this thesis.

My great thanks firstly goes to Prof. Dirk P. Kroese, who offered me an opportunity to study with him and provided generous help during my candidature. Dirk introduced me to Dr. Hanna Kurniawati when I was exploring new research interests. It was Hanna who introduced me to the interesting field of planning. Since then, both Dirk and Hanna have become my principal supervisors. I appreciate such a challenging project and am proud of having been involved in it. During my Ph.D. candidature, I have come to admire Dirk and Hanna for their professionalism in academia. They deliver their passion, knowledge and diligence in every meeting and email.

Next, I would like to acknowledge Dr. Radislav (Slava) Vaisman for his insight and suggestions on Monte Carlo methods when I started my Ph.D. work. My special thanks also go to my colleague Dr. Qibin Duan; our discussions on the rare-event simulation and applications of stochastic optimization methods have extended my knowledge to new domains. I thank Dr. Nan Ye and Dr. Haibo Wang for their discussion on POMDP and comments on this thesis.

Financial acknowledgement goes to the University of Queensland (UQ) and the Australian Research Council Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS) for their supports of tuition fees and living allowance. I also acknowledge the ACEMS, UQ Candidate Development Award, UQ SMP student conference funding and ICAPS 2018 student travel grants, which have supported me to attend academic activities in Australia and overseas.

Personally, I express my deepest gratitude to my family for their constant trusts. They have brought so much joy and balance to my life. Thank you to my wife Dr. Bing Cheng for her love. Her caring and understanding have made me a better person.

Erli Wang
Brisbane, 2019

Financial support

This research was supported by UQ International Scholarship (UQI) scheme, the Australian Research Council Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS).

Keywords

Multi-armed bandit, partially observable Markov decision processes, Cross-Entropy method, sequential decision making, planning under uncertainty

Australian and New Zealand Standard Research Classifications (ANZSRC)

ANZSRC code: 080110 Simulation and Modelling, 50%
ANZSRC code: 010401 Applied Statistics, 30%
ANZSRC code: 010203 Calculus of Variations, Systems Theory and Control Theory, 20%

Fields of Research (FoR) Classification

FoR code: 0801 Artificial Intelligence and Image Processing, 50%
FoR code: 0104 Statistics, 30%
FoR code: 0102 Applied Mathematics, 20%

Contents

List of Figures xii

List of Tables xiv

List of Abbreviations xv

List of Symbols xvi

1 Introduction 1

1.1 Motivation . . . 1

1.2 Background . . . 3

1.3 Contributions . . . 6

1.4 Outline . . . 7

2 Decision Making Under Uncertainty 9

2.1 Multi-Armed Bandit . . . 9

2.2 Markov Decision Process . . . 11

2.2.1 Fundamentals . . . 11

2.2.2 Value Iteration . . . 15

2.2.3 Monte Carlo Tree Search . . . 18

2.3 Partially Observable Markov Decision Process . . . 22

2.3.1 Fundamentals . . . 22

2.3.2 Solving the Tiger Problem . . . 25

2.3.3 Partially Observable Monte Carlo Planning . . . 29

2.4 Cross-Entropy Method . . . 31

3 CEMAB: A Sample-based Method for Large-scale MABs 36
3.1 Background . . . 37
3.2 Related Algorithms . . . 38
3.3 Cross-Entropy-based Multi-Armed Bandit (CEMAB) . . . 40
3.3.1 The Method . . . 40
3.3.2 Example: Ten-Arm MAB illustration . . . 42
3.3.3 Time Complexity and Convergence Properties . . . 44
3.4 Experimental Results . . . 45
3.4.1 Small-Scale MABs . . . 46
3.4.2 Large-scale MABs . . . 50
3.5 Discussion . . . 53

4 QBASE: An On-line Planner for Large-scale POMDPs 56
4.1 Background . . . 57
4.2 Related Algorithms . . . 58
4.3 Quantile-Based Action Selector (QBASE) . . . 59
4.3.1 Overview . . . 59
4.3.2 Expanding the Belief Tree . . . 60
4.3.3 Sampling Actions . . . 63
4.4 Experimental Results . . . 66
4.4.1 RockSample(n, k) . . . 68
4.4.2 Navigation(d, n) . . . 70
4.4.3 Hunting(n, u, v) . . . 71
4.5 Discussion . . . 77

5 APS-QBASE: An Adaptive Parameter Sampling 79
5.1 Background . . . 80
5.2 Related Algorithms . . . 81
5.2.1 POMCP . . . 81
5.2.2 QBASE . . . 81
5.2.3 POMCP-PW . . . 82

5.3 Adaptive Parameter Sampling (APS) 82
5.3.1 The Method 82
5.3.2 Details 83
5.4 Experimental Results 86
5.4.1 Hunting-Normal(n, u, v) 86
5.4.2 InventoryControl(K) 86
5.4.3 Performance Comparison 88
5.4.4 Sensitivity Study 91
5.5 Discussion 92

6 Summary **94**

A Details of Scenarios **97**
A.1 RockSample(n, k) 97
A.2 Navigation(d, n) 98
A.3 Hunting(n, u, v) 99

References **101**

List of Figures

1.1 Illustration of the Navigation problem in a grid world 2

2.1 Illustration of the Navigation MAB problem 11
2.2 Illustration of the Navigation MDP problem 14
2.3 Off-line and on-line algorithms comparison 15
2.4 The optimal policy and associated value with $\gamma = 0.95$ and $\varepsilon = 0.1$ 18
2.5 The optimal policy and associated value with $\gamma = 0.95$ and $\varepsilon = 0.001$ 18
2.6 The optimal policy and associated value with $\gamma = 0.1$ and $\varepsilon = 0.001$ 18
2.7 MDP tree with two actions and two states 20
2.8 One iteration of UCT 21
2.9 Illustration of the Navigation POMDP problem 24
2.10 AND-OR tree with two actions and two observations 26
2.11 Tiger AND-OR tree for $b_0 = (b_0(s_l), b_0(s_r)) = (0.5, 0.5)$ and $N = 2$. 28
2.12 Tiger AND-OR tree for $b_1 = (b_1(s_l), b_1(s_r)) = (0.85, 0.15)$ and $N = 2$ 29
2.13 One iteration of POMCP 31

3.1 An example of ten-arm bandit with Gaussian reward function 43
3.2 The average total regret on six Bernoulli reward problem sets 48
3.3 The average total regret on four Gaussian reward problem sets 49

4.1 Correspondence between a history and a path in $\mathcal{T}$ 61
4.2 A batch simulation of QBASE 65
4.3 Illustration of ***RockSample***$(20, 50)$ 68
4.4 Performance with different planning times per step in RockSample 69
4.5 Illustration of ***Navigation***$(2, 30)$ 70

4.6 Performance with different planning times per step in Navigation 71
4.7 Illustration of ***Hunting***$(11, 3, 3)$ 72
4.8 Performance with different planning times per step 73
4.9 Collaborations strategy in ***Hunting-normal***$(11, 3, 3)$ generated by QBASE . . 76

5.1 Illustration of ***InventoryControl***(1) 87
5.2 Performance with different planning times per step in Hunting-normal and Inventory Control problem 90
5.3 Sensitivity Study 91

List of Tables

1.1 Approximate POMDP solvers. 4

3.1 Evolution of $\boldsymbol{p}$ by CEMAB method 44
3.2 Bx refers to Bernoulli distributions and Gx to truncated Gaussian distributions. 46
3.3 Parameter Range 47
3.4 Large-scale MAB Settings 50
3.5 The average total reward for large MABs with Bernoulli reward functions. . . 52
3.6 The average total reward for large MABs with Gaussian reward functions. . . . 53

4.1 Simulation Results 67
4.2 Success rate of completing ***Hunting-normal***$(11, 5, 5)$ 74
4.3 Summary of results in ***Hunting-normal*** with different number of agents and targets with 10 sec planning per step. 75

5.1 Settings of ***InventoryControl***(6) 89
5.2 Performance variation in the each algorithm's parameter domain in ***Hunting-normal***$(11, 5, 5)$ 92
5.3 Performance variation in the each algorithm's parameter domain in ***InventoryControl***(6) 92

List of Abbreviations

ABT	Adaptive Belief Tree
APS	Adaptive Parameter Sampling
CE	Cross-Entropy
CEMAB	Cross-Entropy-based Multi-Armed Bandit
MAB	Multi-Armed Bandit
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
POMCP	Partially Observable Monte Carlo Planning
PW	Progressive Widening
QBASE	Quantile-Based Action Selector
UCB	Upper Confidence Bound
VI	Value Iteration

List of Symbols

$\sim$	is distributed according to
Ber	Bernoulli distribution
N	Gaussian distribution
U	uniform distribution
Beta	beta distribution
$\mathbf{x}, \mathbf{y}$	vectors
$\mathbf{X}, \mathbf{Y}$	random variables
$\mathcal{X}, \mathcal{Y}$	sets
$\mathcal{S}$	State space
$\mathcal{K}$	Arm space
R	Reward function
$\mathcal{A}$	Action space
$\mathcal{O}$	Observation space
T or $p(s' \mid s, a)$	State-transition function
Z or $p(o \mid s, a)$	Observation function
b	Belief state
V	Value function
γ	Discount factor, or threshold

N	Horizon, or sample size
ρ	Elite ratio
N_e	Elite sample size
$\alpha, \boldsymbol{\alpha}$	Learning rate
N_s	Size of subset of action space
M	Evaluation budget
β	A parameter in the learning rate

Chapter 1

Introduction

1.1 Motivation

Decision making is important and widespread in engineering, computer science, operation research, economics and so on. In many problems, an agent has to make decisions *sequentially* to complete a given task. In other words, the decision made now has to consider not only the results of immediate effect but also the future impacts. Moreover, the agent inevitably has to deal with various types of *uncertainties* in the real world. For example, the effects of actions are not precisely known, sensors and sensing are erroneous, and information about the operating environment is imperfect. How should the agent decide what action to take now, so that it will reach its goal optimally and reliably, despite these types of uncertainties?

As an illustration, consider a Navigation problem where the agent moves in a grid world (Figure 1.1). The robot starts from the cell marked with *start* and tries to reach the cell marked with money with the minimum cost. If it decides how to move at the start only based on one action, moving `up` and `right` has equal benefits. However, in the long-run, moving `right` from the *start* will force the agent to take a more expensive route to the goal. Therefore, a better approach is to consider *sequences* of actions to reach the goal, rather than a single step action.

Now, consider the sequential decision making problem when the agent is experiencing two types of uncertainties: uncertainty in the effects of performing actions and uncertainty in percep-

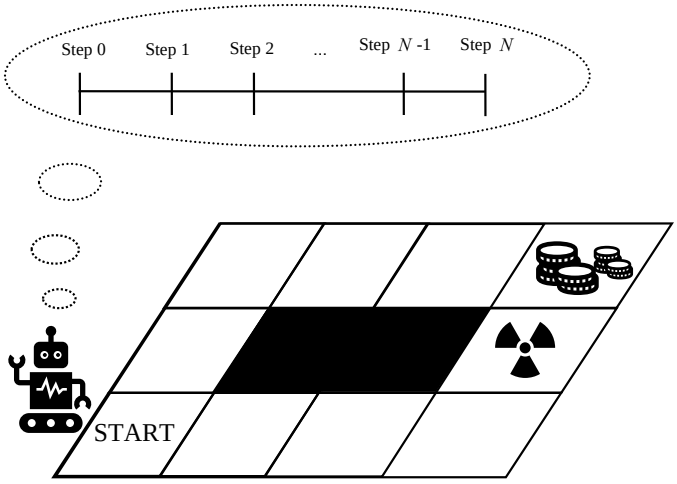


Figure 1.1: Illustration of the Navigation in a grid world. The agent has four possible movements (`up`, `down`, `left`, `right`). Moving to the obstacle cell (black) is prohibited. The agent receives a reward of +1 when it reaches the cell with "money", and incurs a cost of 1 when it enters the cell labelled with "warning". Moving out of any other cells incurs a cost of 0.04.

tion. For instance, the mobile robot may move slower than expected (from its motion equation) due to the unknown friction coefficient of the surface, and it may not know its exact location on the map because the GPS signal is influenced by high-rise buildings. Many practical problems of interest share one or more properties motivated by the Navigation example. Those properties are:

- Sequential. The decision at each step depends on the previous decisions, and thus influences future decisions.
- Uncertain. Three types of uncertainties are considered:
  - Reward. The outcome of reward function can be noisy.
  - Transition. The effect of an action can be stochastic.
  - Perception. The agent cannot determine its exact position.
- Large-Scale. For example, the state space can be very large (say 10^{32} possible states) and/or the action space can be large as well (e.g., a million possible discrete actions).

1.2 Background

The Partially Observable Markov Decision Process (POMDP) (Sondik, 1971) is a mathematically principled model for sequential decision making under uncertainty. A POMDP can be described as follows. At each step, an agent is in some hidden true state and takes an action. It moves to the next state according to some state-transition function. This next state is partially perceived via an observation function. After each step, the agent receives a reward. This process repeats. One possible objective function is to maximize its expected (discounted) total reward. Now, due to the uncertainty in the effects of actions and in perception, the agent never knows its exact state. Instead, it maintains an estimate of the current state in the form of a belief. Even though a POMDP is a nice mathematical framework for such robust sequential decision making problems, it has been considered impractical for decades (Papadimitriou and Tsitsiklis, 1987a; Madani et al., 1999). Generally, four components lead to high computational complexity in solving POMDP problems: long planning horizons, large state spaces, large action spaces, and large observation spaces.

Pioneering works of POMDP solvers are proposed (Aoki, 1965; Astrom, 1965; Smallwood and Sondik, 1973; Sondik, 1978) in the 1960s. Some of the early surveys on POMDP algorithms are reviewed in Monahan (1982); Lovejoy (1991); White (1991). Not long ago, the best solvers are only available to provide solutions for tiny problems with less than 100 states and less than 20 observations (Littman et al., 1995a,b). Fortunately, starting from Pineau et al. (2003), approximate POMDP solvers developed rapidly in the past decade. By trading off optimality for efficiency, approximate POMDP solvers significantly improve the scalability of problems without sacrificing too much of the solution quality within a reasonable time. There are two types of solvers in general: off-line and on-line. The off-line methods build a mapping for each belief state, while the on-line solvers do a planning phase, execute one step, get an observation and replan again. Some of the off-line and on-line methods are reviewed in (Ross et al., 2008; Shani et al., 2013). Table 1.1 provides a brief summary of the scalability of the current approximate POMDP solvers.

Table 1.1: Approximate POMDP solvers.

Type	Algorithms	Scalability
Off-line	2003, PBVI (Pineau et al., 2003)	The first point-based approximate solver; Finds a good solution for a benchmark called Tag (870 discrete states)
	2004, HSVI (Smith and Simmons, 2004)	Significantly speeds up compared with PBVI; Provides a benchmark called Rocksample (10 times larger than Tag)
	2005, HSVI2 (Smith and Simmons, 2005)	Scales up to $> 10^5$ discrete states
	2005, Perseus (Spaan and Vlassis, 2005)	Handles continuous action spaces
	2008, SARSOP (Kurniawati et al., 2008)	Dramatically reduces planning time of finding a good policy within 6 seconds in Tag, where HSVI2 takes 2 hours
	2010, MCVI (Bai et al., 2010)	Handles continuous observation spaces
	2011, MiGS (Kurniawati et al., 2011)	Handles long time horizons up to 100 steps
On-line	2010, POMCP (Silver and Veness, 2010)	Scales up to 10^{56} discrete states and 1024 discrete observations
	2013, ABT (Kurniawati and Yadav, 2013)	Handles dynamic environments
	2013, DESPOT (Somani et al., 2013)	Scales up to 10^6 discrete observations
	2015, GPS-ABT (Seiler et al., 2015)	Significantly outperforms ABT with discretization in action spaces; Handles up to 3-D continuous action spaces
	2018, POMCPOW (Sunberg and Kochenderfer, 2018)	Focuses on handling continuous observation spaces

Since then, many complex decision problems have become solvable in practice, for example, the nurseBot in nursing facilities (Porta et al., 2006), the airplane collision avoidance system (Temizer et al., 2009, 2010), the system of handwashing assistance (Hoey et al., 2010), conservation planning (Chadès et al., 2011), multi-robot cooperation to tracking (Capitan et al., 2013), clinical decision-making (Bennett and Hauser, 2013), autonomous drones (Floreano and Wood, 2015), and the *in-silico* behavior discovery system in ethology (Wang et al., 2015).

Despite these developments, most of the solvers up to now can only solve POMDP problems with a small number of actions (< 100). In order to be applied to more applications, it is crucial for the solvers to handle much larger action spaces. For example, the action size of a real problem in a spoken dialog system is at least 10 billion (Young et al., 2013). Problems with large action spaces are unlikely to be handled even by the best solvers today. Finding the best strategy in POMDPs involves, firstly, the estimation of the expected total reward (called Q-value) for executing some action at the current belief, and, secondly, the optimization of the Q-value over the action space. The difficulty here is that the evaluation of the expectation itself is expensive because it cannot be computed in a closed form. Furthermore, it will be more difficult if we want to find the best action that maximizes the Q-value. For this reason, most successful solvers resort to enumerating all possible actions. When the size of action space is large, such enumeration will significantly reduce the planning horizons that the solver can look ahead within a given planning time and then reduce the quality of the solution.

In fact, the action selection is a fundamental issue in solving POMDP problems. The challenge is how to select the best action sequence even if the consequence of each action may not be exactly known. We investigate its simpler one-step decision problem as a start. The action selection from a given belief is akin to choosing which slot machine to play from a number of choices. This simplified problem can be studied as a Multi-Armed Bandit (MAB) problem (Robbins, 1952). Although MAB is the simplest version of POMDP, it reflects the core of reinforcement learning (Sutton and Barto, 1998). The key is to balance between making decisions that have been giving good rewards in the past (often called exploitation) and choices that have not been tried before (often called exploration).

Inspired by our preliminary work on MAB, the primary goal of the thesis is to alleviate the difficulty of solving POMDPs with large action spaces. It is conducted by adopting a sampling-based stochastic optimization method and carefully constructing the sampling technique in the action space, so that we are able to obtain the optimal actions from only a small number of samples, thus creating a significant improvement in computational efficiency. The way of constructing the sampling strategy is motivated by recent developments in stochastic optimization methods, e.g., the Cross-Entropy method (Rubinstein and Kroese, 2004).

1.3 Contributions

The research work documented in this thesis provides a number of contributions as follows.

Chapter 3 presents a quantile-based sampling approach called Cross-Entropy-based Multi-Armed Bandit (CEMAB) to address MAB problems with a large number of arms. One of the variants is guaranteed to converge to the optimal arm by applying results on CE for noisy optimization. This property holds under certain conditions on the reward function. Different existing strategies and CEMAB are tested on ten small problems and four large problems, including discrete and continuous reward functions. Empirical results show that the proposed method can significantly outperform established methods when the number of arms is large.

Although computing the exact optimal POMDP decision is computationally intractable, approximate solvers can compute good decision strategies for problems with increasingly large state and observation spaces, enabling POMDPs to become practical. Despite these advances, finding good strategies for problems with large action spaces remains difficult. To alleviate this difficulty, Chapter 4 proposes a novel approximate on-line POMDP solver, called Quantile-Based Action Selector (QBASE). Extensive numerical experiments on different robotics tasks with up to 100,000 actions illustrate that QBASE can generate substantially better strategies than a state-of-the-art method.

With the aim of computing the optimal policy, most POMDP solvers, including QBASE, have to identify the best parameters as a prior. Invariably, the performance of parameters is instance-dependent, which is unknown in advance. This is often done off-line by running a

set of preliminary tests for each problem. The test set is a grid search within given parameter ranges. Such search strategy is repetitive, costly and tedious. In Chapter 5, a parameter search mechanism is built on top of QBASE, allowing it to find a good performing parameters automatically. Numerical experiments show that the proposed method can significantly reduce the difficulty to set QBASE's parameters.

1.4 Outline

This thesis is organized as follows.

Chapter 2 provides an overview of background knowledge for three types of mathematical models for decision making in the presence of uncertainties: multi-armed bandit in Section 2.1, Markov decision processes in Section 2.2 and partially observable Markov decision processes in Section 2.3. The framework of the Cross-Entropy method as a stochastic optimization method is also reviewed in Section 2.4.

Chapter 3 presents a sampling-based method to solve large-scale MAB problems. A number of existing algorithms are reviewed in Section 3.2. We introduce the proposed CEMAB and analyze its time complexity and convergence properties in Section 3.3. The comparison results are reported in Section 3.4 with five well-known methods.

Chapter 4 presents a new on-line planner for POMDP with large discrete action spaces. We review some recent developments on POMDPs with large action spaces in Section 4.2. A detailed description of the proposed QBASE is given in Section 4.3. We assess the performance of QBASE by comparing with a leading POMDP solver on a wide range of robotics tasks including *RockSample*, *Navigation*, and *Hunting* in Section 4.4. In particular, we output more results on *Hunting* in Section 4.4.3, such as the success rate and collaboration strategies.

Chapter 5 presents an extension of QBASE with a mechanism of identifying running parameters automatically. A simple modification on POMCP to handle large action space is introduced in Section 5.2. The description of APS-QBASE is given with details in Section 5.3. The proposed method is tested on two tasks and compared with state-of-the-art algorithms in

Section 5.4. Except for a robotics task, we introduce a partially observable version of inventory control problem in Section 5.4.2. A sensitivity study of APS-QBASE's parameter is given in Section 5.4.4.

Chapter 6 briefly summarizes the contributions in this thesis.

Chapter 2

Decision Making Under Uncertainty

In this chapter, we review several models for decision making under uncertainty and consider various methods for solving those models as well. We also introduce a stochastic optimization technique that does not rely on gradient information.

2.1 Multi-Armed Bandit

The Multi-Armed Bandit (MAB) problem was first described in Robbins (1952). In this problem, a gambler has to decide which of several slot machines (often called arms) to play with, where each machine gives a random reward according to some unknown probability distribution. The goal is to maximize the total reward of all the plays. Ideally, the player should only consider the arm that yields the highest reward on average. The difficulty of arm (or action) selection is that the reward function is not known and can only be evaluated by sampling. Thus, due to the fixed number of trials, the effects of the noise cannot completely be eliminated before the decision is made. In the decision process, one has to face the dilemma of the exploration and exploitation. In this thesis, we only consider MAB with stochastic reward functions.

Definition 2.1.1 (MAB)**.** A multi-armed bandit can be described as 2-tuple $\langle \mathcal{K}, R \rangle$ consisting of

- Arm space: $\mathcal{K}$ is the set of possible arms. Each arm $k \in \mathcal{K}$ corresponds to an unknown reward distribution D_k with support $[0, 1]$ and expectation μ_k.
- Reward: R_k is a random reward for taking arm k. □

We assume that the reward distribution of each arm to be stationary and independent of the other arms. The goal of MAB is to maximize the total payout over a fixed number of trials. Mathematically, it can be defined as follows.

Definition 2.1.2 (Objective Function)**.** Let K_t be the arm at step t, $t = 1, \ldots$. The expected total rewards over a fixed number of plays is given by

$$\mathbb{E}\left[\sum_{t=1}^{T} R_{K_t}\right], \tag{2.1}$$

where T is the fixed number of plays. □

The expectation takes over stochastic outcomes of plays and the random selection made by the decision maker. A policy in MAB selects the next arm to play based on the history of past plays and rewards.

Equivalently, we can minimize a total regret. The total regret is the gap between not playing the optimal arm with the highest expectation during T plays. Specifically, a pseudo-regret (Bubeck et al., 2012) is defined as

$$T \max_{k \in \mathcal{K}} \mu_k - \mathbb{E}\left[\sum_{t=1}^{T} R_{K_t}\right]. \tag{2.2}$$

Example 2.1.1 (Navigation MAB problem)**.** Imagine an agent who stands in front of four closed doors (Figure 2.1). Behind each door is a machine. Every time a door is visited, the corresponding machine will return a reward. Due to the unreliability of the machines, the rewards from each machine are not deterministic and follow an unknown distribution. The agent has four choices (door 1, 2, 3 and 4) in each run. Immediately after it takes an action and receives a reward, the problem restarts. The goal of the agent is to maximize the total reward over a fixed number of trials. □

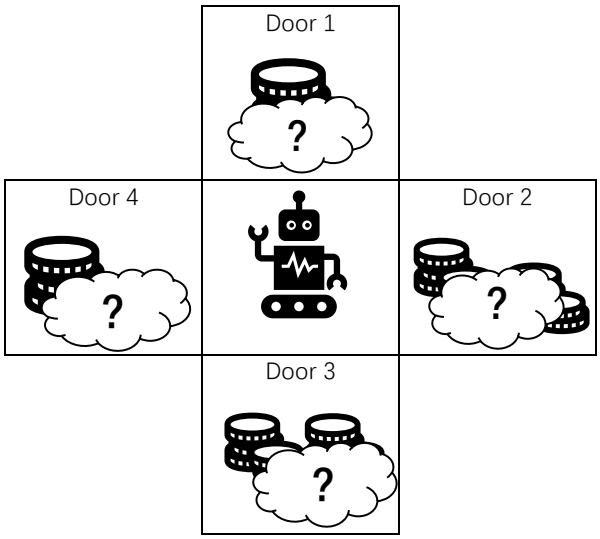


Figure 2.1: Illustration of the Navigation MAB problem

To find out the (near) best door as soon as possible, we can use different strategies to interact with the unknown environment. For example, one possibility is to try out all doors to obtain some initial understandings among those doors and then play with the promising doors more often in the future. The 'promising doors' refers to those doors that have been giving good rewards up to now. We provide more detailed explanations in different methods to balance the exploration and exploitation and compare their performance from small (like $|\mathcal{K}| = 2$) to large arm sets (such as $|\mathcal{K}| = 10,000$) in Chapter 3.

2.2 Markov Decision Process

2.2.1 Fundamentals

A Markov Decision Process (MDP) (Puterman, 1994) is a discrete-time stochastic control process. Although MDPs with continuous-time domain exists, this thesis focuses only on MDPs with discrete-time domains. At each time, the process is in some state and the decision maker can choose one of possible actions. As a result, the process will move to a new state and supply the decision maker with an associated reward. MDPs are generally used to model sequential

decision making − answering the question "what action should I take next to maximize a long-term goal".

The multi-armed bandit problem can be viewed as a particular case of Markov decision processes where there is only one state.

Definition 2.2.1 (MDP)**.** A Markov decision process can be described as a 4-tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ consisting of

- State space: $\mathcal{S}$ is the set of possible system states.
- Action space: $\mathcal{A}$ is the set of possible actions.
- State-transition function: $T(s, a, s') = p(s' \mid s, a)$ is the probability that the next state is s' given that the current state is s and action a is taken.
- Reward: $R(s, a)$ is the immediate reward function of choosing action a in state s. □

Definition 2.2.2 (Decision Rule)**.** A decision rule π_t prescribes a procedure for action selection in each state at a specified decision time t.

$$\pi_t : \mathcal{S} \to \mathcal{A}, \quad t = 0, 1, \ldots, N, \tag{2.3}$$

where $N (N \leq \infty)$ is the planning horizon. In particular, $\pi_t(s)$ is the action $a \in \mathcal{A}$ when the agent is in state s and $N - t$ steps have still to be taken until the horizon. □

Decision rules can be generally classified as deterministic Markovian, randomized Markovian, Markovian history dependent and randomized history dependent. The easiest decision rule is deterministic Markovian. This decision rule is said to be (first-order) Markovian for the reason that the next state depends on the current state (and action) only, rather than on all the previous states and actions, and deterministic because it chooses an action with certainty.

Definition 2.2.3 (MDP Policy)**.** A policy is a sequence of decision rules

$$\boldsymbol{\pi}_N = (\pi_0, \pi_1, \ldots, \pi_N), \quad t = 0, 1, \ldots, N, \tag{2.4}$$

where $N (N \leq \infty)$ is the planning horizon. □

For a given policy, the quality of following this policy, called the *value function*, can be measured by different criteria as follows.

Let S_t be the state at step t, $t = 0, 1, \ldots, N$.

1. *Finite-horizon reward*. For horizon $N(1 < N < \infty)$, the expected reward of finite-horizon model for a given policy $\boldsymbol{\pi}_N$ and an initial state s is given by

$$V_N^{\boldsymbol{\pi}_N}(s) = \mathbb{E}\left[\sum_{t=0}^{N} R(S_t, \pi_t(S_t))\middle| S_0 = s\right]. \tag{2.5}$$

   Since the mapping $\pi_t(s)$ possibly changes over time, such policy is called *non-stationary* policy.

2. *Infinite-horizon reward*. To be consistent with the definition in finite-horizon, let $\boldsymbol{\pi}_N$ store the planning history. If N goes to ∞, $\boldsymbol{\pi}_\infty$ would be $\boldsymbol{\pi}_\infty = (\pi_0, \pi_1, \ldots, \pi_k, \ldots)$. Since each π_k has the equal (infinite) horizons to lookahead, there is no difference between $\pi_0, \pi_1, \ldots$. Therefore, it suffices to consider only one π. Such policy is called *stationary policy*.

   The discounted value function for a given stationary policy π and an initial state s is given by

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t))\middle| S_0 = s\right], \tag{2.6}$$

   where $\gamma\,(0 < \gamma < 1)$ is a discount factor.

In what follows, we will only consider stationary policies.

Definition 2.2.4 (The Optimal Policy)**.** Let Π be a policy space. Then the optimal policy can be derived from the optimal value function $V^{\pi}(s)$.

$$\pi^*(s) = \underset{\pi \in \Pi}{\operatorname{argmax}}\, V^{\pi}(s), \quad s \in \mathcal{S}. \tag{2.7}$$

The associated optimal value is

$$V^*(s) = \max_{\pi \in \Pi} V^{\pi}(s), \quad s \in \mathcal{S}. \tag{2.8}$$

□

Example 2.2.1 (Navigation MDP problem (Russell and Norvig, 2016))**.** The decision maker is an agent that moves in a grid world (Figure 2.2). The state space consists of 11 walkable states and one unwalkable state (s_{22}) The agent starts from the cell s_{31} and tries to reach the cell with $+1$ with the lowest cost. The agent has four possible movements {`NORTH`, `WEST`, `SOUTH`, `EAST`}. It cannot move into the obstacle cell. Due to the unknown fraction coefficient of the ground surface, the agent only has 80% moving accuracy. The agent receives a reward of +1 when it reaches to the cell s_{14}, and incurs a cost of -1 when it enters the cell s_{24}. Moving out of any other cells incurs a cost of -0.04. □

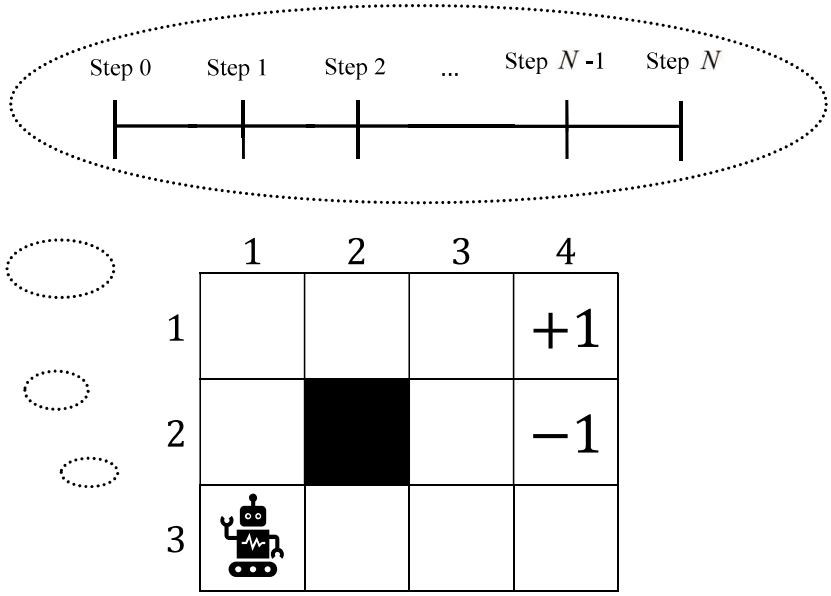


Figure 2.2: Illustration of the Navigation MDP problem

In order to achieve the best total rewards, the agent has to consider sequences of all possible future states after performing the current action. For example, when the agent stays in s_{23} and s_{34}, it has some potential to fall into the penalty cell (s_{24}) due to the movement uncertainty. Such future risks should be considered when the robot decides where to go from the current state. Therefore, go `EAST` should not be better than go `NORTH`.

2.2.2 Value Iteration

Off-line Solvers

To solve an MDP problem, there are two basic approaches: off-line and on-line. The off-line methods build a policy prior to execution, whereas the on-line algorithms interleave a planning phase with an execution step. Figure 2.3 visualizes the different running processes of off-line and on-line solvers.

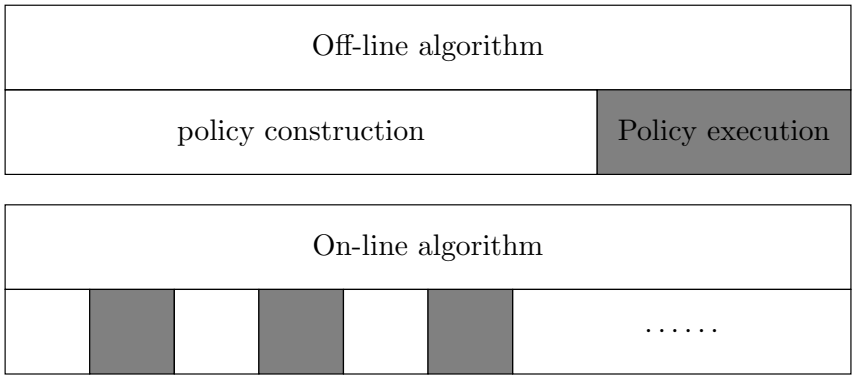


Figure 2.3: Off-line and on-line algorithms comparison

Off-line MDP methods can be generally divided into three families: value iteration, policy iteration, and policy search. For policy search, one directly searches the policy in the policy space Π. Given a policy, it can be directly evaluated by the value function in (2.6). In this type of solvers, the key components depend on policy π parametrization and the optimization used to search for next policies (Williams, 1992; Ng and Jordan, 2000; Mannor et al., 2003; Busoniu et al., 2011; Chang et al., 2013).

Value iteration and policy iteration (Puterman, 1994) explicitly build a mapping from $\mathcal{S}$ to $\mathcal{A}$. Those two methods mainly rely on Bellman's criteria, which is derived as follows:

$$V^*(s) = \max_{a \in \mathcal{A}} \underbrace{\left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s') \right]}_{\text{Q-value: } Q(s,a)}, \tag{2.9}$$

where $Q(s,a)$ denotes the state-action value or Q-value. $Q(s,a)$ provides a measure for selecting the best action given the same state.

Value Iteration

The basic idea of Value Iteration (VI) is simple. For a given state s, it tries to improve its value function $V(s)$ by one-step lookahead. During the local update, the immediate reward and possible future values are backpropagated to the value of the current state s. VI sweeps the entire action space to determine which action is able to yield the highest estimate Q-value according to the right-hand side of (2.9), with V instead of V^*. The associated action forms the decision rule at the current state. We do this update for all state s. The outputs of VI are the approximated value function $\widehat{V}$ and the policy $\widehat{\pi}$. VI essentially is an application of the fix-point theorem to search the (approximate) optimal value function. The overview of the VI algorithm is described in Algorithm 1.

Algorithm 1: Value Iteration for the Infinite-horizon Case

Input: Tolerance $\varepsilon > 0$ and discount factor γ.
Output: $\widehat{V}_t(s)$ and $\widehat{\pi}(s)$ for each $s \in \mathcal{S}$.

Set $t \leftarrow 1$
for $s \in \mathcal{S}$ **do**
  $\widehat{V}_1(s) \leftarrow \max_{a \in \mathcal{A}} R(s, a)$
while $||\widehat{V}_t - \widehat{V}_{t-1}|| > \frac{\varepsilon(1-\gamma)}{2\gamma}$ **do**
  $t \leftarrow t + 1$.
  for $s \in \mathcal{S}$ **do**
    $\widehat{V}_t(s) \leftarrow \max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \widehat{V}_{t-1}(s') \right]$
for $s \in \mathcal{S}$ **do**
  $\widehat{\pi}(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \widehat{V}_t(s') \right]$
return $\widehat{V}_t(s)$ *and* $\widehat{\pi}(s)$

Now, we show how to apply VI to solve the Navigation MDP problem.

Example 2.2.2 (Navigation MDP continued)**.** We can define the problem in an MDP manner.

- $\mathcal{S} = \{s_{11}, s_{12}, s_{13}, s_{14}, s_{21}, s_{23}, s_{24}, s_{31}, s_{32}, s_{33}, s_{34}\}$, where s_{14} is a terminal state.
- $\mathcal{A} = \{\texttt{NORTH}, \texttt{WEST}, \texttt{SOUTH}, \texttt{EAST}\}$.
- the state-transition function $p(s'|s, a)$

– if next state s' is walkable. For example, if the agent in state s_{33} and chooses action "North", its transition function is as follows:

$$p(s' \mid s_{33}, \text{'North'}) = \begin{cases} 0.8, & s' \text{ lies in } s_{23} \\ 0.1, & s' \text{ lies in } s_{32} \\ 0.1, & s' \text{ lies in } s_{34} \end{cases}$$

– if next state s' is not walkable, the state after transition will stay in its original state. For example, the transition function in state s_{21} choosing action "East" is

$$p(s' \mid s_{21}, \text{'East'}) = \begin{cases} 0.8, & s' \text{ will stay in } s_{21} \\ 0.1, & s' \text{ lies in } s_{11} \\ 0.1, & s' \text{ lies in } s_{31} \end{cases}$$

– if state s is in terminal state, it will never move out.

- For the reward function, there is one terminal and ten non-terminals. Specifically,

$$R(s) = \begin{cases} +1, & s = s_{14} \\ -1, & s = s_{24} \\ -0.04, & \text{all other } s \end{cases}$$

□

We compare several optimal policies with different parameters γ and ε. Given the same discount factor $\gamma = 0.95$, the value function of those states far from terminals are able to back-propagate well with a lower tolerance ε. We find the values of s_{31} and s_{32} improve significantly and the optimal policy at s_{32} changes to `WEST` (Figure 2.4 and Figure 2.5). Although it leads to a longer path, it avoids the potential of falling into the penalty cell. If ε is fixed, a larger discount factor γ helps the agent to lookahead further and behave in a relative smart way (Figure 2.5 and Figure 2.6). When γ is small, the agent seems to simply get away from some costs close to its current cells, such as s_{23} and s_{34}.

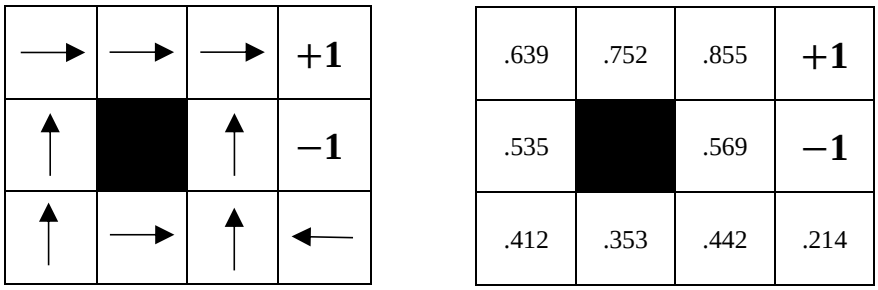


Figure 2.4: The optimal policy and associated value with $\gamma = 0.95$ and $\varepsilon = 0.1$

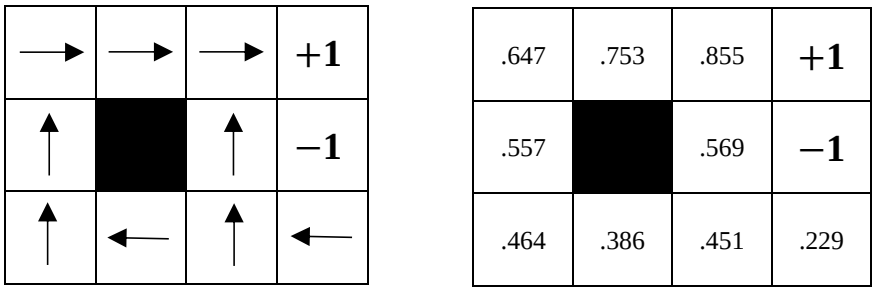


Figure 2.5: The optimal policy and associated value with $\gamma = 0.95$ and $\varepsilon = 0.001$

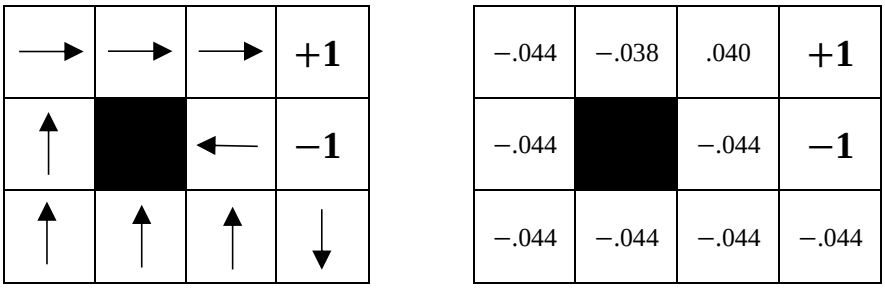


Figure 2.6: The optimal policy and associated value with $\gamma = 0.1$ and $\varepsilon = 0.001$

2.2.3 Monte Carlo Tree Search

On-line Solvers

In off-line solvers, one has to build the mapping π for the entire state space. When the size of the state space is large, constructing policies in such a way can be computationally inefficient. On-line algorithms interleave a planning phase with an execution step. In other words, on-line

solvers only need to figure out the best action (or policy) with respect to the agent's current state. It brings the benefit that the memory requirement of an on-line solver is dramatically reduced.

Most on-line algorithms use a generative model due to two main advantages. Firstly, in many practical problems, parts of the MDP model may not be known explicitly, such as the state-transition function T and/or the reward function R. But we are still able to solve the MDP problem via using such a simulator. Secondly, even if the complete model information is known, it is still beneficial to use a simulator when the model is complicated or extensive. Take VI (Algorithm 1) for example, it is not viable for fully updating V over the whole state space if the state space is large. However, it is efficient to sample the next state from the transition function with a given state and an action.

In this section, we introduce the Monte Carlo tree search. Other common on-line MDP solvers include sparse sampling (Kearns et al., 2002) and real-time dynamic programming (Barto et al., 1995; Bonet and Geffner, 2003).

Policy Tree

A policy in MDPs can be represented in multiple ways. In this section, we adopt the policy tree, which maintains an explicit representation of the policy.

Figure 2.7 illustrates a simple MDP policy tree. It is convenient to view the MDP planning as building an AND-OR tree (Figure 2.7). The AND-OR tree is a tree of reachable future states from the current state. In the tree, there are two types of nodes: state nodes (circles) and action nodes (squares). The root node s_0 represents the current state. Starting from this state, two actions can be taken: a and a'. Action a can lead to two types of next states: s and s'. The state node s is a child resulting from taking action a and observing s, starting from root node s_0. It is important to note that we will use the same notation to refer to a state node and the state it represents. If the action space and state space only have a few elements, it is possible to fully enumerate all possibilities up to a given depth of the policy tree. This is no longer feasible

for large (e.g., continuous) action and observation spaces, so that only a part of the tree can be evaluated.

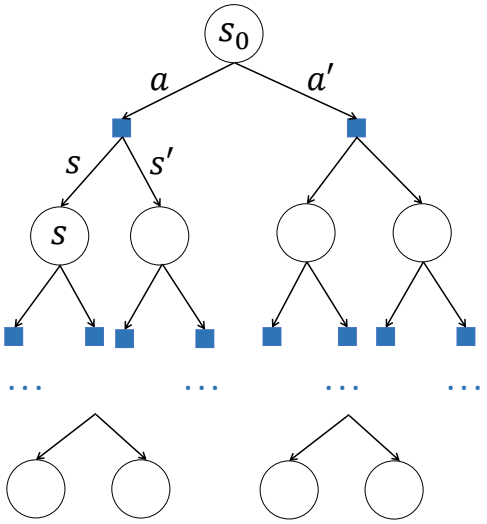


Figure 2.7: MDP tree with two actions and two states

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an on-line solver that generates the optimal policy by using a simulator. It explicitly represents a subset of the policy via a tree, and improves and updates the policy at the same time. Upper Confidence bounds applied to Trees (UCT) (Kocsis and Szepesvári, 2006) is one of the most successful variants. In the search tree, there are two attributes associated with each state node: an estimate of the Q-value $Q(s, a)$ and the number of visits $N(s, a)$ to state s using action a. The basic idea of UCT is to iterate over the following five steps. Based on those steps, one iteration of UCT is visualized in Figure 2.8.

1. Each simulation always starts from the root node. At beginning of the search, the tree contains nothing more than the root s_0.

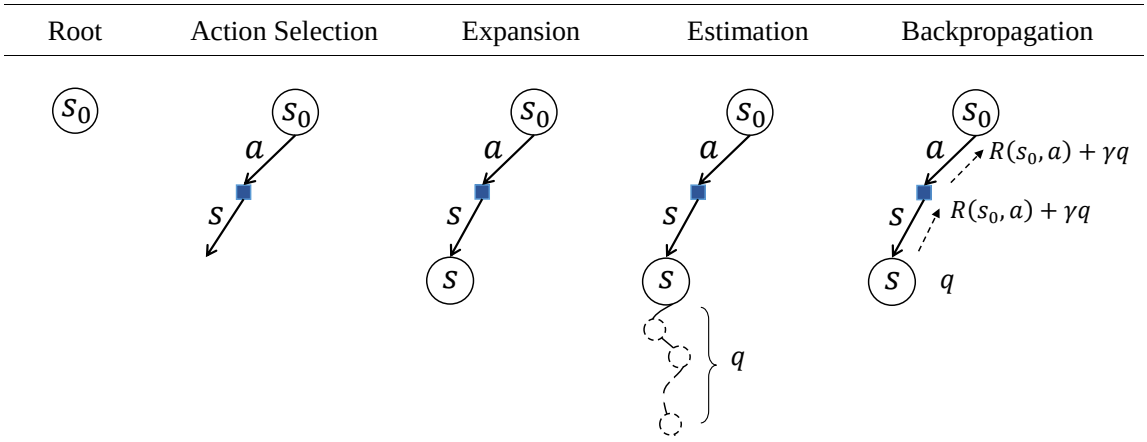


Figure 2.8: One iteration of UCT

2. *Action selection*: For each state node, UCT frames the action selection as a multi-armed bandit problem and applies the UCB1 strategy to balance exploration and exploitation. The action is selected by

$$a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left\{ Q(s,a) + C\sqrt{\frac{\log N(s)}{N(s,a)}} \right\}, \tag{2.10}$$

where C is an exploration constant and $N(s)$ is the number of visits for state s and it is equal to $\sum_{a \in \mathcal{A}} N(s,a)$.

Suppose the current node is associated with state s_0, UCB1 determines the most promising action according to (2.10). With inputs of (s_0, a), the simulator returns the information about the next state and an immediate reward in the form of $(s, R(s_0, a))$. Repeat this process until the next state node does not exist in the search tree.

3. *Expansion*: When a leaf node is reached, we add a new state child node to the current search tree if the expansion condition is met. The default value for $Q(s,a)$ and $N(s,a)$ can be initialized according to our prior knowledge of the problem; otherwise, they can be simply set as 0.

4. *Estimation*: The estimate (say q) of the newly added node can be obtained by running a default policy. It can be done by a random walk up to a pre-defined number of steps. This

estimate can be quite rough but it still can provide some information to bias the search to promising directions.

5. *Backpropagation*: We update the information of $Q(s, a)$ and $N(s, a)$ for all nodes from the expanded node to the root with the information obtained from the new estimate at the bottom.

MCTS has been widely applied to different domains, for example, real-time strategy game (Balla and Fern, 2009), general game playing (Finnsson and Björnsson, 2008) and the game of Go (Silver et al., 2016). A relatively comprehensive survey is reviewed for its methodology, enhancements and applications in Browne et al. (2012).

2.3 Partially Observable Markov Decision Process

2.3.1 Fundamentals

While the MDP is a principled manner to model sequential decision making with uncertainty in the effect of action, a partially observable Markov decision process (POMDP) adds additional uncertainty description over the state space, namely uncertainty effects of perception (Kaelbling et al., 1998).

Definition 2.3.1 (POMDP)**.** A partially observable Markov decision process can be described as a 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$, where $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ are defined as the same way as in the MDP. $\mathcal{O}$ and Z are defined as:

- Observation space: $\mathcal{O}$ is the set of possible observations. Observation $o \in \mathcal{O}$ is usually an imperfect projection of the system state.
- Observation-transition function: $Z(s, a, o) = p(o \mid s, a)$ is the probability of observation o at the next state given action a in state s.

Due to uncertainties in the effects of actions and in sensing, the agent never knows its exact state. Therefore, a POMDP agent maintains an estimate of its current state in the form of a belief b.

Definition 2.3.2 (Belief State)**.** A belief state b is a distribution over the state space. After executing an action a and obtaining an observation o, the next belief state is

$$b' = \tau(b, a, o), \tag{2.11}$$

which can be computed as Bayes' formulate as

$$\begin{aligned} b'(s') &= p(s' \mid b, a, o) \\ &= \frac{1}{p(o \mid b, a)} p(o \mid s', a) \sum_{s \in \mathcal{S}} p(s' \mid a, s) b(s), \end{aligned} \tag{2.12}$$

where $p(o \mid b, a) = \sum_{s' \in \mathcal{S}} p(o \mid s', a) \sum_{s \in \mathcal{S}} p(s'|a, s) b(s)$. □

Note that $p(o \mid b, a)$ represents the probability of perceiving the observation o after performing the action a from the belief b.

A POMDP policy $\pi : \mathcal{B} \to \mathcal{A}$ assigns an action a for each belief $b \in \mathcal{B}$, and induces a value function $V^\pi(b)$ which computes the expected total reward of following policy π at the current belief b.

Definition 2.3.3 (POMDP Infinite-horizon Reward)**.** Let B_t be the belief at step t, $t = 0, 1, \ldots$. The discounted value function for a policy π and an initial state b is given by

$$V^\pi(b) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(B_t, \pi(B_t)) \middle| B_0 = b \right], \tag{2.13}$$

where $\bar{R}(b, a) = \sum_{s \in \mathcal{S}} R(s, a) b(s)$ and γ $(0 < \gamma < 1)$ is a discount factor. □

Similar to MDPs, the Bellman's equation in POMDPs is

$$V^*(b) = \max_{a \in \mathcal{A}} \underbrace{\left[\sum_{s \in \mathcal{S}} R(s,a) b(s) + \gamma \sum_{o \in \mathcal{O}} p(o \mid b, a) V^*(\tau(b,a,o)) \right]}_{Q-\text{value: } Q(b,a)}, \tag{2.14}$$

where $\tau(b, a, o)$ denotes the next belief.

Example 2.3.1 (Navigation POMDP problem)**.** The decision maker is an agent that moves in a grid world (Figure 2.9). The robot does not know where it is; instead, it keeps a belief, which is a distribution over the state space. The different levels of grey cells indicate the likelihood of the agent at each location. At each step, the robot takes an action based on a given belief. It receives an immediate reward and an observation. In this problem, the agent can only observe the existence of walls surrounding it. The goal of the robot is to reach the "+1" cell with a lower cost. □

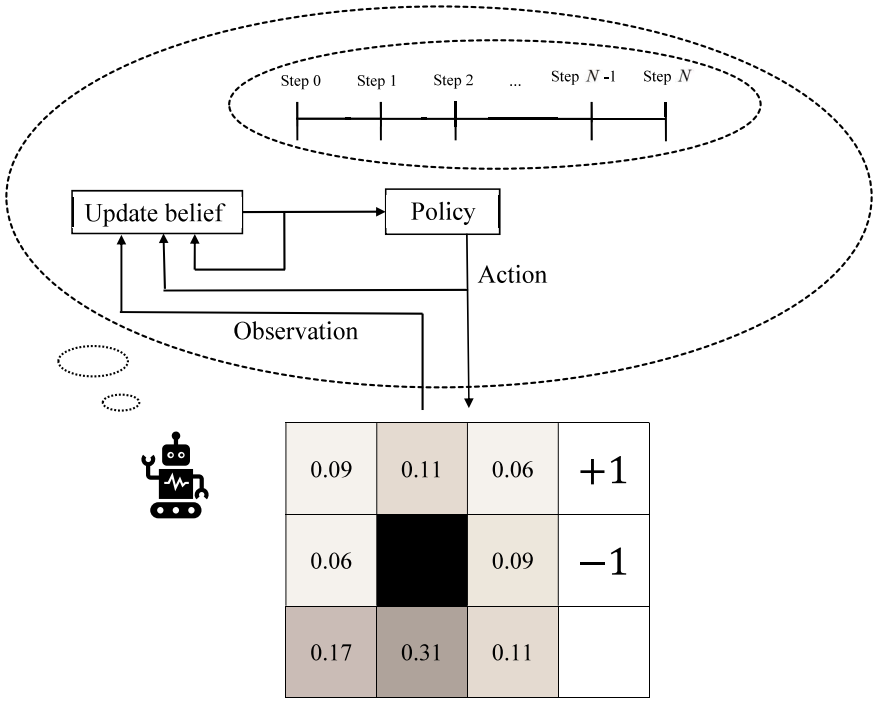


Figure 2.9: Illustration of the Navigation POMDP problem

In this example, the observation only captures local information of the surrounding walls. Even though it knows exactly where walls are, it still can not fully observe the agent's state. This is because multiple states can yield the same observation. We elaborate and solve a much larger version of this example (up to $|\mathcal{S}| \approx 8 \times 10^5$, $|\mathcal{A}| \approx 2,400$ and $|\mathcal{O}| = 256$) in Section 4.4.2 of Chapter 4.

Example 2.3.2 (Tiger Problem (Kaelbling et al., 1998))**.** Imagine an agent that stands in front of two closed doors. Behind one of the doors there is a tiger and behind the other one is a large reward. If the agent opens the door with the tiger, a substantial penalty is received (presumably in the form of some amount of bodily injury). Instead of opening one of the two doors, the agent can listen to gain more information about the location of the tiger. Unfortunately, listening is not free; besides, it is also not entirely accurate. There is a chance that the agent will hear a tiger behind the left-hand door when the tiger is actually behind the right-hand door, and vice versa. □

2.3.2 Solving the Tiger Problem

To motivate the use of an on-line method, let us solve the Tiger problem in this section. Given the current belief, the goal is to figure out the best action for this belief. In this tree, there are two types of nodes: belief nodes (OR-nodes) and action nodes (AND-nodes). The root node b represents the current belief. Starting from this belief b, two actions can be taken: a and a'. Action a leads to two types of observations: o and o'. In this case the same observations can occur for actions a and a' (but with different probabilities). As a result of taking action a and observation o, a belief node $b' = \tau(b, a, o)$ is obtained.

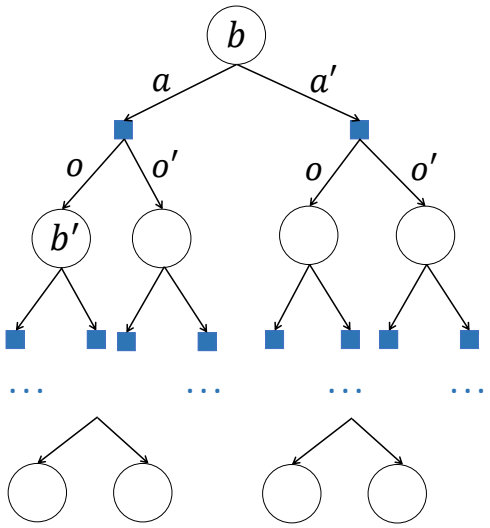


Figure 2.10: AND-OR tree with two actions and two observations

Example 2.3.3 (Tiger problem (cont.))**.** We can describe the problem in a POMDP manner.

- $\mathcal{S} = \{s_l, s_r\}$. The tiger is behind the left door or behind the right.
- $\mathcal{A} = \{\texttt{LEFT}, \texttt{RIGHT}, \texttt{LISTEN}\}$.
- The state-transition function. The `LISTEN` action does not change the state of the world. The `LEFT` and `RIGHT` actions cause a transition to system state s_l with probability 0.5 and to state s_r with probability 0.5 (essentially resetting the problem).
  - $p(s_l \mid \texttt{LISTEN}, s_r) = p(s_r \mid \texttt{LISTEN}, s_l) = 0$.
  - $p(s_l \mid \texttt{LISTEN}, s_l) = p(s_r \mid \texttt{LISTEN}, s_r) = 1$.
  - For the `LEFT` and `RIGHT` actions, $p(\cdot \mid \texttt{LEFT}, \cdot) = p(\cdot \mid \texttt{RIGHT}, \cdot) = 1/2$.
- The reward function.
  - $R(s_l, \texttt{LISTEN}) = R(s_r, \texttt{LISTEN}) = -1$.
  - $R(s_l, \texttt{RIGHT}) = R(s_r, \texttt{LEFT}) = +10$.
  - $R(s_l, \texttt{LEFT}) = R(s_r, \texttt{RIGHT}) = -100$.

- $\mathcal{O} = \{\texttt{TL}, \texttt{TR}\}$. There are only two possible observations: to hear the tiger on the left (TL) and to hear the tiger on the right (TR).

- The observation probabilities. When the tiger is in state s_l, the LISTEN action results in the observation TL with probability 0.85 and the observation TR with probability 0.15; the same is true for state s_r. No matter what state the world is in, the LEFT and RIGHT actions result in an observation with probability 0.5.

  - $p(\texttt{TL} \mid \texttt{LISTEN}, s_l) = p(\texttt{TR} \mid \texttt{LISTEN}, s_r) = 0.85.$
  - $p(\texttt{TR} \mid \texttt{LISTEN}, s_l) = p(\texttt{TL} \mid \texttt{LISTEN}, s_r) = 0.15.$ □

We would like to emphasize that the objective in (2.13) is to generate a policy considering an infinite planning horizon. However, for the convenience of getting a better insight into the planning process, we only consider a few examples with horizon $N = 2$.

Let $b_{d,i}$ denote the $i-$th belief node at depth d of the search tree.

1. The denominator in (2.12) is computed as follows.

$$p(\texttt{TL} \mid b_0, \texttt{LISTEN}) = 0.85b_0(s_l) + 0.15b_0(s_r) \tag{2.15}$$

$$p(\texttt{TR} \mid b_0, \texttt{LISTEN}) = 0.85b_0(s_r) + 0.15b_0(s_l). \tag{2.16}$$

2. The new beliefs $b_{1,2}$ and $b_{1,3}$ are calculated using (2.12) as follows.

$$b_{1,2}(s_l) = \frac{0.85b_0(s_l)}{0.85b_0(s_l) + 0.15b_0(s_r)},\ b_{1,2}(s_r) = \frac{0.15b_0(s_r)}{0.85b_0(s_l) + 0.15b_0(s_r)}. \tag{2.17}$$

   Using similar arguments, we arrive at

$$b_{1,3}(s_r) = \frac{0.85b_0(s_r)}{0.85b_0(s_r) + 0.15b_0(s_l)},\ b_{1,3}(s_l) = \frac{0.15b_0(s_l)}{0.85b_0(s_r) + 0.15b_0(s_l)} \tag{2.18}$$

Note that the above equations in 1 and 2 are valid for any horizon. Thus, they open the way for exact policy calculation for any finite horizon N.

Suppose there is no prior knowledge about the tiger position, namely $b_0 = (b_0(s_l), b_0(s_r)) = (0.5, 0.5)$. We apply (2.14) to compute the Q-value for each belief and actions encountered. The policy tree (Figure 2.11) contains the optimal policy at the root for planning $N = 2$ horizons ahead. The optimal action for b_0 is to `LISTEN`. Once `LISTEN`, the next belief evolves to $(0.85, 0.15)$ if the observation is the `TL`; otherwise the next belief becomes $(0.15, 0.85)$.

Thereafter, suppose the agent gets the `TL` observation, then the next belief b_1 is $(0.85, 0.15)$. The agent will need plan for $N = 2$ horizons again for the new belief. The optimal action in the search tree (Figure 2.12) is still to `LISTEN`, though there is a relatively strong preference that the tiger is on the left door. This process repeats until the agent opens the door.

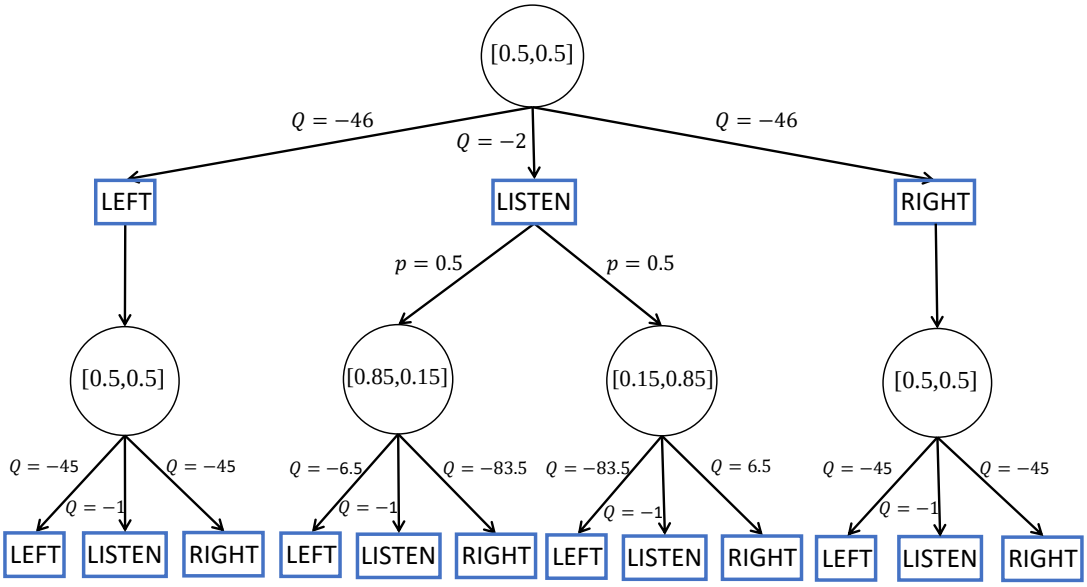


Figure 2.11: Tiger AND-OR tree for $b_0 = (b_0(s_l), b_0(s_r)) = (0.5, 0.5)$ and $N = 2$.

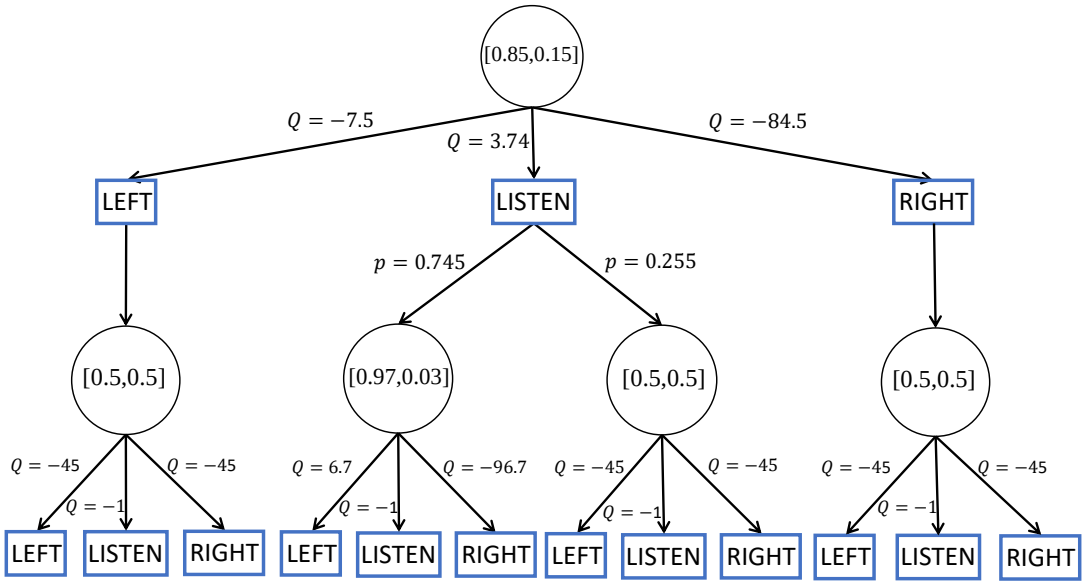


Figure 2.12: Tiger AND-OR tree for $b_1 = (b_1(s_l), b_1(s_r)) = (0.85, 0.15)$ and $N = 2$

2.3.3 Partially Observable Monte Carlo Planning

On-line Solvers

Although POMDPs essentially can be viewed as belief-state MDPs, the computation complexity of finding the optimal policy increases exponentially respect to the number of states, actions, observations and planning horizons. Various solvers can be applied to find the optimal policy (Kochenderfer, 2015), including exact solution methods via the alpha vector, off-line methods and on-line methods. Table 1.1 in Chapter 1 lists recent developments in *off-line* and *on-line* POMDP solvers. In general, on-line solvers scale better and faster than off-line solvers.

In POMDPs, a policy is computed with respect to a belief rather than a single state. The dimension of the planning space is normally high-dimensional. This is known as the curse of dimensionality. To this reason, successful POMDP solvers plan only on a representative and small subset of the belief space. Moreover, the number of belief-contingent plans increase exponentially with the planning horizon. The branching factor of the search tree is of the order $O\left((|\mathcal{A}||\mathcal{Z}|)^N\right)$. Even for a simple problem like Tiger, maintaining such a large tree is com-

putationally infeasible, which is known as the curse of history. As a consequence, we resort to approximation techniques.

POMCP

Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness, 2010) is one of the most widely used on-line POMDP planners. It breaks the two curses mentioned above by using a sampling approach. POMCP basically extends UCT to handle partially observable problems. There are many similarities between UCT and POMCP. Firstly, POMCP adopts a policy tree representation to compute the optimal policy with considering reachable beliefs only. This largely helps to reduce the memory consumption compared with off-line POMDP solvers. Secondly, POMCP applies a simulator to generate solutions instead of backpropagating the value function explicitly. The only difference in a POMDP simulator is to additionally generate an observation due to partial observability.

Instead of keeping track of beliefs exactly, POMCP adopts the particle filter to sample states (Arulampalam et al., 2002). POMCP represents a belief as a collection of unweighted particles, where such a collection is incrementally constructed by sampling. In each simulation, POMCP always starts from the root node say b_0, samples a state $s \sim b_0$ and takes an action a according to UCB1 strategy (Auer et al., 2002a). With inputs of (s, a), one obtains the information about a next state, an observation and a reward in the form of (s', o, r). Specifically, the process of generating samples from a POMDP simulator then become as follows.

1. Sample s from the belief b_0
2. Sample s' according to the state-transition function $p(s' \mid s, a)$
3. Sample o according to the observation function $p(o \mid s, a)$
4. Get a reward from $R(s, a)$

Through the a-o pair, it arrives at a new belief. In particular, the next state s' will be inserted into the next belief node. The process repeats until the next belief node does not exist in the current search tree. If a leaf belief node is reached, POMCP expands a new node and links it to the tree.

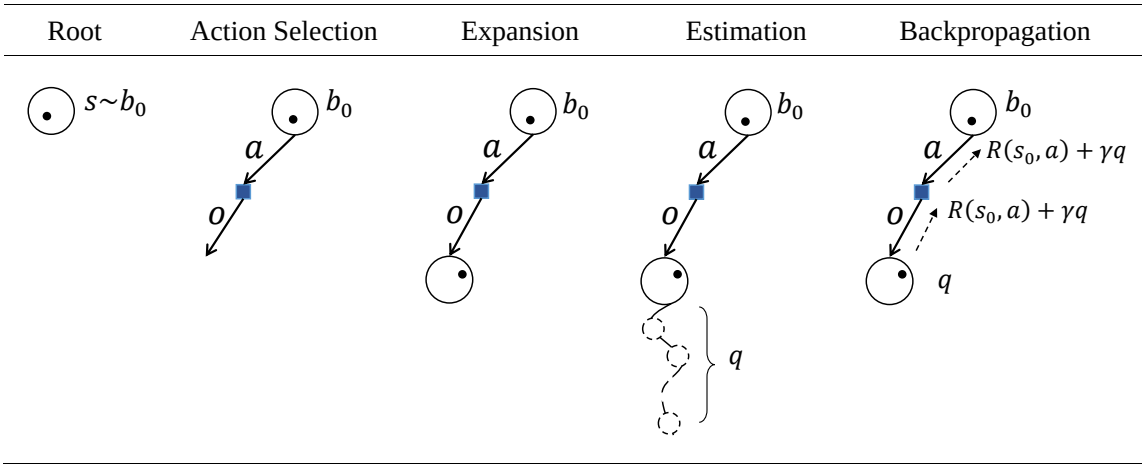


Figure 2.13: One iteration of POMCP

In terms of estimating the newly added node, the simplest way is to run a default policy, such as employing a random walk from this node. The estimate is used to backpropagate the Q-value from this leaf node up to the root. Repeating such simulations many times, the number of states arriving at any belief node in the tree gradually increases and then the set of those states is to approximately represent the belief. A sample iteration of POMCP is visualized in Figure 2.13.

Similar to UCT, POMCP interleaves a planning phase with an execution. POMCP decides the best action to execute from the root node after simulating the policy tree for a pre-defined budget. Depending on the observation, the root of the policy tree resets and is ready to re-plan. However, the number of particles used to represent the next belief decreases dramatically due to effects of branching factor $\mathcal{A}$ and $\mathcal{O}$. To deal with the issue, it will be useful to add an extra number of particles before re-planning the policy for next step. This operation is called particle reinvigoration.

2.4 Cross-Entropy Method

Decision making often involves an optimization component. The goal of the decision models in (2.1) (2.6) (2.13) is inherently to solve noisy (or stochastic) optimization problems, in which the objective function values are unknown and are obtained via Monte Carlo simulations. In

this section, we introduce a randomized algorithm, called the *Cross-Entropy method* (CE) (Rubinstein and Kroese, 2004), to address optimization problems. Other common randomized optimization methods include stochastic approximation, stochastic counterpart method, simulated annealing and evolutionary algorithms (Kroese et al., 2011).

The Cross-Entropy method can be considered as an importance sampling approach to the problem of estimating rare events (Rubinstein, 1997). It is realized that its adaptation mechanism could be applied to solve optimization problems as well, as searching for a small set of solutions to that optimization problem can be treated as a rare-event (Rubinstein, 1999). Suppose the goal is to find the maximum of $S(\mathbf{x})$ on a given set $\mathcal{X}$. Denote the maximum by γ^* corresponding to maximizer $\mathbf{x}^*$, so that

$$S(\mathbf{x}^*) = \gamma^* = \max_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x}). \tag{2.19}$$

Now associate this problem with the estimation of the probability $\ell = \mathbb{P}(S(\mathbf{X}) \geq \gamma)$, where $\mathbf{X}$ is generated from some parametrized distribution $f(\mathbf{x}; \mathbf{v})$ on $\mathcal{X}$ and γ is some threshold. Finding the optimal $\mathbf{x}^*$ then converts into iterating of the rare-event simulation steps until the $\widehat{\gamma}_t$ approaches γ^*. It is crucial to understand that one of the main goals of the CE in optimization is to generate a sequence of pdfs $f(\mathbf{x}; \mathbf{v}_0)$, $f(\mathbf{x}; \mathbf{v}_1), \ldots$ converging to a delta distribution and all samples $\mathbf{X}$ will become almost identical. By doing this, the CE method aims to locate an optimal parametric sampling distribution on $\mathcal{X}$, rather than locating the optimal solution directly.

The description of the CE algorithm for optimization is given in Algorithm 2. For initialization, $\widehat{\mathbf{v}}_0$ denotes the parameter of a sampling density function, $\widehat{\gamma}$ stores the best level from the beginning and N^e is the number of elite sample in each iteration where ρ is elite proportion. CE mainly iterates two stages. First, it generates N samples from the current sampling distribution and receives the performance based on the objective function $S(\mathbf{x})$. According to their performance, CE sorts the batch samples and determines a new threshold $\widehat{\gamma}_t$ by top ρ-quantile (line 2-4). Second, if the new threshold gets improved, CE updates the sampling parameter via (2.20). It can be viewed as the *exponential maximum likelihood estimation* (line 5-7). The CE algorithm repeats those two stages until the stopping criterion (line 1) is met. The condition

can be set as the convergence of a parameter, for example $||\widehat{\mathbf{v}}_t - \widehat{\mathbf{v}}_{t-1}|| < \varepsilon$, or the threshold level goes to stable after a long run such as $\widehat{\gamma}_t = \widehat{\gamma}_{t+1} = \cdots = \widehat{\gamma}_{t+d}$ if for some $t \geq d$, say $d = 5$. At the end of the search, CE returns maximum value found $\widehat{\gamma}$ and its associated parameter of sampling density $\widehat{\mathbf{v}}_t$ (line 8).

Algorithm 2: CE Algorithm for Optimization

Input: Set $\widehat{\mathbf{v}}_0 = \mathbf{u}$ and let $\widehat{\gamma} = \widehat{\gamma}_0$ be a small number. Let $N^e = \lceil \varrho N \rceil$. Set $t = 1$.
Output: $\widehat{\gamma}, \widehat{\mathbf{v}}_t$.

1 **while** $\widehat{\gamma}_t$ *or* $\widehat{\mathbf{v}}_t$ *meets stopping criterion* **do**
2 Sample $\mathbf{X}_1, \ldots, \mathbf{X}_N \overset{\text{i.i.d}}{\sim} f(\mathbf{x}; \widehat{\mathbf{v}}_{t-1})$.
3 Calculate $S_i = S(\mathbf{X}_i)$, and sort them from smallest to largest: $S_{(1)} \leq \cdots \leq S_{(N)}$.
4 Estimate $\widehat{\gamma}_t = S_{(N-N^e+1)}$.
5 **if** $\widehat{\gamma}_t \geq \widehat{\gamma}_{t-1}$ **then**
6 $\widehat{\gamma} \leftarrow \widehat{\gamma}_t$.
7 Use the same sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ and update parameter $\widehat{\mathbf{v}}_t$ by,

$$\widehat{\mathbf{v}}_t = \underset{\mathbf{v}}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq \widehat{\gamma}_t\}} \ln f(\mathbf{X}_i; \mathbf{v}). \tag{2.20}$$

8 **return** $\widehat{\gamma}, \widehat{\mathbf{v}}_t$.

A smooth updating rule is often used, in which the parameter vector $\widehat{\mathbf{v}}_t$ is taken as

$$\widehat{\mathbf{v}}_t = \operatorname{diag}(\boldsymbol{\alpha})\tilde{\mathbf{v}}_t + \operatorname{diag}(1 - \boldsymbol{\alpha})\widehat{\mathbf{v}}_{t-1}, \tag{2.21}$$

where $\tilde{\mathbf{v}}_t$ is the solution to Algorithm 2 and $\boldsymbol{\alpha}$ is a vector of smoothing parameters, with each component in $[0, 1]$. More modifications of CE can be found in (Rubinstein and Kroese, 2017), convergence properties are discussed in (Margolin, 2005; Costa et al., 2007; Hu et al., 2012) and a runtime analysis is provided in (Wu et al., 2017, 2018).

So far, the performance function $S(\mathbf{x})$ is assumed to be deterministic. In term of noisy optimization problems, we consider the objective in (2.19) again and now outcomes of performance function is stochastic. In other words, $S(\mathbf{x}) = \mathbb{E}[\widehat{S}(\mathbf{x})]$ is not available but can be measured via simulations. We are able to re-use the framework of CE (Algorithm 2) with replacing $S(\mathbf{x})$ by $\widehat{S}(\mathbf{x})$.

Since CE was invented, it has been widely applied to a broad in both deterministic and stochastic domains, for example, Markov decision process (Mannor et al., 2003; Busoniu et al., 2011), game of Tetris (Szita and Lörincz, 2006), robotics motion planning (Kobilarov, 2012), Laguerre tessellation (Duan et al., 2014; Spettl et al., 2016), multi-armed bandit problem (Wang et al., 2017), partially observable Markov decision process (Omidshafiei et al., 2016; Wang et al., 2018).

The following publication has been incorporated as Chapter 3.

1. **Erli Wang**, Hanna Kurniawati, and Dirk P. Kroese, CEMAB: A Cross-Entropy-based Method for Large-Scale Multi-Armed Bandits, *Australasian Conference on Artificial Life and Computational Intelligence*, 353-365, 2017

Contributor	Statement of contribution	%
Erli Wang	Conception and design	40
	Implementation of numerical experiments	100
	Drafting and production	30
Hanna Kurniawati	Conception and design	30
	Drafting and production	40
Dirk P. Kroese	Conception and design	30
	Drafting and production	30

Chapter 3

CEMAB: A Sample-based Method for Large-scale Multi-Armed Bandits

The Multi-Armed Bandit (MAB) problem is an important model for studying the exploration–exploitation tradeoff in sequential decision making. In this problem, a gambler has to repeatedly choose between a number of slot machine arms to maximize the total payout, where the total number of plays is fixed. Although many methods have been proposed to solve the MAB problem, most have been designed for problems with a small number of arms. To ensure convergence to the optimal arm, many of these methods, including leading methods such as UCB (Auer et al., 2002a), require sweeping over the entire set of arms. As a result, such methods perform poorly in problems with a large number of arms. This chapter proposes a new method for solving such large-scale MAB problems. The method, called Cross-Entropy-based Multi-Armed Bandit (CEMAB), uses the Cross-Entropy method as a noisy optimizer to find the optimal arm with as lower costs as possible. Experimental results indicate that CEMAB outperforms state-of-the-art methods for solving MABs with a large number of arms.

3.1 Background

A fundamental question in sequential decision making is how to select the best action sequence even if the consequence of each action may not be exactly known. In its simplest form, this question can be studied as a multi-armed bandit (Robbins, 1952) problem. Under this framework, selecting an action from a given state is akin to selecting which slot machine to play from a number of such machines. The question becomes how to balance between playing machines that have been giving good rewards in the past and machines that have not been tried before, such that the total reward received is as close as possible to the total reward that would have been received if the player had always played the highest-paying machine.

Many methods have been proposed to solve the above problem of balancing exploration and exploitation, such as ε-greedy (Watkins, 1989), softmax (Sutton and Barto, 1998), and UCB (Auer et al., 2002a). In fact, many of such methods have become the foundation of today's reinforcement learning (Sutton and Barto, 1998). However, except for a few (Bubeck et al., 2011; Coquelin and Munos, 2007), most methods (Burtini et al., 2015) try and estimate the reward of each and every action, to ensure that the best action is not missed. Therefore, their effectiveness is limited to problems with a relatively small number of arms (e.g., fewer than 20). Unfortunately, this assumption is quickly becoming unrealistic in a growing number of applications. For instance, one can now choose from hundreds of drug cocktails —combinations of various types of drugs at various dosages— in personalized medicine, choose one of hundreds of different combinations of investment portfolios, and select a subset of tens of millions of possible combinations of data and sensors that can be used to analyze consumer preferences. As a result, most of today's methods for solving MABs (Burtini et al., 2015) are no longer effective for solving the more recent large-scale problems.

To alleviate the difficulty of solving MABs with a large number of discrete actions, we propose a novel method called Cross-Entropy-based Multi-Armed Bandit (CEMAB). The key to CEMAB is the use of the Cross-Entropy method (Rubinstein and Kroese, 2004) as a stochastic optimization method to identify the best action. By doing so, CEMAB can significantly reduce the number of actions to test before identifying the best action, assuming that the reward for pulling an arm is retrieved from an unknown fixed distribution. Preliminary results on standard

test cases for MAB indicate that the number of arms to pull before CEMAB identifies the (close to) optimal arms does not directly depend on the number of arms in the problem, which indicates that CEMAB is able to scale up well. This observation is supported by our simulation results, where tests on various MAB problems with up to 10,000 arms indicate that CEMAB outperforms state-of-the-art MAB solvers on large problems.

3.2 Related Algorithms

Various methods for solving MAB have been proposed. Those algorithms share the common structure as listed in Algorithm 3. The main difference depends on what kind of selection strategy is used (line 2).

Algorithm 3: Basic framework of MAB

Input: The number of arms $|\mathcal{K}|$, a (black box) function `reward()` to sample random rewards. Maximum number of plays T.

Output: Total reward G.

1 Set $\boldsymbol{\mu} \leftarrow \mathbf{0}_{1\times|\mathcal{K}|}$, $\texttt{visits} \leftarrow \mathbf{0}_{1\times|\mathcal{K}|}$.
2 **for** $t \leftarrow 1$ **to** T **do**
3 $k \leftarrow$ Choose an arm according to different strategy.
4 $r \leftarrow \texttt{reward}(k)$ `// Draw an immediate reward`
5 $G \leftarrow G + r$.
6 $\texttt{visits}(k) \leftarrow \texttt{visits}(k) + 1$.
7 $\boldsymbol{\mu}(k) \leftarrow \boldsymbol{\mu}(k) + (r - \boldsymbol{\mu}(k))/\texttt{visits}(k)$.
8 **return** G

A brief overview of the well-known methods is reviewed.

ε-greedy (Watkins, 1989). The ε-greedy method is perhaps the simplest way to solve the MAB problem. At each time step, the algorithm has a probability ε to select an arm uniformly at random (exploration) and a probability $1-\varepsilon$ to choose the arm with the highest estimated reward so far (exploitation). In general, this strategy does not converge to the optimal arm.

Softmax (Sutton and Barto, 1998). Softmax picks each arm with a probability according to its empirical performance. The probability of each arm in Softmax can be based on the

Boltzmann distribution $p_k = \mathrm{e}^{\hat{\mu}_k/\mathcal{T}} / \sum_{k=1}^{|\mathcal{K}|} \mathrm{e}^{\hat{\mu}_k/\mathcal{T}}$, where $\hat{\mu}_k$ is an estimate of the expected reward μ_k and $\mathcal{T}$ is the temperature. If $\mathcal{T}$ is very small, the arm with the highest estimated reward will have a large probability of being chosen (exploitation). In contrast, when $\mathcal{T}$ is very large, all $\{p_k\}$ are approximately equal, so that in this case Softmax is purely exploring.

Exp3 (Auer et al., 2002b). Exp3 (*exponential weight algorithm for exploration and exploitation*) is a variant of Softmax. The probability of choosing arm k is defined by $p_k = (1-\gamma)w_k / \sum_{j=1}^{|\mathcal{K}|} w_j + \gamma/|\mathcal{K}|$. The weights $\{w_j\}$ are updated after each step. In particular, after arm k is chosen (yielding reward r_k), the weight w_k is updated as $w_k \leftarrow w_k \exp^{\gamma r_k/p_k|\mathcal{K}|}$. It can be shown that the "weak regret", defined as $\left(T \max_{k\in\mathcal{K}} \mu_k - \sum_{t=1}^{T} r_t\right)$, is bounded under Exp3.

UCB (Auer et al., 2002a). The UCB is a family of algorithms for which optimal logarithmic regret can be achieved uniformly over time, assuming that all reward distributions have a bounded support (Auer et al., 2002a). The simplest member of this family is UCB1. It records the number of times that each arm has been played, $\texttt{visits}(k)$, and after each choice k updates its current estimate of μ_k via $\hat{\mu}_k \leftarrow \hat{\mu}_k + (r_k - \hat{\mu}_k)/\texttt{visits}(k)$. At the beginning, each arm is played once (full sweep). Subsequently, at each time t arm k is chosen that satisfies $k = \mathrm{argmax}_{k=1,\ldots,|\mathcal{K}|} \hat{\mu}_k + C\sqrt{\log t/\texttt{visits}(k)}$, where C is a parameter of explorations.

Thompson Sampling (TS) (Agrawal and Goyal, 2012). Thompson sampling is a Bayesian sampling algorithm based on (Thompson, 1933). For each arm k, the knowledge of the expected reward μ_k is described by a $\mathsf{Beta}(\alpha_k, \beta_k)$ distribution. At time t, random variables $\theta_k, k = 1, \ldots, |\mathcal{K}|$ are generated from each of these distributions. The index k^* corresponding to the largest of the $\{\theta_k\}$ is the arm to play. If r is the corresponding reward, a Bernoulli trial B with the probability r is generated. If $B = 1$, then α_{k^*} is increased by 1, otherwise β_{k^*} is increased by 1.

3.3 Cross-Entropy-based Multi-Armed Bandit (CEMAB)

3.3.1 The Method

The key idea of CEMAB is to search for the optimal arm, rather than optimizing a mapping from the past history to arm space. We transform the MAB problem into a simpler stochastic optimization problem, and then solve this simpler problem using the Cross-Entropy method. To this end, notice that, under the assumption that the reward distributions of the arms do not change over time, the objective is to find,

$$k^* = \underset{k \in \mathcal{K}}{\operatorname{argmax}}\, \mu_k. \tag{3.1}$$

Maximizing $\mu_k, k \in \mathcal{K}$ is simpler than solving the original MAB problem over all policies, or even over all action sequences. This is because the solutions of the simplified problem lie in a space of size $|\mathcal{K}|$, while the solutions of the original problem lie in a space of all possible policies. This difference in computational complexity becomes more pronounced as the number of arms increases. Therefore, to be effective in solving MABs with a large number of arms, CEMAB finds the best sequence of arms by searching the optimal arm to play.

Despite this simplification, the stochastic nature of MAB remains, as the expected reward μ_k of any arm $k \in \mathcal{K}$ is not known as a priori and can only be estimated by playing the arm. Therefore, to keep the total reward high, CEMAB strives to avoid using arms with low rewards as much as possible when searching for the best arm. To this end, CEMAB adopts CE for noisy optimization and modifies it to suit the nature of the MAB problem. It uses the quantile statistics in CE and carefully adapts it to degrade the probability of selecting the bad arms gracefully. In this way, it can find the best arm quickly without starving the arms that might be optimal. The CEMAB algorithm is presented in Algorithm 4.

To find the optimal arm k^*, CE starts by initializing the probability $\boldsymbol{p}$ of pulling a particular arm uniformly (Line 2). It iteratively chooses an arm (say $k \in \mathcal{K}$) to play, based on the probability $\boldsymbol{p}$, receives a reward r, which is drawn randomly from the unknown distribution D_k, and updates the estimated expected reward μ_k (Line 11). This sampling and estimation

Algorithm 4: CEMAB Method

Input: The number of arms $|\mathcal{K}|$, a (black box) function `reward()` to sample random rewards. CE parameters: sample size N, elite ratio $\rho \in (0, 1)$, elite sample size $N_e = \rho N$ and learning rate $\alpha \in (0, 1)$. Maximum number of plays T (for simplicity, we assume $T = MN$).

Output: Total reward G.

1 $\boldsymbol{\mu} \leftarrow \mathbf{0}_{1\times|\mathcal{K}|}$
2 $\boldsymbol{p} \leftarrow (1/|\mathcal{K}|)_{1\times|\mathcal{K}|}$.
3 $\texttt{visits} \leftarrow \mathbf{0}_{1\times|\mathcal{K}|}$
4 **for** $\tau \leftarrow 1$ **to** M **do**
5 $\quad A \leftarrow [\,]$ `// empty matrix`
6 $\quad$ **for** $i \leftarrow 1$ **to** N **do**
7 $\quad\quad$ Draw an arm k from the discrete distribution parameterized by $\boldsymbol{p}$.
8 $\quad\quad r \leftarrow \texttt{reward}(k)$ `// Draw an immediate reward`
9 $\quad\quad G \leftarrow G + r$.
10 $\quad\quad \texttt{visits}(k) \leftarrow \texttt{visits}(k) + 1$.
11 $\quad\quad \boldsymbol{\mu}(k) \leftarrow \boldsymbol{\mu}(k) + (r - \boldsymbol{\mu}(k))/\texttt{visits}(k)$.
12 $\quad\quad A \leftarrow [A; [k, \boldsymbol{\mu}(k)]]$. `// Append row` $[k, \boldsymbol{\mu}(k)]$ `to` A
13 $\quad \tilde{\boldsymbol{p}} \leftarrow \texttt{update}(|\mathcal{K}|, N_e, A)$
14 $\quad$ Using the learning rate α, update $\boldsymbol{p}$ as

$$\boldsymbol{p} \leftarrow (1 - \alpha)\,\boldsymbol{p} + \alpha\,\tilde{\boldsymbol{p}}. \tag{3.2}$$

15 **return** G

process repeats until one is confident that updating the selection probability $\boldsymbol{p}$ will benefit the optimization procedure. Once the probability is updated, the iterative sampling and estimation procedure repeat using the new selection probability.

Key to the performance of CEMAB is how it updates its selection probability (Lines 13–14). A straightforward application of CE for noisy optimization would estimate the expected reward of all of the arms, and only after all estimates are improved, the probability $\boldsymbol{p}$ is updated. However, in the MAB problem, an estimate of the expected reward of any arm can only be improved by playing an arm, and each play incurs a reward. Therefore, CEMAB updates the probability $\boldsymbol{p}$ in an asynchronous manner: It clusters a sequence of N samples of the arm into a single batch and updates the probability $\boldsymbol{p}$ after each batch ends. Note that at the end of each batch the estimated expected reward of some of the arms may not have improved at all. Therefore, a smoothing mechanism (Line 14) is needed, to avoid being overcommitted to the

new estimate of the different arms and also to guarantee that each arm has a non-zero probability of being visited.

Similar to most CE-based algorithms, CEMAB updates the probability $\boldsymbol{p}$ on the basis of the estimate $\boldsymbol{\mu}$ of the samples. The question is how the probability $\boldsymbol{p}$ should be updated based on the set of samples (Line 13). To this end, we propose two strategies: CEMAB-truncated and CEMAB-proportional. In CEMAB-truncated, we use the traditional CE updating formula, ignoring any arm that does not make it to the elite sample set. Specifically, the probability update rule for CEMAB-truncated is in Algorithm 5. In CEMAB-proportional, we assign the probability based on the estimated values $\boldsymbol{\mu}$ of each arm after the batch ends, and never set the probability of selecting an arm to be zero. The description of this update strategy is given in Algorithm 6.

Algorithm 5: `update`$(|\mathcal{K}|, N_e, A)$ for CEMAB-truncated

1 **for** $k \leftarrow 1$ **to** $|\mathcal{K}|$ **do**
2 Rearrange A by sorting its rows according to the second column, from largest to smallest.
3 $\tilde{\boldsymbol{p}}(k) = \frac{1}{N_e} \sum_{j=1}^{N_e} I_{\{A(j,1)=k\}}$
4 **return** $\tilde{\boldsymbol{p}}$

Algorithm 6: `update`$(|\mathcal{K}|, -, A)$ for CEMAB-proportional

1 **for** $k \leftarrow 1$ **to** $|\mathcal{K}|$ **do**
2 For each arm k sampled in A, get the latest estimate $\boldsymbol{\mu}(k)$.
3 $\tilde{\boldsymbol{p}}(k) = \frac{p_k \boldsymbol{\mu}(k)}{\sum_{j=1}^{K} p_j \boldsymbol{\mu}(j)}$.
4 **return** $\tilde{\boldsymbol{p}}$

3.3.2 Example: Ten-Arm MAB illustration

Example 3.3.1. Consider a ten-arm case with truncated Gaussian reward functions. The problem is generated by the following $\boldsymbol{\mu}, \boldsymbol{\sigma}$,

$$\begin{aligned} \boldsymbol{\mu} &= [0.5606, 0.2631, 0.4288, 0.3061, 0.8117, 0.0851, 0.7781, 0.7890, 0.3613, 0.8346], \\ \boldsymbol{\sigma} &= [0.2974, 0.2725, 0.2795, 0.1429, 0.1001, 0.1983, 0.1535, 0.0021, 0.1951, 0.0338]. \end{aligned}$$

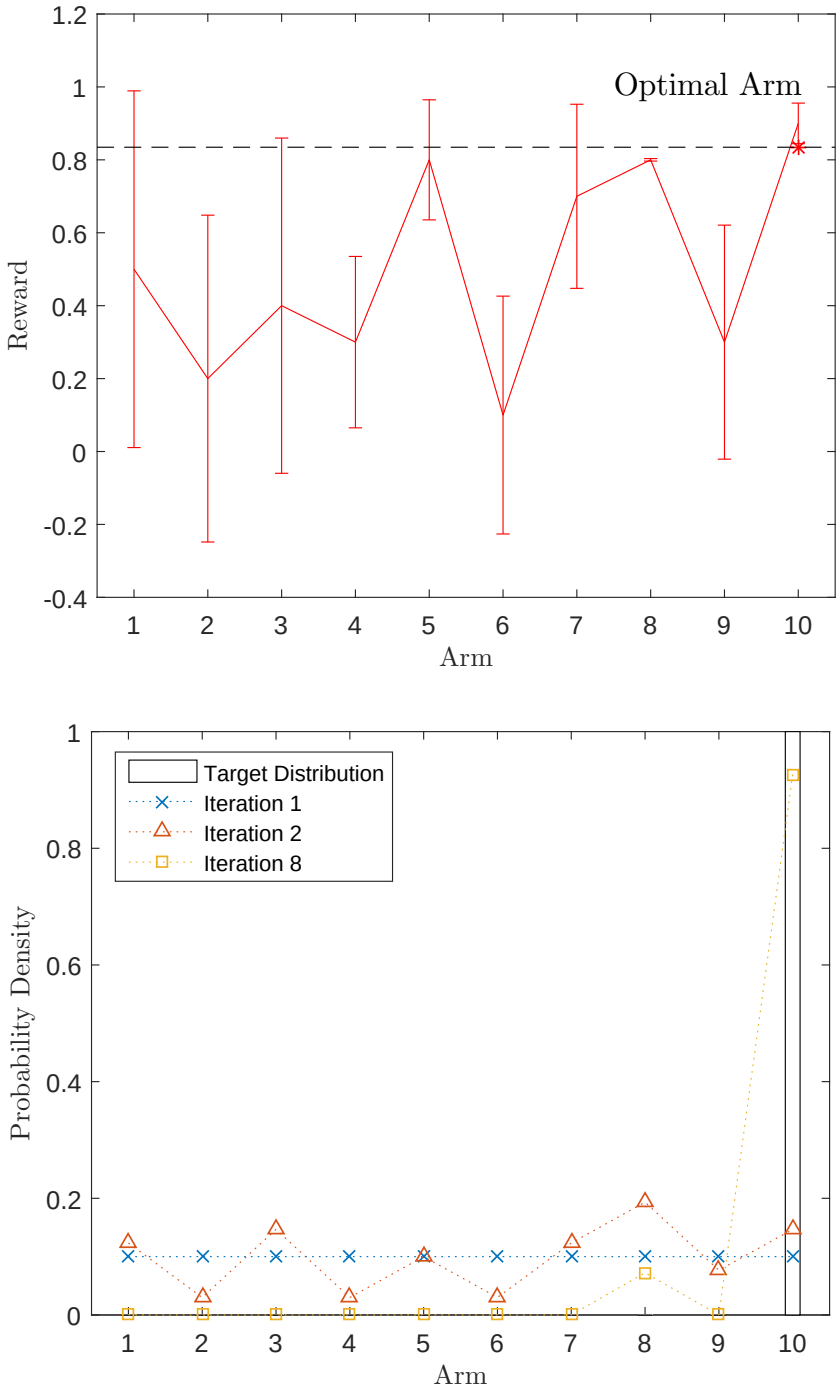


Figure 3.1: An example of ten-arm bandit with Gaussian reward function is shown in the top figure. Several sample iterations of running CEMAB are plotted in the bottom figure.

The Figure 3.1(top) shows this example. The optimal index of the arm is 10. Finding out the optimal arm is not easy, since arms of number 5,7,8 have suboptimal expectations and arms of number 1,3,5,7,8 have high variances overlapping the true optimal expectation.

Figure 3.1(bottom) illustrates some sample iterations of running CEMAB on this example. In CEMAB, finding out the index of the optimal arm is equivalent to assigning the probability of choosing this arm to be 1. Hence, identifying the optimal arm is converted to searching for the optimal distribution. At iteration 1, $\boldsymbol{p}_1$ is uniformly distributed over the set of arms. After a batch of sampling, we use top ρ-quantile of performing samples to update $\boldsymbol{p}_2$. The process repeats until the total number of plays is reached. Table 3.1 shows how the $\boldsymbol{p}$ updates. An interesting observation is the probability changes for the 5-th arm. In first five iterations, it seems that the arm is the optimal arm since it has the similar expectation to the optimal arm. However, as more samples are generated, the estimation gets more accurate. The probability of the truly optimal arm increases after more iterations.

Table 3.1: Evolution of $\boldsymbol{p}$ by CEMAB method

τ	$\boldsymbol{p}(1)$	$\boldsymbol{p}(2)$	$\boldsymbol{p}(3)$	$\boldsymbol{p}(4)$	$\boldsymbol{p}(5)$	$\boldsymbol{p}(6)$	$\boldsymbol{p}(7)$	$\boldsymbol{p}(8)$	$\boldsymbol{p}(9)$	$\boldsymbol{p}(10)$
1	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
2	0.1000	0.0300	0.1233	0.0300	0.2400	0.0300	0.1233	0.1700	0.0300	0.1233
3	0.0300	0.0090	0.0370	0.0090	0.3753	0.0090	0.0603	0.2610	0.0090	0.2003
4	0.0090	0.0027	0.0111	0.0027	0.4626	0.0027	0.0181	0.2416	0.0027	0.2468
5	0.0027	0.0008	0.0033	0.0008	0.5821	0.0008	0.0054	0.0725	0.0008	0.3307
6	0.0008	0.0002	0.0010	0.0002	0.4780	0.0002	0.0016	0.0217	0.0002	0.4959
7	0.0002	0.0001	0.0003	0.0001	0.2601	0.0001	0.0005	0.0065	0.0001	0.7321
8	0.0001	0.0000	0.0001	0.0000	0.0780	0.0000	0.0001	0.0020	0.0000	0.9196
9	0.0000	0.0000	0.0000	0.0000	0.0234	0.0000	0.0000	0.0006	0.0000	0.9759
10	0.0000	0.0000	0.0000	0.0000	0.0070	0.0000	0.0000	0.0002	0.0000	0.9928

3.3.3 Time Complexity and Convergence Properties

Similar to many state-of-the-art methods for solving MABs, such as Exp3 (Auer et al., 2002b) and UCB (Auer et al., 2002a), the most time-consuming part of CEMAB is its update step, i.e., Line 13 of Algorithm 4. For CEMAB-truncated, each update takes $O(N\log(N)+\max(|\mathcal{K}|, N_e))$, where the first component is due to sorting the samples within a batch (Line 2 of Algorithm 5). For CEMAB-proportional, each update will take $O(|\mathcal{K}|)$. Although Thompson sampling, Exp3 and UCB (current state-of-the-art methods) require $O(|\mathcal{K}|)$ for each update too, the number of updates for CEMAB is much less than for these two methods. Exp3 and UCB update their

probability for selecting an arm at each step, but CEMAB updates its probability for selecting the arms only once per batch, i.e., $M = T/N$ times for a total of T plays.

CEMAB-proportional is guaranteed to converge to the optimal expected reward, assuming that the cumulative distribution function of the reward of the optimal arm is strictly increasing. The proof is a straightforward application of the proof of the CE-proportional algorithm for noisy optimization (Goschin et al., 2011). We do not have a theoretical proof that CEMAB-truncated will converge to the optimal expected reward. However, under the aforementioned assumption on the cumulative distribution function, CEMAB-truncated converges to the quantile of the total reward function. This proof is a straightforward application of the proof of the commonly used CE algorithm for noisy optimization in (Goschin et al., 2013). CEMAB-truncated is more aggressive in its distribution update compared to CEMAB-proportional, and therefore we can expect that CEMAB-truncated tends to converge to a particular arm faster than CEMAB-proportional, which is good if the quantile function of the total reward is equivalent to the expected total reward.

3.4 Experimental Results

The goal of our experiments are two-fold: The first goal is to test the proposed methods against existing MAB methods on well-known benchmarks and understand the properties of the proposed methods better (Section 3.4.1). This also helps us in setting the parameters for tests on large MAB problems. The second and ultimate goal is to test the performance of our proposed methods on large MAB problems (Section 3.4.2).

We compare the empirical performance of ε-greedy (with 0 initialization), ε-greedy (play once), Softmax, Exp3, and UCB1, with our proposed CE-based methods on discrete (Bernoulli) and continuous (truncated Gaussian) reward distributions. Note that we use two types of ε-greedy: One initializes the estimate of the expected reward to zero (denoted as E1), while the other initializes the estimate of the expected reward based on the reward received when playing the arm once (denoted as E2). The reason for these two versions is that we found significant performance differences between ε-greedy with these two different initializations in large-scale problems.

3.4.1 Small-Scale MABs

Experimental Setup

We test our methods and comparators on ten small-scale MAB problems, with up to ten arms. Table 3.2 details the reward distributions of these problems.

Table 3.2: Bx refers to Bernoulli distributions and Gx to truncated Gaussian distributions.

	1	2	3	4	5	6	7	8	9	10
Mean of B1	0.9	0.6								
Mean of B2	0.9	0.8								
Mean of B3	0.55	0.45								
Mean of B4	0.9	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
Mean of B5	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6
Mean of B6	0.55	0.45	0.45	0.45	0.45	0.45	0.45	0.45	0.45	0.45
Mean of G1	0.3	0.6								
Std of G1	0.2	0.2								
Mean of G2	0.3	0.6								
Std of G2	0.6	0.2								
Mean of G3	0.5	0.2	0.4	0.3	0.8	0.1	0.7	0.8	0.3	0.9
Std of G3	0.3	0.2	0.3	0.1	0.1	0.2	0.5	0.4	0.2	0.1
Mean of G4	0.5	0.2	0.4	0.3	0.8	0.1	0.7	0.8	0.3	0.9
Std of G4	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

The first six problems (i.e., B1–B6) are MABs with discrete reward distributions, which is the benchmark used in (Auer et al., 2002a). The reward of each arm in each of these problems is sampled from a Bernoulli distribution, where the success probability corresponds to the probability of generating a reward of 1 and the failure probability corresponds to the probability of generating a reward of 0. B1–B3 specify MABs with two arms and B4–B6 define MABs with ten arms. Note that B3 and B6 are relatively "difficult", because the reward of the optimal arm has a higher variance and the gaps $\mu^* - \mu_k$, $k = 1, \ldots, 10$ are small.

The last four problems (i.e., G1–G4) are MABs with continuous reward distributions, in particular truncated normal distributions with support $[0, 1]$. Table 3.2 specifies the mean and

standard deviation of the Gaussian distribution of each arm in each MAB problem. In this set of problems, G2 and G4 are quite challenging. The standard deviations of the reward distributions in these two problems are large and the support of these distributions also overlap significantly, which makes it difficult to distinguish between the best arm and bad arms.

Parameters Selection

To set the parameters for comparison, we first run a set of preliminary tests for each algorithm on each problem in Table 3.2 with a wide range of parameters. The parameters are summarized in Table 3.3. For each algorithm, the best parameters are those that maximize the most problems across the 10 MAB problems described above.

Table 3.3: Parameter Range

Method	Parameters Tested	Best Parameter
CEMAB-truncated	$N \in \{50, 100\}, \rho \in \{0.3, 0.4, 0.5\}, \alpha \in \{0.7, 0.8, 0.9\}$	$N = 50, \rho = 0.5, \alpha = 0.8$
CEMAB-proportional	$N \in \{50, 100\}, \alpha \in \{0.7, 0.8, 0.9\}$	$N = 50, \alpha = 0.7$
ε-greedy (E1 and E2)	$\varepsilon \in \{0.4, 0.35, \ldots, 0.1, 0.05\}$	$\varepsilon = 0.1$ (E1), $\varepsilon = 0.05$ (E2)
Softmax	$t \in \{0.5, 0.45, ...0.1, 0.05\}$	$t = 0.1$
Exp3	$\gamma \in \{0.7, 0.65, ...0.15, 0.1\}$	$\gamma = 0.2$
UCB	$c \in \{3, 2.75, ...0.5, 0.25, 0.1, 0.05\}$	$C = 0.1$

Results

Figure 3.2 and Figure 3.3 present the performance of CEMAB and the comparator methods in ten small-scale testing cases. The general pattern we can get from the results is that CEMAB-truncated converges faster than CEMAB-proportional when the uncertainty is relatively small. While CEMAB-proportional has a better ability to handle noise in rewards since it will consider all data in the updating step rather than elite set only. Even though CEMAB does not always get the best performance in the small-scale cases, it is still able to achieve above-average performance. When the size of the arm becomes large in the next section, its advantage becomes more obvious.

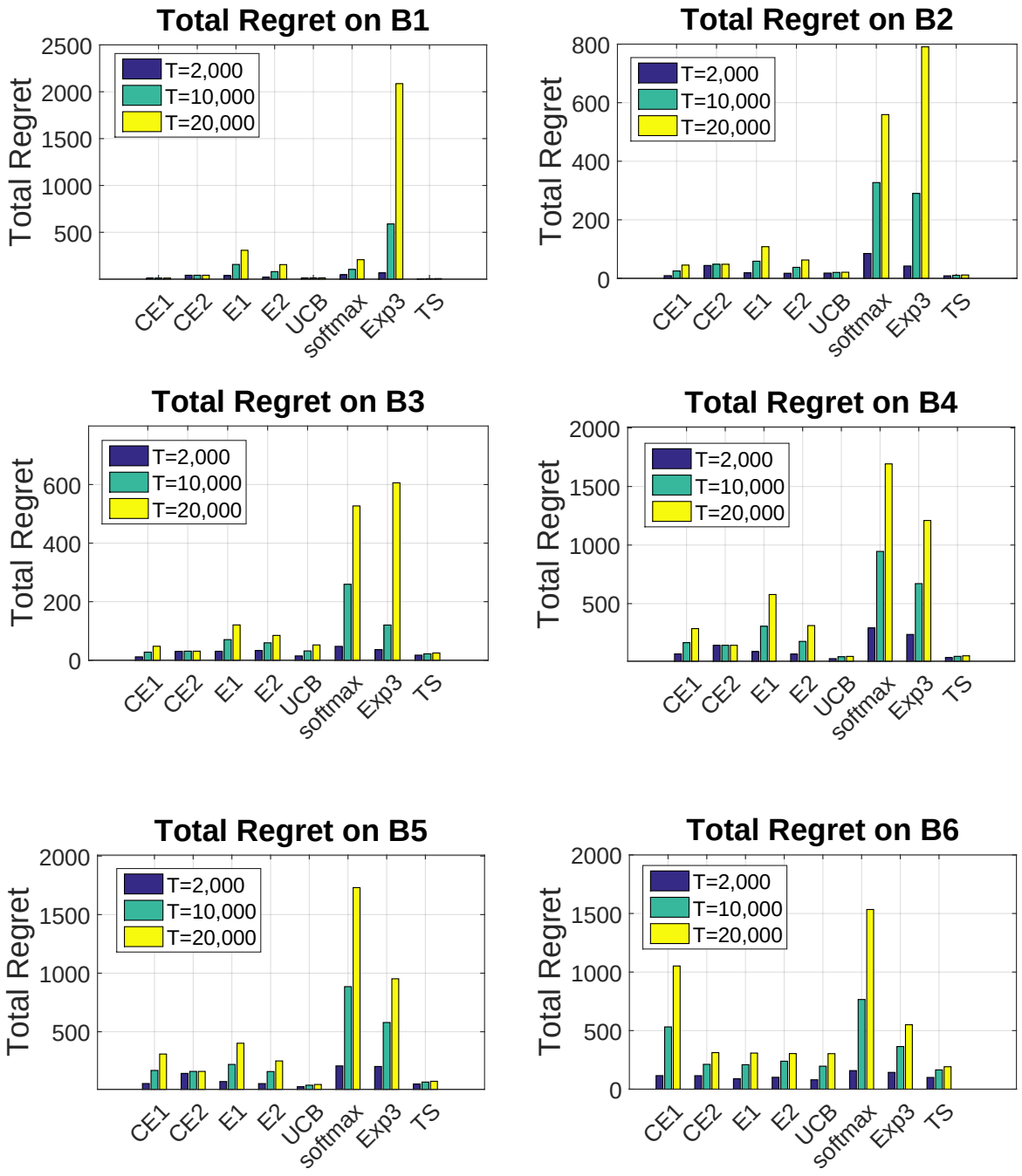


Figure 3.2: The average total regret on six Bernoulli reward problem sets. All algorithms use the best parameters and all experiments are repeated for 50 times.

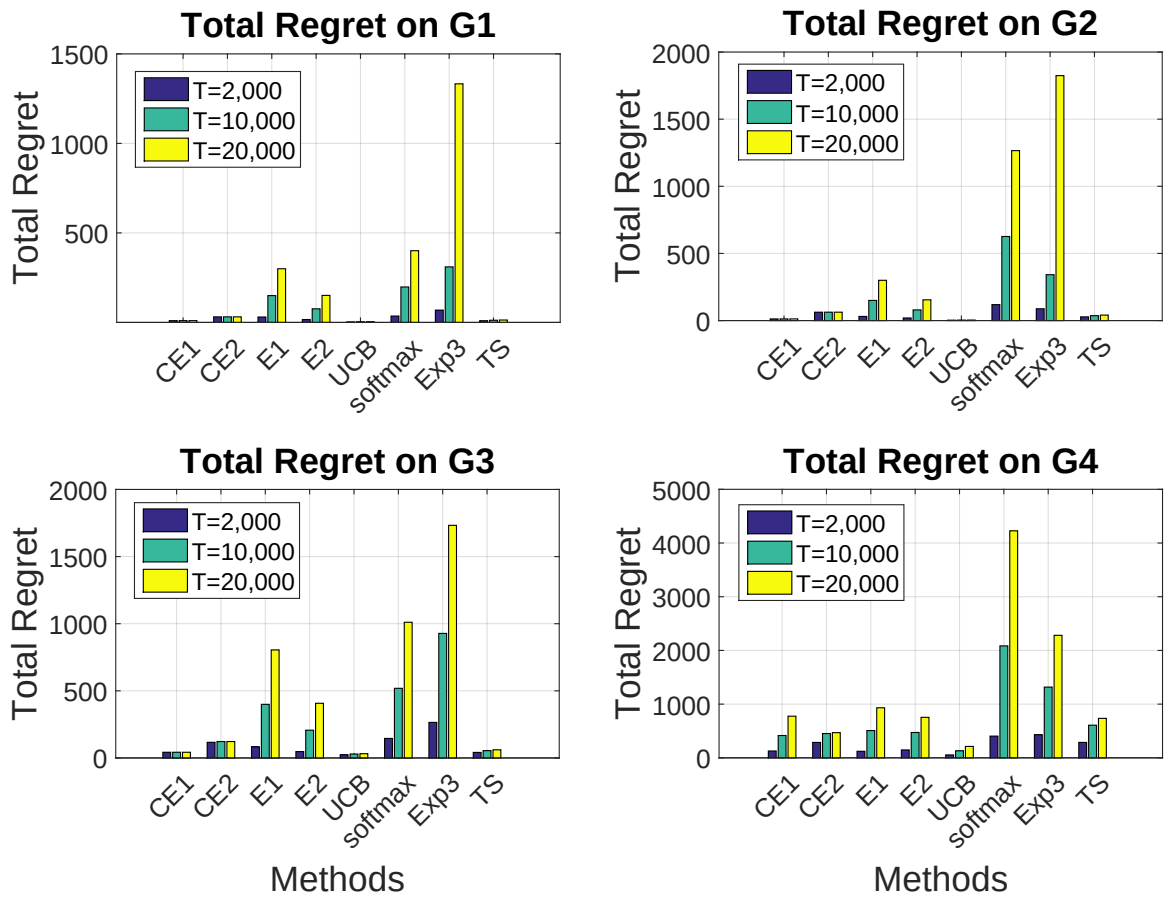


Figure 3.3: The average total regret on four Gaussian reward problem sets. All algorithms use the best parameters and all experiments are repeated for 50 times.

Among the ten small problems defined in Table 3.2, B5, B6, G3, and G4, which are the more difficult problems. In this set of problems, TS achieves the lowest total regret in B5 and B6, followed by UCB and both CEMABs. In G3 and G4, UCB takes the first place, while one of the CEMABs is the second. The reason is that in B6 the best arm (i.e., arm 1) does not show significant performance difference at the beginning, and it is quite easy for CE-truncated to underestimate this arm and set a probability zero during the updating step. Once this happens, CEMAB-truncated fails to identify the best arm, and as time progresses the difference in the total regret becomes more apparent. However, by avoiding this aggressive update, CEMAB-proportional perform well and is similar to UCB in G4. It is important to note that the best empirical parameter for UCB here is $C = 0.1$, rather than the default value $C = 2$.

Softmax and Exp3 have the worst performance. In Softmax, the use of the Boltzmann distribution is likely to exaggerate an arm with a "good" estimate. For Exp3, it is important to note that this method is designed for non-stochastic MAB problems. A particular arm is highly influenced by the current sample reward rather than the current estimate of the reward for each arm, which is a downside for stochastic problems

It is also interesting to note that the performance of the simplest algorithm, ε-greedy, differs significantly when applying different initializations. Variant E1 initializes the reward estimate of each arm with 0, while variant E2 initializes the reward estimate based on the reward received when playing the arm once. The performance of E2 is better in this set of small-scale problems. However, in the next section we will see that E1 is better for large-scale problems.

3.4.2 Large-scale MABs

Experimental Setup

To assess the performance of CEMAB for large problems, we test the algorithms on problems with an increasing number of arms. For each number of arms, we test the algorithms on four different MAB problems, as shown in Table 3.4, which consists of two LB (Large Bernoulli) that represent MABs whose reward distributions are Bernoulli distributed and two LG (Large Gaussian) that represent MABs whose reward distributions are truncated Gaussian with support $[0, 1]$.

Table 3.4: Large-scale MAB Settings

LB1	$\mu_k \sim \mathsf{U}(0, 1)$
LB2	10% of $\mu_k \sim \mathsf{U}(0.75, 1)$ and the rest of $\mu_k \sim \mathsf{U}(0, 0.25)$
LG1	$\mu_k \sim \mathsf{U}(0, 1), \sigma_k \sim \mathsf{U}(0, 0.25)$
LG2	10% of $\mu_k \sim \mathsf{U}(0.75, 1)$ and the rest of $\mu_k \sim \mathsf{U}(0, 0.25), \sigma_k \sim \mathsf{U}(0, 0.25)$

For LB1, the success probability (i.e., the probability of sampling a reward of 1) of each reward distribution is uniformly sampled from $(0, 1)$. For LB2, 10% of the arms have rewards drawn from a Bernoulli distribution whose success probability is sampled from $(0.75, 1)$ and 90% have rewards drawn from a Bernoulli distribution whose success probability is sampled

from $(0, 0.25)$. For LG1, the means are uniformly sampled from interval $(0, 1)$, while for LG2, 10% of the means are sampled from $(0.75, 1)$ uniformly at random and 90% are sampled from $(0, 0.25)$ uniformly at random. The standard deviations for both LG1 and LG2 are sampled uniformly at random from $(0, 0.25)$ for each arm. All of these parameters for the reward distributions are sampled independently for each arm. Apparently, LB2 and LG2 are harder than LB1 and LG1, since they require a strategy that has a good capability of exploring, rather than keep playing the best arm so far.

Results

For these tests, each algorithm uses the best parameters as found in Table 3.3. All algorithms use the best parameters and all experiments are repeated for 200 times. The best method is highlighted in boldface. If the difference between the best and second best method is not statistically significant (meaning that one method lies in the 95% confidence interval of the other), we highlight both of them. The results of these tests for $|\mathcal{K}| = 100, 1000,$ and 10000 are summarized in Table 3.5 and Table 3.6. The results indicate that CEMAB outperforms all other methods as the number of arms increases. The reason for the significantly decreasing performance of UCB is that each arm has to be played at least once to estimate the performance of each arm, so that it can converge to the optimal solution. However, exactly because of this, its performance becomes impractical as the number of arms increases. On the other hand, CEMAB incrementally improves its estimate on the performance of the arms based on sampling, without ever requiring to play the entire set of arms at first. This causes the convergence property of CEMAB to be weaker than UCB, but its empirical performance to be significantly better in large problems.

It is also interesting to note that the simple ε-greedy with zero initial estimate (E1) is a relatively strong competitor. In fact, for problems with a large number of arms, this simple methods is a stronger competitor than the state-of-the-art UCB. Note that for the Gaussian reward case, the gap between rewards is much less than for the Bernoulli case. As a result, even if an arm that is played is not very good, the reward obtained by playing a better arm will not be much higher. This could be a reason why the performance of E1 is comparable to CEMAB's in the LG1 and LG2, while it loses in the LB1 and LB2.

Table 3.5: The average total reward for large MABs with Bernoulli reward functions.

		LB1				LB2			
		The number of plays T				The number of plays T			
$\|\mathcal{K}\|$	Method	$1,000$	$5,000$	$10,000$	$20,000$	$1,000$	$5,000$	$10,000$	$20,000$
100	CE1	**893**	**4749**	**9569**	19207	718	4106	8342	16817
	CE2	806	4612	9463	19184	741	4192	8571	17341
	E1	864	4572	9230	18547	**820**	4139	8288	16597
	E2	868	4686	9460	19010	767	4167	8437	17022
	UCB	833	4674	9538	19225	787	**4311**	**8788**	**17796**
	softmax	859	4407	8866	17798	801	4127	8305	16686
	Exp3	580	3564	7702	16285	241	2610	6202	13550
	TS	856	4695	9557	**19323**	678	4168	8634	17615
$1,000$	CE1	**896**	**4788**	**9651**	**19380**	784	4493	9130	18402
	CE2	810	4655	9559	**19409**	792	**4538**	**9297**	18850
	E1	868	4510	9131	18496	771	4352	8885	18019
	E2	500	4129	8802	18532	197	3852	8635	18217
	UCB	499	3812	8358	17906	197	3730	8568	18361
	softmax	875	4470	8989	18043	**853**	4445	8976	18074
	Exp3	507	2666	5643	12378	202	1122	2601	7114
	TS	580	4027	8957	18886	305	3949	8925	**18897**
$10,000$	CE1	**895**	**4784**	**9644**	**19367**	795	4554	9251	18649
	CE2	809	4655	9554	**19393**	**800**	**4574**	**9358**	**18969**
	E1	863	4479	9025	18158	761	4322	8812	17818
	E2	515	2492	5013	14126	181	1004	2007	11022
	UCB	515	2486	5007	12135	182	1004	2007	9880
	softmax	832	4341	8786	17734	776	4264	8725	17734
	Exp3	500	2510	5040	10154	201	1006	2026	4109
	TS	511	2707	5811	13091	207	1191	3081	10433

For CEMAB, the performance in LB1 and LG1 seems to be better than that in LB2 and LG2. This is because the abundance of good arms in LB1 and LG1 is more than LB2 and LG2. It helps CEMAB to sample good arm easily.

Table 3.6: The average total reward for large MABs with Gaussian reward functions.

		LG1				LG2			
		The number of plays T				The number of plays T			
$\|\mathcal{K}\|$	Method	$1,000$	$5,000$	$10,000$	$20,000$	$1,000$	$5,000$	$10,000$	$20,000$
100	CE1	**885**	**4624**	**9297**	**18644**	772	4378	8884	17897
	CE2	800	4484	9150	18512	767	4435	9068	18341
	E1	868	4463	8973	17995	783	4411	8977	18109
	E2	877	4562	9177	18415	**879**	4685	9443	18956
	UCB	804	4450	9110	18506	863	**4777**	**9705**	**19578**
	softmax	833	4242	8513	17067	813	4282	8682	17551
	Exp3	559	3449	7457	15737	281	2752	6678	14878
	TS	785	4406	9071	18477	851	**4779**	**9706**	**19580**
$1,000$	CE1	**876**	**4599**	**9254**	18564	778	**4405**	8937	18004
	CE2	782	4441	9109	18499	763	4381	**8990**	18236
	E1	865	4532	9180	18542	768	4340	**8895**	18043
	E2	499	4358	9183	**18835**	238	4051	8818	**18351**
	UCB	500	3560	7750	16852	238	3440	8071	17711
	softmax	811	4125	8281	16609	**816**	4235	8535	17165
	Exp3	504	2624	5493	11865	242	1287	2831	7095
	TS	563	3713	8186	17729	312	3670	8378	18004
$10,000$	CE1	**877**	**4612**	**9281**	18618	**779**	**4394**	8913	17951
	CE2	788	4470	9153	18572	770	4369	**8934**	**18113**
	E1	876	4568	**9247**	**18656**	**774**	4313	**8815**	17894
	E2	488	2475	4997	14736	241	1207	2394	11942
	UCB	488	2477	4997	12390	240	1207	2393	9131
	softmax	794	4102	8275	16648	740	4030	8227	16690
	Exp3	501	2506	5027	10108	239	1200	2412	4865
	TS	507	2661	5637	12469	244	1339	3164	9783

3.5 Discussion

In this chapter, we propose a new approach, called CEMAB, for solving MABs with a large number of discrete arms. It uses the CE method as a noisy optimization method to search for the best arm with as little regret as possible. We present and evaluate the CEMAB algorithm with two variants for the updating procedure. Using results on CE for noisy optimization, one

of the variants is guaranteed to converge to the optimal arm, under certain conditions on the reward function. Empirical results on a number of MAB problems with an increasing number of arms indicate that CEMAB outperforms state-of-the-art methods.

However, there are still several shortcomings. Firstly, CEMAB-truncated can only guarantee converging to the quantile of the total reward function, rather than the expectation. Although CEMAB-proportional is able to converge to the expected total reward function, there is no theoretical proof for the convergence rate. Secondly, the number of parameters in CEMAB is more than the other methods, which makes finding the best performing parameters difficult. To alleviate these problems, it is worthy for future research.

The following publication has been incorporated as Chapter 4.

1. **Erli Wang**, Hanna Kurniawati, and Dirk P. Kroese, An On-Line Planner for POMDPs with Large Discrete Action Space: A Quantile-Based Approach, *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 273-277, 2018

Contributor	Statement of contribution	%
Erli Wang	Conception and design	50
	Implementation of numerical experiments	100
	Drafting and production	30
Hanna Kurniawati	Conception and design	35
	Drafting and production	40
Dirk P. Kroese	Conception and design	15
	Drafting and production	30

Chapter 4

QBASE: An On-line Planner for Large-scale POMDPs

A key difference between MAB and POMDP problems is the estimation part. Since a MAB only has one state, after sampling an arm, the sampled reward immediately improves the estimated expected reward of that arm. However, in POMDP, an action will move the agent to a new belief. Therefore the Q-value must be estimated by considering many possible sequences of future beliefs of the agent, which are affected by the future actions selected and observations perceived. This intertwined effect makes balancing the belief and the action sampling essential for solving POMDPs with large action spaces. To alleviate this difficulty, we propose an on-line approximate solver, called *Quantile-Based Action Selector (QBASE)*. Similar to most on-line solvers, QBASE uses sampling to construct a belief tree. However, it uses a quantile-based approach, derived from the Cross-Entropy method for optimization, to adaptively construct a small subset of the action space, so as to avoid full enumeration of the action space without sacrificing the quality of the generated decision strategies. Extensive numerical experiments on three different robotics tasks with up to 100,000 actions illustrate that QBASE can generate substantially better strategies than a leading method.

4.1 Background

Decision making under partial observability is critical to autonomous systems. Except in highly engineered scenarios, an autonomous agent must decide what to do "next" to get good long-term outcomes, despite not knowing its exact state due to uncertainty in the effects of actions, errors in sensors and perception.

The Partially Observable Markov Decision Process (POMDP) is a mathematically principled model for decision making under uncertainty; see, e.g., Sondik (1971). It quantifies this uncertainty using probability distribution functions, and computes the best action to perform with respect to distributions over the state space, called *beliefs*, rather than over a single state.

Unfortunately, finding the exact optimal decision strategy under a POMDP framework is so computationally expensive (Papadimitriou and Tsitsiklis, 1987b) that it has been deemed impractical and is often abandoned at the expense of reliability and robustness. Loosely speaking, four components have hindered the practicality of POMDP: long planning horizons and large state, action, and observation spaces. Fortunately, the past decade has seen significant advances in approximate POMDP solvers. As a consequence, we can now find good decision strategies for POMDPs with large state spaces (Pineau et al., 2003; Smith and Simmons, 2004, 2005; Porta et al., 2006; Shani et al., 2007; Kurniawati et al., 2008; Silver and Veness, 2010; Kurniawati and Yadav, 2013; Somani et al., 2013; Luo et al., 2016) and large observation spaces (Hoey and Poupart, 2005; Bai et al., 2014; Sunberg and Kochenderfer, 2018), as well as up to a hundred look-ahead planning steps (Kurniawati et al., 2011) in less than an hour. This has enabled POMDPs to start becoming practical for solving various problems (Hoey et al., 2010; Horowitz and Burdick, 2013; Young et al., 2013).

Despite these advances, finding good decision strategies for problems with large action spaces remains difficult. A POMDP solver must compute, for each belief, an action that maximizes the expected total return (possibly discounted). Most successful solvers today compute such a mapping only for a small set of sampled beliefs and estimate the expected total return via sampling too, thereby trading optimality with approximate optimality for faster speed and reduced memory requirements. However, finding the maximum action is more difficult, and

most solvers resort to enumerating all possible actions. When the action space is large, such enumeration is no longer feasible.

In this chapter, we present an on-line POMDP solver, called Quantile-Based Action Selector (QBASE), which alleviates the difficulty of finding good decision strategies for problems with large action spaces. It extends the Cross-Entropy method for optimization (Rubinstein and Kroese, 2004) to partially enumerate the action space, so as to avoid full enumeration of all possible actions without sacrificing the quality of the resulting strategy too much. QBASE adaptively constructs a small subset of the action space, based on quantile statistics of the current estimate of the expected total return of the actions and uniform sampling. This allows more computational resources to be allocated to obtain more refined estimates of the expected total return of actions that are likely to perform well, as well as ensuring that all actions will eventually be visited. The method has been tested on three problems with an increasing number of actions. It has shown significant improvements compared to a state-of-the-art method, POMCP (Silver and Veness, 2010), for problems with large action spaces.

4.2 Related Algorithms

The overview of the recent developments in approximate POMDP solvers has been discussed in Table 1.1. In this chapter, we mainly focus on large action spaces.

According to Bellman's equation (2.14), finding the best action from the current belief is affected by the estimation of $Q(b, a)$ and the strategy of optimizing the action over the action space $\mathcal{A}$. Solving the action selection is essentially a stochastic optimization problem. Since computing a good estimate of the Q-value function is costly, optimization methods that rely on gradients would be expensive to compute.

If restrictive conditions can be assumed on POMDP models, linearized solvers can be applied to reduce the computation complexity (Van Den Berg et al., 2011; Sun et al., 2015; Hoerger et al., 2016). As a result, it will help to scale up for solving problems with large action spaces.

In terms of the optimization problem, prior work (Seiler et al., 2015) has attempted to alleviate this problem via the simplest stochastic optimization method that does not require gradient information, i.e., generalized pattern search. It performs well for problems with up to 3-dimensional continuous action spaces. However, its convergence relies on a continuity property of the gradient of the value function, which is unlikely to be satisfied for general POMDP problems.

A method to solve MDP (the fully observable version of POMDP) with continuous action space has been proposed in Mansley et al. (2011). This method uses the hierarchical optimistic optimization tree, in which the action space is adaptively partitioned into regions, and actions are evaluated per region, with more promising actions evaluated first. This approach assumes that nearby actions are likely to generate similar results, which is generally correct for continuous action spaces but not for discrete action spaces.

In this chapter, we aim to relax the requirement of gradient by using a quantile-based approach, which is adapted from the Cross-Entropy method (Rubinstein and Kroese, 2004). Recent advances in solving multi-armed bandit provide a suitable tool to break such assumption (Wang et al., 2017; Chaudhuri and Kalyanakrishnan, 2017). The crucial observation is that CEMAB (Wang et al., 2017), a quantile based sampling approach, significantly outperforms established methods when the size of space is large, given a fixed budget. It empirically indicates such sample-based method is capable of scaling up well.

4.3 Quantile-Based Action Selector (QBASE)

4.3.1 Overview

QBASE is an on-line POMDP solver. At each step it aims to compute the best action to perform from the current belief b for a fixed time limit, executes this action, and updates its belief to $b' = \tau(b, a, o)$, where $a \in \mathcal{A}$ is the computed best action and $o \in \mathcal{O}$ is the observation the agent perceives right after performing a from b. The process then repeats from the new belief b', until a termination condition is met.

To find the best action to perform from a belief, QBASE constructs a *belief tree*, denoted as $\mathcal{T}$. A belief tree is a tree where the nodes are sampled beliefs. For compactness, we denote the nodes of $\mathcal{T}$ and the beliefs they represent in the same way. An edge labeled (a, o) from a belief b to a belief b' in $\mathcal{T}$ means there is an action $a \in \mathcal{A}$ and an observation $o \in \mathcal{O}$ such that $b' = \tau(b, a, o)$. QBASE represents each sampled belief as a set of particles (states) and estimates the value of each sampled belief via Monte Carlo backup. The best action is then the action that induces the best estimated value.

Algorithm 7: QBASE

Input: POMDP model $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$ or a corresponding black-box simulator $\mathcal{P}$, parameters of QBASE $\langle \varrho, N_s, M_b, \beta \rangle$ (described in subsection "Sampling Action")

Initialize $\mathcal{T}$ with b_0 as the root node
INITINODE(b_0)
while *running* **do**
  while *there is still time for planning* **do**
    GROWTREE ($\mathcal{T}$, $\mathcal{P}$, $\langle \varrho, N_s, M_b, \beta \rangle$) `// build tree` $\mathcal{T}$
  Perform action a, such $a = \arg\max_{a \in \mathcal{A}} b.\mathbf{P}(a)$
  $o =$ get observation
  $b = \tau(b, a, o)$ `// update belief`

Algorithm 8: INITINODE(b)

for $a \in \mathcal{P}.\mathcal{A}$ **do** $b.N(a) = 0$; $b.\mathbf{P}(a) = \frac{1}{|\mathcal{P}.\mathcal{A}|}$
$b.A_s =$sample N_s actions uniformly at random from A
for $a \in b.\mathcal{A}_s$ **do** $b.\mathbf{P}_s(a) = \frac{b.\mathbf{P}(a)}{\sum_{a \in b.\mathcal{A}_s} b.\mathbf{P}(a)}$

Algorithm 7 presents the overall algorithm. The next subsection presents how the belief tree is constructed, while the subsequent subsections detail the key to QBASE's capability in solving problems with large action spaces, which is the action sampling strategy for constructing $\mathcal{T}$.

4.3.2 Expanding the Belief Tree

To construct the belief tree $\mathcal{T}$, QBASE follows a strategy similar to POMCP (Silver and Veness, 2010) and ABT (Kurniawati and Yadav, 2013). It initializes the tree by selecting the initial belief (or setting the current belief) b_0 as the root of $\mathcal{T}$. The tree is then iteratively grown (expanded) by

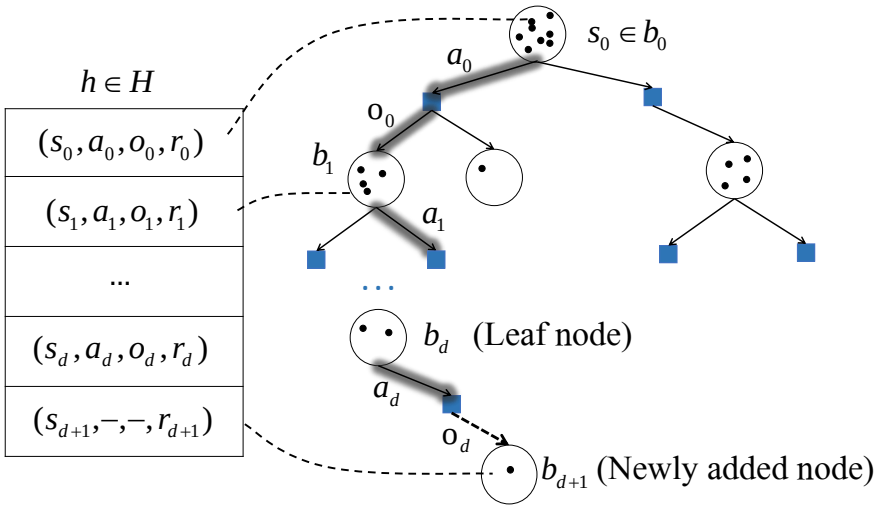


Figure 4.1: Correspondence between a history and a path in $\mathcal{T}$

simulating histories that start from the root of the tree. A *history* is a sequence of ⟨state, action, observation, reward⟩ 4-tuples. To explain the growing of the tree, suppose that the "current" tree is $\mathcal{T}$. Generate the first tuple of a new history h by first sampling a state $s_0 \sim b_0$, then an action $a_0 \in \mathcal{A}$ (detailed in the next subsection), followed by a new state $s_1 \sim T(s_0, a_0, S)$, and finally an observation $o_0 \sim Z(s_1, a_0, O)$. The tuple $\langle s_0, a_0, o_0, r_0 \rangle$ is associated with the root of the tree, b_0. The process is repeated for the next state s_1 to generate the second entry $\langle s_1, a_1, o_1, r_1 \rangle$ in h. This second tuple is then associated with the belief node b_1, where the edge from b_0 to b_1 is labeled by (a_0, o_0). And the state s_1 is added to the set of particles that represent b_1. This process keeps repeating until the associated belief node is a leaf node of the current tree. Suppose $\langle s_d, a_d, o_d, r_d \rangle$ is associated with leaf node b_d in $\mathcal{T}$. Then, at this point, a new tuple $\langle s_{d+1}, -, -, r_{d+1} \rangle$ is added as the last element of h, and a new belief node, denoted as b_{d+1}, is constructed, added as a child of b_d via an edge labeled (a_d, o_d), and associated with this last tuple of h. Figure 4.1 illustrates the association between a history and a path in $\mathcal{T}$. The process of sampling a history from the root node is repeated until the time for planning is over. The set of all sampled histories is denoted as H.

QBASE estimates the Q-value $Q(b, a)$ as

Algorithm 9: GROWTREE($\mathcal{T}$, $\mathcal{P}$, $\langle \varrho, N_s, M_b, \beta \rangle$)

Set $d = 0$, $h = \emptyset$, $b =$ root of $\mathcal{T}$, $I_{rollout} =$ **false**
$s \sim b$
while $\gamma^d > \varepsilon$ *and* $I_{rollout} ==$ ***false*** **do**
    $a =$ SAMPLEACT(b, $\langle \varrho, N_s, M_b, \beta \rangle$)
    $(s', o, r) =$ SIMULATOR($\mathcal{P}, s, a$)
    Append $\langle s, a, o, r \rangle$ to h
    Associate $\langle s, a, o, r \rangle$ with b
    $b.particles = b.particles \cup \{s\}$
    $b.\mathcal{A}_v = b.\mathcal{A}_v \cup \{a\}$
    $s = s'$; d++ ; $b.N(a)$++ ; $b.N$++ ; $b' = \tau(b, a, o)$
    if b' *is not in* $\mathcal{T}$ **then**
        add b' as a child of b and set $I_{rollout} =$ **true**
    $b = b'$
if $\gamma^d > \varepsilon$ **then**
    $E_h = 0$
else
    INITINODE(b) ; $b.particles = \{s\}$
    $E_h =$ROLLOUT-POLICY(s) `// rollout L steps`
Append $\langle s, -, -, E_h \rangle$ to h
Associate $\langle s, -, -, E_h \rangle$ with b
UPDATEVALUES($\mathcal{T}$,h,E_h) `// Back propagate to root`

$$\widehat{Q}(b, a) = \frac{1}{\left|H_{(b,a)}\right|} \sum_{h \in H_{(b,a)}} V(h, l). \tag{4.1}$$

Here, $H_{(b,a)} \subseteq H$ is the set of histories that correspond to paths in $\mathcal{T}$ that starts from the root node, pass through node b and then follow action a. Also, $l = l(b)$ denotes the depth level of node b in $\mathcal{T}$. The function $V(h, l)$ is the value of a history h starting from the l^{th} element, and is computed as $\left(\sum_{i=l}^{|h|} \gamma^{i-l} R(h_i.s, h_i.a)\right) + E_h$, where γ is the discount factor and R is the reward function. E_h is an estimate of the value of the last state in h, and is computed as the total discounted reward of a random walk for a pre-defined number of steps (denoted as L) from this last state. Algorithm 9 presents the tree expansion strategy.

4.3.3 Sampling Actions

Key to QBASE is the way it samples actions when generating histories to construct the belief tree $\mathcal{T}$. Given a belief node to expand, most successful POMDP solvers today enumerate the entire action space $\mathcal{A}$ to estimate the Q-values of these actions, and then choose the action with the highest utility — generally, computed as a combination of the estimated value and an exploration component (Silver and Veness, 2010; Kurniawati and Yadav, 2013) or an upper bound of the Q-value (Somani et al., 2013). In contrast, for each node, QBASE adaptively constructs a probability distribution function over $\mathcal{A}$ and uses this distribution to sample actions, so as to avoid full enumeration of the action space.

The question is, what distribution should QBASE use to sample from $\mathcal{A}$? Ideally, we want to assign high probability mass to good actions and zero mass to bad actions. Such a distribution allocates enough computational resources to estimate the Q-values of good actions to the level that the best action can be identified quickly, without wasting resources estimating the Q-values of bad actions. Of course, which actions are good and which are bad are a priori unknown, as otherwise the problem would have been solved. Therefore, QBASE aims to adaptively construct a distribution proportional to the Q-value function, interleaving estimating Q-values of sampled actions with adaptively improving the distribution in a Cross-Entropy method fashion.

For each node b of $\mathcal{T}$, QBASE maintains and updates the distribution (denoted as $b.\mathbf{P}$) over $\mathcal{A}$ in a batch manner. At the beginning of the first batch, the probability is uniform over $\mathcal{A}$. Within each batch, QBASE is given a pre-specified budget (denoted as M_b) to sample actions and update their Q-values. Algorithm 10 presents the algorithm. Let $b.N$ be the number of simulations for belief b, which is computed as $b.N = \sum_{a \in \mathcal{A}} b.N(a)$. At the end of each batch (line 1), QBASE is going to conduct two-stage update.

To increase the chances that each batch improves the Q-values estimates of good actions without starving any of the others, QBASE constructs a subset $\mathcal{A}_s \subset \mathcal{A}$ at the start of the batch. This set has a pre-defined size (denoted as N_s) and consists of two components: The top ϱ-quantile of $b.\mathbf{P}$ and $N_s - \varrho|\mathcal{A}|$ exploration components, sampled uniformly without repetition from action space A without the exploitation components (line 2-6). Actions for generating histories, and hence expanding the belief tree, are then sampled from the subset $\mathcal{A}_s$ rather than

Algorithm 10: SAMPLEACT(b, $\langle \varrho, N_s, M_b, \beta \rangle$)

1 **if** $b.N > 0$ *and* $(b.N \bmod M_b) == 0$ **then**
2 $\quad \mathcal{E} =$ the top $\lfloor \varrho \cdot |\mathcal{A}| \rfloor$ elements of sorted $b.\widehat{Q}$
3 $\quad b.\mathcal{A}_s = \mathcal{E}$
4 $\quad$ **while** $|b.\mathcal{A}_s| < N_s$ **do**
5 $\quad\quad a \sim \mathsf{U}(\mathcal{A} \backslash \mathcal{E})$
6 $\quad\quad$ Set $b.\mathcal{A}_s = b.\mathcal{A}_s \bigcup \{a\}$
7 $\quad m = \min_{a \in b.\mathcal{A}_v} b.\widehat{Q}(a)$
8 $\quad M = \max_{a \in b.\mathcal{A}_v} b.\widehat{Q}(a)$
9 $\quad$ **for** $a \in b.\mathcal{A}_v$ **do**
10 $\quad\quad \mathbf{W}(a) = \boldsymbol{\alpha}_b(a) \frac{b.\widehat{Q}(a) - m}{M - m}$
11 $\quad$ **for** $a \in b.\mathcal{A}_v$ **do**
12 $\quad\quad b.\mathbf{P}(a) = \frac{|b.\mathcal{A}_v|}{|\mathcal{A}|} \frac{\mathbf{W}(a)}{\sum_{a \in b.\mathcal{A}_v} \mathbf{W}(a)}$
13 $\quad$ **for** $a \in b.\mathcal{A}_s$ **do**
14 $\quad\quad b.\mathbf{P}_s(a) = \frac{b.\mathbf{P}(a)}{\sum_{a \in b.\mathcal{A}_s} b.\mathbf{P}(a)}$
15 $a \sim b.\mathbf{P}_s$
16 **return** a

from the entire $\mathcal{A}$. The size N_s is set *a priori* to be much smaller than $\mathcal{A}$, but slightly larger than $\lfloor \varrho \cdot |\mathcal{A}| \rfloor$ to allow exploration. The distribution $b.\mathbf{P_s}$ to sample an action from $\mathcal{A}_s$ is proportional to $b.\mathbf{P}$ for $a \in \mathcal{A}_s$.

The distribution $b.\mathbf{P}$ is updated based on a slight modification of the *proportional Cross-Entropy* (pCE) update (Goschin et al., 2013):

$$b.\mathbf{P}(a) \;\propto\; \boldsymbol{\alpha}_b \, \frac{\widehat{Q}(b,a) - m}{M - m}. \tag{4.2}$$

The smoothing parameter $\boldsymbol{\alpha}_b(a)$ quantifies how much QBASE trusts the new estimate compared to the past probability of sampling a. This increases with the number of visits according to $\boldsymbol{\alpha}_b(a) = b.N(a)/(b.N(a) + \beta)$, where $b.N(a)$ is the number of times the pair (b, a) has been visited and β is a constant parameter. The quantities m and M refer to the minimum and maximum of the estimated Q-values up to this batch, i.e., $\min_{a \in b.\mathcal{A}_v} b.\widehat{Q}(a)$ and $\max_{a \in b.\mathcal{A}_v} b.\widehat{Q}(a)$, where $\mathcal{A}_v$ is the visited action space (line 7-14). Then actions are sampled from the latest prob-

ability function $b.\mathbf{P}_s$ (line 15). Figure 4.2 provides sample simulations to illustrate the concept of two-stage sampling in QBASE.

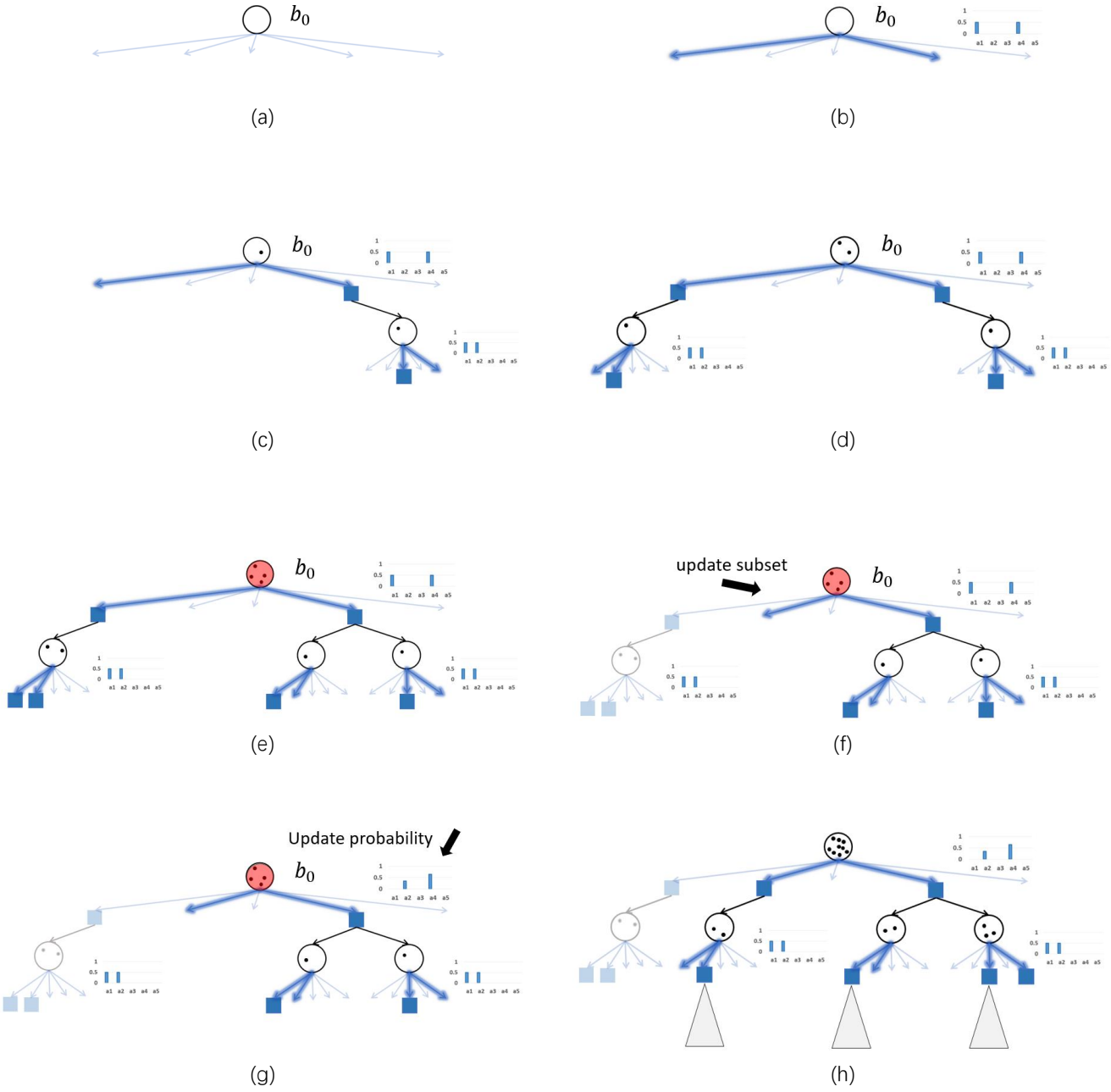


Figure 4.2: A batch simulation of QBASE

4.4 Experimental Results

To assess the applicability of QBASE for solving problems with large action spaces, we test the method on a total of 12 scenarios, classified into 3 different types of robotic tasks. We compare QBASE with the state-of-the-art online POMDP solver, POMCP (Silver and Veness, 2010). For a fair comparison, we ran POMCP using the authors' code[1] and implemented our code in the POMCP code framework using C++. All experiments are run as a single thread process on an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz with 128GB RAM.

To set the parameters of QBASE and POMCP, we first carry out a set of pilot runs for each solver to determine the best one. Specifically, in QBASE, the largest subset N_s is determined by $\min(0.5\,|\mathcal{A}|, 100)$, ϱ is selected from $\{0.3, 0.5, 0.7\}$, M_b from $\{1, 2\}$, and β from $\{5, 10, 20\}$. For POMCP, the exploration constant C is selected from $\{0.1, 1, 10, 100, 1,000, 10,000\}$.

Other parameters of the algorithm are set independently from problem domains. As the scale of the test problems is large, we set the discount factor $\gamma = 0.98$ and the tolerance of the approximate Q-value to $\varepsilon = 0.01$, in order to obtain a relative long planning horizon. As a result, the effective horizon is about $D = \log(\varepsilon)/\log(\gamma) \approx 228$. For a fair comparison, both solvers use the same rollout policy.

Table 4.1 presents the average expected discounted total reward with 95% confidence interval of the $1,000$ simulation runs for each scenario and method. Overall, QBASE outperforms POMCP in all test scenarios. Furthermore, in general, except for ***RockSample***, the gap between QBASE and POMCP increases, as the size of the problem increases. In ***RockSample***, the action spaces are relatively small, that the extra computation of constructing a subset via the quantile-based method that QBASE performs becomes an unnecessary overhead.

In the rest of current section, we will explain each scenarios with necessary details and additional results (e.g., how the planning time per step affects performance).

[1] http://www0.cs.ucl.ac.uk/staff/d.silver/web/Applications.html

Table 4.1: Simulation Results

Scenario	Time/Step (sec)	Method	Reward
Navigation(2, 30)			
$\|\mathcal{S}\| \approx 9 \times 10^2$,	1	POMCP	732.0 ± 5.0
$\|\mathcal{A}\| = 49$, $\|\mathcal{O}\| = 16$		QBASE	741.7 ± 4.7
Navigation(3, 30)			
$\|\mathcal{S}\| \approx 2 \times 10^4$,	2	POMCP	560.5 ± 7.6
$\|\mathcal{A}\| = 343$, $\|\mathcal{O}\| = 64$		QBASE	632.5 ± 8.6
Navigation(4, 30)			
$\|\mathcal{S}\| \approx 8 \times 10^5$,	5	POMCP	-8.2 ± 4.9
$\|\mathcal{A}\| = 2,401$, $\|\mathcal{O}\| = 256$		QBASE	91.3 ± 10.9
RockSample(7, 8)			
$\|\mathcal{S}\| = 12,544$,	1	POMCP	17.7 ± 0.4
$\|\mathcal{A}\| = 13$, $\|\mathcal{O}\| = 3$		QBASE	18.8 ± 0.4
RockSample(20, 50)			
$\|\mathcal{S}\| \approx 4 \times 10^{17}$,	2	POMCP	17.6 ± 0.6
$\|\mathcal{A}\| = 55$, $\|\mathcal{O}\| = 3$		QBASE	19.6 ± 0.7
RockSample(20, 100)			
$\|\mathcal{S}\| \approx 4 \times 10^{32}$,	5	POMCP	14.2 ± 0.8
$\|\mathcal{A}\| = 105$, $\|\mathcal{O}\| = 3$		QBASE	15.1 ± 0.9
Hunting-smart(11, 2, 2)			
$\|\mathcal{S}\| \approx 10^8$,	1	POMCP	-72.0 ± 5.7
$\|\mathcal{A}\| = 100$, $\|\mathcal{O}\| = 4$		QBASE	-69.7 ± 4.3
Hunting-smart(11, 3, 3)			
$\|\mathcal{S}\| \approx 10^{12}$,	5	POMCP	-179.1 ± 7.0
$\|\mathcal{A}\| = 1,000$, $\|\mathcal{O}\| = 8$		QBASE	-94.3 ± 5.7
Hunting-normal(11, 2, 2)			
$\|\mathcal{S}\| \approx 10^8$,	1	POMCP	42.4 ± 2.9
$\|\mathcal{A}\| = 100$, $\|\mathcal{O}\| = 4$		QBASE	42.3 ± 3.9
Hunting-normal(11, 3, 3)			
$\|\mathcal{S}\| \approx 10^{12}$,	5	POMCP	26.1 ± 8.6
$\|\mathcal{A}\| = 1,000$, $\|\mathcal{O}\| = 8$		QBASE	96.1 ± 7.3
Hunting-normal(11, 4, 4)			
$\|\mathcal{S}\| \approx 10^{16}$,	10	POMCP	$-1,572.7 \pm 33.7$
$\|\mathcal{A}\| = 10,000$, $\|\mathcal{O}\| = 16$		QBASE	67.7 ± 7.1
Hunting-normal(11, 5, 5)			
$\|\mathcal{S}\| \approx 10^{20}$,	60	POMCP	$-2,247.8 \pm 32.7$
$\|\mathcal{A}\| = 100,000$, $\|\mathcal{O}\| = 32$		QBASE	28.6 ± 13.6

4.4.1 RockSample(n, k)

Definition

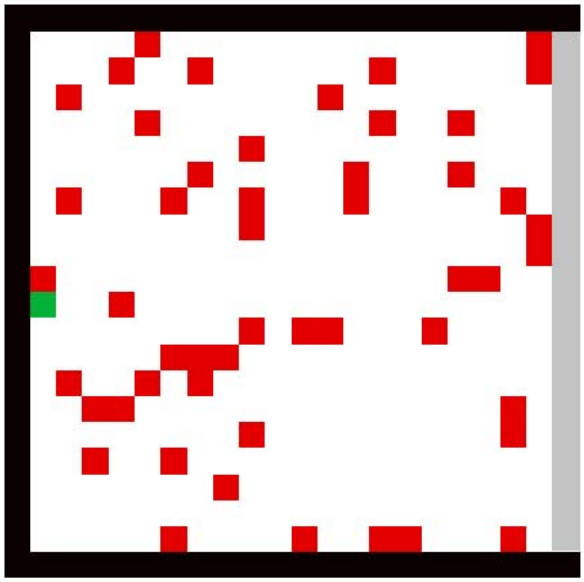


Figure 4.3: Illustration of ***RockSample***$(20, 50)$

.

Figure 4.3 illustrates the scenarios for $(20, 50)$. Rocksample (Smith and Simmons, 2004) is a well-known benchmark for POMDP solvers. A robot must explore an environment of size $n \times n$, populated with k rocks (marked as red squares). The position of the rocks are known exactly, but whether a rock is good or bad is unknown. In fact, at the beginning, each rock has a 0.5 chance of being good or bad. The goal of the robot is to sample as many good rocks as possible as fast as possible. The state space is the Cartesian product of the robot's position and the quality of the rocks, forming a state space of size $n^2 \cdot 2^k$. The robot can move to its North, South, East, and West cell, sample a rock at its current location, or remotely check rock $i = 1, \ldots, k$ to gain more information on whether it is good or bad. Its motion is perfect and the robot's position is fully observed. Checking a rock means that the robot applies a scanner to identify if the rock is good or bad. The reliability of the signal received decreases exponentially with the distance between the robot and the rock. The robot receives a reward of $+10$ if it samples a good rock or if it exits the environment (entering the grey region in the

figure). Sampling a bad rock incurs a penalty of -10. We tested our method on rock sample $(7, 8)$, $(20, 50)$, and $(20, 100)$, increasing the action space from 13 to 105.

Comparison with Different Planning Time

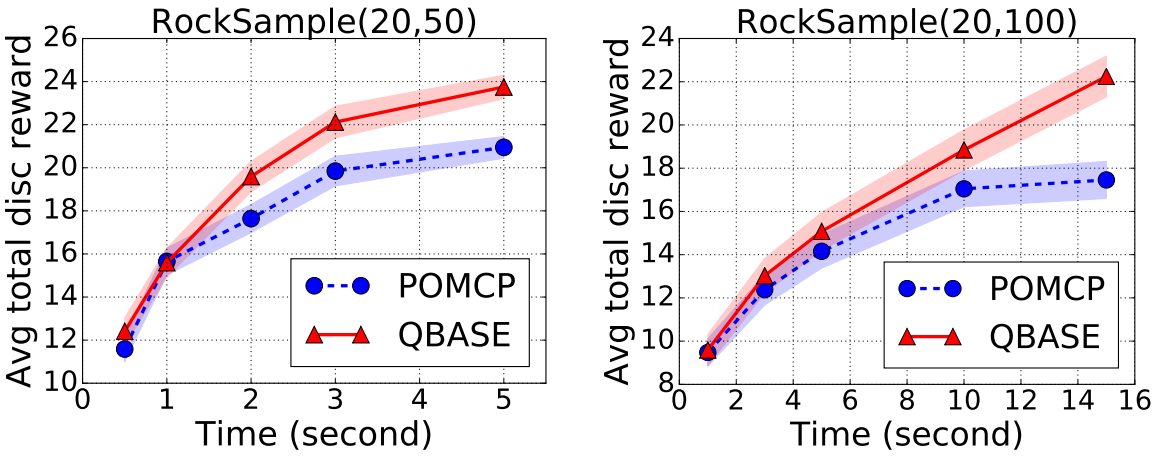


Figure 4.4: Performance with different planning times per step in RockSample

Figure 4.4 shows the trend of performance with increasing planning budgets. The results indicate that, in general, for a small planning time, both POMCP and QBASE perform equally but, as we allow additional planning time, QBASE can signicantly improve on POMCP. The reason is that, when the planning time is limited, both POMCP and QBASE do not have enough time to compute good Q-value estimates and can only build relatively shallow belief trees, causing both to perform equally poorly. However, when more time is allowed and deeper trees can be built, POMCP still needs to sweep the entire action space every time a node is added to the belief tree, which reduces the time it can spend on evaluating good actions. In contrast, QBASE evaluates only a small subset of the action space, guided by quantile-statistics, and can identify faster which actions are more promising and, as a result, can spend more resources on evaluating these actions.

4.4.2 Navigation(d, n)

Definition

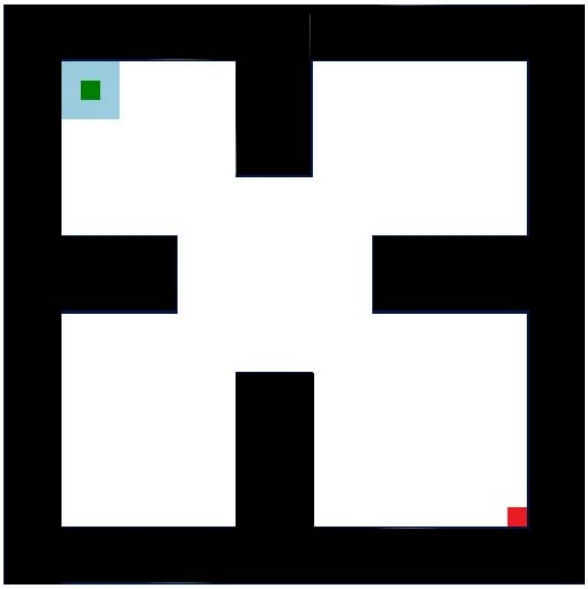


Figure 4.5: Illustration of ***Navigation***$(2, 30)$

.

Figure 4.5 illustrates the navigation scenario $(2, 30)$. An agent must navigate to a goal location (marked as red square) in a d-dimensional grid world populated by obstacles (marked as black regions), where each dimension is discretized into n cells. The agent initial position is not exactly known (the true position is the green square), but it must be at one of the 3^d cells on the upper left (marked by light blue squares). The agent can move to adjacent cells, within 3 cells away from its current position, resulting in 7^d possible actions. Its motion is accurate 90% of the time. The rest of the probability mass is divided equally among the $7^d - 1$ remaining cells. The agent can only observe the existence and position of walls surrounding its current cell, forming 2^{2d} observations. The observation function is perfect, though it is not sufficient to make the state to be fully observable. The agent receives a $+1,000$ reward for reaching the terminal states and incurs a -1 penalty for every movement. In ***Navigation*** problem, we test our methods on problems with $d = 2, 3, 4$ and $n = 30$, which increases the action space from 49 to $2,401$.

Comparison with Different Planning Time

Figure 4.6 shows how the planning time per step affects the performance. It clearly shows QBASE outperforms POMCP at all these three settings. Since Navigation can be viewed as a discretized continuous action space problem, there will be sufficient many similar good actions during our grid discretization. This property helps QBASE relatively easy to find a good action by sampling even it does not visit all possible actions. However, POMCP will always expand every candidate action. In this specific situation, POMCP will waste a significant simulation budget on repetitively searching for the good solution.

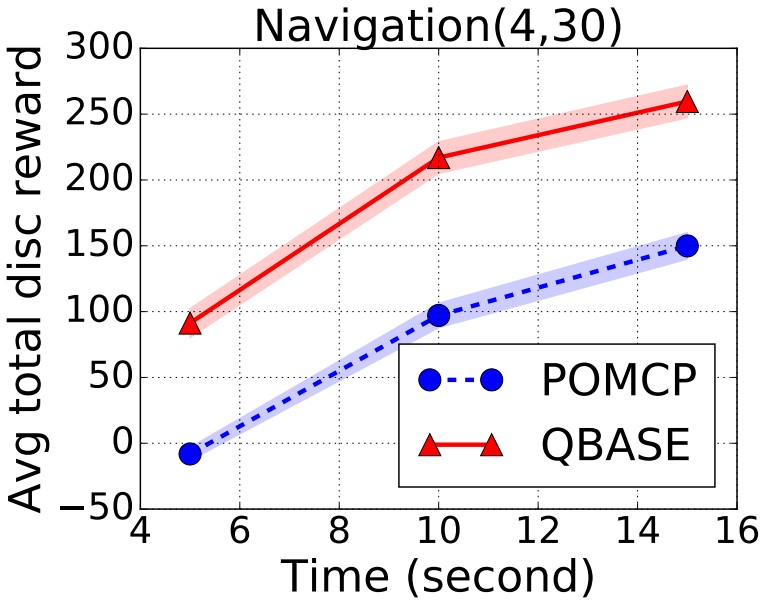


Figure 4.6: Performance with different planning times per step in Navigation

4.4.3 Hunting(n, u, v)

Definition

Figure 4.7 illustrates the scenario for $(11, 3, 3)$. Multiple (u) robots (green squares with letters) controlled by a centralized head, try to catch multiple (v) targets moving in a grid-world of size $n \times n$ populated by obstacles (black regions). At the beginning, the targets' positions are not known, and represented as uniform distributions over the free cells (colored pink). The true

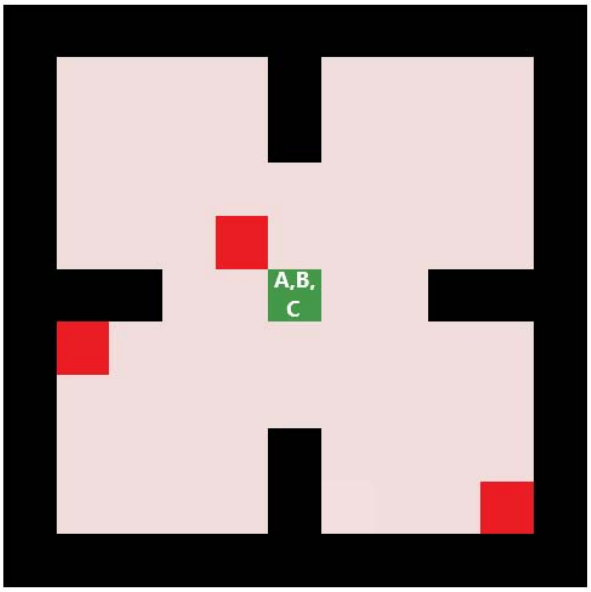


Figure 4.7: Illustration of ***Hunting***$(11, 3, 3)$

.

positions of the targets (which are unknown) are marked by red squares, The state space is the Cartesian product of the positions of the robots and the targets, while the action and observation spaces are the Cartesian products of all of the robots' actions and observations. At each step, each robot can stay where it is, move to one of the 8 cells adjacent to its current position, or catch a target. Their motion has no error. Furthermore, at each step, each robot can perfectly detect whether there is target(s) located in the same cell as itself or in one cell to its North, South, East, or West directions. Note that although its detection is perfect, a robot cannot distinguish which target is being detected nor the exact position (out of the five cells) of the target. A small penalty -1 is imposed on movement action for each robot. The 'catch' action yields a $+100$ reward if the agent is in the same cell as the target(s), otherwise the action incurs a penalty of -100. The targets know exactly the positions of the robots and always move to the direction farthest from the closest robot.

Comparison with Different Planning Time

Figure 4.8 shows how the planning time per step affects performance in some of the scenarios. The trends in ***Hunting-small***(11,2,2) and ***Hunting-small***(11,3,3) are similar to ***Hunting-***

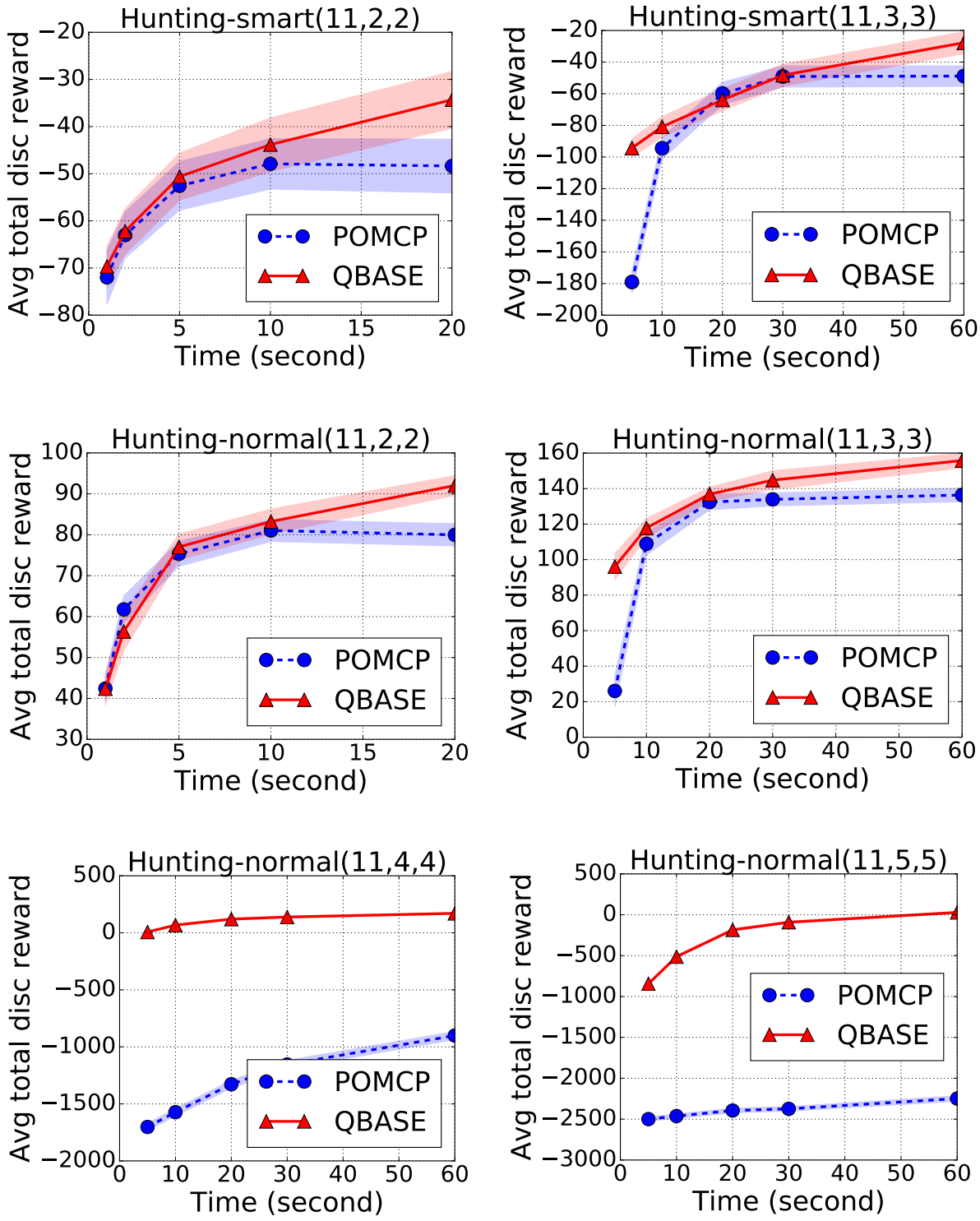


Figure 4.8: Performance with different planning times per step

normal(11,2,2) and ***Hunting-normal***(11,3,3). For ***Hunting-normal***, we report additional comparison for much larger settings upto 10^5 actions.

In the smaller case ***Hunting-small***$(11, 2, 2)$ and ***Hunting-normal***$(11, 2, 2)$, POMCP and QBASE are comparable when the planning is limited. With additional time budget is given, QBASE is able to take the advantage of identifying better strategies. However, POMCP gets stuck to improve its policy for a relative long time. This is observed the performance does not change from $t = 10s$ to $t = 20s$.

A slightly different behavior is shown in ***Hunting-small***$(11, 3, 3)$ and ***Hunting-normal***$(11, 3, 3)$. In this scenario, with 5 seconds planning time, QBASE outperforms POMCP, but POMCP catches up at 20 seconds planning time, before being outperformed by QBASE again as more planning time per step is allowed. The reason is that this problem has a considerably large action space, and therefore POMCP's sweeping of the entire action space already takes a significant portion of the 5 seconds planning time, causing POMCP to perform badly. However, as more time is allowed, POMCP starts to improve its base-line performance into a relatively good and easy to find policy, and then plateaus at this policy. In contrast, QBASE can quickly identify good actions and generate this relatively good and easy to find policy fast, and then takes significant additional time to improve this policy further.

It is worth mentioning that the action space in ***Hunting-normal***$(11, 5, 5)$ is upto 10^5 discrete actions. From the results, QBASE is able to identify good policies faster than POMCP. Besides, if more planning time is given, the improvement of QBASE is much better than POMCP. The success rate in this scenario is provided in Table 4.2. Even though POMCP increases dramatically at $t = 60s$, the success rate is still less than that of QBASE at $t = 5s$.

Table 4.2: Success rate of completing ***Hunting-normal***$(11, 5, 5)$

Method	$t = 5s$	$t = 60s$
POMCP	33%	71%
QBASE	87%	98%

In in Table 4.3, the results show that the success rate of QBASE completing the task does not decrease too much, while that of POMCP drops significantly when the action space is

increasing. Within all successful runs, QBASE is more efficient than POMCP to catch all targets as the average number of steps is significantly less.

Table 4.3: Summary of results in ***Hunting-normal*** with different number of agents and targets with 10 sec planning per step.

$\|A\|$	Success rate		Avg. #steps (of successful runs)	
	POMCP	QBASE	POMCP	QBASE
100	100%	99.9%	31±1	30±1
1,000	99.5%	99.7%	34±1	33±1
10,000	83.9%	98.3%	108±3	55±2
100,000	45.3%	95.8%	150±4	102±3

Collaboration in *Hunting*

Figure 4.9 illustrates some of the interesting behaviors that QBASE generates in ***Hunting-normal***$(11, 3, 3)$. In this scenario, each robot has limited sensing capability, i.e., it can only sense if a target exists in the same cell or in the adjacent cell at the North, South, East, or West direction of the robot. Therefore, they must collaborate to locate the target exactly and avoid the -100 penalty of performing a 'catch' action in the wrong cell.

Specifically, in Figure 4.9(a), agent C observes there is a opponent around it, but it cannot tell the exactly position due to partially observability. Agent C tries to confirm the location of the target by itself. However, it alone cannot succeed and instead it loses the detection of the target. Since ***Hunting*** is a centralized planning scenario, agent A and B come back immediately to help robot C in (c). With a certain chance, they can sense the target again. By communicating with other agent's observation, agent A is sure that the target and itself are in the same cell, therefore robot A is safe to execute '`Catch`' action in (d). After catching the opponent, all robots currently are massed around the right up region. The best decision they can make is to eliminate all other probability of the targets in this area. Agent B and C in (e) simultaneously check the rest cells where other targets could stay. The black arrow shows the motion of each agent. After that, in order to catch the rest of targets efficiently, each agent chooses a different

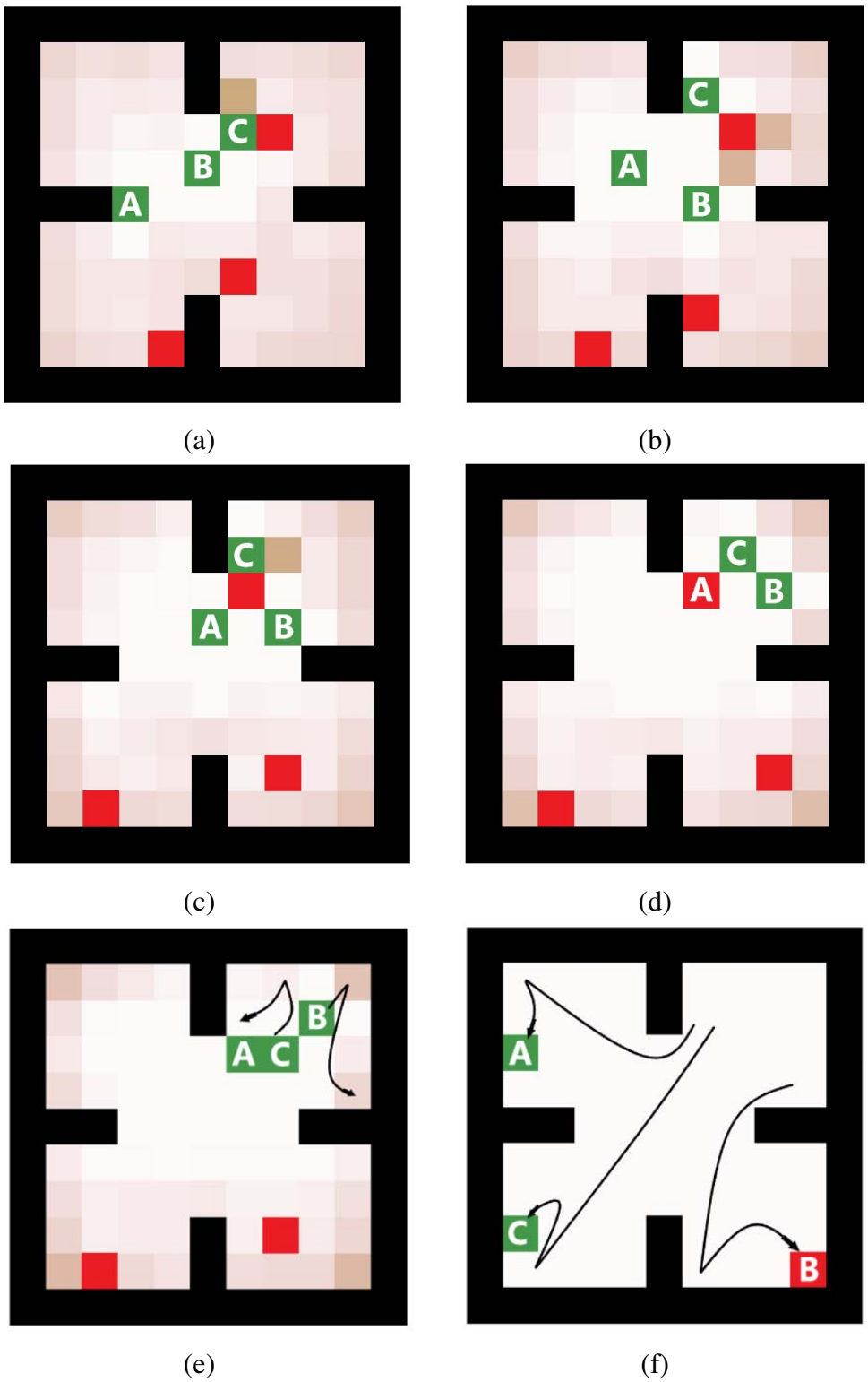


Figure 4.9: Collaborations strategy in ***Hunting-normal***$(11, 3, 3)$ generated by QBASE

region to further search. The last picture (f) shows the robot B moves in the corner diagonally with effect of pushing the target trapped in the corner. At this moment, agent B is on the target. If it captures the target, it will get a big reward and the problem finishes.

4.5 Discussion

In this chapter, we propose a novel online approximate POMDP solver, called Quantile-Based Action Selector (QBASE), that alleviates the difficulty of solving problems that have a large discrete action space. QBASE applies quantile statistics to adaptively construct a subset much smaller than the action space, so that more resources can be given to evaluate the Q-values of more promising actions. Experimental results on a range of robotics benchmark cases with action spaces varying from 13 to 100,000 indicate that QBASE outperforms the state-of-the-art, POMCP, on problems with large action spaces (more than 100).

The two-stage sampling idea underlying QBASE in principle can be generalized to other belief tree search algorithms, for instance ABT (Kurniawati and Yadav, 2013), DESPOT (Somani et al., 2013). We hope this new advancement in solving problems with large action spaces will further advance the practicality of POMDPs and allow more widespread applications of this robust approach to decision making in the presence of uncertainty.

The following manuscript has been incorporated as Chapter 5.

1. **Erli Wang**, Hanna Kurniawati, and Dirk P. Kroese, QBASE: an On-line Planner for Large Action POMDPs

Contributor	Statement of contribution	%
Erli Wang	Conception and design	60
	Implementation of numerical experiments	100
	Drafting and production	60
Hanna Kurniawati	Conception and design	25
	Drafting and production	20
Dirk P. Kroese	Conception and design	15
	Drafting and production	20

Chapter 5

APS-QBASE: An Adaptive Parameter Sampling

The aim of QBASE in the previous chapter was to alleviate the difficulty of handling POMDPs with large action spaces. A good choice of parameters is essential for the performance of QBASE. However, the best performing parameters are often unknown in advance and extra work is required to tune them beforehand. In this chapter, we propose an *adaptive parameter sampling* method called APS-QBASE, which is built on top of QBASE. The new extension enables QBASE to interleave parameter searching with growing the search tree in an on-line manner. Based on the outcomes of selected parameters, APS-QBASE uses the most promising ones more often in the subsequent planning. Two different tasks with up to one million actions are used to assess the performance of the proposed method. The results demonstrate that, given enough planning time, the proposed APS-QBASE can perform as good as or better than QBASE in the same parameter domain. Furthermore, a sensitivity study suggests that the performance is not sensitive to inputs of APS-QBASE. This significantly reduces the difficulty of setting good parameters in QBASE.

5.1 Background

The Partially Observable Markov Decision Process (POMDP) is a general and mathematically principled framework for decision problems that are partially observable and sequential. However, solving a POMDP is notorious for its computational complexity. It has been considered impractical for decades. Recently, various approximate POMDP solvers have significantly improved the capability to solve problems with large state spaces, large observation spaces, and long planning horizons (see Table 1.1). Notably, QBASE in the previous chapter was proposed to solve POMDPs with large discrete action spaces.

In QBASE, each parameter in $\langle \varrho, N_s, M_b, \beta \rangle$ has its own capability of guiding the tree expansion. For example, ϱ balances the preference of exploration and exploitation of sampling actions in a subset of actions and β controls how aggressively the probability function $b.\mathbf{P}$ in (4.2) evolves according to the current estimates of the Q-value. Experiments among different problems in Chapter 4 show that by setting appropriate values for QBASE's parameters, this POMDP solver outperforms one of the fastest POMDP on-line solvers. However, the best combination of $\langle \varrho, N_s, M_b, \beta \rangle$ is difficult to obtain in advance, and evaluating it is costly. The most common way of searching for the best parameters is conducted off-line and then apply them to run POMDP planners on-line. This separation of searching and planning undoubtedly introduces a heavy burden to users. Additionally, once a suitable choice of parameters is identified, the same parameters are used for all nodes in the belief tree. However, these parameters should depend on the current belief state and valid action spaces, which makes a single combination of parameters unlikely to be compatible with different situations. This problem is common in most POMDP solvers today (Silver and Veness, 2010; Kurniawati and Yadav, 2013; Somani et al., 2013; Sunberg and Kochenderfer, 2018). Overall, this suggests that there is a significant scope to further improve QBASE.

Motivated by those considerations, this chapter presents APS-QBASE, a QBASE-based solver with a mechanism of dynamically searching and switching parameters within a predefined planning phase. The basic idea of APS-QBASE is to use the promising sampled parameters more frequently. Considering the planning budget usually is limited (for example, $t = 5s$ per step), we need to take care of the compromise between the bias of the number of parameter

set and the regret of noisy outcomes of each parameter. The way of discovering a new parameter is simply a random search. Bergstra and Bengio (2012) shows the efficiency of random search over a grid search in hyper-parameter optimization. Which parameter to use is determined via UCB1 rule (Auer et al., 2002a) to balance the noisy outcomes. The proposed method has been tested on a robotics task and an operation research scenario with up to one million discrete actions.

5.2 Related Algorithms

Several on-line POMDP algorithms are considered to provide solutions. All solvers are based on MCTS but with different action selection strategies.

5.2.1 POMCP

POMCP (Silver and Veness, 2010) is considered as one of state-of-the-art on-line POMDP solvers today . It frames the action selection as a multi-armed bandit and adopts UCB1 to handle the exploration-exploitation. POMCP provides a baseline performance for various descendants. We refer to Section 2.3.3 for details.

5.2.2 QBASE

POMCP finds the best action by enumerating all possible actions. Therefore, it has to sweep the whole action space whenever a new belief node is expanded. We showed in Section 4.4 that such full enumeration quickly becomes infeasible when the size of action spaces are large (> 100). However, the action selection is essentially a stochastic optimization problem. We elaborate on the idea of QBASE and demonstrate experiments on different tasks in Chapter 4.

5.2.3 POMCP-PW

Like QBASE, instead of interacting with the entire action space directly, progressive widening (PW) (Couëtoux et al., 2011) is gradually expanding a subset of the action space and applying UCB1 over the subset of actions. This technique has recently been introduced to POMDP communities, called POMCP-DPW and POMCPOW(Sunberg and Kochenderfer, 2018). The aim of those methods is to handle continuous observation spaces. In this chapter, all testing scenarios are discrete observation spaces, therefore, the focus of comparison narrows down to only large actions spaces. It is shown in Algorithm 11.

Algorithm 11: SAMPLEACT(b, $\langle C_{\text{pw}}, k_{\text{pw}}, \alpha_{\text{pw}} \rangle$) for POMCP-PW

if $|b.\mathcal{A}_v| \leq k_{\text{pw}}\, b.N^{\alpha_{\text{pw}}}$ **then**
  $a \sim \mathsf{U}(\mathcal{A} \backslash b.\mathcal{A}_v)$
  Set $b.\mathcal{A}_v = b.\mathcal{A}_v \cup \{a\}$
return $\text{argmax}_{a \in b.\mathcal{A}_v}\, b.\widehat{Q}(a) + C_{\text{pw}} \sqrt{\frac{\log b.N}{b.N(a)}}$

5.3 Adaptive Parameter Sampling (APS)

5.3.1 The Method

This section presents a new QBASE-based method for automatically searching for the best performing set of parameters $\langle \varrho, N_s, M_b, \beta \rangle$. Similar to QBASE, APS-QBASE embeds the optimal policy in a belief tree. It shares most of QBASE's structure. Algorithm 12 describes the overview of APS-QBASE. The entry of the proposed algorithm is no longer a fixed parameter set of $\langle \varrho, N_s, M_b, \beta \rangle$. Instead, the inputs of APS-QBASE become $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$ and support D of $\langle \varrho, N_s, M_b, \beta \rangle$. The details on how APS-QBASE uses those inputs are presented in the next subsection.

We use $b.c = (\varrho, N_s, M_b, \beta, n, \hat{\mu}, n_{\text{simTree}})$ to store the information of running QBASE with a specific setting at belief b. The first four elements are QBASE's parameter set. The n counts the number of times that a particular parameter set has been used so far and $\hat{\mu}$ is the estimates

Algorithm 12: APS-QBASE

Input: POMDP model $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R \rangle$ or a corresponding black-box simulator $\mathcal{P}$, parameters of APS-QBASE $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$ and support set D of QBASE's parameters

Initialize $\mathcal{T}$ with b_0 as the root node
INITINODE(b_0, D)
while *running* **do**
  while *there is still time for planning* **do**
    GROWTREE ($\mathcal{T}$, $\mathcal{P}$, $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$, D) `// build tree` $\mathcal{T}$
  Perform action a, where $a = \arg\max_{a \in \mathcal{A}} b.\mathbf{P}(a)$
  $o =$ get observation
  $b = \tau(b, a, o)$ `// update belief`

Algorithm 13: INITINODE(b, D)

for $a \in \mathcal{P}.\mathcal{A}$ **do** $b.N(a) = 0$; $b.\mathbf{P}(a) = \frac{1}{|\mathcal{P}.\mathcal{A}|}$
$b.A_s =$sample N_s actions uniformly at random from A
for $a \in b.\mathcal{A}_s$ **do** $b.\mathbf{P}_s(a) = \frac{b.\mathbf{P}(a)}{\sum_{a \in b.\mathcal{A}_s} b.\mathbf{P}(a)}$
$b.\mathcal{C} = \emptyset$ `// parameter pool`
$\langle \varrho, N_s, M_b, \beta \rangle \sim \mathsf{U}(D)$ `// search a new combination`
$b.\mathcal{C} = b.\mathcal{C} \cup \{b.c\}$, where $b.c = (\varrho, N_s, M_b, \beta, n = 0, \hat{\mu} = 0, n_{\text{simTree}} = 0)$

of the highest Q-value found by applying its QBASE's $\langle \varrho, N_s, M_b, \beta \rangle$. n_{simTree} keeps records of how many tree simulations are conducted by using the parameter set. APS-QBASE initializes a configuration $b.c$ in INITINODE (Algorithm 13). The major modifications in GROWTREE (Algorithm 14) are different inputs of the function and adding a new function called APS, which is described in the next subsection.

5.3.2 Details

Key to APS-QBASE is the way it uses an appropriate parameter when the belief tree is growing. In general, APS-QBASE uses adaptive parameter sampling to create a gradually increasing set of QBASE's parameters as candidates for the best one for each belief node. It iteratively picks the most promising settings and interleaves them with the belief tree expansion. The settings will be used more often in the future if they are able to find a better Q-value. APS function is presented in Algorithm 15.

Algorithm 14: GROWTREE(b, $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$, D)

Set $d = 0$, $h = \emptyset$, $b =$ root of $\mathcal{T}$, $I_{rollout} =$ **false**
$s \sim b$
while $\gamma^d > \varepsilon$ *and* $I_{rollout} ==$ ***false*** **do**
  $a =$ APS(b, $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$, D)
  $(s', o, r) =$ SIMULATOR$(\mathcal{P}, s, a)$
  Append $\langle s, a, o, r \rangle$ to h
  Associate $\langle s, a, o, r \rangle$ with b
  $b.particles = b.particles \cup \{s\}$
  $b.\mathcal{A}_v = b.\mathcal{A}_v \cup \{a\}$
  $s = s'$; d++ ; $b.N(a)$++ ; $b.N$++ ; $b' = \tau(b, a, o)$
  if b' *is not in* $\mathcal{T}$ **then**
    add b' as a child of b and set $I_{rollout} =$ **true**
  $b = b'$
if $\gamma^d > \varepsilon$ **then**
  $E_h = 0$
else
  INITINODE(b,D) ; $b.particles = \{s\}$
  $E_h =$ROLLOUT-POLICY(s) `// rollout L steps`
Append $\langle s, -, -, E_h \rangle$ to h
Associate $\langle s, -, -, E_h \rangle$ with b
UPDATEVALUES($\mathcal{T}$,h,E_h) `// Back propagate to root`

Algorithm 15: APS(b, $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$, D)

$c = b.c$
$a =$ SAMPLEACT(b,$\langle c.\varrho, c.N_s, c.M_b, c.\beta \rangle$) `// same to Algorithm 10`
$c.n_{\text{simTree}} = c.n_{\text{simTree}} + 1$
if $c.n_{\text{simTree}}$ *equals to* $c.M$ **then**
  $v = \max_{a \in \mathcal{A}_s} b.\widehat{Q}(a)$ `// find the best Q-value`
  $c.\hat{\mu} = c.\hat{\mu} + (v - c.\hat{\mu})/c.n$ `// update the measure in (5.1)`
  $c.n = c.n + 1$
  if $|b.\mathcal{C}| \leq \left(\sum_{c \in b.\mathcal{C}} c.n\right)^{\alpha_{\text{aps}}}$ **then**
    $\langle \varrho, N_s, M_b, \beta \rangle \sim \mathsf{U}(D)$
    $b.\mathcal{C} = b.\mathcal{C} \cup \{b.c\}$, where $b.c = (\varrho, N_s, M_b, \beta, n = 0, \hat{\mu} = 0, n_{\text{simTree}} = 0)$
  $b.c = \text{argmax}_{c \in b.\mathcal{C}}\, c.\hat{\mu} + C_{\text{aps}} \sqrt{\log\left(\sum_{c \in b.\mathcal{C}} c.n\right)/c.n}$ `// select next x`
  $c.n_{\text{simTree}} = 0$ `// reset the counter`
return a

At the beginning of the search, there is only one candidate in $b.\mathcal{C}$. By using the current sampled parameters in $b.c$, the SAMPLEACT returns an action that interleaves with evaluation of Q-value (Line 2). The counter n_{simTree} is then increased by one. This process repeats until the update condition is met (Line 4). In other words, APS-QBASE has already employed a specified $\langle \varrho, N_s, M_b, \beta \rangle$ in $b.c$ to simulate the belief tree for a batch. Now, APS-QBASE needs an appropriate way to measure the performance of the selected parameters so that it provides information on parameter selection later. We consider the expected performance of the highest estimates of Q-value. This quantity considers, firstly, the subset involves randomness of actions due to the exploration component in $\mathcal{A}_s$ even if the parameter is the same, and secondly, the simulations of Q-value is noisy. Since our objective function of POMDPs is to find out the optimal policy via inducing from the highest Q-value, this measurement is consistent with this goal. We can simply write the measure as

$$\max_{a \in \mathcal{A}_s} \mathbb{E}\big[b.\widehat{Q}(a)\big]. \tag{5.1}$$

Furthermore, the quality of this measure is affected by the values in $\langle \varrho, N_s, M_b, \beta \rangle$. This is because the size of the current subset $\mathcal{A}_s$ is limited by N_s. The number of exploitative actions in the elite set is controlled by the quantile ϱ. The parameter β will affect how aggressively the probability function learns from the information of the current estimates of Q-value and the number of visits. The budget for evaluating the subset per batch is M_b. In Line 5-7, we update the average mean of the measure and its associated statistics.

The next question is when and how to generate subsequent candidates. Ideally, we need APS-QBASE to explore a new parameter at the right time. If the number of different parameters have been tested is very limited, it causes APS-QBASE to perform poorly because it may not find a good one. On the other hand, the number of different parameters should not explode. Otherwise, APS-QBASE is difficult to focus on the good performing parameter, which could also lead it to perform poorly. To this end, the growth speed of the parameter pool is limited by using the parameter α_{aps} according to the total number of parameters have been tried up to now (line 8). In terms of how to generate the next candidate in the hybrid space with a union of discrete variables N_s, M and continuous variables ϱ, β. The possible parameters are infinite, but

in practice, we can only visit a finite number of them. Various method can be applied here. We simply use random search that a new candidate is uniformly sampled from the support domain D (line 9).

APS-QBASE uses the estimates of Q-value so far to compare the performance of two different parameters. However, Q-value constantly updates throughout the search, therefore the parameter selection from APS-QBASE must be modified accordingly. We model this task as a stochastic multi-armed bandit problem and apply UCB1 to handle the exploration and exploitation tradeoff (line 11).

5.4 Experimental Results

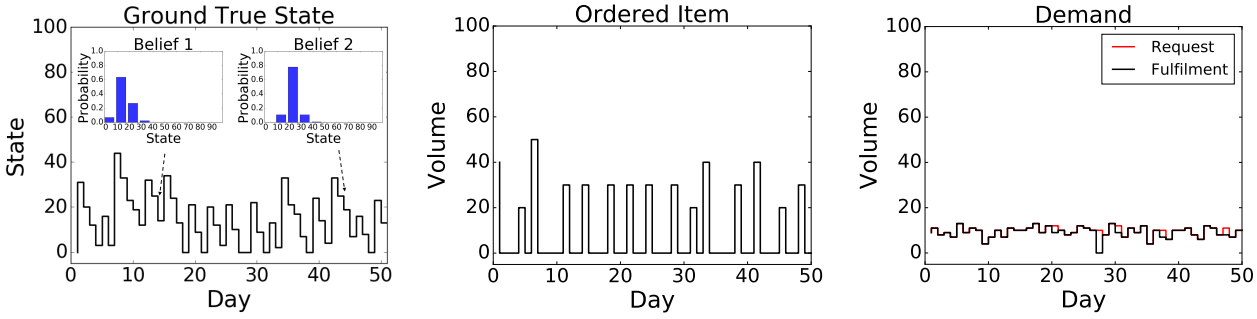
We aim to compare the performance of on-line POMDP solvers with different action selectors in one robotics and one logistics tasks with up to a million discrete actions. In the rest of this section, we will present necessary details of each scenario, describe how different action selectors work and study the sensitivity of $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$ in APS-QBASE approach.

5.4.1 Hunting-Normal(n, u, v)

Hunting-normal(n, u, v) is multi-agent (u) coordination robotics problem as introduced in Section 4.4.3. This task aims to find multiple prior unknown targets (v) with limited sensing as soon as possible in an $n \times n$ obstacle-populated environment. To achieve this goal, the agents in general have to collaborate with each other to systematically explore the space. We test our methods on $(11, 5, 5)$, yielding an action space with $|\mathcal{A}| = 10^5$ actions.

5.4.2 InventoryControl(K)

Inventory control problems have been well-studied in fully observable situations (Resh and Naor, 1963). However, many real-world scenarios involve partial observations, for example,



(a) (b) (c)

Figure 5.1: Illustration of ***InventoryControl***(1). (a) The ground true state is not fully observable. Instead, the agent maintains an estimate of the current state as a probability distribution over the state space. Although the ground truth state is the same (e.g., $s = 25$), the belief over the state space can be quite different. For example, the probability in the belief 1 masses in the [10,20], which leads the decision maker to order new items immediately the next day. The situation is different for the belief 2. The owner believes that there are still sufficient goods so will not replenish at the moment. (b) Considering the cost of ordering, the decision maker should prefer ordering a large number of items at once rather than frequently ordering a small amount. (c) The red line denotes the demand from the customer and the black line shows the actual fulfilment. The profits of the shop owner come only from the sale, therefore, one should avoid insufficient supply. The goal is to keep the stock state at an appropriate level. If the stock level is too high, there is a large holding cost. If the stock is too low, one may miss the sale.

inventory deterioration, misalignment and distributed storage. In this chapter, we assume our observation of the inventory state can only be partially perceived.

Think of an agent in ***InventoryControl***(K) as a shop owner, where the shop provides multiple commodities (K) that are for sale. In order to maximize the cumulative profit, the manager at each step (e.g., day) decides the type and quantity of items to order based on the prediction of future demand. Figure 5.1 shows a sample run of computed policy for one commodity. The state of commodity k denotes as s_k, where $s_k \in \{0, 1, 2, \ldots, 100\}, \forall k = 1, 2, \ldots, K$. The state space is the Cartesian product of the stock level for all commodities. The action $a_k \in \{0, 10, \ldots, 90\}, \forall k = 1, 2, \ldots, K$ corresponds to the amount of stock that is ordered from suppliers for each commodity k. Similarly, the action space is $\mathcal{A} = \{(a_1, \ldots, a_K)\}$ where $a_k \in \{0, 10, \ldots, 90\}, \forall k = 1, 2, \ldots, K$. We assume the number of orderings is perfect and all items ordered will be delivered in good condition on time (i.e., before the opening of a step).

At the start of the next step, inventory is either replenished by ordering certain goods due to the sale from the current stage or is not replenished. During this time period, the customer demand for each commodity is stochastic from Gaussian. This implies that the next state is computed as $\boldsymbol{s}' = \boldsymbol{s} + \boldsymbol{a} - \boldsymbol{D}$, where $\boldsymbol{D} \sim \mathsf{N}(\boldsymbol{\mu}, \Sigma)$. The correlation coefficients ρ_{cor} in Σ are assumed to be greater than 0.

In term of observation, each commodity can be perceived by a categorical result:

$$\tilde{o}_k = \begin{cases} \text{High}, & s'_k \in [66, 100] \\ \text{Normal}, & s'_k \in [33, 66) \\ \text{Low}, & s'_k \in [0, 33) \end{cases}$$

Furthermore, the result of such observation is also noisy, yielding the final observation $o_k \sim \mathsf{N}(\tilde{o}, e_Z)$ where e_Z is observation noise.

At the end of the step, the shop owner will need to compute the profit. The reward is defined as the incomes of sales minus the costs in general. For a given stock level s and ordered action a, a step profit is defined as

$$R(s, a) = \sum_{k=1}^{K} \left[p_k \min\{s_k, D_k\} - I_{\{s_k \geq \delta_k\}} h_k - I_{\{a_k > 0\}} (t_k + b_k\, a_k) \right],$$

where $p_k(> 0)$ is the income of an unit stock, $h_k(> 0)$ is the holding cost, $t_k(> 0)$ is the transition fee if the ordering action happens for commodity k and $b_k(> 0)$ is the cost price.

In ***InventoryControl***, we test our methods on six commodities, yielding an action space with $|\mathcal{A}| = 10^6$ actions.

5.4.3 Performance Comparison

Experimental Setup

The implementation of POMCP follows the authors' code. We build all other methods in POMCP's framework with improving the implementation to support a very large number of

action spaces using C++. The experiments are run as a single thread process on an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz with 128GB RAM.

To set the parameter, we execute a preliminary run to identify the best-performing ones. For POMCP and QBASE, we follow the discretization manners in Section 4.4. In POMCP-PW, the parameter k_{pw} is chosen from $\{50, 200, 500\}$, exploration constant C_{pw} from $\{10, 100, 1000, 10000, 100000\}$ and α_{pw} from $\{0.3, 0.7, 0.9\}$. For APS-QBASE, the parameter C_{aps} is selected from $\{10,\ 100,\ 1000, 10000, 100000\}$ and α_{aps} from $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. The support domain D of $\langle \varrho, N_s, M_b, \beta \rangle$ used in APS-QBASE is the same as running QBASE alone.

In ***Hunting-normal***, we use a random walk for a fixed number of steps as the rollout policy. In ***InventoryControl***, the default policy is computed as the fully observable version of the problem with the assumption that all items are independent. For a fair comparison, all solvers use the same rollout policy for each problem.

Table 5.1 shows settings for a ***InventoryControl***(6) problem. $\widetilde{\boldsymbol{D}}_k$ denotes the marginal distribution of $\boldsymbol{D}$ for each type of item. For other parameters, we let $\rho_{cor} = 0.8$, $e_Z = 5$ and the initial state of all commodities to be 0.

Table 5.1: Settings of ***InventoryControl***(6)

Item	p_k	h_k	δ_k	$\widetilde{\boldsymbol{D}}_k$	t_k	b_k
1	200	5	20	$\mathsf{N}(15, 2.5)$	1000	10
2	200	5	20	$\mathsf{N}(10, 2)$	1000	10
3	300	5	0	$\mathsf{N}(20, 5)$	2000	0
4	300	5	0	$\mathsf{N}(15, 2.5)$	2000	0
5	300	5	0	$\mathsf{N}(15, 2.5)$	500	0
6	200	20	0	$\mathsf{N}(15, 2.5)$	500	10

In particular, the inventory system should be assumed to run forever, as there are no explicitly terminal states. But for the convenience of computing the discounted total profits, the system will be measured over 50 stages.

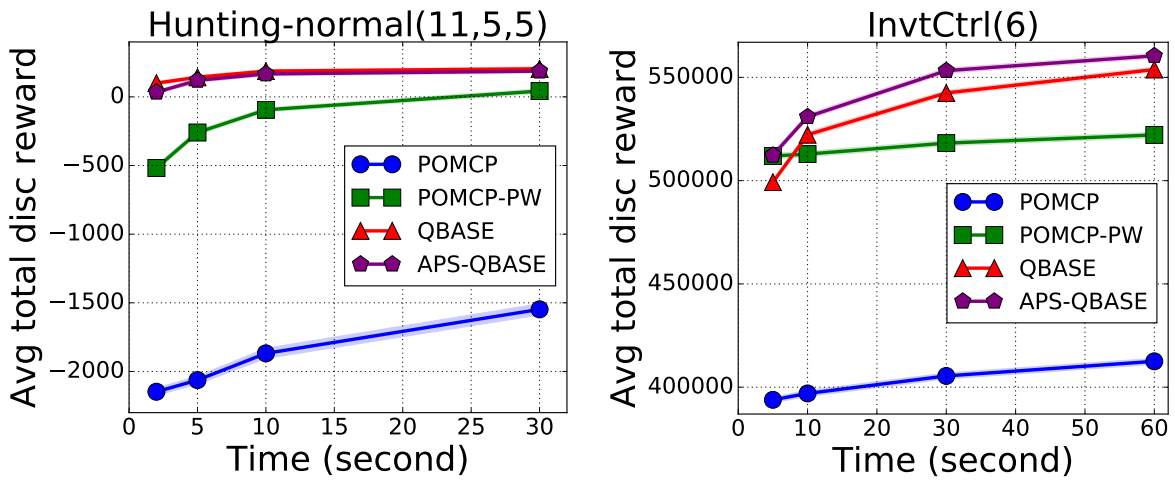


Figure 5.2: Performance with different planning times per step in Hunting-normal and Inventory Control problem

Results

Figure 5.2 presents the average results of expected total discounted rewards with 95% confidence interval over 1000 simulations in each problem. In general, when the planning time is small ($t \leq 5s$), APS-QBASE can only try a limited number of different parameter sets. This is the reason why APS-QBASE performs poorer, for example in ***Hunting-normal***(11,5,5). However, as more planning time is allowed, APS-QBASE is able to identify an appropriate QBASE's parameter to catch up or further boost the total discounted rewards.

POMCP-PW shows its significant improvement over POMCP by simply replacing its full sweeping strategy when the action space is large ($\geq 10^5$). This is because POMCP explodes on expanding new actions in the initialization stage, which leads the tree cannot go deeper to explore any useful information. POMCP-PW further verifies that partial enumeration methods work well for problems with large action spaces.

Interestingly, POMCP-PW gets the best performance in ***InventoryControl***(6) when the planning time is small ($t = 2s$). However, with providing extra budgets, POMCP-PW can only slightly improve performance further. One possible reason is that the PW technique explores a new action at the cost of the number of simulations growing exponentially . It causes the number of actions can be discovered in practice is limited to find a better one. In contrast,

the probability of sampling a starved action will be always a fixed probability in QBASE. This does not prevent QBASE to expand the tree wider to keep increasing the performance when additional planning time is provided.

5.4.4 Sensitivity Study

APS-QBASE aims to reduce the difficulty of setting QBASE's parameters. It is done by replacing the parameters of QBASE $\langle \varrho, N_s, M_b, \beta \rangle$ down to $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$. It is worthy of studying the sensitivity of those two parameters.

We test APS-QBASE for different $\langle C_{\text{aps}}, \alpha_{\text{aps}} \rangle$ values. The planning time for each task is ten second per step. The normalized average results over 50 repetitions are presented in Figure 5.3. The results show that APS-QBASE is not sensitive to α_{aps} to get good results when the exploration constant C_{aps} is large.

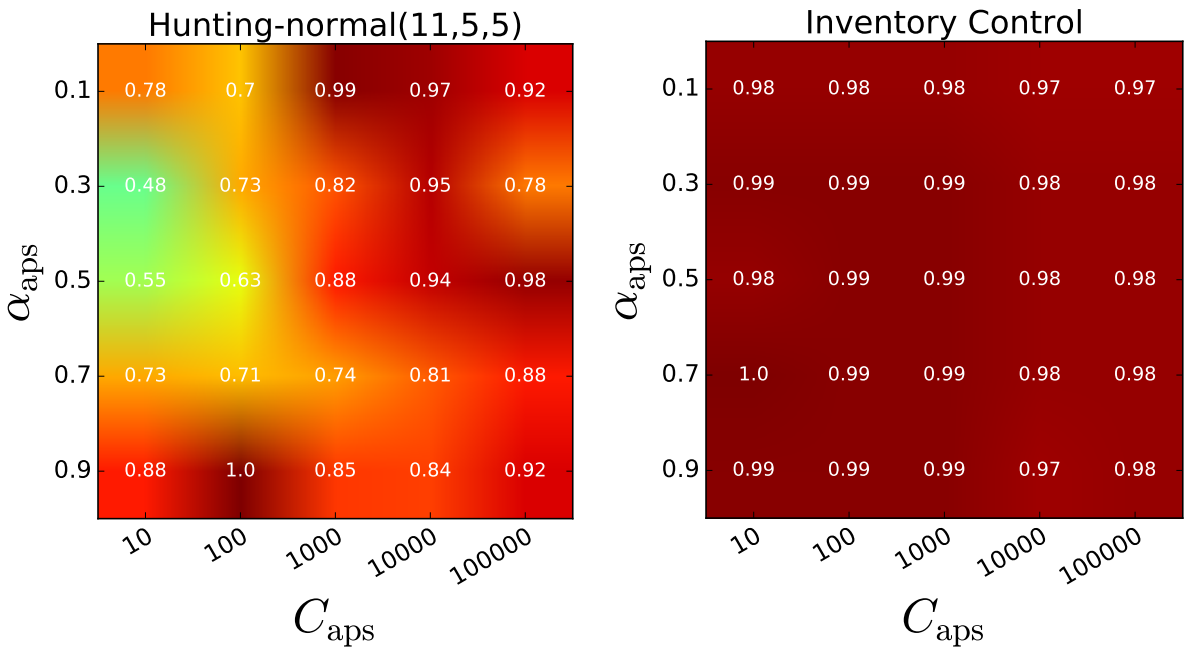


Figure 5.3: Sensitivity Study

In Table 5.2 and Table 5.3, we show the minimum and maximum of the expected discount total rewards found in each algorithm's parameter domains. The relative variation of the results in ***InventoryControl***(6) is small, because the default policy used in this task is the information

of solving a fully observable version of the original problem. Such approximation provides a higher quality of the estimates of newly added belief node in the tree. Comparing with QBASE, the proposed APS-QBASE can achieve a narrower variation. It indicates that this method is more robust and efficient. Overall, we argue that APS-QBASE significantly reduces the difficulty of tuning QBASE's parameters.

Table 5.2: Performance variation in the each algorithm's parameter domain in ***Hunting-normal***$(11, 5, 5)$

Method	POMCP	POMCP-PW	QBASE	APS-QBASE
min $\widehat{V}$	-1868.24	-549.48	69.37	95.56
max $\widehat{V}$	-1546.43	-94.32	189.74	198.22

Table 5.3: Performance variation in the each algorithm's parameter domain in ***InventoryControl***(6)

Method	POMCP	POMCP-PW	QBASE	APS-QBASE
min $\widehat{V}$	392573.29	486544.72	453966.0	525819.08
max $\widehat{V}$	403138.68	512261.73	536978.78	538058.14

5.5 Discussion

In this chapter, we propose an extension of QBASE, called APS-QBASE, for reducing the difficulty of setting good parameters in QBASE. The algorithm creates a dynamic parameter pool to search for the best parameter so that each belief node is able to identify the most suitable running settings according to its own situation. The adaptive parameter sampling mechanism is general and in principle can be applied to any POMDP solvers. Experiments on two different tasks with up to one million discrete actions indicate APS-QBASE can outperform one of state-of-the-art POMDP solvers and its enhancement for handling large action spaces.

There is still scope for further work in APS-QBASE. One limitation of the method is that the exploring strategy for a new candidate is purely random search. It may sample in less promising areas, causing unnecessary waste during the planning. A better data-efficient way is

to build a probabilistic model on the available observations, for example, Bayesian optimization (Srinivas et al., 2010; Brochu et al., 2010; Hoang et al., 2018). Another possible direction is to design a cost-sensitive measure of assessing a specific parameter. Thirdly, the idea of APS can be applied to POMCP and POMCP-PW to reduce the difficulty of setting good parameters. Finally, considering the evaluation of APS-QBASE with a particular parameter is expensive, we simply discretized the parameter space of APS-QBASE into a finite set and ran them for 50 times as an approximate sensitivity analysis. However, the ideal investigation is to study the continuity property of the performance of APS-QBASE, which is reserved for future study.

Chapter 6

Summary

Rational actions are crucial to autonomous systems in the presence of uncertainty. The uncertainties mean that the effects of actions are not precisely known, sensors and sensing are erroneous, and information about the operating an environment can be imperfect. Except for uncertainties, the problems of our interest are often sequential and large-scale. However, the computation complexity of solving such problems are notorious. The goal of this thesis is to develop general strategies to further advance the practicality of POMDPs and allow more widespread applications of this robust approach to decision making. The principal methodology is combining an appropriate structure of problems (e.g., the abundance of good solutions) with recent advances in Monte Carlo methods (e.g., the Cross-Entropy method). By using sampling techniques, it tends to be simple, flexible and scalable to break the heavy burden of computation.

This thesis starts from how to model uncertainties in Chapter 2. Depending on the sources of uncertainties, we review three types of mathematical models. Among those models, the partially observable Markov decision process is the general and principled framework for decision problems that are partially observable and sequential. It is followed by reviewing a stochastic optimization without requiring gradient information.

The main contributions of the thesis are as follows.

Firstly, the stochastic multi-armed bandit, viewed as the simplest version of POMDPs, is a well-known decision making under uncertainty. Traditionally, classical algorithms work well when the number of available pulls is often large compared to the number of arms. However, in many practical applications, the number of arms that can be selected from is very large. It causes existing algorithms can perform poorly. For example, UCB1 requires the agent to play all arms once to set initializations. In Chapter 3, CEMAB represents the goodness of arms as a probability distribution function and adopts the Cross-Entropy method as a core optimizer to search for the optimal arm. It deals with exploration and exploitation in the form of sampling an arm according to its probability. The probability function is updated in a batch mode, which helps to focus on a subset of the entire arm space. Numerical experiments with up to 10^4 arm space show that CEMAB is superior to various existing competitors.

Inspired by CEMAB, we propose QBASE in Chapter 4 — an on-line planner for POMDP with large discrete action spaces — by keeping the idea of 'partial enumeration' in mind. Exactly because of Q-value in POMDP cannot be computed in a closed form, it has to consider many possible sequences of future actions selected and observations obtained. To get better estimates, QBASE avoids complete enumeration by using a quantile-based stochastic optimization approach to focus on going deeper to the promising sub-trees first. Simulation results on problems of *RockSample*, *Navigation*, *Hunting-normal*, and *Hunting-smart* indicate that QBASE outperforms POMCP, one of the fastest POMDP solvers today. When the action space is large (e.g., 10^5 possible actions), QBASE can generate substantially better policies.

Although QBASE has demonstrated its improvement over POMCP, the hyper-parameters in QBASE need to be set carefully and it requires a long time testing. Once the parameter is found, it will be applied to all belief nodes to handle the exploration and exploitation, which is less likely to capture the information of state and valid action spaces. Chapter 5 presents an enhancement of QBASE with setting parameters automatically, named APS-QBASE. It interleaves parameter sampling with belief sampling in the tree. Based on the outcomes of sampled parameters, APS-QBASE uses the settings more often with the highest expected performance. The whole process is completely performed within the planning phase. We conduct experiments on one robotics and one logistics tasks with up to one million actions. The comparison indicates

that APS-QBASE is able to achieve higher quality results with less effort than POMCP and its variant with an enhancement of large action spaces.

The theme of this thesis is about decision making under uncertainty. Most of our efforts focus on alleviating the difficulty of solving problems with large-scale action spaces. In addition to such developments, there are many possible directions for further research.

All investigations conducted in this thesis are simulation experiments. It is crucial to understand how well the proposed strategies work in more practical scenarios. For example, even though CEMAB's performance in Chapter 3 does not tie to any specific dataset, it would be good to test CEMAB on more realistic datasets, for example, Yahoo! Front Page Today Module User Click Log Dataset R6 [1].

The way of selecting a distribution over the action space in QBASE is versatile. For example, to handle discrete action spaces, we used in Chapter 4 a multivariate Bernoulli distribution. However, there is no barrier to adopt other types of distributions including continuous ones when dealing with continuous actions spaces. Such observations help generalize QBASE to a wider applications.

The quality of estimates for newly added nodes in the belief tree is crucial to influence the expansion of the search tree. Although we use a better default policy in Chapter 5 to improve the estimates, there is no interaction between state values computed during on-line planning and values from the default policy. Xiao et al. (2018) exploits a fixed size for the information of past states, to predict the value for a newly added state in an on-line manner. Embedding a machine learning approach to generalize the value function during the planning may significantly improve the quality of solving POMDPs in general. Moreover, all experiments conducted in this thesis were run as a single thread process. By leveraging parallelization, it would be useful to handle much larger problems, for example, HyP-DESPOT (Cai et al., 2018).

[1] https://webscope.sandbox.yahoo.com/

Appendix A

Details of Scenarios

A.1 RockSample(n, k)

- $\mathcal{S} = \{\big(s_{\text{agent}}, s_{\text{rock}(1)}, \ldots, s_{\text{rock}(k)}\big)\}$, where $s_{\text{agent}} \in \{(1,1), \ldots, (n,n)\}$ and $s_{\text{rock}(i)} \in \{\text{Good}, \text{Bad}\}, i = 1, \ldots, k$.
- $\mathcal{A} = \{\texttt{North}, \texttt{West}, \texttt{South}, \texttt{East}, \texttt{Sample}, \texttt{Check}_1, \ldots, \texttt{Check}_k\}$ for the agent.
- $\mathcal{O} = \{\texttt{Good}, \texttt{Bad}, \texttt{Null}\}$.
- $T(s_{\text{agent}}, a, s'_{\text{agent}})$ is deterministic, for $a \in \mathcal{A}$.
- $Z(s'_{\text{agent}}, a, o) = \text{Ber}(\eta)$, where

$$\eta = \frac{1}{2}(1 + 2^{-d(s'_{\text{agent}}, s_{\text{rock}(a)})/\phi}),$$

  for $a \in \{\texttt{Check}_1, \ldots, \texttt{Check}_k\}$. The function $d(\cdot, \cdot)$ measures the distance (i.e., Euclidean distance) between the state of the agent and the rock intended to scan and ϕ is a constant parameter.

- $R(s_{\text{agent}}, a)$ is defined as

  ▷ $R(s_{\text{agent}}, \text{Sample}) = +10$, if the rock is good

▷ $R(s_{\text{agent}}, \text{Sample}) = -10$, if the rock is bad

▷ $R(s_{\text{agent}}, a) = -1$, for $a \in \{\texttt{North}, \texttt{West}, \texttt{South}, \texttt{East}\}$

▷ $R(s_{\text{agent}}, a) = 0$, for $a \in \{\texttt{Check}_1, \ldots, \texttt{Check}_k\}$

The positions of rocks in ***RockSample***$(7, 8)$ exactly follows code implemented in POMCP. While for ***RockSample***$(20, 50)$ and ***RockSample***$(20, 100)$, the placements of rocks are set as follows.

The placements of rocks for ***RockSample***$(20, 50)$ are set at (16, 14), (5, 3), (10, 8), (8, 12), (6, 18), (16, 10), (19, 12), (12, 13), (3, 18), (2, 3), (11, 8), (6, 6), (5, 13), (11, 17), (3, 9), (13, 16), (1, 6), (0, 10), (5, 7), (1, 17), (18, 13), (16, 16), (7, 2), (3, 5), (8, 15), (8, 4), (14, 0), (8, 8), (19, 18), (18, 5), (19, 11), (6, 7), (5, 0), (17, 10), (4, 16), (2, 5), (10, 0), (18, 4), (8, 13), (4, 6), (1, 13), (18, 0), (12, 14), (7, 7), (13, 0), (15, 8), (6, 14), (13, 18), (4, 19), (19, 19).

The placements of rocks for ***RockSample***$(20, 100)$ are set at (8, 14), (11, 16), (18, 2), (2, 9), (0, 4), (8, 15), (11, 6), (18, 16), (5, 3), (10, 17), (15, 18), (6, 3), (1, 1), (8, 10), (0, 0), (13, 16), (2, 18), (3, 14), (10, 4), (12, 11), (7, 18), (12, 17), (16, 12), (14, 15), (7, 16), (11, 11), (4, 0), (14, 5), (6, 8), (1, 8), (6, 17), (6, 1), (0, 6), (3, 3), (17, 4), (13, 14), (17, 5), (5, 4), (17, 2), (4, 4), (19, 16), (8, 7), (4, 13), (17, 18), (7, 8), (10, 12), (14, 19), (16, 8), (7, 13), (1, 6), (4, 18), (15, 5), (18, 5), (11, 18), (18, 9), (11, 5), (19, 1), (15, 3), (3, 6), (10, 19), (12, 15), (17, 13), (12, 16), (19, 8), (2, 14), (5, 9), (9, 16), (2, 8), (4, 17), (3, 0), (13, 19), (6, 5), (15, 0), (10, 3), (4, 9), (13, 17), (0, 5), (11, 15), (19, 3), (7, 14), (11, 4), (18, 1), (5, 17), (16, 13), (3, 19), (17, 19), (5, 10), (16, 18), (16, 9), (3, 17), (19, 0), (5, 2), (15, 14), (16, 7), (9, 7), (18, 12), (2, 0), (2, 7), (17, 1), (0, 13).

A.2 Navigation(d, n)

- Define a grid world as $\mathcal{W} = [1, n]^d$, where n is the size and d is the dimension. $\mathcal{W}$ contains two parts: $\mathcal{W}_{\text{free}}$ and $\mathcal{W}_{\text{obstacle}}$. The way to generate $\mathcal{W}_{\text{free}}$ and $\mathcal{W}_{\text{obstacle}}$ follows four steps. For example in $d = 2$ navigation, we start with an empty gird world $\mathcal{W}$. Then

  1. add boundaries with thickness m_b,

2. add obstacles with thickness m_o in the middle by: i) setting walls at $x = \{\lceil \frac{n+1}{2} \rceil - m_o+1, \ldots, \lceil \frac{n+1}{2} \rceil+1\}$ for any y; ii) setting walls at $y = \{\lceil \frac{n+1}{2} \rceil - m_o+1, \ldots, \lceil \frac{n+1}{2} \rceil+1\}$ for any x,
3. delete obstacles with thickness m_f to get $\mathcal{W}_{\text{free}}$ if a cell $(x, y) \in \mathcal{W}$ is unwalkable, $\forall x, y \in \{\lceil \frac{n+1}{2} \rceil - m_f + 1, \ldots, \lceil \frac{n+1}{2} \rceil + \frac{m_f}{2} + 1\}$,
4. set the goal cell in a corner.

After setting the grid, we can obtain a grid world including walkable cells and unwalkable ones as well. For $d > 2$ problem, the way of constructing environment is the same.

- $\mathcal{S} = \mathcal{W}_{\text{free}}$, where $m_b = 3, m_o = 2$ and $m_f = 6$.
- $\mathcal{A} = \{(\mathcal{A}_{\text{dim}_1}, \ldots, \mathcal{A}_{\text{dim}_d})\}$, where each $\mathcal{A}_{\text{dim}} = \{-3, -2, -1, 0, 1, 2, 3\}$ respect to the distance current state in this dimension.
- $\mathcal{O} = \{(\mathcal{O}_{\text{face}_1}, \ldots, \mathcal{O}_{\text{face}_{2d}})\}$, where each $\mathcal{O}_{\text{face}} = \{\texttt{Wall}, \texttt{NoWall}\}$.
- $T(s_{\text{agent}}, a, s'_{\text{agent}})$ has 90% accuracy of arriving at the desired s'_{agent} after executing action a, while arriving (wrongly) at other states with equal probability.
- $Z(s'_{\text{agent}}, a, o)$ is deterministic.
- $R(s_{\text{agent}}, a)$ is defined as
  - ▷ $R(s_{\text{agent}}, a) = +1,000$, if action a lead the agent reach pre-defined goal,
  - ▷ $R(s_{\text{agent}}, a) = -1$, otherwise.

A.3 Hunting(n, u, v)

- $\mathcal{W}$ is shown in Figure 4.7. It contains the forbidden area $\mathcal{W}_{\text{obstacle}}$ (marked in black) and walkable area $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \mathcal{W}_{\text{obstacle}}$.
- $\mathcal{S} = \{ \left(s_{\text{agent}_1}, \ldots, s_{\text{agent}_u}, s_{\text{target}_1}, \ldots, s_{\text{target}_v}\right)\}$, where $s_{\text{agent}_i} \in \mathcal{W}_{\text{free}}, i = 1, \ldots, u$ and $s_{\text{target}_j} \in \mathcal{W}_{\text{free}}, j = 1, \ldots, v$.

- $\mathcal{A} = \{(\mathcal{A}_{\text{agent}_1}, \ldots, \mathcal{A}_{\text{agent}_u})\}$, where each $\mathcal{A}_{\text{agent}_i} = \{\texttt{Stay}, \texttt{North}, \texttt{Northwest}, \texttt{West}, \texttt{Southwest}, \texttt{South}, \texttt{Southeast}, \texttt{East}, \texttt{Northeast}, \texttt{Catch}\}, i = 1, \ldots, u$.
- $\mathcal{O} = \{(\mathcal{O}_{\text{agent}_1}, \ldots, \mathcal{O}_{\text{agent}_u})\}$, where each $\mathcal{O}_{\text{agent}} = \{\texttt{Yes}, \texttt{No}\}$.
- $T(s_{\text{agent}}, a, s'_{\text{agent}})$ is deterministic. $T(s_{\text{target}}, a, s'_{\text{target}})$ in *Hunting-smart* follows the strategy in *Tag* (Pineau et al., 2003). That is, whenever a robot and a target occupy the same cell, the target can still get away, unless the robot performs the action 'catch'. While in *Hunting-normal*, once a robot and a target are in the same cell, the target cannot escape.
- $Z(s'_{\text{agent}}, a, o)$ has the deterministic effect of detecting targets around the agent
- $R(s_{\text{agent}}, a)$ is defined as
  - ▷ $R(s_{\text{agent}}, \texttt{Catch}) = +100$, if the target is tagged correctly,
  - ▷ $R(s_{\text{agent}}, \texttt{Catch}) = -100$, if there is no target,
  - ▷ $R(s_{\text{agent}}, a) = -1$, otherwise.

References

Agrawal, S. and Goyal, N. (2012). Analysis of thompson sampling for the multi-armed bandit problem. In *Conference On Learning Theory (COLT)*, pages 39.1–39.26.

Aoki, M. (1965). Optimal control of partially observable Markovian systems. *Journal of The Franklin Institute*, 280(5):367–386.

Arulampalam, M. S., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188.

Astrom, K. J. (1965). Optimal control of Markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 10(1):174–205.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2002b). The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77.

Bai, H., Hsu, D., and Lee, W. S. (2014). Integrated perception and planning in the continuous space: A POMDP approach. *The International Journal of Robotics Research*, 33(9):1288–1302.

Bai, H., Hsu, D., Lee, W. S., and Ngo, A. (2010). Monte Carlo Value Iteration for Continuous-State POMDPs. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Balla, R.-K. and Fern, A. (2009). UCT for tactical assault battles in real-time strategy games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 40–45.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138.

Bennett, C. C. and Hauser, K. (2013). Artificial intelligence framework for simulating clinical decision-making: A Markov decision process approach. *Artificial intelligence in medicine*, 57(1):9–19.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bonet, B. and Geffner, H. (2003). Labeled RTDP: Improving the convergence of real-time dynamic programming. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 3, pages 12–21.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

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., Stoltz, G., and Szepesvari, C. (2011). X-armed bandits. *Journal of Machine Learning Research*, 12(May):1655–1695.

Burtini, G., Loeppky, J., and Lawrence, R. (2015). A survey of online experiment design with the stochastic multi-armed bandit. *arXiv preprint arXiv:1510.00757*.

Busoniu, L., Ernst, D., De Schutter, B., and Babuska, R. (2011). Cross-Entropy optimization of control policies with adaptive basis functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(1):196–209.

Cai, P., Luo, Y., Hsu, D., and Lee, W. S. (2018). Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. In *Robotics: Science and Systems (RSS)*.

Capitan, J., Spaan, M. T., Merino, L., and Ollero, A. (2013). Decentralized multi-robot cooperation with auctioned POMDPs. *The International Journal of Robotics Research*, 32(6):650–671.

Chadès, I., Martin, T. G., Nicol, S., Burgman, M. A., Possingham, H. P., and Buckley, Y. M. (2011). General rules for managing and surveying networks of pests, diseases, and endangered species. *Proceedings of the National Academy of Sciences*, 108(20):8323–8328.

Chang, H. S., Hu, J., Fu, M. C., and Marcus, S. I. (2013). *Simulation-based algorithms for Markov decision processes*. Springer Science & Business Media.

Chaudhuri, A. R. and Kalyanakrishnan, S. (2017). PAC identification of a bandit arm relative to a reward quantile. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1777–1783.

Coquelin, P.-A. and Munos, R. (2007). Bandit algorithms for tree search. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 67–74.

Costa, A., Jones, O. D., and Kroese, D. (2007). Convergence properties of the Cross-Entropy method for discrete optimization. *Operations Research Letters*, 35(5):573–580.

Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., and Bonnard, N. (2011). Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization (LION)*, pages 433–445. Springer.

Duan, Q., Kroese, D. P., Brereton, T., Spettl, A., and Schmidt, V. (2014). Inverting Laguerre tessellations. *The Computer Journal*, 57(9):1431–1440.

Finnsson, H. and Björnsson, Y. (2008). Simulation-based approach to general game playing. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 8, pages 259–264.

Floreano, D. and Wood, R. J. (2015). Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460.

Goschin, S., Littman, M. L., and Ackley, D. H. (2011). The effects of selection on noisy fitness optimization. In *The Genetic and Evolutionary Computation Conference (GECCO)*, pages 2059–2066. ACM.

Goschin, S., Weinstein, A., and Littman, M. L. (2013). The Cross-Entropy method optimizes for quantiles. In *International Conference on Machine Learning (ICML)*, pages 1193–1201.

Hoang, T. N., Hoang, Q. M., Ouyang, R., and Low, K. H. (2018). Decentralized high-dimensional bayesian optimization with factor graphs. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hoerger, M., Kurniawati, H., Elfes, A., and Bandyopadhyay, T. (2016). Linearization in motion planning under uncertainty. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Hoey, J. and Poupart, P. (2005). Solving POMDPs with continuous or large discrete observation spaces. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1338.

Hoey, J., Poupart, P., von Bertoldi, A., Craig, T., Boutilier, C., and Mihailidis, A. (2010). Automated handwashing assistance for persons with dementia using video and a partially observable Markov decision process. *Computer Vision and Image Understanding*, 114(5):503–519.

Horowitz, M. and Burdick, J. (2013). Interactive Non-Prehensile Manipulation for Grasping Via POMDPs. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Hu, J., Hu, P., and Chang, H. S. (2012). A stochastic approximation framework for a class of randomized optimization algorithms. *IEEE Transactions on Automatic Control*, 57(1):165–178.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134.

Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine learning*, 49(2):193–208.

Kobilarov, M. (2012). Cross-Entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871.

Kochenderfer, M. J. (2015). *Decision making under uncertainty: theory and application*. MIT press.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European conference on machine learning (ECML)*, pages 282–293. Springer.

Kroese, D. P., Taimre, T., and Botev, Z. I. (2011). *Handbook of Monte Carlo methods*, volume 706. John Wiley & Sons.

Kurniawati, H., Du, Y., Hsu, D., and Lee, W. (2011). Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*, 30(3):308–323.

Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems (RSS)*.

Kurniawati, H. and Yadav, V. (2013). An online POMDP solver for uncertainty planning in dynamic environment. In *International Symposium on Robotics Research (ISRR)*. Springer.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995a). Efficient dynamic-programming updates in partially observable Markov decision processes. *Tech Report CS-95-19, Brown University*.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995b). Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–65.

Luo, Y., Bai, H., Hsu, D., and Lee, W. S. (2016). Importance sampling for online planning under uncertainty. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Madani, O., Hanks, S., and Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *AAAI/IAAI*, pages 541–548.

Mannor, S., Rubinstein, R. Y., and Gat, Y. (2003). The Cross Entropy method for fast policy search. In *International Conference on Machine Learning (ICML)*, pages 512–519.

Mansley, C. R., Weinstein, A., and Littman, M. L. (2011). Sample-based planning for continuous action Markov decision processes. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Margolin, L. (2005). On the convergence of the Cross-Entropy method. *Annals of Operations Research*, 134(1):201–214.

Monahan, G. (1982). State of the art — A survey of Partially Observable Markov Decision Processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16.

Ng, A. Y. and Jordan, M. I. (2000). Pegasus: A policy search method for large MDPs and POMDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 406–415, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Omidshafiei, S., Agha-Mohammadi, A.-A., Amato, C., Liu, S.-Y., How, J. P., and Vian, J. (2016). Graph-based Cross Eentropy method for solving multi-robot decentralized pomdps. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5395–5402. IEEE.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987a). The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987b). The complexity of Markov Decision Processes. *Math. of Operation Research*, 12(3):441–450.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 3, pages 1025–1032.

Porta, J. M., Vlassis, N., Spaan, M. T., and Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367.

Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Resh, M. and Naor, P. (1963). An inventory problem with discrete time review and replenishment by batches of fixed size. *Management Science*, 10(1):109–118.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535.

Ross, S., Pineau, J., Paquet, S., and Chaib-Draa, B. (2008). Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704.

Rubinstein, R. Y. (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112.

Rubinstein, R. Y. (1999). The Cross-Entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190.

Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross-Entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer.

Rubinstein, R. Y. and Kroese, D. P. (2017). *Simulation and the Monte Carlo method*, volume 10. John Wiley & Sons.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Seiler, K., Kurniawati, H., and Singh, S. (2015). An Online and Approximate Solver for POMDPs with Continuous Action Space. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Shani, G., Brafman, R. I., and Shimony, S. E. (2007). Forward search value iteration for POMDPs. In *IJCAI*, pages 2619–2624.

Shani, G., Pineau, J., and Kaplow, R. (2013). A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51.

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.

Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2164–2172.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088.

Smith, T. and Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Smith, T. and Simmons, R. (2005). Point-based POMDP algorithms: Improved analysis and implementation. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Somani, A., Ye, N., Hsu, D., and Lee, W. S. (2013). DESPOT: Online POMDP planning with regularization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1772–1780.

Sondik, E. J. (1971). *The optimal control of partially observable Markov processes*. PhD thesis, Stanford University.

Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations research*, 26(2):282–304.

Spaan, M. T. J. and Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of artificial intelligence research*, 24:195–220.

Spettl, A., Brereton, T., Duan, Q., Werz, T., Krill III, C. E., Kroese, D. P., and Schmidt, V. (2016). Fitting Laguerre tessellation approximations to tomographic image data. *Philosophical Magazine*, 96(2):166–189.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on International Conference on Machine Learning (ICML)*, pages 1015–1022.

Sun, W., Patil, S., and Alterovitz, R. (2015). High-frequency replanning under uncertainty using parallel sampling-based motion planning. *IEEE Transactions on Robotics*, 31(1):104–116.

Sunberg, Z. and Kochenderfer, M. (2018). Online algorithms for POMDPs with continuous state, action, and observation spaces. In *Internatonal Conference on Automated Planning and Scheduling (ICAPS)*, pages 259–263.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Szita, I. and Lörincz, A. (2006). Learning Tetris using the noisy Cross-Entropy method. *Neural computation*, 18(12):2936–2941.

Temizer, S., Kochenderfer, M., Kaelbling, L., Lozano-Pérez, T., and Kuchar, J. (2010). Collision avoidance for unmanned aircraft using Markov decision processes. In *AIAA guidance, navigation, and control conference*, page 8040.

Temizer, S., Kochenderfer, M., Kaelbling, L. P., Lozano-Pérez, T., and Kuchar, J. K. (2009). Unmanned Aircraft Collision Avoidance Using Partially Observable Markov Decision Processes. Project Report ATC-356, MIT Lincoln Laboratory, Advanced Concepts Program, Lexington, Massachusetts, USA.

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):285–294.

Van Den Berg, J., Abbeel, P., and Goldberg, K. (2011). LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913.

Wang, E., Kurniawati, H., and Kroese, D. P. (2017). CEMAB: A Cross-Entropy-based method for large-scale multi-armed bandits. In *Australasian Conference on Artificial Life and Computational Intelligence (ACALCI)*, pages 353–365. Springer.

Wang, E., Kurniawati, H., and Kroese, D. P. (2018). An on-line planner for POMDPs with large discrete action space: A quantile-based approach. In *Internatonal Conference on Automated Planning and Scheduling (ICAPS)*, pages 273–277.

Wang, H., Kurniawati, H., Singh, S. P., and Srinivasan, M. V. (2015). In-silico behavior discovery system: An application of planning in ethology. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 296–305.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge England.

White, C. C. (1991). A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research*, 32(1):215–230.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wu, Z., Kolonko, M., and Möhring, R. H. (2017). Stochastic runtime analysis of the Cross-Entropy algorithm. *IEEE Transactions on Evolutionary Computation*, 21(4):616–628.

Wu, Z., Möhring, R. H., and Lai, J. (2018). Stochastic runtime analysis of a Cross-Entropy algorithm for traveling salesman problems. *Theoretical Computer Science*, 724:69–86.

Xiao, C., Mei, J., and Müller, M. (2018). Memory-augmented Monte Carlo tree search. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Young, S., Gašić, M., Thomson, B., and Williams, J. D. (2013). POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.