上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：多目标深度强化学习算法及其在任务型对话系统中的应用

学生姓名：杨闰哲

学生学号：5140309562

专　　业：计算机科学与技术（致远荣誉计划）

指导教师：俞凯教授

学院(系)：致远学院

多目标深度强化学习算法及其在任务型对话系统中的应用

摘　要

本文对多目标的强化学习问题背景、算法设计、实际应用进行了系统性地探究。在多目标强化学习问题中，智能体所接受的奖励信号是一些编码了许多可能冲突的目标的向量信号，而目标之间的相对重要在算法学习阶段是未知的。这种多目标的设定具有非常好的研究价值与应用前景。传统的强化学习限定奖励函数为一个需要精巧设计的标量函数，这限制了强化学习算法在优化目标较为复杂的实际问题中的适用性和有效性。而向量化奖励的设定提供了一种直接简洁的强化学习奖励函数设计方式，来建模复杂的优化目标。在这一设定中，智能体需要在学习阶段找到所有帕累托最优的累积回馈和对应的最优策略，并且在执行阶段能根据实时的赋予的偏好来调整策略。为实现这一目标，我们利用推广后的贝尔曼方程以及深度神经网络的表达能力，创新地提出了两种多目标强化学习算法，来高效地学习可控参数化的最优策略的表示。针对多目标强化学习算法在学习阶段和执行阶段的表现，我们提出了算法“覆盖率”和“迁移质量”两个量化指标，并在合成任务以及任务型对话系统对话系统中进行了测评。实验结果验证了我们算法的有效性和可应用性。

关键词：强化学习　多目标优化　神经网络　收缩映射定理　任务型对话系统

DEEP MULTI-OBJECTIVE REINFORCEMENT LEARNING AND ITS APPLICATION IN TASK-ORIENTED DIALOGUE SYSTEMS

ABSTRACT

In the thesis, we study a new type of reinforcement learning (RL) problem where an agent receives vectorial rewards that encode many possible competing objectives with unknown relative importance during learning. This setting is worth exploring since the complicate scalarized reward engineering of conventional reinforcement learning is often infeasible in practice. The goals of a multi-objective reinforcement learning agent are two-fold: first, in the learning phase, to find all the potential optimal rewards in the convex coverage set of environment's Pareto frontier; second, in the execution phase, to adapt its policy optimally corresponding to any real-time specified preferences. We propose two novel algorithms which exploit generalized Bellman equations and the expressiveness on neural networks for sample-efficiently learning single controllable parametric representation of all optimal policies. Quantitative evaluation on synthetic tasks and a task-oriented dialogue system with proposed two metrics, coverage ratio and adaptation quality, verifies the validity and applicability of our algorithms.

KEY WORDS: Reinforcement Learning, Multi-Objective Optimization, Neural Networks, Banach Fixed-Point Theorem, Task-Oriented Dialogue Systems

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

$s \in \mathcal{S}$	State in State Space
$a \in \mathcal{A}$	Action in Action Space
$\mathcal{P}(s'\|s,a)$	Stochastic Transition Kernel
$\boldsymbol{r}(s,a)$	Vectorized Immediate Reward Function
m	number of objectives
γ	Discount Factor
τ	Trajectory
$\pi(a\|s)$	Policy
Π	Stationary Policies
$\hat{\boldsymbol{r}}$	Discounted Cumulative Reward
$\boldsymbol{V}^{\pi}(s)$	Value Function
$\boldsymbol{Q}^{pi}(s,a)$	Q-Value Function
$\boldsymbol{V}^{*}(s)$	Optimal Value Function
$\boldsymbol{Q}^{*}(s,a)$	Optimal Q-Value Function
$\mathcal{T}_{\pi}$	Evaluation Operator
$\mathcal{T}$	Optimality Operator
$f_{\omega}(\cdot)$	Linear Preference Function
$\boldsymbol{\omega} \in \Omega$	Relative Importance Weights
$\mathcal{F}^{*}$	Pareto Frontier
Π^{*}	Pareto Optimal Policies
CCS	Convex Coverage Set of Pareto Frontier
Π_{CCS}	Convex Optimal Policies
$\Pi_{\mathcal{L}}$	Linear Optimal Policies
N_{ω}	Number of Sampled Preferences
$\mathcal{D}_{\omega}$	Preference Distribution
$L(\theta)$	Loss function
∇_{θ}	Gradient Operator
$\mathbb{E}[\cdot]$	Expectation
p_{λ}	Homotopy Path

Chapter 1 Introduction

Although we have been impressed by those popular scientific fictions telling that the rise of artificial intelligence will bring an end to our human civilization, in reality, we still envision a bright future where humans and intelligent systems exist and depend on each other in perfect harmony. Intrigued by the various challenges on the path towards this better version future, researchers have developed many theories and techniques for intelligence systems of service to humanity. One central concept that these theories and techniques emerge around is the idea of the autonomous agent. An agent, whose definition in the textbook is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors"[1], plays a crucial role in artificial intelligence research in the past decades. Autonomous agents can help us in many meaningful ways. For example, agents can control manufacturing machines, in order to produce products for a company; drive cars without of intervention from human drivers, trade goods or services on markets and assist in scientific discovery. Hence, autonomous agents have enormous potential impacts on improving the quality of our social and personal life.

So far, we are still on this promising path of designing better autonomous agents, typically computer software as well as their supportive hardware, that interact with the environment with little or without human control or intervention. Recent progress in deep reinforcement learning has witnessed the incomparable success of many exciting and challenging areas. With this technology, we have built many autonomous agents, to master the game of GO without human knowledge[2], to beat the human in many video games[3], to automate synthesis planning in chemistry[4], and assist neuroscientist to study the function of grid cells[5]. We can even use this technology to build conversational systems[6] to directly talk with us to satisfy our emotional or informational demands. The future described in those scientific fictions is seemingly coming true step by step. Should we worry about its terrible side, the horrible "out of control"?

One case that we should be on alert for is when an autonomous agent needs to make a decision under multiple competing objectives. Consider a man crossing the road in front of an autonomous car with a passenger at high speed. The car has to make a decision between avoiding hitting the man by risking its passenger's life or kill that man to ensure its passenger's safety. These two objectives are conflicting. No matter which decision the car makes eventually, this will become a terrifying story — the autonomous agent decides to take human's life. We have no chance to prevent this kind of tragedy, particularly if the autonomous car is empowered by the current reinforcement learning technology. A reinforcement learning agent always aims at maximizing a specific scalar reward signal, which ignores such extreme case, and is designed as some fixed combination of multiple objectives.

The harmonious future for humans and intelligence systems relies on how we take care of the multi-objective issue. First, the relative importance weight of each objective is unknown when we design and train an autonomous agent. Second, if the designer fixes the weights according to its prior knowledge or experience, that weight assignment for objectives may lead to unexpected results in practice. An acceptable

solution is that we should let users, not the agent designer, to express their preferences for different objectives to control the agent's behavior. Moreover, the agents are therefore required to perform optimally under any preference given by users when execution.

In this thesis, we scholarly discuss the possibilities to realize this idealized solution, based on the recent development of deep reinforcement learning. We formalize a simplified scenario of multi-objective reinforcement learning, where the agent needs to adapt its policy to the user's preferences. Theoretical analysis and experimental evaluation are performed on two proposed algorithms aiming at solving that particular scenario. We also apply our algorithms in task-oriented dialogue systems to verify the validity and applicability of our approaches.

1.1 Motivating Scenarios

Just think about how we learn to do everything, e.g., swimming. Our coaches never tell the only objective of swimming is the speed, or the fancy style. Swimming is a task involves many objectives — speed, stability, endurance, energy efficiency, the beauty of style, etc. And our coaches never teach us which one among all of these objects is most important, let along the relative importance weights for them. But we swim well.

We learn how to swim not by maximizing a certain criterion. Conversely, we learn it by exploring the possibilities for all objectives. If we try to swim fast, we will soon be tired and therefore the endurance and stability objectives will be impaired. If we try to swim for a long time, we need to keep better energy efficiency and move at a relatively low speed. In fact, we learn swimming by learning how to make compromises among many objectives. We can adapt to different environment, because we know how to make these compromises to gain better utilities under some preferences. Like when our coaches are watching us swimming, we will pay more attention to the beauty of style, and try to be the fastest one in the class, even though we are almost exhausted.

Reflections on how we learn to give us inspiration. Learning for a task involving multiple objectives is not a simple optimization problem. The reward is vectorized, and it requires a period during which the learner can freely explore, and after that it can adapt to the environment with different preferences. This implication is important to build a controllable or adaptive intelligent agent in the multi-objective settings. The agent needs to explore and learn all potentially optimal solutions with unknown preferences, then reacts corresponding to certain user preferences. In this type of scenario, the agent can truly master the task.

There are several real-world application scenarios support this explore-and-execute setting for multi-objective learning. Think about a mining company mines gold and silver from several mines located in the mountains. This company has only a few vans for transporting miners, and must decide to which mine each van should go every morning. Miners are more efficient if there are more colleagues at the same mine. Since the company aims to maximize revenue, the best strategy depends on the fluctuating prices of gold and silver. To maximize revenue, The mining company wants to use an autonomous agent, which learns from simulation with every price change, to respond with optimal real-time strategies without recomputing.

As the second example, a task-oriented dialogue system is developed without specified preferences on dialogue success rate and dialogue brevity. Therefore, the system needs to explore all possible optimal

conversational strategies under different preferences. The preference will be given by users and specific application scenes after that. Like when a user is trying to ask information about the weather, then the user probably hope the dialogue system can offer her thorough information. And in this case, the success objective is more important than the brevity objective. However, when a user is driving and what to ask for direction information, then the brevity of the dialogue would be critical since the drive does not want to interact too many rounds. In the thesis, we summarize these explore-and-execute settings as *delayed linear preference scenarios*, and study how to build efficient learning algorithms for these scenarios.

1.2 Research Questions

In the thesis, our goal is to answer the following question:

"Can we design efficient and practical multi-objective reinforcement learning algorithms in the delayed linear preference scenario, for acquiring all potentially optimal policies in the learning phase, and adapting optimally to any real-time specified preference in the execution phase?"

To answer this question, we need to formalize the delayed linear preference scenario and build a theoretical framework to analyze and design new types of reinforcement learning algorithms in the multi-objective settings. Besides, how to evaluate the performance of proposed algorithms also involves many critical research problems. What evaluation environment we should use to test our algorithms? What is the evaluation metric? Finally, the applicability of our proposed algorithms needs to be verified in a task-oriented dialogue policy learning task.

1.3 Contribution and Outlines

In this section, we online the organization of this thesis and contributions we present. Our contributions can be divided into three categories: *theory contributions*, *evaluation contributions*, and *application contributions*.

In Chapter 1 (this introduction), we have introduced and motivated multi-objective reinforcement learning problems with kind of philosophical thinking and motivated the explore-and-execute setting for a realistic learning process. Furthermore, we have summarized our research questions.

Chapter 2 provides the background on multi-objective reinforcement learning and the definition of *delayed linear preference scenario*, which characterizes most realistic situations for multi-objective reinforcement learning. In this scenario, we only consider linear preferences parameterized by the relative importance weights of objectives. The delayed linear preferences scenario begins with a learning phase, where the agent is free to access the historical trajectory records or interacts with the simulated environment to learn about the rewards, while the preference is unknown. The aim of the agent in this phase is to find all linear optimal policies. Then in an analysis phase, the user can analyze the trade-off between multiple objectives of the task for determining a suitable preference using the information learned by the agent. The last phase of the delayed linear preference scenario is the execution phase. In this phase, a specific linear preference function will be given to the agent, while learning is no longer allowed. The agent needs to respond with an optimal policy achieving the best utility to the given preference. We also introduce related work and

state our research focus in this chapter.

Chapter 3 presents theoretical results we have. We propose a theoretical framework for developing and analysis value-based reinforcement learning algorithms. This framework is inspired by the Banach's fixed point theorem. Five essential components are involved in this framework for designing value-base reinforcement learning algorithms. Many existing value-based reinforcement learning algorithms can be explained under this theoretical framework. We use the framework to help us develop two value-based algorithms, *naive deep MORL algorithm* and *envelope deep MORL algorithm*, for solving the delayed linear preference scenario. Our proposed two algorithms are novel in the field of multi-objective reinforcement learning.

Chapter 4 evaluates the performance of deep multi-objective reinforcement learning in the delayed linear preferences scenario, by proposing two quantitative evaluation metrics and creating several synthetic environments. The *coverage ratio* (CR) is proposed to measure a MORL agent's ability to find all the potentially optimal solutions in the convex coverage set in the learning phase, and the *adaptation quality* (AQ) is proposed measure agent's ability to adapt its policy to real-time specified preferences in the execution phase These two metrics are technically sound and would be useful to future MORL algorithms research, especially for the ones focusing on delayed linear preference scenarios. Three synthetic multi-objective reinforcement learning environments are created to test our deep MORL algorithms. We compared proposed two deep MORL algorithm in two synthetic environments.

Chapter 5 demonstrates an application of our deep MORL algorithms in the task-oriented spoken dialogue system. Adaption to user preferences and balancing multiple objectives is rarely considered for task-oriented dialogue policy learning. We consider the dialogue learning process as a delayed linear preference scenario of multi-objective reinforcement learning. The rewards in this setting are vectors describing dialogue length penalty for the brevity objective and task success reward for the success objective. Our proposed deep MORL algorithm can learn almost all optimal policies during a learning phase, and adapt to user preference without further learning when the system is deployed online. Our experiments confirm the applicability and show potential advantages of our proposed deep MORL algorithms in dialogue policy learning. The task-oriented dialogue system using our deep MORL algorithm will be more favored by users in real application scenarios.

Chapter 6 enumerates the main conclusions and contributions of this thesis, discusses the implications for further work in multi-objective reinforcement learning and dialogue policy learning, and identifies potential directions for future work.

Chapter 2 Background

This chapter provides background on multi-objective reinforcement learning problems. First, in Section 2.1, we introduce the mathematical descriptions of traditional reinforcement learning setting, *Markov decision processes* (MDPs), as well as how classical planning and learning algorithms work in this single objective setting, and how they can be extended to more general large state space cases with the help of deep neural networks. We then, along parallel lines, formulate the multi-objective reinforcement learning problem as a *Multi-Objective Markov Decision Processes* (MOMDPs) in Section 2.2. Distinct from the traditional reinforcement learning problems, which can be treated as optimization problems, the optimality concepts in the multi-objective reinforcement learning can be defined in various ways and selected by different types of preference functions. Our thesis only pays attention to the cases where the preference functions are linear, and unknown during learning processes. This special case is named *delayed linear preference scenario* and defined triphasically in this section. We then discuss differences between our works and other related works focusing on improving reward function design of reinforcement learning in Section 2.3. Finally, in Section 2.4, we highlight the focuses of our multi-objective reinforcement learning research.

2.1 Reinforcement Learning

Reinforcement Learning[7], generally speaking, is a process of learning how to make sequential decisions — how an intelligent agent responds to situations with actions, to maximize cumulative numerical reward signals. The learner carrying this kind of learning process must discover which actions may yield the greatest cumulative reward in the future by trial-and-error search. Besides, the reward singles are often delayed in the most cases of interest, which means a greedy policy always maximizing the immediate reward would be suboptimal. To address the problem of reinforcement learning, scholars often formalize it using ideas from the theory of dynamical systems, as the optimal control of *Markov decision processes* (MDPs). Details of this standard mathematical formalization would be introduced in Section 2.1.1. The goal of traditional single objective reinforcement learning is to find the optimal policy. Therefore, evaluation and control are two elements invented mathematically to achieve this goal. We introduce this classical value-based reinforcement learning approach in Section 2.1.2. Other policy-based approaches are also briefly mentioned in this section to provide readers with a comprehensive overview.

Recent advances in reinforcement learning have brought the incomparable success to many interesting and challenging areas, such as mastering the game of GO without human knowledge[2], beating human in many video games[3], automating synthesis planning in chemistry[4], and assisting neuroscientist to study the function of grid cells[5]. All of these great successful applications of reinforcement learning are due to the reinvention of artificial neural networks, which is much deeper and more affordable than that of two decades ago. This essential driving power is so-called *deep learning*[8]. Deep learning enables the machine to learn to approximate functions, especially, when meeting with reinforcement learning, to approximate the

value functions or parametric policies (as introduced in Section 2.1.2). We show readers how these basic deep reinforcement learning algorithms work in Section 2.1.3. All the content mentioned in Section 2.1 is fundamental to begin our voyage on deep multi-objective reinforcement learning.

2.1.1 Markov Decision Processes (MDPs)

In a reinforcement learning problem, we consider an agent interacting with the environment in a Markovian setting: at each step, the agent determines which action to take based on its current observation about the environment and itself, to which the environment reacts with a reward signal and transits to the next status. Just think about a robot trying to move from the desk to the door in a room. Valid actions it may take at each step are movements in four directions. It has to decide which direction it should go every step, base on the observation of its current location in the room. The assumption behind this setting is that the environment is *observable*, and all the necessary information for decision making is encoded in this observation of status, or usually, we call this observation a "state". In the robot example, the states are possible locations of an agent in that room. The reward signal in the conventional reinforcement learning settings should be a scalar. Like for the robot routing in the room, the reward signal could be defined as the opposite number of distance between its current location and the door. The closer to the door the robot is, the stronger the reward signal will be.

More specifically, a *Markov decision process* (MDP) models interactions between an agent and an environment with single objective, which can be defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$. In the tuple,

- $\mathcal{S}$ is a *state space*. $s \in \mathcal{S}$ is a state. $\mathcal{S}$ can be a discrete set or a continuous space depending on the task. For some tasks, we may specify a state $s_0 \in \mathcal{S}$ as the fixed *initial state*, which the interaction always starts with.
- $\mathcal{A}$ is an *action space*. $\mathcal{A}$ also can be discrete or continuous depending on the task.
- $\mathcal{P}$ is a *stochastic transition kernel*. $\mathcal{P}(s'|s, a)$ depicts the transition probability from the state $s \in \mathcal{S}$ to the next state $s' \in \mathcal{S}$ when the action $a \in A$ is taken.
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function*. $r(s, a)$ describes the immediate reward the will receive when taking an action $a \in \mathcal{A}$ on a state $s \in \mathcal{S}$. r can be a stochastic function. However, in the most cases, we prefer a deterministic form of r, and returns of this kind of reward function can be interpreted as expected immediate rewards.
- $\gamma \in [0, 1)$ is a *discount factor*, which quantifies the relative importance between short-term and long-term rewards. We will explain this part later.

The *discounted total reward* or *discounted cumulative reward* is often calculated by

$$\hat{r}_\tau := \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t), \tag{2–1}$$

when the agent execute a *trajectory* $\tau = \{(s_t, a_t)\}_{t=0}^{\infty}$. Here we assume the interaction has infinite rounds. Therefore, this model is also called infinite-horizon Markov decision process. The finite cases can be regarded as special cases of this model. Discount factor γ plays a role in the equation 2–1. When t goes larger, γ^t becomes smaller and smaller, which impair the relative importance of the reward from the far future. From another perspective, adding a discount factor $\gamma \in [0, 1)$ guarantees the convergence of equation 2–1.

The goal of the reinforcement learning agent is to find a mapping between states and actions which can produce trajectories maximizing this discounted total reward. This mapping is usually named as a *policy*. We will discuss it in the next section.

2.1.2 Policy: Evaluation and Control

An agent performs its actions under a certain strategy. This strategy instructs it to select actions based on current state, to maximize the expected total reward according to the description of Markov decision process mentioned above. This strategy can vary over time or be steady. Commonly, we desire to obtain a time-independent strategy, or say, a *stationary policy*.

Stationary policies Π contains all the functions that map each state $s \in \mathcal{S}$ to a probability distribution over action space $\mathcal{A}$. For any $\pi \in \Pi$, $\pi(a|s)$, or $\pi(s, a)$, indicates the probability that an agent will take the action $a \in \mathcal{A}$ in the state $s \in \mathcal{S}$. We define the *value function* of a stationary policy π (only shortly say policy in the rest of thesis) given the underlying MDP $\mathcal{M}$,

$$V^{\pi}(s) := \mathbb{E}_{\tau \sim (\mathcal{P},\pi)|s_0=s}\left[\hat{r}_{\tau}\right] = \mathbb{E}_{\tau \sim (\mathcal{P},\pi)|s_0=s}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]. \tag{2–2}$$

The value function denotes the expected discounted total reward gained by performing a policy π. This concept is commonly used for evaluating and analyzing a policy, or searching for *optimal policy*. In reinforcement learning, the stochastic transition kernel $\mathcal{P}$ is often unknown (some people call this model-free reinforcement learning), the latter goal could be formulate as the task for solving $\arg\max_{\pi \in \Pi} \tilde{V}^{\pi}(s_0)$, where $\tilde{V}^{\pi}(s_0)$ is an estimation of the real value function and s_0 is some fixed initial state. We also define $V^*(s) := \max_{\pi \in \Pi} V^{\pi}(s)$ and π^* as the corresponding optimal policy.

How do we find the optimal policy? This question actually can be decomposed into two basic questions: first, how do we evaluate a policy, estimate its value function? Second, how do we control and improve the current policy, if the value function estimation is acquirable. Almost all the existing model-free reinforcement learning approaches are solving the problem by answering these two questions. According to their differences on this two dimensions, our approaches can generally be divided into two categories[9], policy-based approaches and value-based approaches.

The policy-based approaches, such as REINFORCE[10], is that they directly compute $\tilde{V}^{\pi}(s_0) = \frac{1}{N}\sum_{i=1}^{N} \hat{r}_{\tau_i}$ or other equivalent estimators by sampling for evaluating a policy π, and directly optimize the quantity of interest via ascending along the policy gradients

$$\boldsymbol{\eta} \propto \nabla_{\theta} \mathbb{E}_{\pi_{\theta}}\left[\tilde{V}^{\pi_{\theta}}(s_0)\right] = \mathbb{E}_{\pi_{\theta}}\left[\nabla_{\theta} \log \pi_{\theta} \cdot \tilde{V}^{\pi_{\theta}}(s_0)\right], \tag{2–3}$$

where $\boldsymbol{\eta}$ is the step size for updating policy parameters by $\theta \leftarrow \theta + \eta$. These approaches are straightforward to the optimization task, that is a primary advantage. However, their main drawback is sample inefficiency. Because they estimate the policy gradient from rollouts for evaluation, the variance is usually extremely large. A popular solution to this problem is to use other value-based evaluation to replace rollout estimates, in turn at the cost of bias. This solution is called actor-critic methods[11, 12] We will introduce value-based evaluation

later. Another drawback is the inherent sample inefficiency of *on-policy* learning, the policy evaluation used for optimization is strictly for and can only obtain with the current policy, otherwise significant biases will be incurred[13].

By contrast, value-based approaches indirectly evaluating and optimizing the policy, like Q-learning[14], can overcome this *on-policy* limitation. These methods support the *off-policy* settings to learn from any trajectory sampled from the same environment. The fundamental idea to the value-based approaches is the use of *Bellman's equation*[15], which describes the relationship between a value function, and a so-called *Q-value function*. Similar to the definition of the value function, a Q-value function of a policy π is

$$Q^{\pi}(s,a) := \mathbb{E}_{\tau\sim(\mathcal{P},\pi)|s_0=s,a_0=a}\left[\hat{r}_{\tau}\right] = \mathbb{E}_{\tau\sim(\mathcal{P},\pi)|s_0=s,a_0=a}\left[\sum_{t=0}^{\infty}\gamma^{t}r(s_t,a_t)\right], \tag{2–4}$$

which describes the expected discounted total reward from taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, then acting according to policy π. As we expand their definitions, the value and Q-value have following alternatively recursive relations

$$(i.)\ Q^{\pi}(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}V^{\pi}(s'), \quad (ii.)\ V^{\pi}(s') = \mathbb{E}_{a'\sim\pi(\cdot|s')}Q^{\pi}(s',a'). \tag{2–5}$$

Combining equation 2–5 (i) and (ii), we derive the *Bellman expectation equation*

$$Q^{\pi}(s,a) = r(s,a) + \gamma\mathbb{E}_{\mathcal{P},\pi}Q^{\pi}(s',a'). \tag{2–6}$$

This recursive equation of Q-value function provide us with an efficient way to evaluate a policy using out latest guess about its Q-value $Q_k^{\pi}(s,a)$, that is

$$Q_{k+1}^{\pi}(s,a) \leftarrow r(s,a) + \sum_{s'\in\mathcal{S},a'\in\mathcal{A}}\mathcal{P}(s'|s,a)\pi(a'|s')Q_k^{\pi}(s',a'). \tag{2–7}$$

As we repeat the equation 2–7, the estimate of Q-value function will be monotonically preciser, which has a theoretical guarantee that we will explain in Section 3.1.

Moreover, let $Q^*(s,a)$ denote the optimal Q-value function, similar recursive equations describing the relationship between the optimal value function and the optimal Q-value function,

$$(i.)\ Q^{*}(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}V^{*}(s'), \quad (ii.)\ V^{\pi}(s') = \max_{a'\in\mathcal{A}}Q^{*}(s',a'). \tag{2–8}$$

Equations (i) and (ii) in 2–8 inspires us to conclude the *Bellman optimality equation*,

$$Q^{*}(s,a) = r(s,a) + \gamma\mathbb{E}_{\mathcal{P}}\max_{a'\in\mathcal{A}}Q^{*}(s',a'). \tag{2–9}$$

The classical Q-learning utilizes this equation to perform

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\left(r(s,a) + \gamma\max_{a'\in\mathcal{A}}Q_k(s',a') - Q_k(s,a)\right), \tag{2–10}$$

when $\{s, a, s'\}$ is a step in a given trajectory, to converge to the optimal Q-value $Q^*(s,a)$, where α is a step size parameter, and the term $r(s,a) + \gamma\max_{a'\in\mathcal{A}}Q_k(s',a') - Q_k(s,a)$ is called *time-difference error* (TD-error). The corresponding deterministic optimal policy can be extracted by $\pi(s) = \arg\max_{ain\mathcal{A}}Q^{*}(s,a)$.

The key drawback of value-based methods is that we have to maintain the value function or the Q-value function. When the state and action spaces are small and discrete, we can use a tabular method to maintain the Q-value function. However, when the state space is large or continuous, we need to seek help from function approximation. That is why deep neural networks can bring so much vigor to reinforcement learning. And we will see how deep learning and reinforcement learning interact in the next section.

2.1.3 Deep Neural Networks as Function Approximators

In the real world, intelligent agents are confronted with tasks associated with high-dimensional sensory inputs. To tackle these real tasks, traditional value-based reinforcement learning is limited by its ability to derive an efficient representation of the environment state, store the information of value function, and then generalize past experience to new situations. Fortunately, the recent emergence of deep neural networks provides us with great tools to represent the high-dimensional state, approximate the value function, and generalize on new unseen states. By feeding sufficient data into deep learning models, they can learn better representations than using handcraft features[16]. This motivates the new approach of deep reinforcement learning[17].

In this approach, a neural network function approximator with parameters θ is used to estimate the Q-value function, as defined in Section 2.1.2. We refer this neural network as a *Q-network*. A Q-network can be trained by minimizing a series of loss functions

$$L_k(\theta) = \mathbb{E}_{s,a}\left[(y_k - Q(s,a;\theta))^2\right], \tag{2–11}$$

which changes at each iteration k, where $y_k = \mathbb{E}_{s'}\left[r(s,a) + \gamma \max_{a'} Q(s',a';\theta_k)\right]$ is the target of iteration k. The target is fixed during optimizing the loss function. And the parameters of the Q-network are updated by $\theta_{k+1} \leftarrow \theta_k - \eta$, where

$$\eta \propto \nabla_{\theta=\theta_k} L_k(\theta) = -\mathbb{E}_{s,a,s'}\left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q(s',a';\theta_k) - Q(s,a;\theta_k)\right) \nabla_{\theta=\theta_k} Q(s,a;\theta)\right]. \tag{2–12}$$

This algorithm is model-free and off-policy. In practice, the exploration is made via ϵ-greedy strategy which follows the greedy strategy $a_{t+1} = \max_{a \in \mathcal{A}} Q(s_t, a_t; \theta_k)$ with probability $1-\epsilon$ and selects action randomly with probability ϵ. A technique known as *experience replay* is also used in the deep reinforcement learning. The agent's trajectories associated with immediate rewards are stored in a replay buffer $\mathcal{D} = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\}_{t,i}$. When applying minibatch Q-network updates, we sample experiences from $\mathcal{D}$ to estimate equation 2–12.

This approach brings several advantages to traditional reinforcement learning. First, due to the experience reply, each step of experience can be used in many times of Q-network updates, which improves sample efficiency. Second, the state space of reinforcement learning now can be easily extended to any high-dimensional space, such as visual input. The flexibility of the architecture design in deep learning easily extends to the design of Q-network. Therefore, deep reinforcement learning would be a popular method for solving sequential decision-making problem in a wide range of scientific and industrial fields.

Although we have witnessed the great success of deep reinforcement learning, the mainstream reinforcement learning approaches are restricted in scalarized reward settings, where the reward function has to return a scalar. This restriction indicates that we have to undertake the heavy reward engineering to combine different

attributes otherwise the reinforcement learning may fail[18]. A better solution to encode multiple potentially competing objectives into the reward function design is needed. Nevertheless, in some applications, the relative importance weights of different attributes may change over time. If a reinforcement learning agent is trained using one reward with specific relative weights, its policy is not easily adaptive. Therefore, in the next section, we discuss an extension of conventional reinforcement learning, which allows the reward singles to be vectors encoding multiple attributes directly.

2.2 Problem Formulation

In this section, we give the mathematical formulation of the multi-objective reinforcement learning (MORL), where the agent interacts with environment receiving vectorial rewards. In contrast with conventional reinforcement learning settings, this reward design would enable the encoding of the reward for multiple potentially conflict objectives without specifying their relative importance in advance. Similar to the conventional reinforcement learning, the whole process of interaction can be modeled as an extended Markov decision process, known as multi-objective Markov decision process (MOMDP). We will introduce the details of this mathematical model in Section 2.2.1 parallel to the previous introduction of MDP. Since we are using vectorial rewards in this problem setting, our goal is no longer a straightforward optimization problem. There might be many *optimal* policies in the multi-objective reinforcement learning. To select the desired policy, we have to specify a certain preference. Optimality concepts and preferences functions are described in Section 2.2.2. Finally, multi-objective reinforcement learning may have different application scenarios, according to the whether the preferences will be given or not. Here we focus on one kind of common scenario — *delayed linear preference scenario*, defined in Section 2.2.3, where preferences are unknown during the learning phase and will be revealed during the execution phase.

2.2.1 Multi-Objective Markov Decision Processes (MOMDPs)

Let us consider that robot example again. Now the task for it is not to move from the desk to the door, but to move to some location on the line segment between the desk and the door in a room. However, when learning by itself, the robot does not know the destination, unless its owner gives him an instruction, “Go to the one-third of that line, closer to the door!". Of course, the owner will not be crazily patient to instruct robot for training to move to *every* where on the line segment. The robot should learn all possible solutions by itself. And when its owner is back to give an order, it should be able to perform perfectly.

As your notice, the state space is not changed at this time. It is still a set containing possible locations in that room. And the action space and the transition kernel is also not changed. The actions are movements in four directions, and the transition kernel is common Euclidean rules plus some noise. The only thing changed is the reward function — it is no longer a single scalar describing how far it is to the door. The reward function here should at least encode two objectives: the distance to the door, and the distance to the desk. These two objectives are conflicting, since the physical constraint does not allow a robot to be both close to the door and the desk at the same time. While learning, the relative importance weights for these two objectives

are unknown, unless an owner specifies them when execution. This setting, similar to the previous MDP formulation, can be formalized as a *multi-objective Markov decision process*[19] (MOMDP), which describes interactions between an agent and an environment with multiple objectives. We provide formal definitions.

Definition 2.1 (MOMDPs)**.** The multi-objective Markov decision process is defined by a five-element tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \boldsymbol{r}, \gamma \rangle$. In the tuple,

- $\mathcal{S}$ is a *state space*. $s \in \mathcal{S}$ is a state. $\mathcal{S}$ can be a discrete set or a continuous space depending on the task. For some tasks, we may specify a state $s_0 \in \mathcal{S}$ as the fixed *initial state*, which the interaction always starts with.
- $\mathcal{A}$ is an *action space*. $\mathcal{A}$ also can be discrete or continuous depending on the task.
- $\mathcal{P}$ is a *stochastic transition kernel*. $\mathcal{P}(s'|s, a)$ depicts the transition probability from the state $s \in \mathcal{S}$ to the next state $s' \in \mathcal{S}$ when the action $a \in A$ is taken.
- $\boldsymbol{r} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$ is the *reward function*, where m is the number of potentially competing objectives encoded in the reward. $r(s, a)$ describes the immediate reward the will receive when taking an action $a \in \mathcal{A}$ on a state $s \in \mathcal{S}$. r can be a stochastic function. However, in the most cases, we prefer a deterministic form of r, and returns of this kind of reward function can be interpreted as expected immediate rewards.
- $\gamma \in [0, 1)$ is a *discount factor*, which quantifies the relative importance of short-term and long-term rewards, as we will explain in above article.

Definition 2.2 (Discounted Total Rewards)**.** The *discounted total reward* or *discounted cumulative reward* of an agent acting following a MOMDP is the following summation

$$\hat{\boldsymbol{r}}_\tau := \sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}(s_t, a_t),$$

when the agent execute a *trajectory* $\tau = \{(s_t, a_t)\}_{t=0}^{\infty}$, where $\gamma \in [0, 1)$ is the discount factor.

As in conventional Markov decision process, in multi-objective Markov decision process, we also aim at finding the good time-independent policy. A policy and its corresponding value-function in the multi-objective Markov decision process are defined as follows.

Definition 2.3 (Stationary Policies)**.** The set of stationary policies Π of a given multi-objective Markov decision process $\mathcal{M}$ is all the functions that map from state space to a probability distribution over actions space $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. For any $\pi \in \Pi$, $\pi(a|s)$, or $\pi(s, a)$, indicates the probability that an agent following policy π will take the action $a \in \mathcal{A}$ in the state $s \in \mathcal{S}$. The the policy is *deterministic*, we will simply use the notation $\pi(s) = a$ to describe that the agent will take the action a in the state s with probability one.

Definition 2.4 (Value Functions)**.** Give policy π and the underlying MOMDP $\mathcal{M}$, the value function of π is defined by

$$\boldsymbol{V}^\pi(s) := \mathbb{E}_{\tau \sim (\mathcal{P}, \pi)} [\hat{\boldsymbol{r}}_\tau] = \mathbb{E}_{\tau \sim (\mathcal{P}, \pi)} \left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}(s_t, a_t) \right],$$

which indicates the expected discounted total reward when performing the policy π with the initial state s. Notice that the value function returns a vector. We use $V_i^\pi(s)$ to denote the total reward of the i-th objective.

In the goal of reinforcement learning is always finding the *optimal* policy. In the single-objective settings, this goal can be expressed as $\arg\max_\pi V^\pi(s_0)$, a straightforward optimization problem. However, in the multi-objective settings, the definition of policy optimality is not clear. The vectorized value functions do not keep a linear order to support maximization. We will discuss the optimality concepts of multi-objective reinforcement learning policy in the next section.

2.2.2 Optimality Concepts and Preferences Functions

In multi-objective reinforcement learning settings, it is hard to assert one vectorial return is superior to all the others. In fact, in MOMDPs, a single policy dominating all others often does not exist. When conflicting objectives are involved, no policy could simultaneously maximize all of those objectives. For this reason, people usually employs a different dominance concept based on *Pareto optimality*.

Definition 2.5. (Pareto Optimality) A policy π is *strongly* dominated by another polity π', we often denote it as $\pi \succ \pi'$, if and only if it outperforms π on all m objectives. i.e.,

$$\pi' \succ \pi \Leftrightarrow \forall i \in [m], V_i^{\pi'}(s_0) > V_i^{\pi}(s_0).$$

Similarly, we can define the *weak* dominance $\pi' \succeq \pi$ as

$$\pi' \succeq \pi \Leftrightarrow \forall i \in [m], V_i^{\pi'}(s_0) \geq V_i^{\pi}(s_0).$$

If there is no policy π' such that $\pi' \succ \pi$, then we say that the policy π is a *Pareto optimal policy*, and its corresponding value $\boldsymbol{V}^\pi(s_0)$ is a *Pareto optimal solution*. The set of all Pareto-optimal policies is denoted by $\Pi^* := \{\pi \mid \nexists \pi' \in \Pi, \pi' \succ \pi\}$, which maps to the *Pareto frontier* $\mathcal{F}^* := \{\boldsymbol{V}^\pi(s_0) \mid \pi \in \Pi^*\}$. Sometimes we will also use the definition $\mathcal{F}^* := \{\hat{\boldsymbol{r}}_\tau \mid \tau \sim (\mathcal{P}, \pi), \pi \in \Pi^*\}$ for the *Pareto frontier* alternatively.

Figure 2–1 (a.) illustrates an two dimensional example of Pareto frontier. Every node in this figure indicates an solution (value or real discounted total reward) consisting of quantities of two objectives. Nodes {A, B, C, D, E, F, G, H} are obviously Pareto optimal solutions, since they are on the outer boundary. And the node L is obviously not a Pareto optimal solutions since it is dominated by many other solutions in this figure. One thing worth our attention is that the node K is actually a Pareto optimal solution. No solution in this figure dominates node K on both objectives 1 and 2. The Pareto frontier is unnecessary to be *convex*.

As we see, there would be many optimal solutions and their corresponding optimal policies in the multi-objective reinforcement learning problem. This causes ambiguity when an agent tries to perform an action since in a sequential decision-making process, an agent can only perform *one* policy which leads to *one* result. What if that result is optimal but not *desired*? Back to the robot example, any policy that efficiently moves to the line segment between the door and the desk is optimal regarding Pareto optimality. However, in the end, where should the robot go in one specific execution?

In fact, in the multi-objective reinforcement learning, an agent has to serve one *preference* in one execution. In the robot example, the preference is expressed by the owner. Mathematically, we would say a preference is a function which maps the m-dimensional reward to a one-dimensional scalar. With a preference

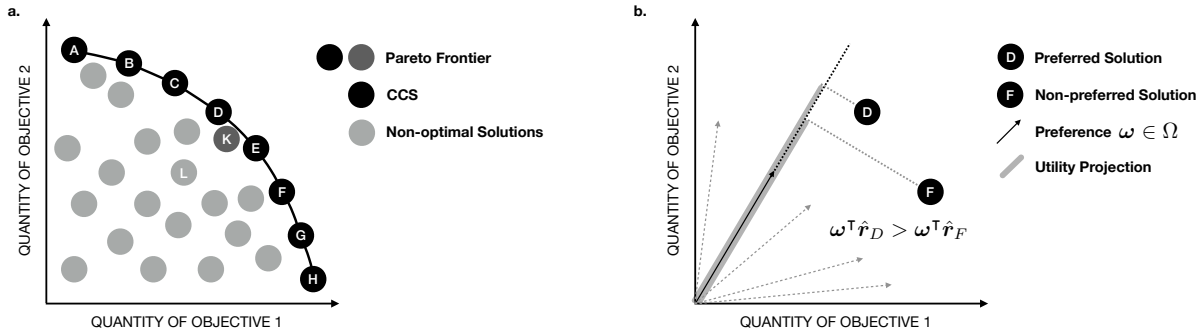

Figure 2–1 Pareto frontier, convex coverage set, and linear preference functions. (a.) Pareto frontier may encapsulate local concave parts, whereas CCS is a convex subset of Pareto frontier. (b.) Linear preferences select optimal solution from CCS with highest utility, which can be represented by the projection length along the direction of preference.

function, all the optimal solutions are comparable so that an agent can select the most desired one and perform the corresponding optimal policy towards it.

Definition 2.6 (Preference Functions)**.** A preference function $f : \mathbb{R}^m \to \mathbb{R}$ maps the value or reward consisting of quantity of m objectives in to one real scalar. Given value function $\boldsymbol{V}^\pi$ or discounted total rewards $\hat{\boldsymbol{r}}_\tau, \tau \sim \pi$, we name the real value $f \circ \boldsymbol{V}^\pi(s)$ or $f(\hat{\boldsymbol{r}}_\tau)$ the policy's *utility* under preference f.

There are many types of preference functions. Such as a max preference $f(\boldsymbol{r}) = \max_{i\in[m]} r_i$, to express a greater liking to the one performing extremely outstanding on some objective, a min preference $f(\boldsymbol{r}) = \min_{i\in[m]} r_i$, to express a greater liking to the one has no flaw on all the objective, and a linear preference $f_\omega(\boldsymbol{r}) = \boldsymbol{\omega}^\intercal \boldsymbol{r}$, which is parameterized by the *relative importance weights* $\boldsymbol{\omega} \in \Omega$, and Ω is an $(m-1)$-simplex. The goal of the agent is to find policy maximizing the gained reward singles under a certain preference f, or say, to maximize the *utitility*. Therefore, we can rewrite the this goal when a preference f is given as $\arg\max_\pi f \circ \boldsymbol{V}^\pi(s_0)$, and its solution π_f is called *preferred optimal policy*.

Suppose we already know the preference function f, how much would the MOMDP be different from the single-objective MDP? Intuitively, since the preference function provides a viable scalarization scheme, can we simply conver a MOMDP to an MDP by replacing multi-dimensional immediate rewards $\boldsymbol{r}(s,a)$ with $f \circ \boldsymbol{r}(s,a)$? At the first glance, we may believe the answer to be yes, but that positive result does not hold true for all preference function. Note that given a reward sequence $\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots$, and its discounted cumulation $\hat{\boldsymbol{r}} = \sum_{t=0}^\infty \gamma^t \boldsymbol{r}_t$, we will find $f(\hat{\boldsymbol{r}}) \neq \sum_{t=0}^\infty \gamma^t f(\boldsymbol{r}_t)$ when f is not additive or homogeneous. In this case, a simple conversion may not exist. Therefore, different preference functions may lead different levels of difficulty for solving MOMDPs.

The majority of multi-objective reinforcement learning algorithms proposed so far are in the scenario that f is specified[20]. Since they all aim at finding a single policy which is most preferred on the Pareto frontier, these algorithms are categorized into so-called *single-policy* algorithms. One fundamental distinction among the single-policy algorithms proposed so far is the form of f in which these preferences are expressed.

What intrigues us most is the case when preference f is not (or cannot be) specified initially. This

situation would be more common in the real application and substantially more challenging. For example, autonomous traders search for optimal policies to invest different goods in an open market and their returns constantly fluctuate. It is hard to afford expensive real-time learning corresponding to the preference defined by current returns and then react in a short time. Thus, the learning needs to be done in advance, when more computational resources are available. And after being deployed to the real market, it is able to quickly respond the real-time price with the optimal trading policy. In the next section, we will discuss a scenario closer to this kind of realistic situation. And our thesis mainly focuses on that scenario.

2.2.3 Delayed Linear Preference Scenarios

In this thesis, we are interested in the non-trivial *delayed linear preferences scenarios*. As its name indicates, in these scenarios, we only consider *linear preferences* (see definition 2.6). Suppose the agent receives a multi-objective reward $\boldsymbol{r}(s_t, a_t)$ at time t. Its *real utility* increases by $f_\omega(\boldsymbol{r}(s_t, a_t)) = \boldsymbol{\omega}^\intercal \boldsymbol{r}(s_t, a_t)$, where *relative importances weights* $\boldsymbol{\omega} \in \Omega$, and Ω is an $(m-1)$-simplex.

In this special case, a MOMDP collapses into a classical MDP once we knew a linear preference function. Furthermore, when we restrict preference functions to be linear, the preferred solutions can only come from a *convex coverage set*[20] (CCS) of the Pareto frontier.

Definition 2.7 (Convex Coverage Sets)**.** In an MOMDP, given its Pareto frontier $\mathcal{F}^*$, the convex coverage set is defined by

$$\mathsf{CCS} := \{\hat{\boldsymbol{r}} \in \mathcal{F}^* \mid \exists \boldsymbol{\omega} \in \Omega \text{ s.t. } \boldsymbol{\omega}^\intercal \hat{\boldsymbol{r}} \geq \boldsymbol{\omega}^\intercal \hat{\boldsymbol{r}}', \forall \hat{\boldsymbol{r}}' \in \mathcal{F}^*\}.$$

We use Π^*_{CCS} to denote the set of all corresponding policies.

Figure 2–1 (a.) shows an example of CCS and Pareto frontier. The CCS is a subset of the Pareto frontier, containing all the solutions settled on the outer convex boundary. When a specific linear preference is given, as Figure 2–1 (b.) shows, one solution in the CCS with the largest projection length along the direction of the relative importances weights will be selected as the preferred optimal solution.

Another key feature of a delayed linear preference scenario is that the linear preference is initially unrevealed, until a later phase of this scenario. Formally, the delayed linear preference scenario contains three consecutive phases: *learning phase*, *analysis phase*, and *execution phase*.

Learning Phase In the learning phase, the agent is free to access the historical trajectory records or interacts with the simulated environment to learn about the rewards. Here, the computational resources are relatively adequate, while the linear preferences are unknown. We are interested in finding the *linear optimal policies* $\Pi_{\mathcal{L}}$ that generate the whole CCS. A policy $\pi \in \Pi_{\mathcal{L}}$ only if it there exists at least one linear preference function f_ω, such that under this preference function, no other policy π' generates higher real utility. Put in mathematical language, this means that

$$\pi \in \Pi_{\mathcal{L}} \Rightarrow \exists\, \boldsymbol{\omega} \in \Omega, \text{s.t. } \forall \pi' \in \Pi, \boldsymbol{\omega}^\intercal \boldsymbol{v}^{\pi}(s_0) \geq \boldsymbol{\omega}^\intercal \boldsymbol{v}^{\pi'}(s_0),$$

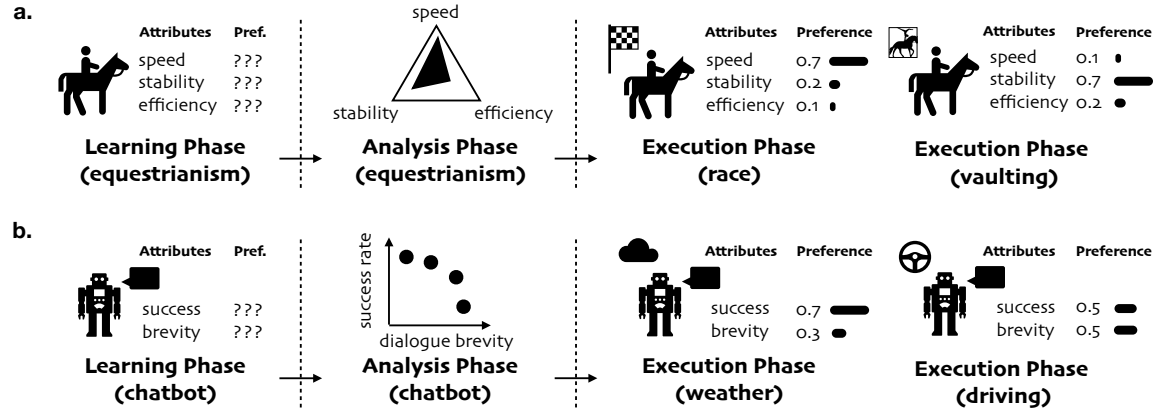

Figure 2–2 Delayed linear preference scenarios. (a.) In equestrianism (sports of horse riding) example we want to train a horse to walk and run. Three objectives, speed, stability, and efficiency are involved. (b.) A task-oriented chatbot example. Users may expect their dialogue with the chatbot to be brief or to be more informative depending on different scenes.

where s_0 is a fixed initial state. The agent in this phase is suppose to acquire $\Pi_{\mathcal{L}} \subseteq \Pi^*_{\text{CCS}}$ such that $\mathcal{F}_{\Pi_{\mathcal{L}}} = \text{CCS}$. Thus for any relative importance weight $\boldsymbol{\omega}$, one of policies in the learned $\Pi_{\mathcal{L}}$ is optimal. This phase prepares the multi-objective reinforcement learning agent for tackling various future preferences.

Analysis Phase In the analysis phase, the commander or the user can analyze the trade-off between multiple objectives of the task for determining a preference by using the information learned by the agent during the learning phase. This phase sometimes can be skipped, since the in some cases, the preference is totally up to the environment, such as the prices in a fluctuant market, which is not specified by the commander. However, for the most cases, users wish to express a preference to control or adjust the behavior of the agent, where an analysis phase is important. Available information an learned agent can offer includes the predicted or sampled CCS of objectives and the corresponding optimal real utilities for some relative importance weights of interest.

Execution Phase In the execution phase, a specific linear preference function $f_{\boldsymbol{\omega}}(\cdot)$ will be given to the agent, while learning in this phase is not allowed. The agent needs to respond with an optimal policy $\pi_{\boldsymbol{\omega}}$ from the learned set of policies $\Pi_{\mathcal{L}}$ to the given preference, using limited computational resources. The process of selecting the specific preferred optimal policy is called *policy adaptation*. The goal of efficiently selecting the preferred optimal policy by using the previously learned Π_{ω} is often not trivial since the CCS can be very large or continuous. A parametric representation $\Pi_{\mathcal{L}}(\boldsymbol{\omega}) = \pi_{\boldsymbol{\omega}}$ is more favorable in this case.

Our delayed linear preference scenario also capture a lot of realistic scenarios. Figure 2–2 presents two motivative real-life examples of delayed linear preference scenario. In the first example (a.), when training an equestrian, learning how to trade off speed, stability, and efficiency of horse riding is important, while the relative weights are unclear during training. After training, the horse rider can analysis the strength and the weakness of her horse. Then in a specific contest, the horse rider needs to perform the best riding strategy

according to a preference. In a race, the speed is the most important. However, in a vaulting, the stability is the first priority. Similarly, in the second example, a task-oriented dialogue system is developed in the learning phase without specifying preferences on dialogue success rate and dialogue brevity. The preference should be given by users and specific application scenes. Like when a user is trying to ask information about the weather, then the user probably hope the dialogue system can offer her thorough information. And in this case, the success objective is more important than the brevity objective. However, when a user is driving and what to ask for direction information, then the brevity of the dialogue would be critical since the drive does not want to interact too many rounds.

The *goal* of multi-objective reinforcement learning in delayed linear preference function scenarios is to answer two questions:

- How to efficiently learn representations for *all* linear optimal policies in the learning phase? i.e., How do we solve the multi-objective optimization and maintain potentially optimal policies?
- How do we efficiently select an optimal policy from learned representations corresponding to a specified preference in the execution phase? i.e., How to align preferences with policies?

In Chapter 3, we propose two novel value-base algorithms aiming at solving the above two questions of multi-objective reinforcement learning. And in Chapter 5, the policy learning for task-oriented dialogue systems can be abstracted as a *delayed linear preference scenario*, to which we apply our algorithms.

2.3 Related Works

In spite of the wide success in reinforcement learning theory and applications, multi-objective reinforcement learning is still a relatively unexplored topic. In this section, we introduce mainstream methods to *detour* the multi-objective reward design, as well as existing affirmative methods for dealing with multi-objective reinforcement learning problems.

2.3.1 Inverse Reinforcement Learning

Instead of directly dealing with a multi-attribute reward function, most research studies the *inverse reinforcement learning* (IRL)[18, 21] problem to learn a scalar reward function from expert's demonstrations, or directly imitate the expert's policy, bypassing the intermediate steps for solving a scalar reward function[22]. IRL is effective when the unrevealed preference among objectives is fixed, and expert's demonstrations are available. In our delayed linear preference scenarios, the unrevealed preference is *not* fixed, which can change up to different situations. Therefore, inverse reinforcement learning cannot fully solve our settings.

2.3.2 Existing Multi-Objective Reinforcement Learning Algorithms

The existing multi-objective reinforcement learning (MORL) algorithms can be roughly divided into two main categories: *single-policy* methods and *multiple-policy* methods[23, 24]. The majority of MORL algorithms belong to single-policy methods, which target at finding the optimal policy under a *known* preference among the objectives. These methods differ in forms of preference functions[25, 26]. When preferences are unknown,

multiple-policy methods are invented to practically learn a set of policies for obtaining the approximate Pareto frontier, especially the approximate CCS, when we only interested in linear preferences. The most common method is to perform multiple runs of a single-policy method by varying the preferences[27-29]. Reusing the previously learned policies and selecting next weights can improve its sample efficiency greatly. Besides, few multiple-policy methods aim at learning a set of policies $\Pi_{\mathcal{L}}$ in parallel. Several value-based reinforcement learning algorithms employ the extended Bellman equation which maintains the convex hull of the desecrate Pareto frontier[30-32]. The time and space consumption of these methods will increase problematically as the number of solution grows. Policy-based methods[33, 34] learn a manifold in the policy parameters space such that its image in the objectives space is a continuous CCS approximation.

However, all algorithms discussed above cannot fully solve the MORL in a delayed linear preference scenario, since the learned set of policies $\Pi_{\mathcal{L}}$ in convex hull methods is limited by size, and $\Pi_{\mathcal{L}}$ in manifold-based methods is difficult for policy adaptation to a given $\boldsymbol{\omega}$ in the execution phase. *Multi-objective fitted Q-iteration* (MOFQI)[35, 36] encapsulates the preference $\boldsymbol{\omega}$ in the input to the Q-function approximatior and uses expanded historical trajectories to learn multiple policies, so that it is able to construct the preferred optimal policy $\pi_{\boldsymbol{\omega}} \approx \Pi_{\mathcal{L}}(\boldsymbol{\omega})$ for any given $\boldsymbol{\omega}$. It provides a feasible approach to achieve goals of both learning and execution phases in the unknown preferences scenarios, while its Q-function approximatior can be more expressive by using deep neural networks, and the batch of trajectories could be more efficiently utilized.

2.4 Focuses

We state the focuses of our thesis as follows:

- **Model-Free Learning.** As we introduce in Section 2.1, we refer reinforcement learning as learning process based on trial-and-error. The model, especially the transition kernel and the reward function of the environment is unknown to an agent. Otherwise, the whole process is *planning* not *learning*. In multi-objective settings, we still keep this distinction and focus on the model-free learning.
- **Delayed Linear Preference Scenarios.** In the multi-objective reinforcement learning, preferences are keys to express the optimality concepts. Preferences can be given or not given at the beginning of reinforcement learning. Preference can also have many types. In this thesis, we focus on one special case called delayed linear preference scenario as stated in Section 2.2.
- **Value-Based Multiple-Policy Methods.** Different from the most existing multi-objective reinforcement learning algorithms, we focus on value-based multiple-policy methods. We choose value-based methods since it can support off-policy settings inherently and we have theoretical tools to develop this kind of algorithms (see Chapter 3). We emphasize multi-policy methods because our goals are fist finding all potentially optimal policies in the learning phase, and then selecting the one with the greatest utility under a certain preference in the execution phase.
- **Applications in Task-Oriented Dialogue Systems.** We expected our proposed algorithms to have the real impact in real life. The policy learning in task-oriented dialogue systems can be treated as a multi-objective reinforcement learning in delayed linear preference scenario. We apply our deep MORL algorithms to improve its on-line dialogue policy learning.

Chapter 3 Value-Based Methods

This chapter is about a theoretical framework of value-based deep multi-objective reinforcement learning. Our framework is inspired by Banach's fixed point theorem, which guarantees the existence and uniqueness fixed point of a *contraction* on a complete metric space. This theoretical framework provides a topological insight into how previous successful deep Q-learning works in single-objective settings, which is introduced in Section 3.1. Equipped with this theoretical tool, we develop two novel value-base multi-objective reinforcement learning algorithms in Section 3.2 and 3.3. Finally, in Section 3.4, we summarize pros and cons of these value-based approaches.

3.1 Theoretical Framework for Value-Based Reinforcement Learning

In this section, we introduce a theoretical framework for analyzing and designing the value-based multi-objective reinforcement learning algorithms. This framework is based on the famous Banach's fixed-point theorem, which guarantees the existence and uniqueness of fixed-point of a *contraction* on a *complete metric space*. Therefore, generalize this theorem a bit, we can think all value functions of reinforcement learning are in some metric space, and find the optimal value or policy is to find the fixed point of a certain contraction on that space. Let us first have a quick pass to several fundamental mathematical concepts.

3.1.1 Banach's Fixed-Point Theorem

As we are unable to display all technical details about the mathematical background in the thesis, we highly recommend readers to refer to the blog [37], where more examples are created to illustrate math ideas.

Definition 3.1 (Metric Space)**.** A metric space is an ordered pair (X, d) consists of an *underlying set* X and a real-valued function $d(x, y)$, called *metric*, defined for $x, y \in X$ such that for any $x, y, z \in X$ the following conditions are satisfied:

1. non-negativity: $d(x, y) \geq 0$;
2. identity of indiscernibles: $d(x, y) = 0 \Leftrightarrow x = y$;
3. symmetry: $d(x, y) = d(y, x)$;
4. triangle inequality: $d(x, y) \leq d(x, z) + d(z, y)$.

Think about the common Euclidean space, where the underlying set is $\mathbb{R}^3$, and its distance function $d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}$ is a valid metric.

When we construct a metric space (X, d), it is common to expect our metric space has no point "missing" from it. For example, the space $\mathbb{Q}$ of rational numbers, with the metric defined by the absolute value $|\cdot|$ of numbers' differences, is not complete. Just consider the rational sequence $x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}, x_1 = 1$, whose elements become arbitrarily close to each other as the sequence progresses, but it doesn't converge to any point in space $\mathbb{Q}$. Therefore, the space $\mathbb{Q}$ is incomplete, since it obviously misses points.

A sequence like that in our example is called *Cauchy sequence*. Formally, we define the Cauchy sequence in and the completeness of a metric space (X, d) as follows:

Definition 3.2 (Cauchy Sequence)**.** A sequence $\{x_n\}$ is said to be *Cauchy sequence* if for each $\epsilon > 0$ there exists an N_ϵ such that $d(x_n, x_m) < \epsilon$ for any $n, m \geq N_\epsilon$.

Definition 3.3 (Complete Metric Space)**.** A metric space is said to be *complete* if each Cauchy sequence is a convergent sequence in it.

The significance of complete metric space is hard to be overemphasized. In many application, it is easier to show a given sequence is a Cauchy sequence than to show it is convergent. If the underlying metric space is complete, to show a given sequence is a Cauchy sequence is sufficient for proving its convergence.

Based on the concept of completeness, we now introduce the *contraction mapping*, which is a key concept in our value-based theoretical framework.

Definition 3.4 (Contraction)**.** Let (X, d) be a metric space. We say that $\mathcal{T}$ is a *contraction*, if there is a real number $\gamma \in [0, 1)$ such that

$$d(\mathcal{T}(x), \mathcal{T}(x')) \leq \gamma d(x, x') \tag{3–1}$$

for all points $x, x' \in X$, where γ is called a Lipschitz coefficient for the contraction $\mathcal{T}$.

Theorem 3.1 (Banach's Fixed-Point Theorem)**.** *Let (X, d) be a complete metric space and let $\mathcal{T} : X \to X$ be a contraction. Then there is one and* ***only one*** *fixed point $x^* \in X$ such that $\mathcal{T}(x^*) = x^*$. Moreover, if x is any point in X and $\mathcal{T}^n(x)$ is inductively defined by $\mathcal{T}^n(x) = \mathcal{T}(\mathcal{T}^{n-1}(x))$, then we have $\mathcal{T}^n(x) \to x^*$ as $n \to \infty$.*

Proof. Let us first prove every sequence $\{\mathcal{T}_n(x)\}$ convergent to a fixed point. Since the metric space (X, d) is complete, we only need to prove every $\{\mathcal{T}_n(x)\}$ is a Cauchy sequence. By the triangle inequality and symmetry of metric, for all $x, y \in X$,

$$\begin{aligned} d(x, y) &\leq d(x, \mathcal{T}(x)) + d(\mathcal{T}(x), \mathcal{T}(y)) + d(\mathcal{T}(y), y) \\ &\leq d(\mathcal{T}(x), x) + \gamma d(x, y) + d(\mathcal{T}(y), y) \\ \Rightarrow d(x, y) &\leq [d(\mathcal{T}(x), x) + d(\mathcal{T}(y), y)]/(1 - \gamma) \end{aligned}$$

Consider two points $\mathcal{T}^\ell(x), \mathcal{T}^m(x)$ in the sequence. Their distance is bounded by

$$\begin{aligned} d(\mathcal{T}^\ell(x), \mathcal{T}^m(x)) &\leq [d(\mathcal{T}^{\ell+1}(x), \mathcal{T}^\ell(x)) + d(\mathcal{T}^{m+1}(x), \mathcal{T}^m(x))]/(1 - \gamma) \\ &\leq [\gamma^\ell d(\mathcal{T}(x), x) + \gamma^m d(\mathcal{T}(x), x)]/(1 - \gamma) \\ &= \frac{\gamma^\ell + \gamma^m}{1 - \gamma} d(\mathcal{T}(x), x) \end{aligned}$$

Since $\gamma \in [0, 1)$ the distance converges to zero as $\ell, m \to \infty$, we prove that $\{\mathcal{T}^n(x)\}$ is a Cauchy sequence. Because the space (X, d) is complete, $\{\mathcal{T}^n(x)\}$ converges.

We now show that $x^* = \lim_{m\to\infty} \mathcal{T}^m(x)$ is a fixed point, and it is unique. Since $\mathcal{T}$ is a contraction, by definition it is continuous. Therefore,

$$\mathcal{T}(x^*) = \mathcal{T}(\lim_{m\to\infty} \mathcal{T}^m(x)) = \lim_{m\to\infty} \mathcal{T}(\mathcal{T}^m(x)) = \lim_{m\to\infty} \mathcal{T}^m(x) = x^*.$$

The uniqueness of x^* can be proved by contradiction. Assume x^* and y^* are two distinct fixed points of $\mathcal{T}$. We then get the contradiction

$$0 < d(x^*, y^*) = d(\mathcal{T}(x^*), \mathcal{T}(y^*)) \leq \gamma d(x^*, y^*) < d(x^*, y^*).$$

Hence $\mathcal{T}$ has only one fixed point and $\{\mathcal{T}^n(x)\}$ converges to it for any $x \in X$. □

3.1.2 General Value-Based Reinforcement Learning Framework

The above introduced Banach fixed-point theorem provides us with an iterative method for converging quickly to any desired solution in the large solution space, by repeatedly applying a properly designed contraction. For example, the foundation to standard value-based single-objective reinforcement learning is the use of Bellman's optimality equation[15]:

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \sup_{a'\in\mathcal{A}} Q^*(s',a'), \tag{3–2}$$

where $\gamma \in [0,1)$ is the discount factor and the optimal Q-value function $Q^*(s,a)$ is the desired solution in the space $\mathcal{Q} \subseteq \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ consisting of all the bounded functions with ℓ_∞-distance metric

$$d(Q,Q') := \sup_{s\in\mathcal{S},a\in\mathcal{A}} |Q(s,a) - Q'(s,a)|. \tag{3–3}$$

Since the all the function in this space is bounded, we can prove that with this ℓ_∞-distance metric, the space $(\mathcal{Q}, d)$ is complete. Besides, according to the equation (3–2), we can design an *Bellman optimality operator* $\mathcal{T}$ such that

$$(\mathcal{T}Q)(s,a) := r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \sup_{a'\in\mathcal{A}} Q(s',a'), \tag{3–4}$$

which can be shown as a contraction on $(\mathcal{Q}, d)$. Many popular value-based reinforcement algorithms, such as *deep Q-learning*, can be seen as asynchronous iteration methods with approximately applied contraction.

We first verify that the Bellman optimality operator $\mathcal{T}$ indeed is a contraction on $(\mathcal{Q}, d)$.

Theorem 3.2 (Bellman Optimality Operator of is a Contraction)**.** *Let Q, Q' be any two Q-value function in the space $(\mathcal{Q}, d)$, the Lipschitz condition $d(\mathcal{T}Q, \mathcal{T}Q')$ holds.*

Proof. Without the loss of generality, we assume $\sup_{a\in\mathcal{A}} Q(s,a) \geq \sup_{a\in\mathcal{A}} Q'(s,a)$ in some state s of interest.

Expand the expression of $d(\mathcal{T}Q, \mathcal{T}Q')$ we have

$$\begin{aligned}
d(\mathcal{T}Q, \mathcal{T}Q')(s,a) &= \sup_{s\in\mathcal{S}, a\in\mathcal{A}} |(\mathcal{T}Q)(s,a) - (\mathcal{T}Q')(s,a)| \\
&= \sup_{s\in\mathcal{S}, a\in\mathcal{A}} \left| r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} \sup_{a'\in\mathcal{A}} Q(s',a') - r(s,a) - \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} \sup_{a''\in\mathcal{A}} Q'(s',a'') \right| \\
&= \gamma \cdot \sup_{s\in\mathcal{S}, a\in\mathcal{A}} \left| \mathbb{E}_{s'\sim P(\cdot|s,a)} \left[\sup_{a'\in\mathcal{A}} Q(s',a') - \sup_{a''\in\mathcal{A}} Q'(s',a'') \right] \right| \\
&\leq \gamma \cdot \sup_{s\in\mathcal{S}, a\in\mathcal{A}} \mathbb{E}_{s'\sim P(\cdot|s,a)} \left[\left| \sup_{a'\in\mathcal{A}} Q(s',a') - \sup_{a''\in\mathcal{A}} Q'(s',a'') \right| \right] \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S}} \left| \sup_{a'\in\mathcal{A}} Q(s',a') - \sup_{a''\in\mathcal{A}} Q'(s',a'') \right|
\end{aligned}$$

Now according to our assumption, let a' be the action maximizing the value of Q in some state s' of interest, then we derive

$$\begin{aligned}
d(\mathcal{T}Q, \mathcal{T}Q')(s,a) &\leq \gamma \cdot \sup_{s'\in\mathcal{S}} \left| Q(s',a') - \sup_{a''\in\mathcal{A}} Q'(s',a'') \right| \\
&= \gamma \cdot \sup_{s'\in\mathcal{S}} \left| Q(s',a') - Q'(s',a') + Q'(s',a') - \sup_{a''\in\mathcal{A}} Q'(s',a'') \right| \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S}} |Q(s',a') - Q'(s',a')| \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S}, a'\in\mathcal{A}} |Q(s',a') - Q'(s',a')| = \gamma d(Q,Q')
\end{aligned}$$

It completes our proof. □

It is easy to verify that the optimal value function Q^* is a fixed point in $(\mathcal{Q}, d)$. Therefore, applying the optimality operator iteratively many times on any initial Q-value function, we can find the unique optimal one. Similarly, we can also define *Bellman evaluation operator* $\mathcal{T}_\pi$ using the Bellman expectation equation 2–5, which is also a contraction.

Knowing that optimality operator is a contraction is a great step to understanding value-based algorithms. But how could we use it to help us develop a doable algorithm? How is the classical deep Q-learning associated with this fact? One biggest obstacle is that in deep Q-learning we use a minibatch to update previous Q-value function approximated by neural networks (see Section 2.1.3), not update for all states and actions. Thus the updated Q-value function is not a strict $\mathcal{T}Q$, but only close to $\mathcal{T}Q$ on some state and action pairs. Is there still a theoretical guarantee?

Fortunately, we can prove that even if the update each step is mini-batchified and approximated, an iterative algorithm can still converge to a promising result, counting for some extra assumptions.

Definition 3.5 (Minibatch Iteration)**.** Consider the Q-value function Q as a composition of $\{Q_{S,A}\}_{S\subseteq\mathcal{S}, A\subseteq\mathcal{A}}$ such that in each iteration,

$$Q^{k+1}_{S,A}(s,a) = \begin{cases} \mathcal{T}Q^k_{S,A}(s,a), & \text{if } s\in S \text{ and } a\in A; \\ Q^k_{S,A}(s,a), & \text{otherwise.} \end{cases}$$

Theorem 3.3 (Minibatch Convergence Theorem)**.** *Suppose each restricted Q-value function $Q_{S,A}$ can be update infinite times, and there is a sequence of nonempty subsets $\{\mathcal{Q}^k\} \subseteq \mathcal{Q}$ with $\mathcal{Q}^{k+1} \subseteq \mathcal{Q}^k$, $k = 0, 1, \ldots$ such that if $\{Q^k\}$ is any sequence with $Q^k \in \mathcal{Q}^k$ for all $k \geq 0$, the $\{Q^t\}$ converges pointwisely to Q^*. Assume further the following:*

1. *Convergence Condition: We have*

$$\forall Q \in \mathcal{Q}^k, \mathcal{T}Q \in \mathcal{Q}^{k+1} \tag{3–5}$$

2. *Box Condition: For all $k, \mathcal{Q}^k$ is a Cartesian product of the form*

$$\mathcal{Q}^k = \times_{s\in\mathcal{S}, a\in\mathcal{A}} \mathcal{Q}^k_{\{s\},\{a\}} \tag{3–6}$$

where $\mathcal{Q}^k_{S,A}$ is a set of bounded real-valued functions on states S and actions A. Then for every $Q^0 \in \mathcal{Q}^0$ the sequence $\{Q^k\}$ generated by the minibatch iteration algorithm converges to Q^*[38].

Proof. To show that the algorithm converges is equivalent to show that the iterates from $\mathcal{Q}^k$ will get in to $\mathcal{Q}^{k+1}$ eventually, i.e., for each $k \geq 0$, there is a time t_k such that $Q^t \in \mathcal{Q}^k$ for all $t \geq t_k$. We can prove it by mathematical induction.

When $k = 0$, the statement is true since $Q^0 \in \mathcal{Q}^0$. Assuming the statement is true for a given k, we will show there is a time t_{k+1} with the required properties. For each $s \in \mathcal{S}, a \in \mathcal{A}$, let set $L_{s,a} = \{t : s^t = s, a^t = a\}$ record the time a minibatch update happens on the state s and action a. Let $t(s, a)$ be the first element in $L_{s,a}$ such that $t(s, a) \geq t_k$. Then by the convergence condition, we have $\mathcal{T}Q^{t(s,a)} \in \mathcal{Q}^{k+1}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. In the view of the box condition, it implies $Q^{t+1}(s, a) \in \mathcal{Q}^{k+1}_{s,a}$ for all $t \geq t(s, a)$ for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Therefore, let $t_{k+1} = \max_{s,a} t(s, a) + 1$, using the box condition, we have $Q^t \in \mathcal{Q}^{k+1}$ for all $t \leq t_{k+1}$. By mathematical induction, the statement holds, and $\mathcal{Q}^k$ will shrink in to $\mathcal{Q}^{k+1}$ eventually. Hence, we have proved $\{Q^t\}$ converges to Q^* finally. □

The above theorem indicates that minibatch update with experience reply will not affect the convergence of iteratively applying optimality operator $\mathcal{T}$. This gives the flexibility to design value-based algorithm's updating scheme. Besides, even though we use deep Q-network to approximate the Q-value function and update it by using gradient descent, this will not impair the magical function of optimality operator too much. If after n-round neural network updates, we always have $\sup_{s\in\mathcal{S}, a\in\mathcal{A}} |Q^{t+n} - \mathcal{T}Q^t| \leq \epsilon$, where n is an arbitrary bounded integer, by applying the triangle inequality we assert that the final error $d(Q_{final}, Q^*) \leq \epsilon/(1 - \gamma)$ is bounded. And by now we have put the whole deep Q-learning into a general theoretical framework.

In a high-level overview, this framework for value-based reinforcement learning consists of five elements:

- **Value Space** $\mathcal{X}$: A common choice of $\mathcal{X}$ is the space of value functions in $\mathbb{R}^{\mathcal{S}}$ or the space of Q-value functions in $\mathbb{R}^{\mathcal{S}\times\mathcal{A}}$. There are many other choices such as the space of ordered pair of $(\mathcal{V}, \mathcal{Q})$ (Example 2.6.2 in book [39]), or the space of multi-dimensional value functions as we will show in the following sections.
- **Value Metric** d: It defines the "distance" between two points in the value space. Besides the basic four requirements, the metric d should ensure a complete metric space $(\mathcal{X}, d)$ to validate the Banach's fixed-point theorem. A compatible selection of value metric will make our analysis much easier.

- **Evaluation Operator** $\mathcal{T}_\pi$: We have constructed a recursive expression of a certain value point in the value space associated with some policy, e.g., the Bellman expectation equation, to depict the value of a policy v_π as a fixed point we desire. Carefully verify that the contraction property holds for $\mathcal{T}_\pi$.
- **Optimality Operator** $\mathcal{T}$: Write down recursive expression of the optimal point in the value space, e.g., the Bellman optimality equation. Note that when the metric d is the supremum of the absolute value of the difference, and $\mathcal{T}_\pi$ is a contraction, we can prove the contraction property of $\mathcal{T}$ is always automatically satisfied.
- **Updating Scheme**: It is the most tricky part. For those small-scale problems, the minibatch iteration works well. However, to make our reinforcement learning algorithm practical and scalable, we have to consider much more factors. For example: How do we approximate the value and policies? If it is an online algorithm, how do we trade off exploration and exploitation? All these details will significantly influence the performance of our algorithm on real-world tasks.

To sum up, we propose five essential components for analyzing and designing general value-base reinforcement learning algorithms: **(1) value space**, **(2) value metric**, **(3) evaluation operator**, **(4) optimality operator**, and **(5) updating scheme**. In fact, there is some work [40] developing distributional reinforcement learning almost under this framework. We will discuss how to design these five elements of our framework to develop multi-objective reinforcement learning algorithms in the next two sections.

3.2 Value-Based Deep MORL Algorithm: Naive Version

A straightforward extension of the value-based reinforcement learning algorithm for the multi-objective tasks in the delayed linear preference scenario is to consider a new value space $\mathcal{Q} \subseteq \mathbb{R}^{\mathcal{S}\times\mathcal{A}\times\Omega}$ complete in ℓ_∞ value metric.

$$d(Q, Q') := \sup_{\substack{s\in\mathcal{S}, a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} |Q(s, a, \boldsymbol{\omega}) - Q'(s, a, \boldsymbol{\omega})|, \tag{3–7}$$

where Ω is the parameter space of linear preference functions, and $Q(s, a, \boldsymbol{\omega})$ stands for the real utility of preforming some trajectories after taking the action a at initial state s to satisfy the preference $f_{\boldsymbol{\omega}}(\cdot) = \langle \boldsymbol{\omega}, \cdot \rangle$.

We refer this kind of Q-value functions *utility-based multi-objective value functions*. What are the evaluation operator and the optimality operator for this value space design? How should we approximate this utility-based Q-value function and update it?

3.2.1 Multi-Objective Bellman Optimality Operator

In this section, we give the evaluation operator and the optimality operator under the space $(\mathcal{Q}, d)$ stated above. The evaluation operator is a simple extension of that of single-objective reinforcement learning. Given a policy π, the evaluation operator is defined by

$$(\mathcal{T}_\pi Q)(s, a, \boldsymbol{\omega}) := \boldsymbol{\omega}^\intercal \boldsymbol{r}(s, a) + \gamma \mathbb{E}_{\tau\sim(\mathcal{P},\pi)} Q(s', a', \boldsymbol{\omega}) \tag{3–8}$$

We can image it as a “parallel" version of single-objective Bellman expectation equation for different importances weights. It is a contraction simply because it is a contraction for every fixed $\boldsymbol{\omega}$.

As for the optimality operator, for the convenience of future analysis, we hope to define an *optimality filter* $\mathcal{H}$ by $(\mathcal{H}Q)(s,\boldsymbol{\omega}) := \sup_{a'\in\mathcal{A}} Q(s,a',\boldsymbol{\omega})$. It can also be regarded as a parallel version of the simple maximization operation in the classical Q-learning. And then the optimality operator $\mathcal{T}$ is defined in terms of the optimal filter:

$$(\mathcal{T}Q)(s,a,\boldsymbol{\omega}) := \boldsymbol{\omega}^\intercal \boldsymbol{r}(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}(\mathcal{H}Q)(s',\boldsymbol{\omega}). \tag{3–9}$$

Theorem 3.4 (Fixed Point of Naive Optimality Operator for MORL)**.** *Use definitions above. Let $Q^* \in \mathcal{Q}$ be the preferred optimal value function in the value space, such that*

$$Q^*(s,a,\boldsymbol{\omega}) = \sup_{\pi\in\Pi} \mathbb{E}_{\tau\sim(\mathcal{P},\pi)|s_0=s,a_0=a}\left[\sum_{t=0}^{\infty}\gamma^t \boldsymbol{\omega}^\intercal \boldsymbol{r}(s_t,a_t)\right], \tag{3–10}$$

then $Q^* = \mathcal{T}Q^*$ holds.

Proof. Our proof starts with substituting the definition of Q^* into $\mathcal{T}Q^*$.

$$\begin{aligned}
\mathcal{T}Q^*(s,a,\boldsymbol{\omega}) &= \boldsymbol{\omega}^\intercal \boldsymbol{r}(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}(\mathcal{H}Q)(s',\boldsymbol{\omega}) \\
&= \boldsymbol{\omega}^\intercal \boldsymbol{r}(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \sup_{a'\in\mathcal{A}} Q^*(s',a',\omega) \\
&= \boldsymbol{\omega}^\intercal \boldsymbol{r}(s,a) + \gamma \sup_{\pi} \mathbb{E}_{\substack{\tau\sim(\mathcal{P},\pi)\\ s_0\sim\mathcal{P}(\cdot|s,a)}}\left[\sum_{t=0}^{\infty}\gamma^t \boldsymbol{\omega}^\intercal \boldsymbol{r}(s_t,a_t)\right]
\end{aligned}$$

The last step is because the expectation over s' does not affect the supremum over a', and the supremum over a' is absorbed by the supremum over π. Then by the definition of Q^* again, we have

$$\mathcal{T}Q^*(s,a,\boldsymbol{\omega}) = \sup_{\pi\in\Pi} \mathbb{E}_{\tau\sim(\mathcal{P},\pi)|s_0=s,a_0=a}[\sum_{t=0}^{\infty}\gamma^t \omega^T r(s_t,a_t)] = Q^*(s,a,\boldsymbol{\omega}).$$

This completes the proof. The preferred optimal value function is a fixed point of the proposed optimality operator. □

Theorem 3.4 tells us the preferred optimal value function is a fixed-point in the value space. If we can show that this $\mathcal{T}$ is a contraction, than iteratively applying $\mathcal{T}$ can lead us to the desired solutions for all $\boldsymbol{\omega}$.

Theorem 3.5 (Naive Optimal Operator is a Contraction)**.** *Let Q, Q' be any two Q-value functions in the value space $\mathcal{Q}$ as defined above, the Lipschitz condition $d(\mathcal{T}Q,\mathcal{T}Q') \le \gamma d(Q,Q')$ holds, where $\gamma \in [0,1)$ is the discount factor of the underlying MOMDP $\mathcal{M}$.*

Proof. Similar to the proof of Theorem 3.2, without the loss of generality, we assume $\sup_{a\in\mathcal{A}} Q(s,a,\boldsymbol{\omega}) \ge$

$\sup_{a\in\mathcal{A}} Q'(s,a,\boldsymbol{\omega})$ for some state s and $\boldsymbol{\omega}$ of interest. Expand the expression of $d(\mathcal{T}Q,\mathcal{T}Q')$ we have

$$\begin{aligned}
d(\mathcal{T}Q,\mathcal{T}Q')(s,a) &= \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} |(\mathcal{T}Q)(s,a) - (\mathcal{T}Q')(s,a)| \\
&= \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} |\boldsymbol{\omega}^\intercal \boldsymbol{r}(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}(\mathcal{H}Q)(s',\boldsymbol{\omega}) - \boldsymbol{\omega}^\intercal \boldsymbol{r}(s,a) - \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}(\mathcal{H}Q')(s',\boldsymbol{\omega})| \\
&= \gamma \cdot \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} \left| \mathbb{E}_{s'\sim P(\cdot|s,a)} \left[\sup_{a'\in\mathcal{A}} Q(s',a',\boldsymbol{\omega}) - \sup_{a''\in\mathcal{A}} Q'(s',a'',\boldsymbol{\omega}) \right] \right| \\
&\leq \gamma \cdot \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} \mathbb{E}_{s'\sim P(\cdot|s,a)} \left[\left| \sup_{a'\in\mathcal{A}} Q(s',a',\boldsymbol{\omega}) - \sup_{a''\in\mathcal{A}} Q'(s',a'',\boldsymbol{\omega}) \right| \right] \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S},\boldsymbol{\omega}\in\Omega} \left| \sup_{a'\in\mathcal{A}} Q(s',a') - \sup_{a''\in\mathcal{A}} Q'(s',a'') \right|
\end{aligned}$$

Now according to our assumption, let a' be the action maximizing the value of Q for some state s' and preference $\boldsymbol{\omega}$ of interest, then we derive

$$\begin{aligned}
d(\mathcal{T}Q,\mathcal{T}Q')(s,a) &\leq \gamma \cdot \sup_{s'\in\mathcal{S},\boldsymbol{\omega}\in\Omega} \left| Q(s',a',\boldsymbol{\omega}) - \sup_{a''\in\mathcal{A}} Q'(s',a'',\boldsymbol{\omega}) \right| \\
&= \gamma \cdot \sup_{s'\in\mathcal{S},\boldsymbol{\omega}\in\Omega} \left| Q(s',a',\boldsymbol{\omega}) - Q'(s',a',\boldsymbol{\omega}) + Q'(s',a',\boldsymbol{\omega}) - \sup_{a''\in\mathcal{A}} Q'(s',a'',\boldsymbol{\omega}) \right| \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S},\boldsymbol{\omega}\in\Omega} |Q(s',a',\boldsymbol{\omega}) - Q'(s',a',\boldsymbol{\omega})| \\
&\leq \gamma \cdot \sup_{\substack{s'\in\mathcal{S},a'\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} |Q(s',a',\boldsymbol{\omega}) - Q'(s',a',\boldsymbol{\omega})| = \gamma d(Q,Q')
\end{aligned}$$

This ends our proof. $\mathcal{T}$ is a contraction. □

Corollary 3.6. *As a direct consequence of Theorems 3.1, 3.4, and 3.5, applying the optimality $\mathcal{T}$ iterative on any initial Q-value function derives the preferred optimal Q-value function Q^* for any preference $\boldsymbol{\omega}$.*

3.2.2 Updating Scheme: Hindsight Experience Replay

Deep neural networks empower us to approximate any bounded functions in $\mathcal{Q} \subseteq \mathbb{R}^{\mathcal{S}\times\mathcal{A}\times\Omega}$. Like deep Q-learning[17], we employ a neural network function approximator with parameters θ to estimate the utility-based multi-objective Q-value function, as defined above. We refer to this neural network as a *utility-based multi-objective Q-network*. A utility-based multi-objective Q-network can be trained by minimizing a series of loss function

$$L_k(\theta) = \mathbb{E}_{s,a,\boldsymbol{\omega}} \left[(y_k - Q(s,a,\boldsymbol{\omega};\theta))^2 \right], \tag{3–11}$$

which changes at each iteration k, where $y_k = \mathbb{E}_{s'} \left[\boldsymbol{\omega}^\intercal \boldsymbol{r}(s,a) + \gamma(\mathcal{H}Q)(s',a',\boldsymbol{\omega};\theta_k) \right]$ is the target of iteration k. The target is fixed during optimizing the loss function. And the parameters of the utility-based multi-objective Q-network are updated by $\theta_{k+1} \leftarrow \theta_k - \eta$, where

$$\eta \propto \nabla_{\theta=\theta_k} L_k(\theta) = -\mathbb{E}_{s,a,s'} \left[(\boldsymbol{\omega}^\intercal \boldsymbol{r} + \gamma(\mathcal{H}Q)(s',a',\boldsymbol{\omega};\theta_k) - Q(s,a,\boldsymbol{\omega};\theta_k)) \nabla_{\theta=\theta_k} Q(s,a,\boldsymbol{\omega};\theta) \right]. \tag{3–12}$$

Algorithm 3–1 Naive Deep MORL with Hindsight Experience Replay

Given

- a preference sampling distribution $\mathcal{D}_\omega$,
- minibatch sizes for transitions N_τ and for preference N_ω.
- a utility-based multi-objective Q-network Q parameterized by θ.

Initialize Q with random parameters.

Initialize replay buffer $\mathcal{D}_{tau}$.

for episode = $1, \ldots, M$ **do**

  Sample a linear preference $\boldsymbol{\omega} \sim \mathcal{D}_\omega$.

  for t = $0, \ldots, N$ **do**

    Sample an action a_t using behavioral policy for trading off exploration and exploitation, i.e.

$$a_t = \begin{cases} \text{random action in } \mathcal{A}, & \text{with probability } \epsilon; \\ \max_{a \in \mathcal{A}} Q(s_t, a, \boldsymbol{\omega}; \theta), & \text{with probability } 1 - \epsilon. \end{cases}$$

    Execute the action a_t and observe new state s_{t+1}.

    Store transition $(s_t, a_t, \boldsymbol{r}_t, s_{t+1})$ in $\mathcal{D}_{tau}$.

    if update **then**

      Sample random minibatch of transitions $(s_j, a_j, \boldsymbol{r}_j, s_{j+1})$ from $\mathcal{D}_\tau$.

      Sample N_ω preferences $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_{N_\omega}\} \sim \mathcal{D}_\omega$.

      Compute targets $(\mathcal{T}Q)_{ij} = \begin{cases} \boldsymbol{\omega}_i^\intercal \boldsymbol{r}_j. & \text{for terminal } s_{j+1}; \\ \boldsymbol{\omega}_i^\intercal \boldsymbol{r}_j + \gamma \max_{a \in \mathcal{A}} Q(s_j, a, \boldsymbol{\omega}_i; \theta), & \text{for non-terminal } s_{j+1}. \end{cases}$

      Update Q_θ by descending its stochastic gradient according to equation 3–12:

$$\nabla_\theta L(\theta) = -\frac{1}{N_\tau N_\omega} \sum_{i \in [N_\omega], j \in [N_\tau]} \left[\left((\mathcal{T}Q)_{ij} - Q(s_j, a_j, \boldsymbol{\omega}_i; \theta) \right) \nabla_\theta Q(s_j, a_j, \boldsymbol{\omega}_i; \theta) \right].$$

    end if

  end for

end for

return the utility-based Q-net Q_θ

Similar to the classical deep Q-learning, we use a experience replay buffer $\mathcal{D}_\tau = \{(s_t^i, a_t^i, \boldsymbol{r}_t^i, s_{t+1}^i)\}_{t,i}$ to store each step of sampled trajectories. When updating, we sample a minibatch of transition records from this replay buffer. Theorems 3.3 and 3.5 guarantees the convergence of this minibatch updating, with an extra assumption that we can update as equation 3–12 for each $\boldsymbol{\omega} \in \Omega$ infinite times. How can we ensure this?

Paper [41] presents a technique for training a reinforcement learning to serve multiple goals. For each episode, the agent performs following a policy according to a randomly sampled goals. When updating, the agent uses the past trajectory update for multiple other goals in parallel. They refer this method as *hindsight experience replay*. Though there are huge differences between our settings and theirs, we use a similar method

to update utility-based multi-objective Q-network here.

In the learning phase of the delayed linear preference scenario, for each training episode, the MORL agent randomly sample a preference $\boldsymbol{\omega}$ from a certain distribution $\mathcal{D}_\omega$. When updating the utility-based multi-objective Q-network according to equation 3–12, for each sampled transition record $(s_t^i, a_t^i, \boldsymbol{r}_t^i, s_{t+1}^i)$ from replay buffer $\mathcal{D}_\tau$, we associate it with N_ω preferences $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_{N_\omega}\}$ sampled from $\mathcal{D}_\omega$. The update is apply to an expended batch of size $\texttt{minibatch\_size} \times \texttt{N}_\omega$. Notice that the preferences sampled for sampling trajectories only influence the agent's actions but not the environment dynamics. Therefore we can replay each trajectory with an arbitrary preference. Besides, our utility-based multi-objective Q-network can be replaced with any model similar to that in single-objective off-policy RL algorithms like DDPG[42], NAF[43] or SDQN[44]. Actually, in the experiment, we also use other deep reinforcement learning techniques such as double Q-learning[45] with a target network and prioritized experience replay[46], which stabilize and speed up our algorithms a lot. We present the skeleton of our naive deep MORL as algorithm 3–1.

3.2.3 Problems and Limitations of the Naive Version Algorithm

Algorithm 3–1 is an approach better than existing methods[29, 35, 36] for delayed linear preference scenarios of deep multi-objective reinforcement learning for two reasons. First, it is a single-model multiple-policy method. It maintains real utility for each linear preference $\boldsymbol{\omega}$ using one neural network and updates them in parallel. In the execution phase, it can directly respond with the optimal policy according to a user input preference $\boldsymbol{\omega}$ without extra search. Second, this method is compatible with many advanced techniques for single-objective deep reinforcement learning, such as double[45] or duel[47] Q-learning, prioritized experience replay[46], and other actor-critic methods[11].

However, in spite of this merits of our naive attempt, there are some limitations of this method. The major problems of our naive deep MORL algorithm are two-fold: first, the prediction of utility-based multi-objective Q-network is uninformative for the analysis phase; second, sample inefficiency.

In the naive deep MORL algorithm, the value space is a utility space. The agent learns a mapping

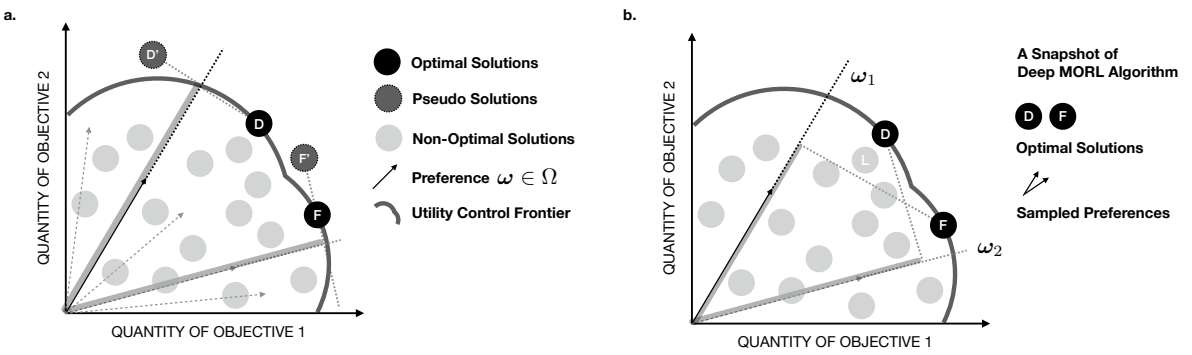

Figure 3–1 Two limitations of naive deep MORL algorithm.(a.) Several predicted utilities are not enough to recover the Pareto optimal solutions, unless we have known the whole utility frontier. (b.) At some stage, the naive algorithm finds the optimal solutions while they are not aligned with preference. It still requires many iterations for the value-preference alignment.

between different preferences to different real utilities during the learning phase. In the analysis phase, a user can ask the agent for the optimal utility for a certain preference, to evaluate whether she should command the agent with that preference in the execution phase. However, since utility is a scalar indicates the length of projection of optimal solution along the preference direction, we cannot use this information to know what is the specific solution this preference will result in. All the possibilities along the vertical line of the utility projection, called pseudo solutions, are indistinguishable. The user needs to know the whole utility control frontier, or let agent sample trajectories in a simulated environment to known the real corresponding multi-objective solutions. This would be very costly.

Furthermore, the naive deep MORL algorithm is sample inefficient. As illustrated in Figure 3–1 (b.), at some stage of the learning process, the naive deep MORL algorithm finds the optimal solution D, F in some state s, while stored in $\max_a Q(s, a, \boldsymbol{\omega}_2)$ and $\max_a Q(s, a, \boldsymbol{\omega}_1)$, respectively. These optimal solutions are not aligned with preferences. However, in the future iterations, the naive algorithm cannot use the information of $\max_a Q(s, a, \boldsymbol{\omega}_1)$ (which finds F) to update the optimal solution aligned with $\boldsymbol{\omega}_2$. It also can not use the information of $\max_a Q(s, a, \boldsymbol{\omega}_2)$ (which finds D) to update the optimal solution aligned with $\boldsymbol{\omega}_1$, even when they are sampled in the same batch, because of the updating rule. To improve the utility under the preference $\boldsymbol{\omega}_1$, it still needs to search along this direction, meeting with non-optimal L, then D, even it has seen solution D before under other preferences. This problem cannot be solved even if we come up with a wiser updating scheme. That is because we only maintain the optimal utility for each $\boldsymbol{\omega}$, while to help the agent align an $\boldsymbol{\omega}$ to a found solution requires the memory of that multi-objective solution.

3.3 Value-Based Deep MORL Algorithm: Envelope Version

The naive deep MORL algorithm is capable of solving delayed linear preference scenarios of multi-objective reinforcement learning. However, two limitations of that algorithm restrict its applicability and performance in practice. Aiming at solving two problems stated in Section 3.2.3, we design a new algorithm called envelope deep MORL algorithm. Following the value-based theoretical framework introduced in Section 3.1, our key idea to upgrade the naive algorithm is to consider a different value space, where every Q-value function is a mapping to multi-objective solutions, not utilities, and therefore maintains the necessary information for prediction in the analysis phase. Furthermore, we enhance the optimality filter to use that information to boost up the alignment in the learning phase.

For the envelope version algorithm, we consider a new value space $\boldsymbol{Q} \subseteq (\Omega \to \mathbb{R}^m)^{\mathcal{S}\times\mathcal{A}}$, containing all bounded functions $\boldsymbol{Q}(s, a, \omega)$ returning the estimates of preferred expected total rewards under preference $\boldsymbol{\omega}$, which are m-dimensional vectors. Besides, we employ a value metric d defined by

$$d(\boldsymbol{Q}, \boldsymbol{Q}') := \sup_{\substack{s\in\mathcal{S}, a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} |\boldsymbol{\omega}^\intercal(\boldsymbol{Q}(s, a, \boldsymbol{\omega}) - \boldsymbol{Q}'(s, a, \boldsymbol{\omega}))|. \tag{3–13}$$

Notice that this metric is a pseudo-metric, since the identity of indiscernibles in definition 3.1 does not hold for it. It is easy to show that metric space $(\boldsymbol{Q}, d)$ is complete.

We refer the Q-value functions in this $\boldsymbol{Q}$ *multi-objective value functions*. Similar to the naive one, we

design an evaluation operator and an optimality operator for this envelope version algorithm. As for the updating scheme, we also use hindsight experience replay in this case, but with a homotopy optimization trick.

3.3.1 Multi-Objective Bellman Optimality Operator

In this section, we give the evaluation operator and the optimality operator in the envelope version value space $(\mathcal{Q}, d)$ as stated above. The evaluation operator now is even simpler than that of the naive version. Give a policy π, the evaluation operator is defined by

$$(\mathcal{T}_\pi \boldsymbol{Q})(s, a, \boldsymbol{\omega}) := \boldsymbol{r}(s, a) + \gamma \mathbb{E}_{\tau \sim (\mathcal{P}, \pi)} \boldsymbol{Q}(s', a', \boldsymbol{\omega}) \tag{3–14}$$

Since now the multi-objective Q-value function is also in a vector form, this evaluation operator is almost the same to that of the single-objective reinforcement learning. It can be easily verified as a contraction.

As for the envelope version of optimality operator, we employ a stronger optimality filter $\mathcal{H}$, defined by $(\mathcal{H}\boldsymbol{Q})(s, \boldsymbol{\omega}) := \arg_Q \sup_{a' \in \mathcal{A}, \boldsymbol{\omega}' \in \Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s, a, \boldsymbol{\omega}')$, where the $\arg_Q$ takes the multi-objective value corresponding to the supremum. This filter is solving the convex envelope of the current solution frontier, therefore we name this algorithm as "envelope" version. We can write the optimality operator $\mathcal{T}$ in terms of the optimal filter:

$$(\mathcal{T}\boldsymbol{Q})(s, a, \boldsymbol{\omega}) := \boldsymbol{r}(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} (\mathcal{H}\boldsymbol{Q})(s', \boldsymbol{\omega}). \tag{3–15}$$

Theorem 3.7 (Fixed Point of Evelope Optimality Operator for MORL)**.** *Use above definitions in the envelope version value space. Let $\boldsymbol{Q}^* \in \mathcal{Q}$ be the preferred optimal value function in the value space, such that*

$$\boldsymbol{Q}^*(s, a, \boldsymbol{\omega}) = \arg_Q \sup_{\pi \in \Pi} \boldsymbol{\omega}^\intercal \mathbb{E}_{\tau \sim (\mathcal{P}, \pi) | s_0 = s, a_0 = a} \left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}(s_t, a_t) \right], \tag{3–16}$$

where the $\arg_Q$ takes the multi-objective value corresponding to the supremum, then $\boldsymbol{Q}^* = \mathcal{T}\boldsymbol{Q}^*$ holds.

Proof. First, we observe that $d(\boldsymbol{Q}^*, \mathcal{T}\boldsymbol{Q}^*) = \sup_{s \in \mathcal{S}, a \in \mathcal{A}, \boldsymbol{\omega} \in \Omega} |\boldsymbol{\omega}^\intercal (\boldsymbol{Q}^*(s, a, \boldsymbol{\omega}) - \mathcal{T}\boldsymbol{Q}^*(s, a, \boldsymbol{\omega}))| = 0 \Leftrightarrow \boldsymbol{\omega}^\intercal \mathcal{T}\boldsymbol{Q}^*(s, a, \boldsymbol{\omega}) = \boldsymbol{\omega}^\intercal \boldsymbol{Q}^*(s, a, \boldsymbol{\omega})$ for all $s, a, \boldsymbol{\omega}$. Then, by substituting the definition of Q^* into $\mathcal{T}Q^*$,

$$\begin{aligned}
\boldsymbol{\omega}^\intercal \mathcal{T} Q^*(s, a, \boldsymbol{\omega}) &= \boldsymbol{\omega}^\intercal \boldsymbol{r}(s, a) + \gamma \cdot \boldsymbol{\omega}^\intercal \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} (\mathcal{H}Q)(s', \boldsymbol{\omega}) \\
&= \boldsymbol{\omega}^\intercal \boldsymbol{r}(s, a) + \gamma \cdot \boldsymbol{\omega}^\intercal \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \arg_Q \sup_{a' \in \mathcal{A}, \boldsymbol{\omega}' \in \Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s, a', \boldsymbol{\omega}') \\
&= \boldsymbol{\omega}^\intercal \boldsymbol{r}(s, a) + \gamma \sup_{\pi} \mathbb{E}_{\substack{\tau \sim (\mathcal{P}, \pi) \\ s_0 \sim \mathcal{P}(\cdot|s,a)}} \left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\omega}^\intercal \boldsymbol{r}(s_t, a_t) \right]
\end{aligned}$$

The last step is because the elimination of $\boldsymbol{\omega}^\intercal$ and $\arg_Q$, and the expectation over s' does not affect the supremum over a', and the supremum over a' is absorbed by the supremum over π. Then by the definition of Q^* again, we have

$$\boldsymbol{\omega}^\intercal \mathcal{T}\boldsymbol{Q}^*(s, a, \boldsymbol{\omega}) = \sup_{\pi \in \Pi} \mathbb{E}_{\tau \sim (\mathcal{P}, \pi) | s_0 = s, a_0 = a} [\sum_{t=0}^{\infty} \gamma^t \omega^T r(s_t, a_t)] = \boldsymbol{\omega}^\intercal \boldsymbol{Q}^*(s, a, \boldsymbol{\omega}).$$

By our observation stated at the beginning, $d(\boldsymbol{Q}^*, \mathcal{T}\boldsymbol{Q}^*) = 0$. The preferred optimal value function is a fixed point of the proposed envelope version optimality operator. □

Theorem 3.7 tells us the preferred optimal value function is one of the fixed-points of envelope optimality operator $\mathcal{T}$ in the value space. And we still need to show that this $\mathcal{T}$ is a contraction.

Theorem 3.8 (Envelope Optimal Operator is a Contraction)**.** *Let $\boldsymbol{Q}, \boldsymbol{Q}'$ be any two multi-objective Q-value functions in the envelope value space $\mathbb{Q}$ as defined above, the Lipschitz condition $d(\mathcal{T}\boldsymbol{Q}, \mathcal{T}\boldsymbol{Q}') \leq \gamma d(\boldsymbol{Q}, \boldsymbol{Q}')$ holds, where $\gamma \in [0, 1)$ is the discount factor of the underlying MOMDP $\mathcal{M}$.*

Proof. Similar to the proof of Theorem 3.2, without the loss of generality, we assume $\sup_{a\in\mathcal{A},\boldsymbol{\omega}'\in\Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s, a, \boldsymbol{\omega}') \geq \sup_{a\in\mathcal{A},\boldsymbol{\omega}'\in\Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}'(s, a, \boldsymbol{\omega}')$ for some state s and $\boldsymbol{\omega}$ of interest. Expand the expression of $d(\mathcal{T}\boldsymbol{Q}, \mathcal{T}\boldsymbol{Q}')$ we have

$$
\begin{aligned}
d(\mathcal{T}\boldsymbol{Q}, \mathcal{T}\boldsymbol{Q}')(s, a) &= \sup_{\substack{s\in\mathcal{S}, a\in\mathcal{A} \\ \boldsymbol{\omega}\in\Omega}} |\boldsymbol{\omega}^\intercal((\mathcal{T}\boldsymbol{Q})(s, a) - (\mathcal{T}\boldsymbol{Q}')(s, a))| \\
&= \sup_{\substack{s\in\mathcal{S}, a\in\mathcal{A} \\ \boldsymbol{\omega}\in\Omega}} |\gamma \cdot \boldsymbol{\omega}^\intercal \mathbb{B}_{s'\sim P(\cdot|s,a)}(\mathcal{H}\boldsymbol{Q})(s', \boldsymbol{\omega}) - \gamma \cdot \boldsymbol{\omega}^\intercal \mathbb{B}_{s'\sim P(\cdot|s,a)}(\mathcal{H}\boldsymbol{Q}')(s', \boldsymbol{\omega})| \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S}, \boldsymbol{\omega}\in\Omega} \left| \boldsymbol{\omega}^\intercal \left[\arg_Q \sup_{a'\in\mathcal{A}, \boldsymbol{\omega}'\in\Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s', a', \boldsymbol{\omega}') - \arg_Q \sup_{a''\in\mathcal{A}, \boldsymbol{\omega}''\in\Omega} \boldsymbol{\omega}^\intercal Q'(s', a'', \boldsymbol{\omega}'') \right] \right| \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S}, \boldsymbol{\omega}\in\Omega} \left| \sup_{a'\in\mathcal{A}, \boldsymbol{\omega}'\in\Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s', a', \boldsymbol{\omega}') - \sup_{a''\in\mathcal{A}, \boldsymbol{\omega}''\in\Omega} \boldsymbol{\omega}^\intercal Q'(s', a'', \boldsymbol{\omega}'') \right|
\end{aligned}
$$

Now according to our assumption, let a' and $\boldsymbol{\omega}'$ be the action and preference chosen to maximize the value of $\boldsymbol{\omega}^\intercal \boldsymbol{Q}$ for some state s' and preference $\boldsymbol{\omega}$ of interest, then we derive

$$
\begin{aligned}
d(\mathcal{T}\boldsymbol{Q}, \mathcal{T}\boldsymbol{Q}')(s, a) &\leq \gamma \cdot \sup_{s'\in\mathcal{S}, \boldsymbol{\omega}\in\Omega} \left| \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s', a', \boldsymbol{\omega}') - \sup_{a''\in\mathcal{A}, \boldsymbol{\omega}''\in\Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}'(s', a'', \boldsymbol{\omega}'') \right| \\
&= \gamma \cdot \sup_{s'\in\mathcal{S}, \boldsymbol{\omega}\in\Omega} |\boldsymbol{\omega}^\intercal \boldsymbol{Q}(s', a', \boldsymbol{\omega}') - \boldsymbol{\omega}^\intercal \boldsymbol{Q}'(s', a', \boldsymbol{\omega}') \\
&\quad + \boldsymbol{\omega}^\intercal \boldsymbol{Q}'(s', a', \boldsymbol{\omega}') - \sup_{a''\in\mathcal{A}, \boldsymbol{\omega}''\in\Omega} \boldsymbol{\omega}^\intercal \boldsymbol{Q}'(s', a'', \boldsymbol{\omega}'')| \\
&\leq \gamma \cdot \sup_{s'\in\mathcal{S}, \boldsymbol{\omega}\in\Omega} |\boldsymbol{\omega}^\intercal \boldsymbol{Q}(s', a', \boldsymbol{\omega}') - \boldsymbol{\omega}^\intercal \boldsymbol{Q}'(s', a', \boldsymbol{\omega}')| \\
&\leq \gamma \cdot \sup_{\substack{s'\in\mathcal{S}, a'\in\mathcal{A} \\ \boldsymbol{\omega}\in\Omega}} |\boldsymbol{\omega}^\intercal \boldsymbol{Q}(s', a', \boldsymbol{\omega}') - \boldsymbol{\omega}^\intercal \boldsymbol{Q}'(s', a', \boldsymbol{\omega}')| = \gamma d(\boldsymbol{Q}, \boldsymbol{Q}')
\end{aligned}
$$

This completes our proof that $\mathcal{T}$ is a contraction. □

Remember that in our design, envelope version value distance d is a pseudo-metric. In a pseudo-metric space iteratively applying contraction may not shrink to the desired fixed point. To assert the convergence effectiveness of our design for optimality operator, we need to investigate a generalized Banach's Fixed-Point Theorem in the pseudo-metric space.

3.3.2 Generalized Banach's Fixed-Point Theorem

Theorem 3.9 (Generalized Banach Fixed-Point Theorem)**.** *Given that $\mathcal{T}$ is a contraction mapping with Lipschitz coefficient γ on the complete pseudo-metric space $\langle \mathbb{Q}, d \rangle$, and $\boldsymbol{Q}^*$ is defined as that in Theorem 3.7, it is always true that $\lim_{n\to\infty} d(\mathcal{T}^n\boldsymbol{Q}, \boldsymbol{Q}^*) = 0$ for any $\boldsymbol{Q} \in \mathbb{Q}$.*

Proof. By the symmetry and triangle inequality of pseudo-metric, for any $\boldsymbol{Q}, \boldsymbol{Q}' \in \mathcal{Q}$,

$$\begin{aligned} d(\boldsymbol{Q}, \boldsymbol{Q}') &\leq d(\boldsymbol{Q}, \mathcal{T}\boldsymbol{Q}) + d(\mathcal{T}\boldsymbol{Q}, \mathcal{T}\boldsymbol{Q}') + d(\mathcal{T}\boldsymbol{Q}', \boldsymbol{Q}') \\ &\leq d(\boldsymbol{Q}, \mathcal{T}\boldsymbol{Q}) + \gamma d(\boldsymbol{Q}, \boldsymbol{Q}') + d(\mathcal{T}\boldsymbol{Q}', \boldsymbol{Q}') \\ \Rightarrow d(\boldsymbol{Q}, \boldsymbol{Q}') &\leq [d(\mathcal{T}\boldsymbol{Q}, \boldsymbol{Q}) + d(\mathcal{T}\boldsymbol{Q}', \boldsymbol{Q}')]/(1-\gamma) \end{aligned}$$

Consider two points $\mathcal{T}^\ell \boldsymbol{Q}, \mathcal{T}^m \boldsymbol{Q}$ in the sequence $\{\mathcal{T}^n \boldsymbol{Q}\}$. Their distance is bounded by

$$\begin{aligned} d(\mathcal{T}^\ell \boldsymbol{Q}, \mathcal{T}^m \boldsymbol{Q}) &\leq [d(\mathcal{T}^{\ell+1}\boldsymbol{Q}, \mathcal{T}^\ell \boldsymbol{Q}) + d(\mathcal{T}^{m+1}\boldsymbol{Q}, \mathcal{T}^m \boldsymbol{Q})]/(1-\gamma) \\ &\leq [\gamma^\ell d(\mathcal{T}\boldsymbol{Q}, \boldsymbol{Q}) + \gamma^m d(\mathcal{T}\boldsymbol{Q}, \boldsymbol{Q})]/(1-\gamma) \\ &\leq \frac{\gamma^\ell + \gamma^m}{(1-\gamma)} d(\mathcal{T}\boldsymbol{Q}, \boldsymbol{Q}) \end{aligned}$$

since $\gamma \in [0, 1)$ the distance $d(\mathcal{T}^\ell \boldsymbol{Q}, \mathcal{T}^m \boldsymbol{Q})$ converge to 0 as $\ell, m \to \infty$, proving that $\{\mathcal{T}^n \boldsymbol{Q}\}$ is a Cauchy sequence. Because $\langle \mathcal{Q}, d \rangle$ is a complete pseudo-metric space, $\lim_{n\to\infty} d(\mathcal{T}^n \boldsymbol{Q}, \boldsymbol{Q}^\diamond) = 0$ for some $\boldsymbol{Q}^\diamond \in \mathcal{Q}$. Therefore,

$$d(\mathcal{T}\boldsymbol{Q}^\diamond, \boldsymbol{Q}^\diamond) = \lim_{n\to\infty} d(\mathcal{T}^{n+1}\boldsymbol{Q}, \boldsymbol{Q}^\diamond) = \lim_{n\to\infty} d(\mathcal{T}^n \boldsymbol{Q}, \boldsymbol{Q}^\diamond) = 0$$

We claim that $\boldsymbol{Q}^\diamond$ and $\boldsymbol{Q}^*$ must lie in the same equivalent class partitioned by relation $d(\cdot, \cdot) = 0$. Suppose $d(\boldsymbol{Q}^\diamond, \boldsymbol{Q}^*) \neq 0$, then we can get a contradiction

$$\begin{aligned} d(\boldsymbol{Q}^\diamond, \boldsymbol{Q}^*) &= d(\boldsymbol{Q}^\diamond, \mathcal{T}\boldsymbol{Q}^\diamond) + d(\mathcal{T}\boldsymbol{Q}^\diamond, \mathcal{T}\boldsymbol{Q}^*) + d(\mathcal{T}\boldsymbol{Q}^*, \boldsymbol{Q}^*) \\ &\leq 0 + \gamma d(\boldsymbol{Q}^\diamond, \boldsymbol{Q}^*) + 0 \\ &< d(\boldsymbol{Q}^\diamond, \boldsymbol{Q}^*) \end{aligned}$$

This proves our claim. Therefore, $\lim_{n\to\infty} d(\mathcal{T}^n \boldsymbol{Q}, \boldsymbol{Q}^*) = 0$ for any $\boldsymbol{Q} \in \mathcal{Q}$. □

In other words, Theorem 3.9 guarantees that iteratively applying optimal operator $\mathcal{T}$ on any multi-objective Q-value function, the algorithm will terminate with a function $\boldsymbol{Q}^\diamond$ which is equivalent to $\boldsymbol{Q}^*$ under the measurement of pseudo-metric d. Actually, these $\boldsymbol{Q}^\diamond$'s are as good as $\boldsymbol{Q}^*$, since they all have the same utilities for each $\boldsymbol{\omega}$, and only differ in the real value when the utility corresponds a recess in the utility control frontier (see Figure 3–1 for example).

3.3.3 Updating Scheme: Homotopy Optimization

We use deep neural networks to approximate bounded functions in $\mathcal{Q} \subseteq (\Omega \to \mathbb{R}^m)^{\mathcal{S}\times\mathcal{A}}$ with parameters θ. We refer to this neural network as a *multi-objective Q-network*. To drag $\boldsymbol{Q}$ close to $\mathcal{T}\boldsymbol{Q}$ at each update step, the multi-objective Q-network can be trained by minimizing a series of loss functions

$$L_k^{\mathtt{A}}(\theta) = \mathbb{E}_{s,a,\boldsymbol{\omega}} \left[\| \boldsymbol{y}_k - \boldsymbol{Q}(s, a, \boldsymbol{\omega}; \theta) \|_2^2 \right], \tag{3–17}$$

which changes at each iteration k, where $\boldsymbol{y}_k = \mathbb{E}_{s'} \left[\boldsymbol{r}(s, a) + \gamma(\mathcal{H}\boldsymbol{Q})(s', a', \boldsymbol{\omega}; \theta_k) \right]$ is the target of iteration k. The target is fixed during optimizing this loss function. And the parameters of the utility-based multi-objective

Q-network are updated by $\theta_{k+1} \leftarrow \theta_k - \eta$, where

$$\eta \propto \nabla_{\theta=\theta_k} L_k(\theta) = -\mathbb{E}_{s,a,s'}\left[\left(\boldsymbol{r} + \gamma(\mathcal{H}\boldsymbol{Q})(s', a', \boldsymbol{\omega}; \theta_k) - \boldsymbol{Q}(s, a, \boldsymbol{\omega}; \theta_k)\right)^\intercal \nabla_{\theta=\theta_k} \boldsymbol{Q}(s, a, \boldsymbol{\omega}; \theta)\right]. \quad (3\text{–}18)$$

Minimizing the loss function $L^{\mathbb{A}}$ is trying to drag the vector $\boldsymbol{Q}$ close to $\mathcal{T}\boldsymbol{Q}$. This ensures the correctness of our algorithm, to predict a $\boldsymbol{Q}$ as the real solution, while this means the square error is hard to be optimized in practice. To address this problem, we use a sequence of auxiliary loss function $L^{\mathbb{B}}$ to directly optimized the value metric d, which is defined by

$$L_k^{\mathtt{B}}(\theta) = \mathbb{E}_{s,a,\omega}[|\boldsymbol{\omega}^\intercal \boldsymbol{y}_k - \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s, a, \boldsymbol{\omega}; \theta)|] \quad (3\text{–}19)$$

Our final loss function sequence $L_k(\theta) = (1 - \lambda_k) \cdot L_k^{\mathtt{A}}(\theta) + \lambda_k \cdot L_k^{\mathtt{B}}(\theta)$, where λ_k is a weight to trade off between losses $L_k^{\mathtt{A}}$ and $L_k^{\mathtt{B}}$. We increase the value of λ_k from 0 to 1, to shift our loss function from $L^{\mathtt{A}}$ to $L^{\mathtt{B}}$. This method called *homotopy optimization*[48] is effective since for each update step, it uses the optimization result from the previous step as the initial guess. In the envelope deep MORL algorithm, $L^{\mathtt{A}}$ first ensure the prediction of $\boldsymbol{Q}$ is close to any real expected total reward, though it is hard to be optimal. And then $L^{\mathtt{B}}$ can provide an auxiliary force to pull the current guess along the direction with better utility. Figure 3–2 illustrate an explanation for this homotopy optimization.

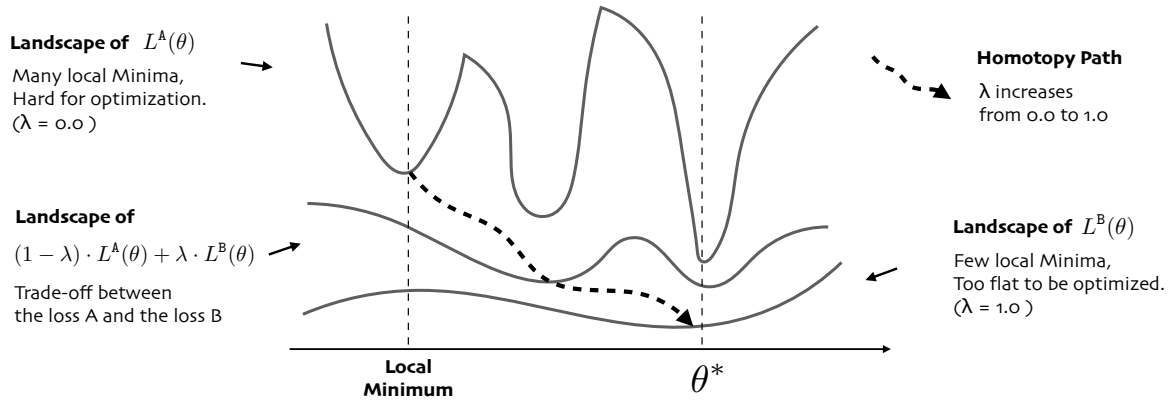

Figure 3–2 An explanation for homotopy optimization method used in the envelope deep MORL algorithm. The MSE loss $L^{\mathtt{A}}$ is hard for optimization since there are many local minima over its landscape. Although the value metric loss $L^{\mathtt{B}}$ has fewer local minima, it is also hard for optimization since there are many vectors $\boldsymbol{Q}$ minimizing value metric d. The landscape of $L^{\mathtt{B}}$ is too flat. The homotopy path connecting $L^{\mathtt{A}}$ and $L^{\mathtt{B}}$ provides better opportunities to find the global optimal parameters θ^*

Similar to the naive deep MORL algorithm, we use a experience replay buffer $\mathcal{D}_\tau = \{(s_t^i, a_t^i, \boldsymbol{r}_t^i, s_{t+1}^i)\}_{t,i}$ to store each step of sampled trajectories. When updating, we sample a minibatch of transition records from this replay buffer. Theorems 3.3 and 3.5 guarantees the convergence of this minibatch updating, with an extra assumption that we can update as equation 3–18 for each $\boldsymbol{\omega} \in \Omega$ infinite times. We use *hindsight experience replay* (HER) to ensure this. As introduced in Section 3.2.2, in the learning phase of the delayed linear preference scenario, the MORL agent randomly sample a preference $\boldsymbol{\omega}$ from a certain distribution $\mathcal{D}_\omega$ for each training episode. When updating the multi-objective Q-network according to equation 3–18, for each sampled

transition record $(s_t^i, a_t^i, \boldsymbol{r}_t^i, s_{t+1}^i)$ from replay buffer $\mathcal{D}_\tau$, we associate it with N_ω preferences $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_{N_\omega}\}$ sampled from $\mathcal{D}_\omega$. The update is apply to an expended batch of size $\texttt{minibatch\_size} \times \texttt{N}_\omega$. Notice that we will apply our optimality filter on this expended batch. Therefore the cost of solving the convex envelope is acceptable. Our multi-objective Q-network can also be replaced with other models similar to those in single-objective off-policy RL algorithms. In the experiment, we also use some popular deep reinforcement learning techniques to stabilize and speed up our algorithms. The skeleton of our envelope deep MORL is shown as algorithm 3–2.

3.4 Summary

In this chapter, we propose a theoretical framework for developing and analysis value-based reinforcement learning. This framework is inspired by the Banach's fixed point theorem, which says a contraction in a complete metric space has a unique fixed point. In this framework, five essential components are involved for designing general value-base reinforcement learning algorithms: (1) value space, (2) value metric, (3) evaluation operator, (4) optimality operator, and (5) updating scheme. With the help of this theoretical tool, we develop two value-based algorithms for the delayed linear preferences scenario of multi-objective reinforcement learning.

The first algorithm is called naive deep MORL algorithm, which is a straightforward extension of the value-based reinforcement learning algorithm. It maintains the utility estimation for all state, action, and preferences, and update them in parallel by using hindsight experience replay algorithm. This algorithm is easy to implement, and is compatible with many advanced reinforcement learning techniques. However, two limitations, the uninformative prediction and sample inefficiency, restrict the applicability and performance of our naive algorithm.

The second envelope deep MORL algorithm solves above two limitations of the naive algorithm by directly maintaining the multi-objective solutions, and solving a convex envelope of the explored value frontier each update step. This modification incurs a theoretical issue about the convergence analysis of our envelope algorithm, which requires us to generalize the Banach's theorem to pseudo-metric space. Besides, when updating, we optimize a weighted average of two loss functions. The balance weight for the loss will change over time, referred to as a homotopy optimization method. Although the envelope deep MORL algorithm is more complex to implement, and will cost more time for each update step, it is more sample-efficient during the learning phase and can provide with more informative prediction during the analysis phase.

Algorithm 3–2 Envelope Deep MORL with HER and Homotopy Optimization

Given

- a preference sampling distribution $\mathcal{D}_\omega$,
- minibatch sizes for transitions N_τ and for preference N_ω.
- a multi-objective Q-network $\boldsymbol{Q}$ parameterized by θ.
- a path p_λ for the balance weight λ increasing from 0 to 1.

Initialize Q with random parameters.

Initialize replay buffer $\mathcal{D}_{tau}$.

Initialize $\lambda = 0$.

for episode = $1, \ldots, M$ **do**

  Sample a linear preference $\boldsymbol{\omega} \sim \mathcal{D}_\omega$.

  for t = $0, \ldots, N$ **do**

    Sample an action a_t using behavioral policy for trading off exploration and exploitation, i.e.

$$a_t = \begin{cases} \text{random action in } \mathcal{A}, & \text{with probability } \epsilon; \\ \max_{a \in \mathcal{A}} \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s_t, a, \boldsymbol{\omega}; \theta), & \text{with probability } 1 - \epsilon. \end{cases}$$

    Execute the action a_t and observe new state s_{t+1}.

    Store transition $(s_t, a_t, \boldsymbol{r}_t, s_{t+1})$ in $\mathcal{D}_{tau}$.

    if update **then**

      Sample random minibatch of transitions $(s_j, a_j, \boldsymbol{r}_j, s_{j+1})$ from $\mathcal{D}_\tau$.

      Sample N_ω preferences $W = \{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \ldots, \boldsymbol{\omega}_{N_\omega}\} \sim \mathcal{D}_\omega$.

      Compute $(\mathcal{T}Q)_{ij} = \begin{cases} \boldsymbol{r}_j. & \text{for terminal } s_{j+1}; \\ \boldsymbol{r}_j + \gamma \arg_Q \max_{a \in \mathcal{A}, \boldsymbol{\omega}' \in W} \boldsymbol{\omega}_i^\intercal \boldsymbol{Q}(s_j, a, \boldsymbol{\omega}'; \theta), & \text{for non-terminal } s_{j+1}. \end{cases}$

      Update Q_θ by descending its stochastic gradient according to equations 3–17 and 3–19:

$$\nabla_\theta L(\theta) = (1 - \lambda) \cdot \nabla_\theta L^{\mathtt{A}}(\theta) + \lambda \cdot \nabla_\theta L^{\mathtt{B}}(\theta).$$

      Increase λ along the path p_λ.

    end if

  end for

end for

return the multi-objective Q-net $\boldsymbol{Q}_\theta$

Chapter 4 Evaluation

In this chapter, we evaluate the performance of our proposed value-based approaches in two environments: *Deep Sea Treasure (DST)*[23], a benchmark environment for multi-objective reinforcement learning; *Fruit Tree Navigation (FTN)*, which contains dozens of high-dimensional optimal solutions. We also provide *Muti-Objective Lunar Lander (MOLL)* environment for continuous control. These environments involve various possibly competing objectives to be optimized, while the preference in terms of the importance weight of each objective is unknown during the learning phase. The technical implementation details are discussed in Sections 4.2 and 4.3. Our source code is available at https://github.com/runzheyang/morl.

Recall that in section 2.2.3 we state our goals of solving delayed linear preference scenarios of multi-objective reinforcement learning as two-fold tasks: first, to efficiently learn representations for *all* linear optimal policies in the learning phase, i.e. to solve the multi-objective optimization and maintain potentially optimal policies; second, to efficiently select an optimal policy from learned representations corresponding to a specified preference in the execution phase, i.e. to align preferences with policies. In parallel to these goals, our purposes of synthetic experiments are to investigate the following questions: (1) Can both deep MORL algorithms efficiently find all the optimal solutions during the learning phase? (2) Does the Q-network learned by the envelope version of MORL algorithm has better adaptation ability than that of the naive algorithm in the execution phase? For better evaluate and compare the performance of our algorithms in above two dimensions, we propose two quantitative evaluation metric serving for this two purposes in section 4.1.

4.1 Quantitative Evaluation Metrics

In this section, we propose two quantitative evaluation metrics for multi-objective reinforcement learning. The first metric is *coverage ratio*, to evaluate an algorithm's ability to find optimal policies during the learning phase. The second metric is *adaptation quality*, which assesses the alignment between preferences and optimal policies in the execution phase.

4.1.1 Coverage Ratio

The coverage ratio is a measure of an agent's ability to find all the potential optimal solutions in the convex coverage set of Pareto frontier.

In an MOMDP, let $\mathcal{F}_{\Pi_{\mathcal{L}}} \subseteq \mathbb{R}^m$ be the set of solutions found by the agent (via sampled trajectories), $\mathcal{F}_Q \subseteq \mathbb{R}^m$ be the set of agent's guessed solutions (via predictions from Q-network), and $\mathtt{CCS}$ denote the convex coverage set. We define $\mathcal{F} \cap_\epsilon \mathtt{CCS} := \{x \in \mathcal{F} \mid \exists y \in \mathtt{CCS} \text{ s.t. } \|x - y\|_1 / \|y\|_1 \leq \epsilon\}$ as an intersection with the tolerance of a relative error of ϵ. The coverage ratio in is defined by an F1 score

$$\mathtt{CR}_{\mathrm{F1}}(\mathcal{F}) = 2 \cdot \frac{\mathtt{precision} \cdot \mathtt{recall}}{\mathtt{precision} + \mathtt{recall}}, \tag{4–1}$$

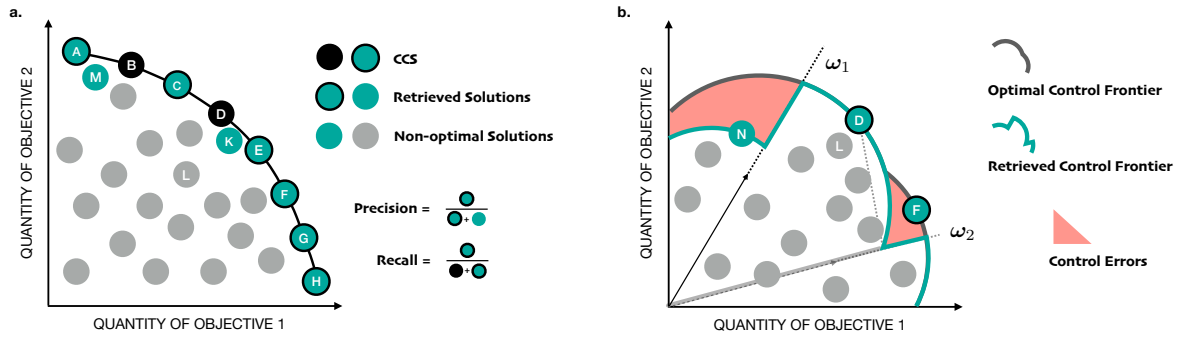

Figure 4–1 Quantitative evaluation metrics for multi-objective reinforcement learning. (a.) Coverage ratio measures an agent's ability to find all the potential optimal solutions in the convex coverage set of Pareto frontier. (b.) Adaptation quality measures an agent's ability of policy adaptation to real-time specified preferences.

where the $\texttt{precision} = |\mathcal{F} \cap_\epsilon \texttt{CCS}|/|\mathcal{F}|$, indicating the fraction of optimal solutions among the retrieved solutions, and the $\texttt{recall} = |\mathcal{F} \cap_\epsilon \texttt{CCS}|/|\texttt{CCS}|$, indicating the fraction of optimal instances that have been retrieved over the total amount of optimal solutions. The F1 score is their harmonic mean. In our evaluation of all the three synthetic tasks, we set $\epsilon = 0.00$ for $\mathcal{F}_{\Pi_\mathcal{L}}$ and $\epsilon = 0.20$ for $\mathcal{F}_Q$. Figure 4–1 (a.) illustrates an example of the computation of coverage ratio.

4.1.2 Adaptation Quality

To investigate an agent's ability of policy adaptation to real-time specified preferences, we introduce the metric of adaptation quality by comparing the optimal and retrieved *control frontiers*.

The optimal *control frontier* of the problem is defined by a function $C_{\text{opt}} : \Omega \to \mathbb{R}$ with $\boldsymbol{\omega} \mapsto \max_{\hat{\boldsymbol{r}} \in \text{CCS}} \boldsymbol{\omega}^\intercal \hat{\boldsymbol{r}}$. Similarly, $C_{\pi_{\boldsymbol{\omega}}} = \boldsymbol{\omega}^\intercal \hat{\boldsymbol{r}}_{\pi_{\boldsymbol{\omega}}}$ is the control frontier an agent can achieve (via sample trajectories), and C_Q is the control frontier guessed by an agent (via predictions from Q-network). The adaptation quality is defined by

$$\texttt{AQ}(C) = \frac{1}{1 + \alpha \cdot \texttt{err}_{\mathcal{D}_\omega}}, \tag{4–2}$$

where the $\texttt{err}_{\mathcal{D}_\omega} = \mathbb{E}_{\boldsymbol{\omega} \sim \mathcal{D}_\omega}[|C(\boldsymbol{\omega}) - C_{\text{opt}}(\boldsymbol{\omega})|/C_{\text{opt}}(\boldsymbol{\omega})]$ is the expected relative error, and α is a scaling coefficient to amplify the discrepancy. In all experiment domains, we use Gaussian distributions which are restricted to be positive part and $\ell 1$-normalized as our $\mathcal{D}_\omega$ and set $\alpha = 10.0$. Figure 4–1 (b.) shows examples of optimal control frontier, retrieved control frontier, and the control discrepancy.

4.2 Synthetic Environments

In this section, we introduce three synthetic environments for evaluating and comparing deep multi-objective reinforcement learning algorithms in delayed linear preference scenarios. The reason we use synthetic environments is that in these cases we have access to the ground truth. The calculation of coverage ratio

and adaptation quality requires the knowledge of optimal CCS and optimal control frontier, while in a real application it is hard to acquire.

4.2.1 Deep Sea Treasure

Our first experiment domain is a grid-world navigation problem, *Deep Sea Treasure*. This episodic problem was first explicitly created to highlight the limitations of linear scalarization[23]. However, in this thesis, we use this environment as a delayed linear preference scenario. We ensure the Pareto frontier of this environment is convex, therefore the Pareto frontier itself is the CCS.

In DST, an agent controls a submarine searching for treasures in a 10×11-grid world. The state s_t consists of the agent's current coordinates (x, y). The grid world contains 10 treasures with different values. Their values increase in as the distance from the starting point $s_0 = (0, 0)$ increase. An agent's action spaces are formed by navigation in four directions. The first dimension indicates a time penalty, which is -1 on all turns. The second dimension is the treasure value which is 0 except when the agent moves into a treasure location. We depicted the map in Figure 4–2 (a.).

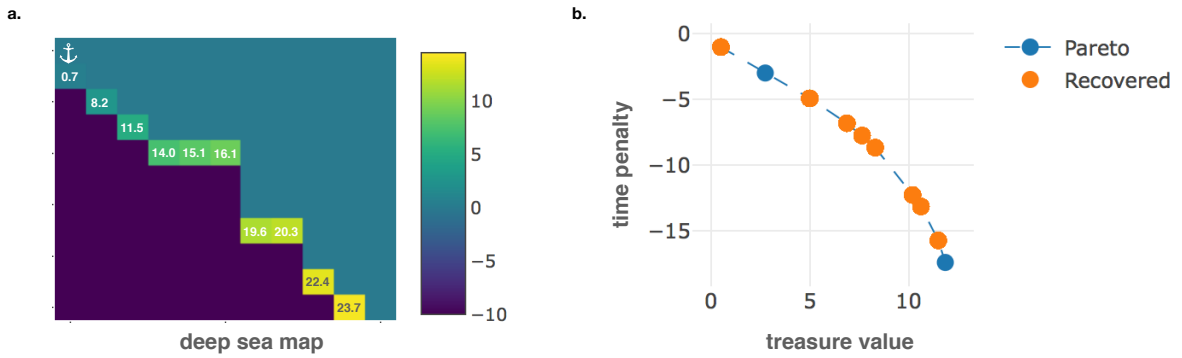

Figure 4–2 Deep Sea Treasure (DST): (a.) An agent controls a submarine searching for treasures in a 10×11-grid world. The state s_t consists of the agent's current coordinates (x, y). An agent's action spaces is navigation in four directions. The reward received by the agent is a 2-dimensional vector, (time penalty, treasure value). (b.) The DST Pareto frontier's convex coverage set is itself. The retrieved solutions by an MORL algorithm is also illustrated in this figure.

In the learning phase, the relative importance weight between the time penalty and the treasure value is unknown. The agent needs to learn and maintain all possible optimal policies which lead the shortest path to the desired treasure value with the highest utility. While in the execution phase, we hope the agent can control the submarine towards the treasure with the highest utility under a certain given preference.

4.2.2 Fruit Tree Navigation

Our second experiment domain, *Fruit Tree Navigation*, is a synthetic full binary tree of depth d with randomly assigned vectorial reward $\boldsymbol{r} \in \mathbb{R}^6$ on its leaf nodes, which encodes the amounts of six different components of nutrition of the *fruits* on the tree: {`Protein`, `Carbs`, `Fats`, `Vitamins`, `Minerals`, `Water`}. The goal of our

multi-objective reinforcement learning agent is to find a path from the root to a leaf node associating with a desired combination of nutrition. The preferred optimal leaf node has the best utility under a certain given preference.

Figure 4–3 shows an instance of the *fruit tree navigation* task when $d = 6$, in which every non-leaf node is associated with zero reward and every fruit is a potential optimal solution in the convex cover set of the Pareto frontier. To construct this, we sample $\boldsymbol{r}^{(i)} = (\boldsymbol{v}_+^{(i)} + \boldsymbol{v}_-^{(i)})/\|\boldsymbol{v}^{(i)}\|_2$, where $\boldsymbol{v}^{(i)} \sim \mathcal{N}_6(\mathbf{0}, \boldsymbol{I})$, for each fruit i on a leaf node. The optimal multiple-policy model for this tree structured MOMDP should contain all the paths from the root to different desired fruits. In experiments, we also test on $d = 5$ and $d = 7$ cases.

In such a simple multi-objective environment, an optimal policy can be easily learned if we know the preference function $f_{\boldsymbol{\omega}}(\cdot) = \langle \boldsymbol{\omega}, \cdot \rangle$ for scalarization. However, here we are interested in evaluating whether a multiple-policy neural network, trained with deep MORL algorithms, can find and maintain all the potential optimal policies (i.e., paths to every leaf node) when the preference function is unknown, and adapt to the optimal policy when a specific preference is given for execution.

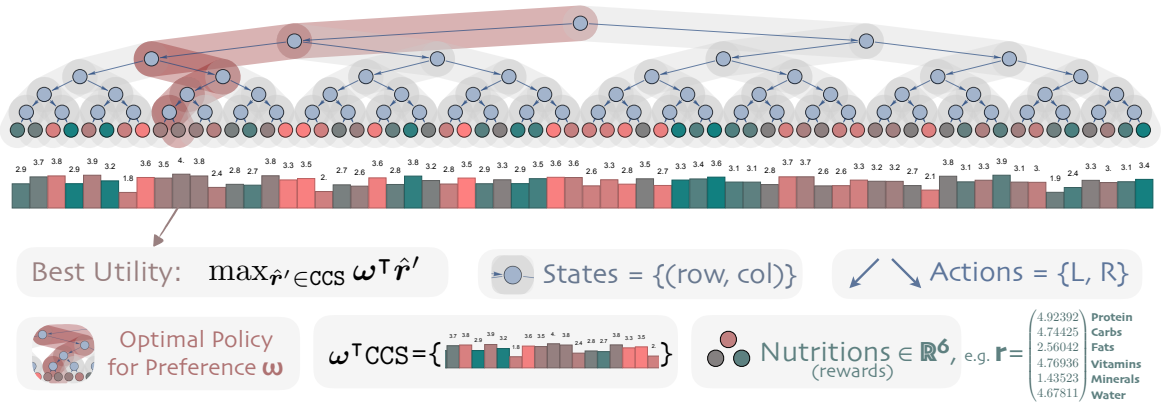

Figure 4–3 Fruit Tree Navigation (FTN): An agent travels from the root node to one of the leaf node to pick a fruit according to a post-assigned preference $\boldsymbol{\omega}$ on the components of nutrition, treated as different objectives. The observation of an agent is its current coordinates (`row`, `col`), and its valid actions are moving to the left or the right child node.

4.2.3 Multi-Objective Lunar Lander

The third synthetic environment, *Multi-Objective Lunar Lander*, is designed for continuous control, where the action space is a continuous space. We create this environment by modifying the single-objective Lunar Lander environment (https://github.com/dennisfrancis/LunarLander-v2).

In multi-objective lunar lander task, an agent controls a spacecraft to land on the mountain on the moon. There are three fixed landing pads on that mountain, of different height. The spacecraft has to land on one of these landing pads with low speed, otherwise it will be crashed. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. The state is an 8-dimensional vector:

- The first two entries s_1, s_2 represent the aircraft's coordinates (x, y).
- The horizontal speed and the vertical speed are s_3, s_4 respectively;
- Here s_4 is the angle of the aircraft, and s_5 is its angular speed;
- s_6, s_7 indicates whether the left leg / right leg makes contact with the ground.

The action space contains three operations, firing the main engine, firing left-side engine, and firing right-side engine. These actions can be discrete, or a continuous real value indicates the fuel consumption speed. The reward for moving from the top of the screen to landing pads consists of three objectives: fuel consumption, landing speed, and height, and has a same crashing penalty in each dimension. Figure 4–4 shows several screenshots of this environment.

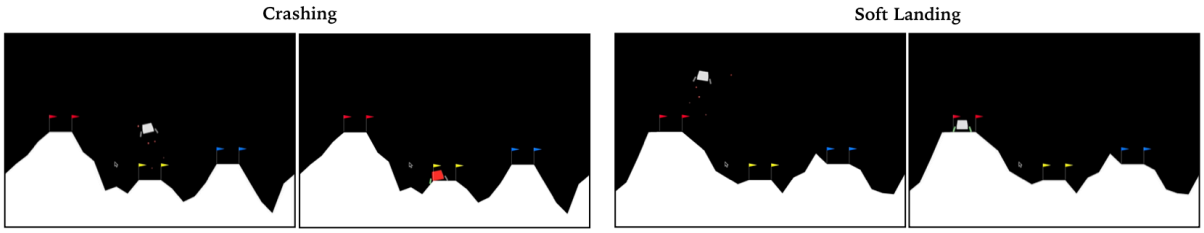

Figure 4–4 Screenshots of a video about the multi-objective reinforcement learning environment "Multi-Objective Lunar Lander". The task involves three objectives $\{\text{fuel consumption}, \text{landing speed}, \text{height}\}$. And this benchmark environment has two modes, compatible with discrete and continuous action space, respectively.

In the learning phase, the agent does not have any preference for those three objectives. It explores and learns how to land the spacecraft on one of the landing pads, with low speed, and low fuel consumption. However, in the execution phase, the agent is required to perform under a given specific preference, such as height is the first priority since in the valley the spacecraft cannot make a scientific observation. Then the agent has to control the spacecraft to move to the highest landing pad on the left, which increases the difficulty of landing task.

This environment is much more complex than previous two environments. In practice we need to employ an actor-critic algorithm rather than directly using the deep MORL algorithms based on deep Q-network. We do not evaluate our prototype algorithms on this environment, while we provide this environment for future research. Furthermore, our proposed deep MORL algorithms are compatible with those actor-critic methods[11] and energy-based methods[49], therefore this environment is suitable for all of these possible combinations.

4.3 Experiments: Naive Version versus Envelope Version

In this section, we evaluate and compare our proposed two deep MORL algorithms in above mentioned synthetic environments with two quantitative evaluation metrics. Our experimental settings are as follows.

Architectures of the Multi-objective Q-Network We implement the (utility-based) Multi-objective Q-networks (MQNs) by 4 fully connected hidden layers with $\{16, 32, 64, 32\} \times (\texttt{dim}(S) + m)$ hidden unites

respectively. The multi-objective Q-networks are similar to Deep Q-Networks (DQNs)[3], but differs on inputs. An input of the multi-objective Q-network is a concatenation of state representation and parameters of a linear preference function. The output layer of the naive MORL algorithm is of size $|\mathcal{A}|$, and that of envelope version is of size $m \times |\mathcal{A}|$. Here $\texttt{dim}(\mathcal{S})$ is the dimensionality of the state space, $|\mathcal{A}|$ is the cardinality of the action set, and m is the number of objectives.

Training with Prioritized Double Q-Learning When training the with our MORL algorithm, we employ techniques of *prioritized experience reply*[46] and *double Q-learning*[45] to speed up the training process and to yield more accurate value estimates. Double Q-Learning introduces a target network Q_{target} to replace the estimate of $\mathcal{T}Q(s,a,\boldsymbol{\omega})$, $y_t(\boldsymbol{\omega}) = \boldsymbol{\omega}^\intercal \boldsymbol{r} + \gamma Q_{\text{target}}(s',a,\boldsymbol{\omega})$ with $y_t^{\text{double}}(\boldsymbol{\omega}) = \boldsymbol{\omega}^\intercal \boldsymbol{r} + \gamma Q_{\text{target}}(s', \arg\max_a Q(s',a,\boldsymbol{\omega}),\boldsymbol{\omega})$ for naive version of algorithm, and similarly we replace $y_t(\boldsymbol{\omega}) = \boldsymbol{\omega}^\intercal \boldsymbol{r} + \gamma \boldsymbol{\omega}^\intercal \boldsymbol{Q}_{\text{target}}(s',a,\boldsymbol{\omega})$ with $y_t^{\text{double}}(\boldsymbol{\omega}) = \boldsymbol{\omega}^\intercal \boldsymbol{r} + \gamma \boldsymbol{\omega}^\intercal \boldsymbol{Q}_{\text{target}}(s', \arg\max_a \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s',a,\boldsymbol{\omega}),\boldsymbol{\omega})$ for envelope version of MORL algorithm. We update the target network by coping from Q-network every 100 steps.

The priority of sampling transition $\tau_i = (s,a,\boldsymbol{r},s')$ is $p_i^{\text{naive}} = |y^{\text{double}}(\boldsymbol{\omega}) - Q(s,a,\boldsymbol{\omega})|$ for the naive version of MORL algorithm, and similarly $p_i^{\text{envelope}} = |y^{\text{double}}(\boldsymbol{\omega}) - \boldsymbol{\omega}^\intercal \boldsymbol{Q}(s,a,\boldsymbol{\omega})|$ for the envelope version of algorithm, where $\boldsymbol{\omega}$ is sampled from the distribution $\mathcal{D}_\omega$. When updating the network, a trajectory is sampled by $\tau_i \sim P(i) = p_i / \sum_i p_i$. The replay memory size is 4000 and the batch size is 32. For the deep tree navigation task, we train each model for total 5000 episodes, and update it by Adam optimizer every step after at least a batch experiences are stored in reply buffer, with a learning rate $\texttt{lr} = 0.001$.

4.3.1 Effectiveness

Both our proposed algorithms are effective for delayed linear preference scenario. In the deep sea treasure environment, we train the agent for 2000 episodes. The training process of the naive deep MORL algorithm in the deep sea treasure environment are shown in Figure 4–5. As it illustrates, the multi-objective agent first

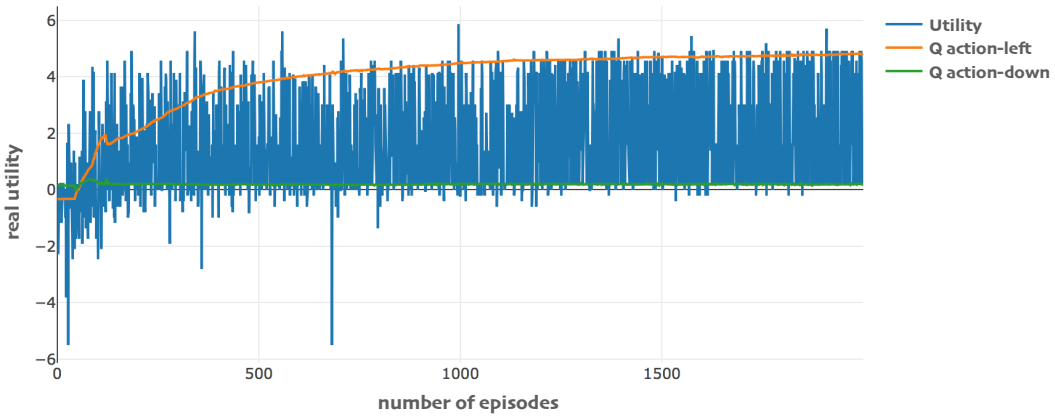

Figure 4–5 The training process of the naive deep MORL algorithm in the deep sea treasure environment. The agent learns moving down can only obtain very low utility under preference ($\texttt{len} = 0.2, \texttt{val} = 0.8$), while the predicted utility of moving up keep increasing during the whole learning process. The discount factor of this task is $\gamma = 0.99$.

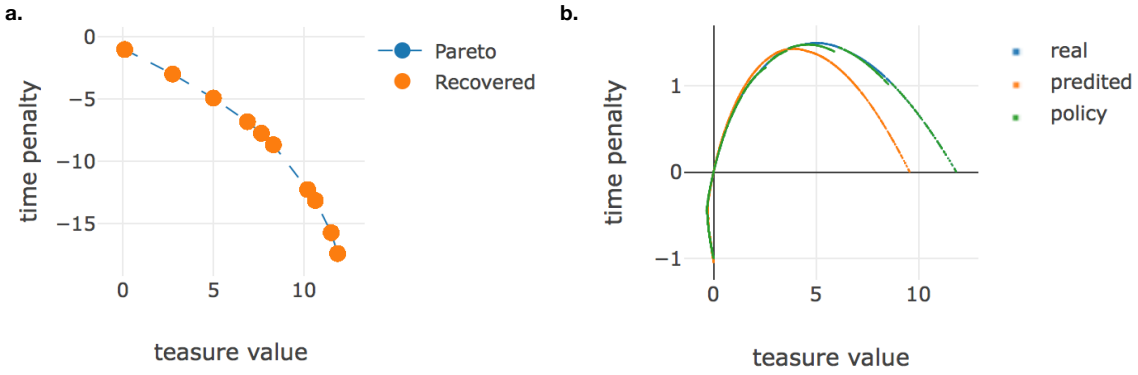

Figure 4–6 The solutions and control frontier found by a naive deep MORL algorithm in the deep sea treasure task. (a.) The real CCS and the retrieved solutions of naive deep MORL algorithm. The naive version algorithm already can efficiently learn all the potentially optimal solutions in this task. (b.) The real control frontier, retrieved control frontier in the execution phase, and the predicted control frontier in the analysis phase.

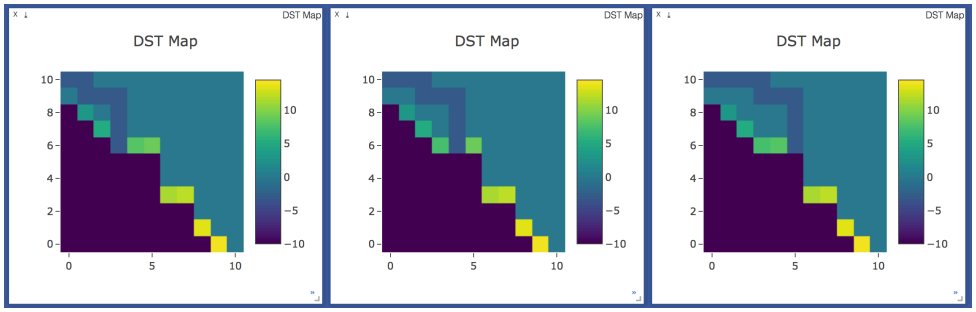

Figure 4–7 Demo of DST. We set there different preferences in the execution phase of deep sea treasure task. The agent can respond to our preferences with corresponding optimal policies.

learns moving down can only obtain very low utility under preference ($\texttt{len} = 0.2, \texttt{val} = 0.8$) quickly, while the predicted utility of moving up keep increasing during the learning process.

After training in the learning phase, our deep MORL algorithms find all the potential optimal solutions and their corresponding policies. Figure 4–6 (a.) presents the real CCS and the retrieved solutions of the naive deep MORL algorithm. The naive algorithm already can find all the whole CCS. Figure 4–6 (b.) illustrates the real control frontier (the blue curve), retrieved control frontier (the green curve) of the naive deep MORL agent, and its predicted control frontier (the orange line). The retrieved control frontier is almost overlapped with the real control frontier, which indicates that the alignment between preferences and optimal policies is perfectly well. The agent can respond any given preference with the policy resulting in best utility.

Figure 4–7 shows a demo of playing deep sea treasure with our multi-objective reinforcement learning agent. We set different preferences in the execution phase of deep sea treasure task. The agent can respond to our preferences with corresponding optimal policies to find the desired treasure by minimal steps.

The classical deep sea treasure task shows the effectiveness of our deep multi-objective reinforcement

learning algorithms, while it is too easy for the agent to find all the good policies. It only contains 10 potentially optimal solutions in the real CCS, therefore simple naive algorithm can efficiently solve this problem. In the next section, we compare the naive version and envelope version algorithms in the slightly more difficult fruit tree navigation task.

4.3.2 Sample Efficiency

In this section, we compare the sample efficiency of the naive version and envelope version deep MORL algorithms in the *Fruit Tree Navigation* (FTN) task as introduced in Section 4.2.2, where the depth of the tree is $d = 6$. For evaluating their coverage ratio (CR), as defined in Section 4.1.1, we trained on 5000 episodes and tested on 2000 episodes. For evaluating their adaptation quality, as defined in Section 4.1.2, we tested on 5000 episodes. The basic setup of the experiments is stated above. For the envelope deep MORL algorithm, we set the homotopy path controlled by λ to be a path where λ increase exponentially from 0.0 to 1.0.

Our approach to compare the sample efficiency of these two algorithms, is to fix the number of training episodes, which is equivalent to the number of interactions in this environment, and varies the number of sampled preferences N_ω for updating models in both algorithms. We increase N_ω from 1 to 128. When $N_\omega = 1$, both algorithms only update under single preference each time, therefore runs fast while it makes wasteful use of interactions. When $N_\omega = 128$, both algorithm needs to update for 128 sampled preferences in a batch. In this case the algorithms run slowly, while can make better use of interactions.

Table 4–1 compares their coverage ratio (precision, recall, f1 score). Each entry in the table is an average of 5 experiments (train and test). Due to the property of FTN task that every solution is potentially optimal, the CR precision is always 1 for both naive and envelope algorithm. While for the recall, which indicates the ability to find unseen optimal solutions in the learning phase, the envelope deep MORL algorithm is better than the naive version for all numbers of sampled preferences, and has relatively lower variances. Therefore the same to F1 scores. As N_ω increases, the CR value increases for both naive version and envelope version algorithms, which verifies aa better use of historical interactions for both algorithm when N_ω is larger. As

	CR Precision		CR Recall		CR F1	
N_ω	Naive	Envelope	Naive	Envelope	Naive	Envelope
1	1 ± 0	1 ± 0	0.4562 ± 0.058	0.8626 ± 0.084	0.625 ± 0.057	0.924 ± 0.051
4	1 ± 0	1 ± 0	0.6254 ± 0.097	0.972 ± 0.007	0.7654 ± 0.077	0.9856 ± 0.004
8	1 ± 0	1 ± 0	0.753 ± 0.101	0.9624 ± 0.014	0.856 ± 0.067	0.9808 ± 0.007
16	1 ± 0	1 ± 0	0.8188 ± 0.096	0.9904 ± 0.009	0.8976 ± 0.062	0.9952 ± 0.004
32	1 ± 0	1 ± 0	0.85 ± 0.061	0.975 ± 0.041	0.914 ± 0.044	0.987 ± 0.021
64	1 ± 0	1 ± 0	0.8968 ± 0.036	0.9812 ± 0.013	0.9452 ± 0.02	0.9904 ± 0.007
128	1 ± 0	1 ± 0	0.8626 ± 0.042	0.9906 ± 0.021	0.9258 ± 0.024	0.9952 ± 0.011

Table 4–1 Sample Efficiency - Coverage Ratio (CR) comparison of the naive deep MORL algorithm and the envelope deep MORL algorithm tested on fruit tree navigation task, where the tree depth $d = 6$. Trained on 5000 episode.

here the initial performance for the envelope version algorithm is good enough, it suddenly surpasses 0.98 of CR F1 score when it can sample more than one preference each update.

	Execution AQ		Prediction AQ	
N_ω	Naive	Envelope	Naive	Envelope
1	0.7037 ± 0.012	0.759 ± 0.066	0.6474 ± 0.054	0.6915 ± 0.105
4	0.7701 ± 0.026	0.9101 ± 0.006	0.7216 ± 0.034	0.7853 ± 0.049
8	0.8205 ± 0.023	0.9261 ± 0.015	0.7714 ± 0.02	0.7675 ± 0.033
16	0.8255 ± 0.044	0.9306 ± 0.007	0.8097 ± 0.016	0.8417 ± 0.007
32	0.8597 ± 0.035	0.9402 ± 0.011	0.7989 ± 0.032	0.8513 ± 0.01
64	0.877 ± 0.031	0.9506 ± 0.001	0.7778 ± 0.032	0.8497 ± 0.018
128	0.8705 ± 0.03	0.9536 ± 0.002	0.8081 ± 0.04	0.868 ± 0.025

Table 4–2 Sample Efficiency - Adaptation Quality (AQ) comparison of the naive deep MORL algorithm and the envelope deep MORL algorithm tested on fruit tree navigation task, where the tree depth $d = 6$. $\alpha = 10$. Trained on 5000 episode.

Table 4–2 compares their adaptation quality in both execution phase and analysis phase. Each entry in the table is an average of 5 experiments (train and test). For all numbers of sampled preferences, the envelope deep MORL algorithm has better execution AQ than the naive algorithm, in spite of better CR of the envelope version, it also indicates the envelope version algorithm has better alignment between preferences and policies. As N_ω increases, the values of execution AQ and prediction AQ of both algorithms keep increase. Notice that even though when $N_\omega = 1$ the execution AQ of the naive version and the envelope version differs only around 0.5, when N_ω increase to 4, the envelope version algorithm better utilizes the sampled preferences to improve the execution AQ to above 0.9. This agrees with our theoretical analysis that our envelope deep MORL algorithm has better sample efficiency than the naive version.

4.3.3 Robustness on Size of the Frontier

	CR Recall		CR F1		CR F1 (Prediction)	
N_ω	Naive	Envelope	Naive	Envelope	Naive	Envelope
1	0.8814 ± 0.041	0.9436 ± 0.051	0.9364 ± 0.023	0.9706 ± 0.027	/	0.1768 ± 0.066
4	0.969 ± 0.031	1 ± 0	0.984 ± 0.016	1 ± 0	/	0.559 ± 0.067
8	0.9938 ± 0.014	1 ± 0	0.9968 ± 0.007	1 ± 0	/	0.6424 ± 0.031
16	1 ± 0	1 ± 0	1 ± 0	1 ± 0	/	0.7338 ± 0.018
32	1 ± 0	1 ± 0	0.9968 ± 0.007	1 ± 0	/	0.769 ± 0.018
64	0.9938 ± 0.014	1 ± 0	0.9968 ± 0.007	1 ± 0	/	0.8012 ± 0.025
128	1 ± 0	1 ± 0	1 ± 0	1 ± 0	/	0.827 ± 0.015

Table 4–3 Sample Efficiency - Coverage Ratio (CR) comparison of the naive deep MORL algorithm and the envelope deep MORL algorithm tested on fruit tree navigation task, where the tree depth $d = 5$. Trained on 5000 episode.

We also investigate how the size of optimality frontiers will affect the performance of our algorithms. We train our deep MORL algorithms on two new FTN environments with $d = 5$ and $d = 7$ respectively. One is smaller than the previous environment, which contains only 32 optimal solutions, the other is larger than the previous environment, containing 128 solutions on the CCS. We fix the number of episode for training as 5000, and test 2000 episode to obtain coverage ratio, and test 5000 episode for policy adaptation quality.

Table 4–3 shows the results of coverage ratio evaluation in the environment of tree depth $d = 5$. As it shows, both naive and envelope algorithms work well in that environment. the CR F1 scores are very close to 1. The envelope version algorithm is more stable than the naive version deep MORL algorithm. Besides, the envelope deep MORL algorithm can predict multi-objective solutions, while the naive algorithm cannot. The prediction ability will also be improved as N_ω increases.

	CR Recall		CR F1		CR F1 (Prediction)	
N_ω	Naive	Envelope	Naive	Envelope	Naive	Envelope
1	0.2312 ± 0.029	0.4674 ± 0.068	0.3748 ± 0.038	0.6348 ± 0.063	/	0.03 ± 0.022
4	0.3452 ± 0.057	0.5234 ± 0.074	0.5112 ± 0.063	0.6846 ± 0.067	/	0.067 ± 0.031
8	0.3688 ± 0.042	0.6032 ± 0.05	0.5376 ± 0.046	0.7516 ± 0.039	/	0.1436 ± 0.022
16	0.4358 ± 0.065	0.5798 ± 0.052	0.6052 ± 0.061	0.7328 ± 0.043	/	0.1406 ± 0.008
32	0.3954 ± 0.06	0.65 ± 0.067	0.5646 ± 0.061	0.7862 ± 0.049	/	0.1596 ± 0.031
64	0.447 ± 0.045	0.597 ± 0.109	0.6168 ± 0.043	0.743 ± 0.086	/	0.2006 ± 0.046
128	0.4624 ± 0.063	0.6876 ± 0.053	0.6308 ± 0.058	0.8138 ± 0.038	/	0.1902 ± 0.009

Table 4–4 Sample Efficiency - Coverage Ratio (CR) comparison of the naive deep MORL algorithm and the envelope deep MORL algorithm tested on fruit tree navigation task, where the tree depth $d = 7$. Trained on 5000 episode.

Table 4–4 shows the results of coverage ratio evaluation in the larger environment of tree depth $d = 7$. As it indicates, both naive and envelope algorithms work badly in that environment within only 5000 episodes for training. However, even in this case, the envelope version algorithm performance than the naive version deep MORL algorithm in the learning phase Besides, the envelope deep MORL algorithm can predict multi-objective solutions, while the naive algorithm cannot. The prediction ability will also be improved as N_ω increases in this case, which means our method is robust in the learning phase on the size of CCS.

As for the adaptation quality, Table 4–5 shows the results of execution AQ and prediction AQ evaluation in the environment of tree depth $d = 5$. As it shows, the envelope deep MORL algorithm in this simple environment can reach above 0.9 execution AQ by sampling few preferences each update. It verifies the better sample efficiency of the envelope version algorithm. Furthermore, though the prediction AQ of these two algorithms in this simple task is compatible, while the envelope version algorithm can make more steady predictions when N_ω goes large.

Table 4–6 presents the results of execution AQ and prediction AQ evaluation in the environment of tree depth $d = 5$. As it shows, the both naive and envelope deep MORL algorithms work not so well in the environment within 5000 episodes for training. However, the envelope version algorithm is constantly better

than the naive version algorithm in this hard task. Moreover, as N_ω increases, the execution AQ and the prediction AQ of both algorithms increase. And the envelope version algorithm gains a better improvement when N_ω increases from 1 to 4 because of its better sample efficiency. These results strongly support the robustness of our algorithms.

	Execution AQ		Prediction AQ	
N_ω	Naive	Envelope	Naive	Envelope
1	0.7943 ± 0.039	0.8578 ± 0.036	0.7153 ± 0.079	0.6254 ± 0.041
4	0.8836 ± 0.01	0.9041 ± 0.005	0.8195 ± 0.021	0.7843 ± 0.055
8	0.8975 ± 0.003	0.9099 ± 0.001	0.8427 ± 0.022	0.7952 ± 0.023
16	0.9047 ± 0.003	0.9109 ± 0.001	0.8698 ± 0.01	0.8488 ± 0.032
32	0.9054 ± 0.003	0.9113 ± 0.001	0.8647 ± 0.014	0.8847 ± 0.006
64	0.9096 ± 0.003	0.9119 ± 0	0.8761 ± 0.024	0.8731 ± 0.008
128	0.9071 ± 0.004	0.9121 ± 0	0.8535 ± 0.033	0.8809 ± 0.026

Table 4–5 Sample Efficiency - Adaptation Quality (AQ) comparison of the naive deep MORL algorithm and the envelope deep MORL algorithm tested on fruit tree navigation task, where the tree depth $d = 5$. $\alpha = 10$. Trained on 5000 episode.

	Execution AQ		Prediction AQ	
N_ω	Naive	Envelope	Naive	Envelope
1	0.624 ± 0.012	0.6646 ± 0.025	0.5847 ± 0.061	0.60 ± 0.029
4	0.6704 ± 0.006	0.730 ± 0.018	0.6969 ± 0.057	0.6544 ± 0.066
8	0.6763 ± 0.022	0.7237 ± 0.018	0.6837 ± 0.097	0.7437 ± 0.04
16	0.6867 ± 0.025	0.7473 ± 0.023	0.6532 ± 0.029	0.7936 ± 0.015
32	0.6828 ± 0.022	0.7204 ± 0.05	0.6829 ± 0.037	0.7997 ± 0.017
64	0.671 ± 0.06	0.7429 ± 0.015	0.7284 ± 0.03	0.8112 ± 0.018
128	0.6916 ± 0.015	0.7586 ± 0.024	0.6719 ± 0.076	0.8199 ± 0.008

Table 4–6 Sample Efficiency - Adaptation Quality (AQ) comparison of the naive deep MORL algorithm and the envelope deep MORL algorithm tested on fruit tree navigation task, where the tree depth $d = 7$. $\alpha = 10$. Trained on 5000 episode.

4.3.4 High-Dimensional Visualization

The multi-objective reinforcement learning problem involves high-dimensional rewards. To visualize the results obtained by our naive and envelope deep MORL algorithms would be very interesting. In this section, we show several visualization examples. This example could provide us insights into the characters of our proposed deep multi-objective reinforcement learning algorithms.

Figure 4–8 illustrates the comparison of CCS and control frontier of deep MORL algorithms. Both figure (a.) and (b.) are measured on a fruit tree navigation task with the depth of 6 which contains total 64

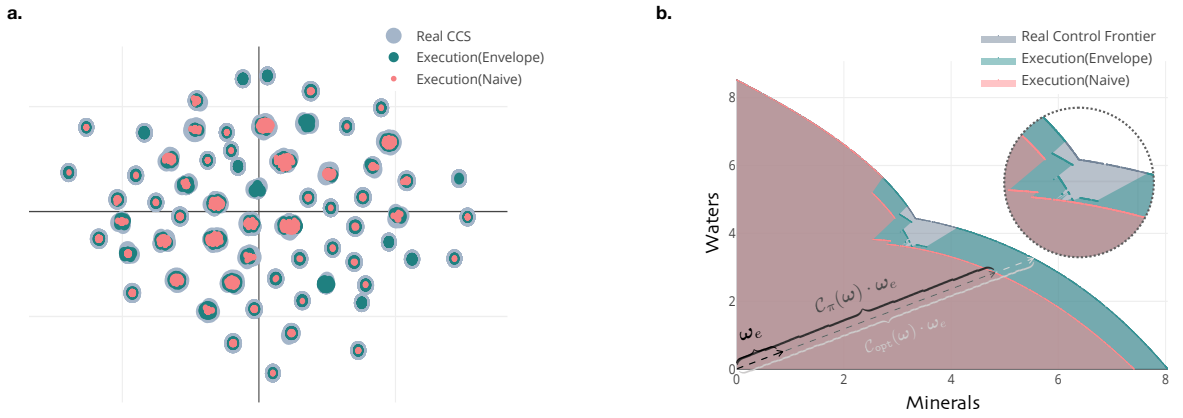

Figure 4–8 Comparison of CCS and control frontiers of deep MORL algorithms. Both figure (a.) and (b.) are measured on a fruit tree navigation task with the depth of 6 which contains total 64 solutions in the real CCS. The left figure is visualizing the real CCS and retrieved CCS of naive and envelope MORL algorithms using t-SNE[50]. The right figure presents the slices of optimal control frontier and the control frontier of naive and envelope MORL algorithms along the `Mineral-Waters` plane.

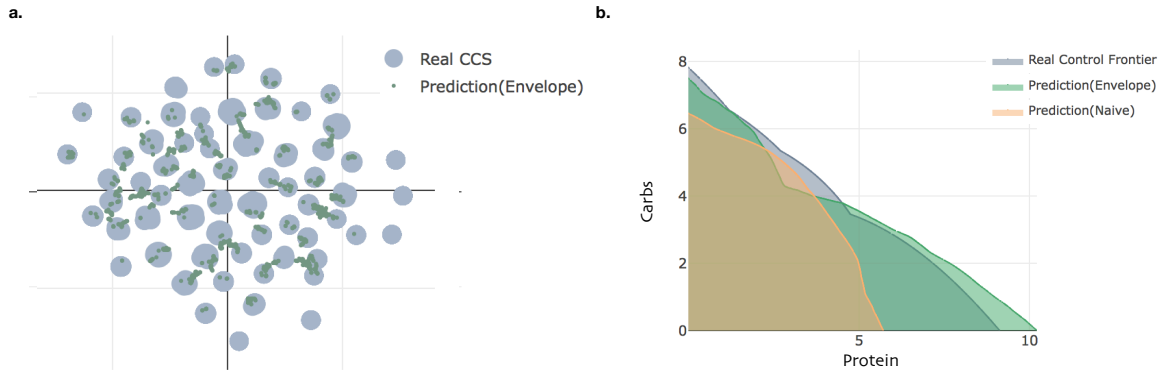

Figure 4–9 Comparison of predicted CCS and control frontiers of deep MORL algorithms. Both figure (a.) and (b.) are measured on a fruit tree navigation task with depth of 6 which contains total 64 solutions in the real CCS. The figure (a.) visualizing the real CCS and predicted CCS of envelope MORL algorithms using t-SNE[50]. The figure (b.) presents the slices of optimal control frontier and the predicted control frontier of naive and envelope MORL algorithms along the `Protein-Carbs` plane.

solutions in the real CCS. The left figure is visualizing the real CCS and retrieved CCS of naive and envelope MORL algorithms using t-SNE[50]. From this figure we can see that the envelope version algorithm (dots in green) covers almost every optimal solutions in the real CCS than the naive algorithm (dots in pink). The right figure presents the slices of optimal control frontier and the control frontier of naive and envelope MORL algorithms along the `Mineral-Waters` plane. The envelope version algorithm retrieves almost the whole real control frontier, while the naive version algorithm has larger control discrepancies. However, there is a small discrepancy between the real control frontier and the one retrieved by envelope version algorithm at the indentation of the frontier. This indicates an alignment issue between the preferences and optimal policies.

Figure 4–9 illustrates the comparison of predicted CCS and predicted control frontiers of deep MORL algorithms. Both figure (a.) and (b.) are measured on a fruit tree navigation task with the depth of 6 which contains total 64 solutions in the real CCS. The left figure visualizing the real CCS and predicted CCS of envelope MORL algorithms using t-SNE[50]. This figure would help us in the analysis phase in the delayed linear preference scenario. The right figure presents the slices of optimal control frontier and the predicted control frontier of naive and envelope MORL algorithms along the `Protein-Carbs` plane. The prediction of the envelope version algorithm is much better than that of the naive version algorithm.

4.4 Summary

In this chapter, we evaluate the performance of our proposed two value-based deep MORL algorithms 3–1 and 3–2. First, we propose two quantitative evaluation metrics, *coverage ratio* (CR) and *adaptation quality* (AQ) in Section 4.1. The coverage ratio is a measure of an agent's ability to find all the potentially optimal solutions in the convex coverage set of the Pareto frontier in the learning phase, which can be defined as an F1 score. The adaptation quality measures an agent's ability of policy adaption to real-time specified preference in the execution phase, which is defined based on the control frontier and control error.

To test our algorithms, we designed three synthetic multi-objective reinforcement learning environments. The reason we use synthetic environments is that in these cases the ground truth of Pareto frontier and control frontier is known. Our first domain is a grid-world problem, *deep sea treasure* (DST), which is a classical benchmark environment for MORL. The second domain is called *fruit tree navigation* (FTN), which contains dozens of high-dimensional optimal solutions. Our third environment, *multi-objective lunar lander* (MOLL) environment, is designed for continuous control and more complex deep Q-learning models.

The experiments on DST gives an appetite to show the effectiveness of our proposed algorithms. Both algorithms can work very well on this simple task in both learning phase and execution phase. To compare the sample efficiency of our algorithms, we test them on the FTN environment. Our results consistently show that the envelope version algorithm has better CR and AQ than the naive version algorithm after training on the same number of episodes. Besides, as the number of sampled preference each update increases, both algorithms get improvement, and the improvement for the envelope version algorithm is more than that of the naive version. These results are robust on the size of the frontiers. We also visualize our results with several dimension reduction methods.

Chapter 5 Applications in Task-Oriented Dialogue Policy Learning

Following the thorough theoretical and experimental analysis about deep multi-objective reinforcement learning algorithms, this chapter provides practical applications of our proposed algorithms in task-oriented dialogue policy learning. We start with a brief introduction to *task-oriented spoken dialogue systems* (SDS) in Section 5.1. The task-oriented dialogue systems consist of many components severing for different functionalities, among which the *dialogue manager* (DM) plays a key role in controlling the interaction with users. This module decides a dialogue system how to respond to user' requests according to a dialogue policy. Designing robust dialogue policies is very time-consuming and expensive.

Section 5.2 discuss the current trends to learn a dialogue policy from interactions by using deep reinforcement learning techniques. Traditional reward design is a weighted average of different objectives. How to balance these objectives is rarely considered. To address this critical problem, we reconsider dialogue policy learning as a delayed linear preference scenario of multi-objective reinforcement learning.

We apply our proposed naive and envelop deep MORL algorithms in the online dialogue learning process, and compare to traditional single-objective methods. Our method can provide the more robust performance to users with different preferences. The experiment settings and results are presented in Section 5.3.

5.1 Task-Oriented Dialogue Systems

The field of task-oriented spoken dialogue systems (Task-Oriented SDS) has developed rapidly in the academic[51-54] and industry[55-57] over the past two decades. Speech-driven interfaces are becoming a common part of daily life through applications such as in-car navigation systems, smart home dictation or telephone-based information services. Task-oriented spoken dialogue systems are no doubt a killer app for artificial intelligence. From the perspective of technology, a task-oriented spoken dialogue system has following three important features comparing to a common chatbot or an information retrieval system.

- It is required to **satisfy user goals**. The task-oriented dialogue system serves the cause of satisfying user's information needs in a certain domain (or relative domains). For example, a task-oriented dialogue system can help users make a restaurant reservation, or answer a weather information query, or give direction information when driving.
- It is required to maintain **multi-round interactions**. Our daily conversational interactions are multi-round processes. A single utterance may lead to ambiguity, and the goal of the task-oriented dialogue may be revealed only after multiple rounds. This requires task-oriented dialogue systems to maintain the context and the user information.
- It is required to deal with **uncertainty**. There are two sources of uncertainty for a conversation, *recognition* and *understanding*. A good task-oriented dialogue system should be robust on errors from speech recognition and language understanding.

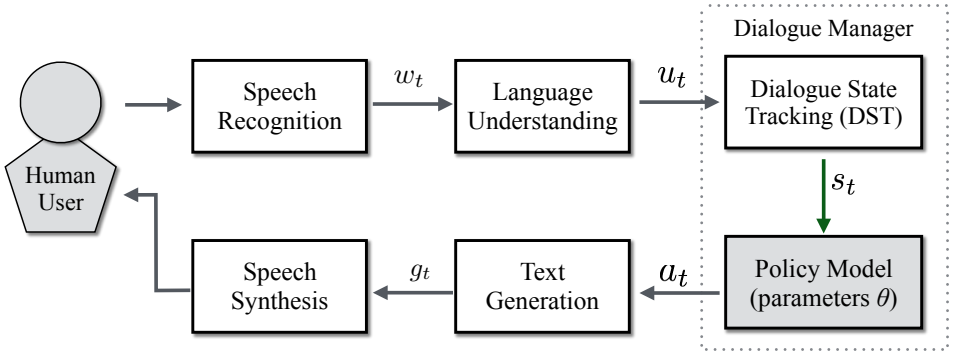

Figure 5–1 The general framework of a task-oriented dialogue system

The general framework of a task-oriented dialogue system is visualized in Figure 5–1. On the input understanding side, the speech recognition module receives the acoustic signal emitted by the human user, maps it to the most k likely sequence of words w_t and their confidence scores. The text output of the speech recognition module is then semantically decoded by a speech understanding module to a structural dialogue act, u_t. The dialogue acts are usually formed by `acttype` – `slot` – `value`, where `acttype` is the type of intent of the utterance action, which can be “inform()", “confirm()", or “request()", etc. `slot` is the semantic object mentioned in the utterance, like “restaurant", “phone number", “price", etc. `value` is the concrete reference of the semantic slot, such as “restaurant=pizzahut", “phone number=233", and “price=$1". For example, “inform(route=61a)" is a dialogue act to inform the system that the user is talking about bus route 61a. On the output generation side, the text generation module translates the high-level representation of the system’s dialogue acts a_t into a sequence of words g_t. And the speech is synthesized using a text-to-speech module to respond to the human user.

The mission of the *dialogue manager* (DM) at the central part of a task-oriented dialogue system is to control the flow of a conversation, tackle the uncertainty arising from speech recognition and understanding errors, and perform forward satisfying user goal. The dialogue manager can be divided into two parts, *dialogue state tracker* and *dialogue policy model*. The dialogue state tracker is designed to interpret probability distribution of user dialogue act u_t, resolves contextual references, and updates the dialogue state s_t. A dialogue state is a structural representation of the user goal, current semantics, and the dialogue history.Based on the state s_t , an appropriate system response a_t is then rendered following the system’s dialogue policy which defines the system’s conversational behavior. In the next section, we will introduce the current trends of constructing dialogue policy module, and explain why a multi-objective reinforcement learning setting is beneficial to the dialogue policy learning.

5.2 Dialogue Policy Learning

Generally, approaches to constructing a policy module can be divided into two categories: rule-based and statistical. Rule-based policies are usually hand-crafted by domain experts which means they are inconvenient

and difficult to be optimized[58, 59]. In recent mainstream statistical studies, *partially observable Markov decision process* (POMDP) framework has been applied to model dialogue management with unobservable states, where policy training can be formulated as a reinforcement learning problem, which enables the policy to be optimized automatically[6, 60].

As a standard single reinforcement learning formulation, the goal of the policy model is to interact with a human user by choosing actions in each turn to maximize future rewards. We define the dialogue state shared by dialogue state tracker in the t-th turn as state s_t. The action taken by policy model under current policy π_θ with parameters θ in the t-th turn as a_t, and $a_t \sim \pi(\cdot|s_t)$. The stochastic transition kernel is unknown but determined by human users or user simulators. In an ideal dialogue environment, once the policy model emits an action a_t , the human user will give an explicit feedback, like a normal response or a feedback of whether the dialogue is successful, which will be converted to a reward signal r_t delivering to the policy model immediately, and then the policy model will transit to next state s_t. The reward signal is an average of values of two objectives, *brevity* and *success*, which can be expressed as

$$r_t = 0.5 \cdot r_t^{\text{turn}} + 0.5 \cdot r_t^{\text{succ}} \tag{5–1}$$

where r_t^{turn} is the turn penalty reward and r_t^{succ} is the dialogue success reward. Typically, r_t^{turn} is fixed for each turn as a negative constant R^{turn}, while r_t^{succ} equals a positive constant R^{succ} only when the dialogue terminates and receives a successful user feedback otherwise it equals zero. Many reinforcement learning algorithm can be therefore apply to dialogue policy learning, such as deep Q-learning[3].

Though RL-based approaches have the potential to improve themselves as they interact more with human users and achieve better performance than rule-based approaches, they are rarely used in real-world applications, especially in on-line scenarios, mainly because of two reasons. The first reason is that the on-line training process is usually unsustainable, and the second reason is that the fixed scalarized reward cannot best fit all users' preferences.

5.2.1 Companion Teaching

This section aims at introducing several new approaches to solve unsustainable and unaffordable on-line dialogue policy learning problem. The main causes of this problem are two-fold:

- Safety issue: the initial policy trained from scratch may lead to terrible user experience at the early training period, thus fail to attract sufficient users for more dialogues to do further policy training.
- Efficiency issue: if the progress of policy learning is not so efficient, it will exhaust users' patience before the policy reaches a desirable performance level.

Prior works have mainly focused on improving *efficiency*, such as Gaussian Processes RL[61], deep RL[62], etc. For deep RL approaches, recent researches on the *student-teacher RL framework* have shown prominent acceleration to policy learning process[63-65]. In such framework, the *teacher agent* instructs the *student agent* by providing suggestions on what actions should be taken next[66].

For the safety issue, paper [67] first proposes a human-in-the-loop framework. Several teaching strategies are devised to answer "how" the human teacher guide the learning process. However, these strategies simply

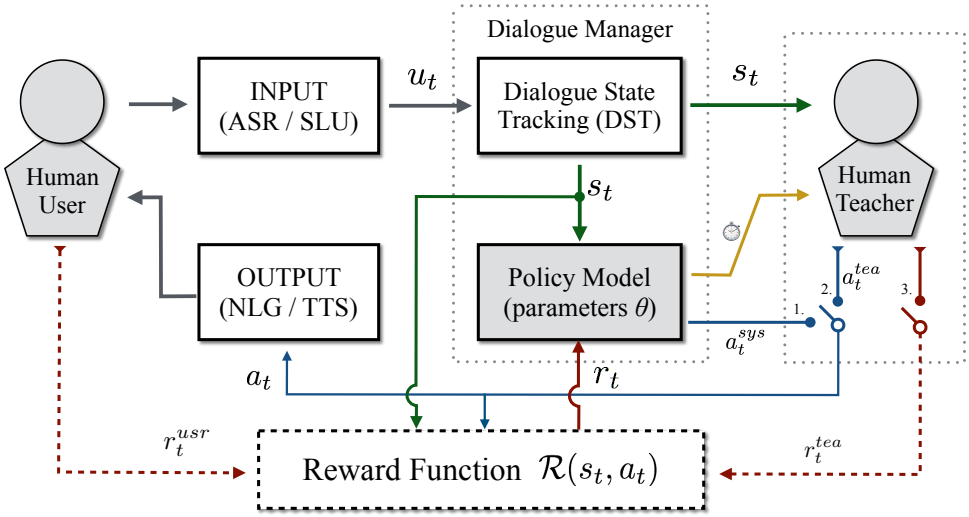

Figure 5–2 Companion teaching framework for on-line dialogue policy learning

exhaust all the budget continuously from the beginning, which is wasteful and causes a heavy workload of the human teacher. An affordable dialogue policy learning with human teaching should require a lighter workload and economically utilize teaching budget. Followed that work, a complete *Companion Teaching* framework[68] is proposed to include “when" to teach in concern. As depicted in Figure 5–2, at each turn, the input module receives speech input from the human user, then produces possible utterances u_t of the speech in text. After that, the dialogue state tracker extracts the dialogue state s_t from possible utterances. This dialogue state will be shared with policy model and human teacher if needed. When the final response a_t , has been determined, the output module will translate this dialogue act to the natural language and reply to the human user. The success signal will be fed back by the user or the human teacher as an important part of system reward, at the end of each session. The human teacher can take the initiative or be activated by *student-initiated heuristic* to give the dialogue guidance with strategies corresponding to different configurations of switches in the illustration. The combination of strategy and heuristic is called *teaching scheme*.

The teacher can choose among three teaching strategies corresponding to different configurations of switches in a wiring diagram as Figure 5–2 shows: The left switch is a Single-Pole, Double-Throw (SPDT) switch, which controls whether the answer is made by the system (connected to 1) or given by the teacher as an example (connected to 2). The right switch is a simple on-off switch, which represents whether there is an extra reward signal from the teacher (ON) or not (OFF). The strategy related to the right switch is called *teaching via Critic Advice* (CA), also known as turn-level reward shaping[69, 70]. When the switch at position 3 is turned on, the teacher will give the policy model extra turn-level reward to distinguish the student’s actions between good and bad actions. Besides, the left switch corresponds to *teaching via Example Action* (EA), which means the teacher gives example action for the student to take according to the student’s state. The other strategy is proposed by[67], which take the advantages of both EA and CA, named *teaching via Example Action with Predicted Critique* (EAPC). With this strategy, the human teacher gives example

actions; meanwhile, a weak action predictor is trained using this teaching information to provide the extra reward even in teacher's absence.

In addition to teaching from the beginning, many other teaching heuristics are explored in this work, such *State Importance based Teaching heuristic* (SIT), *State Uncertainty based Teaching heuristic* (SUT), and *Failure Prognosis based Teaching heuristic* (FPT) using multi-task deep Q-networks. Experiments showed that different teaching schemes have different effects on safety and efficiency dimension. And these teaching schemes also require different workload of the teacher. Among 18 compared teaching schemes, FPT-based heuristics combined with EAPC strategy achieved promising performance on RI and HT, and required relatively slight workload. This result indicates a proper teaching scheme under the companion teaching framework is able to guarantee a sustainable and affordable on-line dialogue policy learning process. The paper [71] discusses a more economic approach for companion teaching, in which the human teachers are replaced by rule-based systems.

5.2.2 Multi-Objective Rewards

Companion teaching provides us with a safe and efficient on-line dialogue learning process. However, as we set the fixed scalarized reward for the dialogue learning, the obtained optimal policy cannot fit all users' preference, and sometimes is out of our expectation. For example, different users may have different preferences to the brevity and the success of conversation with dialogue. Some users interact with the task-oriented dialogue system often when they are driving cars. They hope the dialogue system can provide them a concise response in few turns, rather than a long conversation with very precise information.

Adaptation to user preferences and balancing these objectives is rarely considered. Finding a good balance between multiple potentially competing objectives is usually domain-specific and not straightforward. For example, in the case when the objectives are brevity and success, if the relative importance weight for success is too high, the resulting policy is insensitive to potentially annoying actions such as "repeat()" provided that the dialogue is eventually successful. Conversely, if the relative importance weight for success is too small, the learning algorithm may annoy users by offering inappropriate solutions before fully knowing the user's requirements. Paper [72] proposes a structured method, which is equivalent to our naive method but without hindsight experience replay, for finding the optimal weights for a multi-objective reward function.

The idea to transform the dialogue learning process to a delayed preference scenario multi-objective reinforcement learning is straightforward. With the same definitions of states and actions introduced in Section 5.2, we change the form of reward single to a vector from:

$$\boldsymbol{r}_t = \begin{bmatrix} r_t^{\text{turn}} & r_t^{\text{succ}} \end{bmatrix}^{\intercal} \tag{5–2}$$

where r_t^{turn} is the turn penalty for the brevity objective and r_t^{succ} is the task success reward for the success objective. Typically, r_t^{turn} is fixed for each turn as a negative constant R^{turn}, while r_t^{succ} equals a positive constant R^{succ} only when the dialogue terminates and receives a successful user feedback otherwise it equals zero.

In the learning phase, the linear preference $\boldsymbol{\omega}$ to this two objectives are unknown, while the computational resources are abundant. The task-oriented dialogue system needs to learn all the possible optimal policies with sampled $\boldsymbol{\omega}$'s (achieved by user simulators or collected interactions with real users). While in the execution phase, learning is unaffordable because of the limitation of resources. The task-oriented dialogue system needs to respond the user with a specified user preference. User's utility increase is aligned with the system's utility increase $\boldsymbol{\omega}^\intercal \boldsymbol{r}_t$.

Comparing with single-objective reinforcement learning methods, where the relative importance weights of the system is always fixed in both learning and execution phases. The system may not learn the optimal responses under certain preferences, or execute following a policy mismatched user's preference therefore the user's utility is low. In the next section, we apply our two proposed deep MORL algorithms to the dialogue policy learning, and compare to traditional single-objective methods.

5.3 Experiments

The goal of our experiments in this section is to investigate the applicability of our proposed deep MORL algorithms in task-oriented dialogue policy learning. We are supposed to answer, first, how will the multi-objective reinforcement learning affect the efficiency of training process? Second, what is the optimality frontier for the brevity and success objectives in a dialogue application? Third, how are the policy adaptation abilities of our proposed deep MORL algorithms. Only by investigating all these questions, and comparing MORL algorithms to traditional single-objective methods, we can conclude the benefits we may gain from applying our proposed deep multi-objective reinforcement learning algorithms to the task-oriented dialogue system domain.

5.3.1 Experimental Settings

The multi-objective reinforcement learning algorithm for task-oriented dialogue policy learning is applied to a restaurant lookup hotline to provide information restaurants in Cambridge. There are 3 search constraints, 9 informational items that the user can request, and 110 database entities. The reward R^{turn} is -1 for each turn, and $R^{\text{succ}} = 20$. The maximal length of dialogue is 25.

We use the statistical spoken dialogue toolkit PyDial[73] to generate dialogue simulations, where an agenda-base user simulator[74] with an error model to simulate the recognition and understanding errors arisen in the real system due to in the speech noise and ambiguity. All the single-objective and multi-objective reinforcement learning are trained for 3,000 sessions with 15% simulated error rate. We evaluate learned policies on 5,000 sessions with randomly assigned user preferences. The preference distribution $\mathcal{D}_\omega$ is same as the one we used in previous deep sea treasure experiment (see Section 4.3.1), which is a nearly uniform distribution.

For the single-objective reinforcement learning algorithms, we set three groups of $\boldsymbol{\omega}$ as $\{(0.5, 0.5)$, $(0.2, 0.8)$, $(0.8, 0.2)\}$. For the envelope deep MORL algorithms, the homotopy path is a monotonically increasing track where λ increases from 0.0 to 1.0 exponentially. The number of sampled preferences N_ω

is 32 for both naive and envelope deep MORL algorithms. The exploration policy used for training these reinforcement learning algorithms is ϵ-greedy, where $\epsilon = 0.5$ initially and then decays to zero linearly during the training process. For all the single-objective and multi-objective algorithms, we employ the same deep Q-network architecture, which comprises 3 fully connected hidden layers with $\{16, 32, 32\} \times (\texttt{dim}(S) + m)$ hidden units. The minibatch size is 64 for all. An Adam optimizer is used for updating the parameters of all these algorithms, with an initial learning rate $\texttt{rl} = 0.001$.

5.3.2 Results and Analyses

Evaluation on Training Processes We first investigate the training process of the single-objective and multi-objective method for dialogue policy learning. Figure 5–3 shows the training curves of moving average success rate of single-objective and multi-objective reinforcement learning algorithms for the dialogue policy learning. Each data point is the average over 500 most recent dialogues and three trails. As shown in this figure, the single-objective reinforcement learning methods with different weights of success achieve higher success rates faster than two multi-objective methods. This is because the single-objective methods only need to take care of one fixed preference and always update their parameters for that preference. While the multi-objective method needs to update for all preferences in the learning phase, and can only update for few sampled preference each step. However, in a long-term, the multi-objective methods can achieve competitive success rates to the single-objective method. Especially when we assume we have abundant computational resources in the learning phase, and off-policy learning is always available, this problem will not restrict the applicability of multi-objective reinforcement methods.

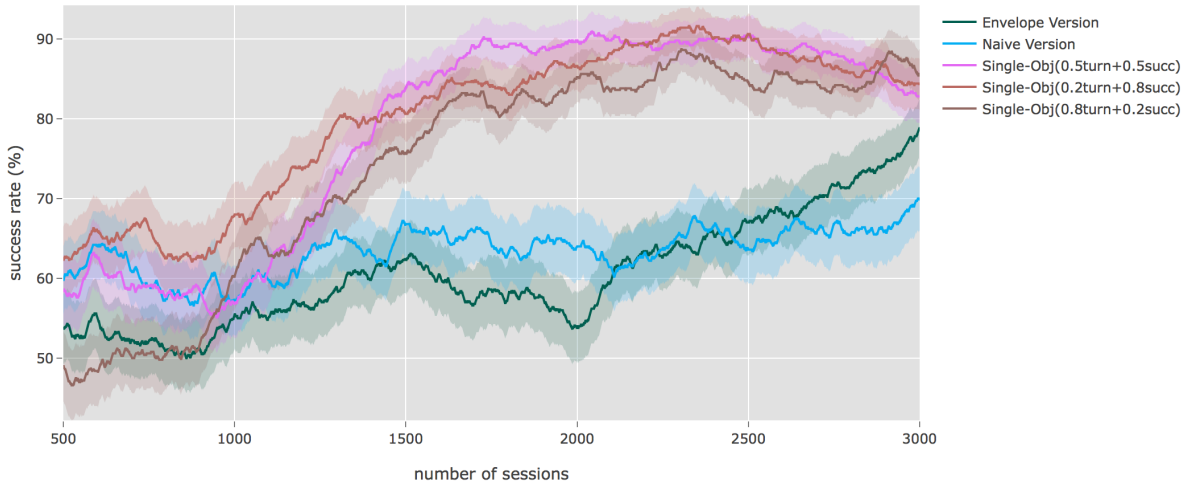

Figure 5–3 Training curves of moving average success rate of single-objective and multi-objective reinforcement learning algorithms for the dialogue policy learning. Each data point is the average over 500 most recent dialogues and three trails.

Figure 5–4 shows the training curves of moving average dialogue length of single-objective and multi-objective reinforcement learning algorithms for the dialogue policy learning. Each data point is the average

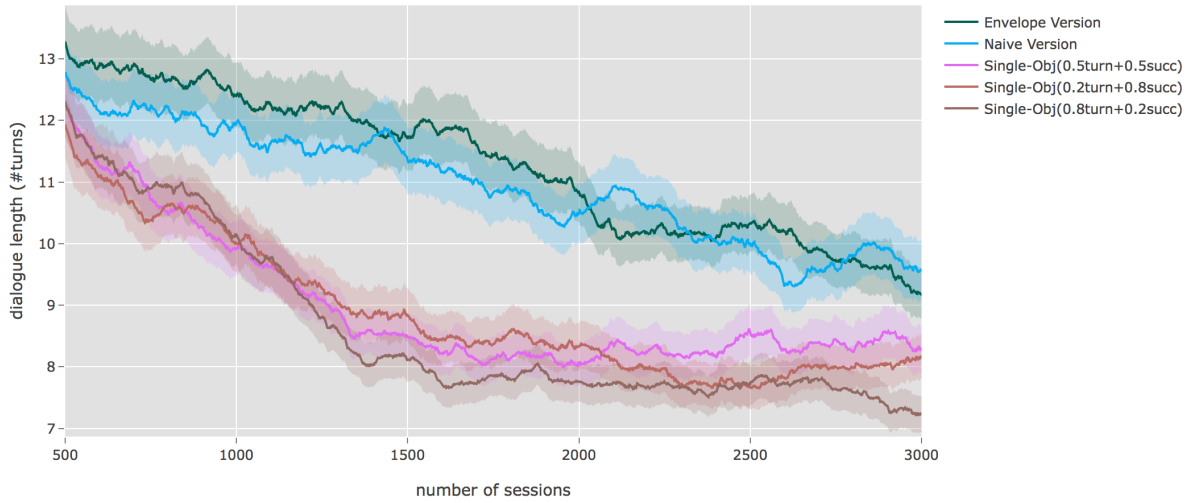

Figure 5–4 Training curves of moving average dialogue length of single-objective and multi-objective reinforcement learning algorithms for the dialogue policy learning. Each data point is the average over 500 most recent dialogues and three trails.

over 500 most recent dialogues and three trails. As the number of learned sessions increases, the dialogue length of each policy the decreases, which indicates the brevity of the conversation for each policy gets better. The single-objective reinforcement learning methods with different weights of success achieve lower dialogue length faster than two multi-objective methods. However, the multi-objective method can achieve the close level eventually.

Evaluation on Different Objectives The Pareto frontier of the dialogue policy learning consists of only one solution ideally, since the cumulative reward $[-2, 20]$ dominates all the other solutions. However, in the real world is almost impossible to build such a policy to satisfy any user's goal within a two-turn interaction. This

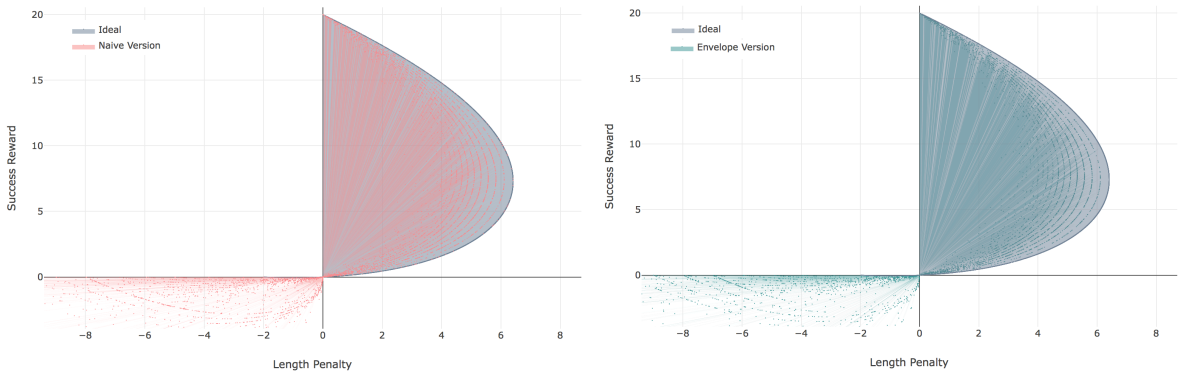

Figure 5–5 Control frontiers of the task-oriented dialogue policy learning. The ideal control frontier is generated by an ideal policy always achieving the ideal optimal reward [-2,20] (make any dialogue successful in two turns). Comparing to it, we draw the execution control frontiers retrieved by naive and envelope deep MORL algorithms, respectively.

	Single-(0.5,0.5)	Single-(0.2,0.8)	Single-(0.8,0.2)	Naive	Envelope
Success Rate	88.18 ± 0.90	85.30 ± 0.98	87.62 ± 0.91	86.38 ± 0.95	**89.52 ± 0.85**
# Turns	8.93 ± 0.13	9.40 ± 0.16	**7.42 ± 0.10**	8.08 ± 0.12	8.08 ± 0.12
User Utility	2.13 ± 0.23	1.84 ± 0.23	2.53 ± 0.22	2.38 ± 0.22	**2.65 ± 0.22**
AQ ($\alpha = 0.1$)	0.660	0.279	0.728	0.614	**0.814**

Table 5–1 The average success rate, number of turns, user utility, and adaptation quality (AQ) (see Section 4.1.2) of policies obtained by single-objective and multi-objective reinforcement learning algorithms after 3,000 training dialogues. We evaluate them on 5,000 dialogues with near-uniformly randomly sampled preference.

ideal solution can be used as the ground truth for evaluating our different MORL algorithms. Since there is only one solution in the CCS, the measurement of coverage ratio is meaningless, while the evaluation related to the control frontier will be still meaningful. Figure 5–5 shows the control frontiers of the task-oriented dialogue policy learning. The ideal control frontier is generated by an ideal policy always achieving the ideal optimal reward $[-2, 20]$ (make any dialogue successful in two turns). Comparing to it, we draw the execution control frontiers retrieved by naive and envelope deep MORL algorithms, respectively. Notice that the control frontiers of these two deep MORL algorithm have many tracks of boundaries. This is because in the dialogue task the initial states are different for different sessions. Some initial states are easier, therefore the system can achieve better utility, while some are harder, the retrieved solution is far worse than the ideal solution. The envelope version algorithm's frontier has thinker outer boundaries than that of the naive version, which indicates the envelope version algorithm behaves better for more initial states. This can also be revealed by our quantitative evaluation metric.

Table 5–1 is the quantitative evaluation results for different dialogue policy learning methods. We evaluate the average success rate, the average dialogue length, the average user utility, and the adaptation quality (AQ) of policies obtained by single-objective and multi-objective reinforcement learning algorithms after 3,000 training dialogues. The evaluation is performed on 5,000 dialogues with near-uniformly randomly sampled preference. Our results are as follows: First, the envelope version MORL algorithm achieves best success rate in average, and the single-objective method with 0.5 weight of success achieves a competitive success rate. The single-objective method with 0.8 success weight has worst performance on success rate eventually. This is might because lack of turn level reward guidance directly optimize for success is very difficult for the single-objective RL learner. Second, the single-objective method with 0.2 success weight (emphasizing the brevity more) achieves the shortest dialogue length. While our two deep MORL algorithms also perform well for the average brevity, and are much better than other two single-objective settings. Third, as for the average user utility, the envelope deep MORL is better than others. The naive deep MORL achieves only an average level of the single-objective results (better than the ones with 0.5, 0.8 success weights, but worse than the one 0.2 with success weight). One possible explanation for this result is that the naive deep MORL requires more sessions for training, while 3,000 dialogues are not enough. This explanation can be supported by our analyses of training processes. Finally, we use the ideal control frontier to compute

the adaptation quality (AQ) with $\alpha = 0.1$ (see its definition in Section 4.1.2) for all learned policies, and the envelope version MORL algorithm reaches the highest score. This supports our statement that MORL algorithm can better adapt to different user preferences. The AQ score of naive version algorithm is close to those of well-trained single-objective methods, and is much better than the one with 0.8 success weight. Applying MORL algorithm can help us avoid choosing improper relative importance weights for training.

Evaluation on Policy's Adaptation Ability We have already known that the deep MORL algorithms can help us train better policy overall without carefully choosing relative importance weights. How do they make it? The answers lie in their policy adaptation ability to different user preferences in the execution phase. Figure 5–6 is the MORL success-weight curves after 3,000 training dialogues. We evaluate policies on 5,000 dialogues with near-uniformly randomly sampled preference. For each curve, each data point is a moving average of closest around 500 dialogues in the interval of around ± 0.05 weight of success over three trained policies. As shown in this figure, the success rates of the MORL algorithms increase as the user's weight of success increases, while the success rates of the single-objective algorithms do not change. This means that our MORL methods can adapt to the user's preference, while the single-objective methods cannot. Furthermore, our envelope deep MORL algorithm outperforms other algorithms on the success rate when the success objective is relatively more important to the user.

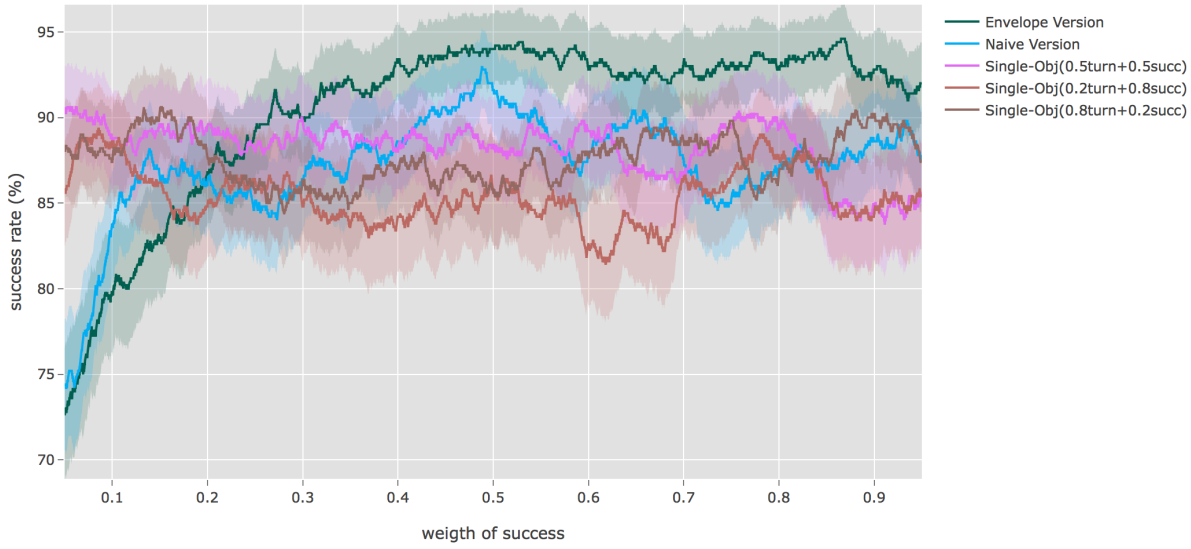

Figure 5–6 The MORL success-weight curves after 3,000 training dialogues. We evaluate policies on 5,000 dialogues with near-uniformly randomly sampled preference. For each curve, each data point is a moving average of closest around 500 dialogues in the interval of around ± 0.05 weight of success over three trained policies.

Figure 5–7 shows similar results for the brevity objective. Although the single-objective method with fixed preference $(0.8, 0.2)$ achieves the shortest dialogue length under all user preferences, two MORL algorithms also achieve a close level. The other single-objective baselines are not so good at brevity due to

Figure 5–7 The MORL success-weight curves after 3,000 training dialogues. We evaluate policies on 5,000 dialogues with near-uniformly randomly sampled preference. For each curve, each data point is a moving average of closest 500 dialogues in the interval of around ± 0.05 weight of success over three trained policies.

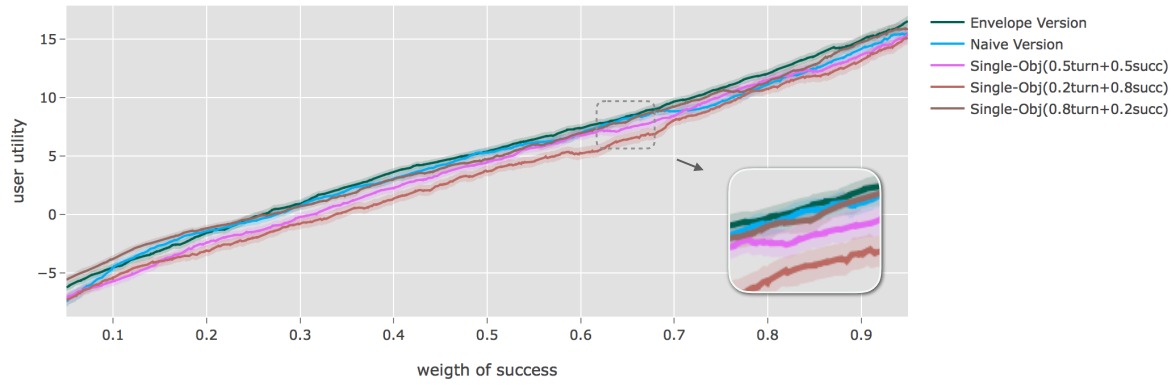

Figure 5–8 The MORL utility-weight curves after 3,000 training dialogues. We evaluate policies on 5,000 dialogues with near-uniformly randomly sampled preference. For each curve, each data point is a moving average of closest 500 dialogues in the interval of around ± 0.05 weight of success over three trained policies.

their training on fixed preferences emphasizing more on the success objective. Moreover, when the user's weight of success is close to 1, which means the length of the dialogue is not important, our MORL algorithms can sacrifice a bit brevity to ensure the success rate is above 90%. As shown in Figure 5–8, which illustrates utility-weight curves, we find the envelope deep MORL algorithm is almost always better than other methods in terms of utility, and the naive version keeps a good level of utility under almost all user preferences. Single-objective methods are good only when the user's weight of success is close to their fixed preferences while training.

5.4 Summary

In this chapter we discuss the applications of proposed deep multi-objective reinforcement learning algorithms in task-oriented dialogue systems. In Section 5.1 we briefly introduce the *task-oriented spoken dialogue system* (SDS). A task-oriented dialogue system is a modular system with many components, among which the dialogue manager plays the central role to control the flow of dialogue. The dialogue manager, consisting of dialogue state tracker and dialogue policy model, interacts with the user on the intention level. We also introduce the structure of dialogue states and the dialogue acts in this section.

Generally speaking, approaches of constructing a policy module can be divided into two categories: rule-based and statistical. We focus on the application in the mainstream statistical dialogue policy learning because of its evolvability. The dialogue learning can be formulated as a Markov decision process and solved by reinforcement learning. We introduce this part in Section 5.2. Although RL-based approaches have the potential to improve themselves, they are rarely used in real-world applications, mainly because first the on-line training process is usually unsustainable, and second the fixed scalarized reward cannot best fit all users' preferences. For addressing the first problem, we introduce a companion teaching framework which takes human in the learning loop. For solving the second problem, we propose a new vectorial reward design to balance and express different importance weights of the brevity objective and success objective for dialogue policy learning.

Our experiments verify the applicability and potential benefits of our previously proposed deep multi-objective reinforcement learning (MORL) algorithms in dialogue policy learning. Though it may take more sessions to train a MORL algorithm in the learning phase, our envelope deep MORL algorithm outperforms all the single-objective reinforcement learning algorithms on average success rate and user utility, and has a short average dialogue length close to that of the single-objective one emphasizing the brevity objective very much. Deep MORL algorithm can learn good policies without carefully specifying the relative importance weights for brevity and success objectives. But for a single-objective method, if we fix improper weights for training, the performance of the resulting policy will be bad. Moreover, the policies learned by deep MORL algorithm can better adapt to user preferences during the execution phase. For example, when the weight of brevity is low and the weight of success is high, the envelope deep MORL algorithm can sacrifices the brevity a bit to ensure the dialogue success rate above 90%. A task-oriented dialogue system using deep MORL algorithm will be more favored by users in practice.

Chapter 6 Conclusion

In this thesis, our goal is to answer the following question: "Can we design efficient and practical multi-objective reinforcement learning algorithms in the delayed linear preference scenario, for acquiring all potentially optimal policies in the learning phase, and adapting optimally to any real-time specified preference in the execution phase?" To answer this question affirmatively, we have discussed a theoretical framework and two proposed algorithms for multi-objective reinforcement learning. We have also discussed the quantitative evaluation in several synthetic environments and one real application in task-oriented dialogue policy learning. In this final chapter, we first reiterate our contributions and discuss their implications for the fields of multi-objective reinforcement learning and dialogue policy learning in Section 6.1. In Section 6.2, we critically review our own work and outline several possibilities for future research.

6.1 Contributions

In this section, we summarize our contributions. Our contributions can be divided into three categories: *theory contributions*, *evaluation contributions*, and *application contributions*. We start with our contributions to multi-objective reinforcement learning theory, then go into the contributions to evaluating MORL algorithm with quantitative measurements and synthetic environments, end with our contributions to apply proposed MORL algorithms to task-oriented dialogue policy learning. For our contributions, we briefly discuss the implications that we think may be advantageous for future research in these research directions.

Theory Contributions

On the theoretical side, we first formulate a general triphasic scenario called *delayed linear preference scenario*, which characterizes most realistic situations for multi-objective reinforcement learning. In this scenario, we only consider linear preferences parameterized by the relative importance weights of objectives. The delayed linear preferences scenario begins with a learning phase, where the agent is free to access the historical trajectory records or interacts with the simulated environment to learn about the rewards, while the preference is unknown. The aim of the agent in this phase is to find all linear optimal policies. Then in an analysis phase, the user can analyze the trade-off between multiple objectives of the task for determining a suitable preference using the information learned by the agent. The last phase of the delayed linear preference scenario is the execution phase. In this phase, a specific linear preference function will be given to the agent, while learning is no longer allowed. The agent needs to respond with an optimal policy achieving the best utility to the given preference. The delayed linear preference scenario is worth studying as a standard scenario for future research in multi-objective reinforcement learning.

We then propose a theoretical framework for developing and analysis value-based reinforcement learning algorithms. This framework is inspired by the Banach's fixed point theorem. Five essential components are

involved in this framework for designing value-base reinforcement learning algorithms: (1) value space, (2) value metric, (3) evaluation operator, (4) optimality operator, and (5) updating scheme. The optimality operator is a contraction in the value space, therefore iteratively applying it we can derive a fixed point in that value space, which is the optimal value function. Many existing value-based reinforcement learning algorithms can be explained under this theoretical framework. We use the framework to help us develop value-based algorithms for the delayed linear preference scenario.

The first algorithm is called naive deep MORL algorithm, which is a straightforward extension of the value-based single-objective reinforcement learning algorithm. It maintains the utility estimation for all state-action pairs and for all relative importance weights. The updating scheme for this algorithm uses the deep neural network to approximate the Q-value function, and hindsight experience replay technique for parallel updating parameters. This algorithm is easy to implement, and is compatible with many advanced reinforcement learning methods. However, as indicated by our analysis, the uninformative prediction and sample inefficiency restrict the applicability the performance of this algorithm.

To address two limitations of the naive version algorithm, we design the envelope deep MORL algorithm by directly maintaining the multi-objective solutions, and solving a convex envelope of the explored value frontier each update step. This modification requires us to generalize the Banach's theorem to pseudo-metric space to proof its convergence. Besides, when updating with hindsight experience reply, we use homotopy method to optimize the weighted average of two loss functions. Although the envelope deep MORL increase the hardness of implementation, and will cost more time for each update step, it is more sample-efficient during the learning phase and can provide with more informative prediction during the analysis phase.

Our proposed two algorithms are novel in the field of multi-objective reinforcement learning. They can be regarded as multiple-policy methods, while having notable differences with existing methods. First, they can not only learn a set of policies for obtaining the approximate optimality frontier, but also adapt learned policy to user preferences in the execution phase. Comparing with the previous manifold-based methods, our methods can better solve the delayed linear preference scenario. Second, by using deep neural networks and hindsight experience replay, our methods use a single neural network to maintain all possible optimal policies, and utilize the batch of trajectories more efficiently, while prior methods require to store multiple models and update those models in different iterations. We hope our two proposed deep MORL algorithms can provide a new paradigm and inspiration for future MORL algorithms research.

Evaluation Contributions

In addition to the theoretical contributions, we make contributions to evaluating the performance of deep multi-objective reinforcement learning in the delayed linear preferences scenario, by proposing two quantitative evaluation metrics and creating several synthetic environments.

Our first evaluation metric is the *coverage ratio* (CR), to measure a MORL agent's ability to find all the potentially optimal solutions in the convex coverage set of the Pareto frontier in the learning phase, which can be defined as an F1 score. The second evaluation metric is the *adaptation quality* (AQ), to measure agent's ability to adapt its policy to real-time specified preferences in the execution phase, which is defined based on

the control frontier and control error. These two metrics are technically sound, and would be useful to future MORL algorithms research, especially for the ones focusing on delayed linear preference scenarios.

Three synthetic multi-objective reinforcement learning environments are created to test our deep MORL algorithms. The reason we user synthetic environments is that in these cases the ground truth of Pareto frontier and control frontier is known. Our first environment is a grid-world navigation problem, *deep sea treasure*, which is a classical benchmark environment for MORL. The second environment is called *fruit tree navigation* (FTN), which contains dozens of high-dimensional optimal solutions. Our third environment, *multi-objective lunar lander* (MOLL) environment, is designed for continuous control and more complex deep Q-learning model. There are few available multi-objective reinforcement learning environments in the community. We make our code open source to provide researchers in MORL community with several standard environments for testing their algorithms in the future.

We compared proposed two deep MORL algorithm in two synthetic environments. The experiments on DST environment shows the effectiveness of our proposed algorithms. Both algorithms can work very well on this simple task in both learning phase and execution phase. By testing two proposed algorithms on the FTN environment, our results consistently show that the envelope version algorithm has better CR and AQ than the naive version algorithm after training on the same number of episodes. Besides, as the number of sampled preference each update increases, both algorithms get improvement, and the improvement for the envelope version algorithm is more than that of the naive version. These results are robust on the size of the frontiers and align with our theoretical analysis that the envelope version algorithm has better sample efficiency. We also visualize our results with several dimension reduction methods. Our visualization methods can also be used in the future research to gain insights into MORL algorithms.

Application Contributions

Finally, our contributions on the application side are demonstrated by an example of the task-oriented spoken dialogue system. A task-oriented dialogue system is a modular system with many components, among which the dialogue manager plays the central role to control the flow of dialogue. Approach to construct a policy model in the dialogue manager can be divided into two categories: rule-based and statistical. We focus on the application in the mainstream statistical dialogue policy learning, which is often formulated as a Markov decision process and solved by reinforcement learning. So far, RL-based approaches are rarely used in real-world applications. One reason is that the fixed scalarized reward balancing multiple objectives, such as brevity and success, limits the performance of dialogue policy learning algorithms. The obtained policy cannot best fit all users' preferences and is sometimes out of our expectation. For example, some users interact with the system often when they are driving cars. These users may hope the system can provide them a concise response in few turns, rather than a long conversation with very precise information.

Adaption to user preferences and balancing multiple objectives is rarely considered. In our thesis, we consider the dialogue learning process as a delayed linear preference scenario of multi-objective reinforcement learning. The rewards in this setting are vectors describing dialogue length penalty for the brevity objective and task success reward for the success objective. Our proposed deep MORL algorithm can learn almost all

optimal policies during a learning phase, and adapt to user preference without extra learning when the system is deployed online.

Our experiments confirm the applicability and show potential advantages of our proposed deep MORL algorithms in dialogue policy learning. Though it may take more sessions for training, our envelope deep MORL algorithm outperforms all the single-objective reinforcement learning algorithms on average success rate and user utility, and has a short average dialogue length close to that of the single-objective one emphasizing the brevity objective very much. In other words, our deep MORL algorithms can learn good policies without carefully specifying the relative importance weights for brevity and success objectives. Moreover, the policies learned by deep MORL algorithm can better adapt to user preferences during the execution phase. For example, when the weight of brevity is low and the weight of success is high, the envelope deep MORL algorithm can sacrifices the brevity a bit to ensure the dialogue success rate above 90%. This kind of task-oriented dialogue system using our deep MORL algorithm will be more favored by users in real application scenarios.

6.2 Discussion and Future Work

In this section, we critically review the work in this thesis, and identify possibilities for future work.

First, all the discussions about multi-objective reinforcement learning in this thesis aim at solving the delayed linear preferences scenarios. The optimality concept in this scenario is the convex coverage set of the Pareto frontier. However, the Pareto frontier is unnecessarily to be convex. And in many cases, we are interested in the non-convex solutions. How we extend our algorithms to general *delayed preference scenarios* would be an interesting research problem.

Second, our proposed two deep MORL algorithms are compatible with many advanced reinforcement learning techniques. However, in this thesis we only explored their combinations with double Q-learning and prioritized experience replay. These algorithms therefore cannot apply to domains with continuous control. Though these two methods are value-based, many state-of-the-art reinforcement learning algorithms mix value-based and policy-based methods can also be embedded into them, such as the deterministic deep policy gradient (DDPG) and the asynchronous advantage actor-critic (A3C). Empowered by these methods, we can run our algorithms in more domains, such as ones requiring continuous control. Our proposed multi-objective lunar lander environment is a good testbed. Embed these advanced algorithms into our deep MORL algorithms is a potential future direction.

Third, in our task-oriented dialogue systems application, the vectorized reward only considers brevity objective and success objective. However, there are many other objectives should be considered for a good conversation, such as “coherence" of the logic of context, “clearness" of the provided information, and the “humor" if users expect the dialogue system to be funny. Moreover, there are also some long-term objectives for the dialogue policy learning, like the safety and efficiency of the learning process itself. How to evaluate and incorporate these objectives into multi-objective dialogue policy learning is a challenging research topic.

Bibliography

[1] RUSSELL S J, NORVIG P. Artificial Intelligence - A Modern Approach (3. internat. ed.)[M/OL]. [S.l.]: Pearson Education, 2010. http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html. ISBN: 978-0-13-207148-2.

[2] SILVER D, SCHRITTWIESER J, SIMONYAN K, et al. Mastering the game of go without human knowledge[J]. Nature, 2017, 550(7676): 354.

[3] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning[J/OL]. Nature, 2015, 518(7540): 529-533. https://doi.org/10.1038/nature14236. DOI: 10.1038/nature14236.

[4] SEGLER M H, PREUSS M, WALLER M P. Planning chemical syntheses with deep neural networks and symbolic AI[J]. Nature, 2018, 555(7698): 604.

[5] BANINO A, BARRY C, URIA B, et al. Vector-based navigation using grid-like representations in artificial agents[J]. Nature, 2018: 1.

[6] YOUNG S J, GASIC M, THOMSON B, et al. POMDP-Based Statistical Spoken Dialog Systems: A Review[J/OL]. Proceedings of the IEEE, 2013, 101(5): 1160-1179. https://doi.org/10.1109/JPROC.2012.2225812. DOI: 10.1109/JPROC.2012.2225812.

[7] SUTTON R S, BARTO A G. Reinforcement learning - an introduction[M/OL]. Adaptive computation and machine learning. [S.l.]: MIT Press, 1998. http://www.worldcat.org/oclc/37293240. ISBN: 0262193981.

[8] LECUN Y, BENGIO Y, HINTON G E. Deep learning[J/OL]. Nature, 2015, 521(7553): 436-444. https://doi.org/10.1038/nature14539. DOI: 10.1038/nature14539.

[9] NACHUM O, NOROUZI M, XU K, et al. Bridging the Gap Between Value and Policy Based Reinforcement Learning[C/OL]// Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. [S.l.]: [s.n.], 2017: 2772-2782. http://papers.nips.cc/paper/6870-bridging-the-gap-between-value-and-policy-based-reinforcement-learning.

[10] WILLIAMS R J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning[J/OL]. Machine Learning, 1992, 8: 229-256. https://doi.org/10.1007/BF00992696. DOI: 10.1007/BF00992696.

[11] KAKADE S. A Natural Policy Gradient[C/OL]// Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]. [S.l.]: [s.n.], 2001: 1531-1538. http://papers.nips.cc/paper/2073-a-natural-policy-gradient.

[12] PETERS J, MÜLLING K, ALTUN Y. Relative Entropy Policy Search[C/OL]// Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010. [S.l.]: [s.n.], 2010. http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1851.

[13] GU S, LILLICRAP T P, GHAHRAMANI Z, et al. Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic[J/OL]. ICLR, 2016, abs/1611.02247arXiv: 1611.02247. http://arxiv.org/abs/1611.02247.

[14] WATKINS C J C H, DAYAN P. Technical Note Q-Learning[J/OL]. Machine Learning, 1992, 8: 279-292. https://doi.org/10.1007/BF00992698. DOI: 10.1007/BF00992698.

[15] BELLMAN R E. Dynamic Programming[M]. Princeton, NJ, USA: Princeton University Press, 1957.

[16] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet Classification with Deep Convolutional Neural Networks[C/OL]// Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States. [S.l.]: [s.n.], 2012: 1106-1114. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[17] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing Atari with Deep Reinforcement Learning[J/OL]. CoRR, 2013, abs/1312.5602arXiv: 1312.5602. http://arxiv.org/abs/1312.5602.

[18] NG A Y, RUSSELL S J. Algorithms for Inverse Reinforcement Learning[C]// LANGLEY P. Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000. Ed. by LANGLEY P. [S.l.]: Morgan Kaufmann, 2000: 663-670.

[19] WHITE D. Multi-objective infinite-horizon discounted Markov decision processes[J]. Journal of mathematical analysis and applications, 1982, 89(2): 639-647.

[20] ROIJERS D M, VAMPLEW P, WHITESON S, et al. A Survey of Multi-Objective Sequential Decision-Making[J/OL]. J. Artif. Intell. Res., 2013, 48: 67-113. https://doi.org/10.1613/jair.3987. DOI: 10.1613/jair.3987.

[21] ABBEEL P, NG A Y. Apprenticeship learning via inverse reinforcement learning[C/OL]// BRODLEY C E. Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004. Ed. by BRODLEY C E. Vol. 69. ACM International Conference Proceeding Series. [S.l.]: ACM, 2004. http://doi.acm.org/10.1145/1015330.1015430. DOI: 10.1145/1015330.1015430.

[22] HO J, ERMON S. Generative Adversarial Imitation Learning[C/OL]// LEE D D, SUGIYAMA M, von LUXBURG U, et al. Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. Ed. by LEE D D, SUGIYAMA M, von LUXBURG U, et al. [S.l.]: [s.n.], 2016: 4565-4573. http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.

[23] VAMPLEW P, DAZELEY R, BERRY A, et al. Empirical evaluation methods for multiobjective reinforcement learning algorithms[J/OL]. Machine Learning, 2011, 84(1-2): 51-80. https://doi.org/10.1007/s10994-010-5232-5. DOI: 10.1007/s10994-010-5232-5.

[24] LIU C, XU X, HU D. Multiobjective Reinforcement Learning: A Comprehensive Overview[J/OL]. IEEE Trans. Systems, Man, and Cybernetics: Systems, 2015, 45(3): 385-398. https://doi.org/10.1109/TSMC.2014.2358639. DOI: 10.1109/TSMC.2014.2358639.

[25] MANNOR S, SHIMKIN N. The Steering Approach for Multi-Criteria Reinforcement Learning[C/OL]// DIETTERICH T G, BECKER S, GHAHRAMANI Z. Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]. Ed. by DIETTERICH T G, BECKER S, GHAHRAMANI Z. [S.l.]: MIT Press, 2001: 1563-1570. http://papers.nips.cc/paper/1986-the-steering-approach-for-multi-criteria-reinforcement-learning.

[26] TESAURO G, DAS R, CHAN H, et al. Managing Power Consumption and Performance of Computing Systems Using Reinforcement Learning[C/OL]// PLATT J C, KOLLER D, SINGER Y, et al. Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007. Ed. by PLATT J C, KOLLER D, SINGER Y, et al. [S.l.]: Curran Associates, Inc., 2007: 1497-1504. http://papers.nips.cc/paper/3251-managing-power-consumption-and-performance-of-computing-systems-using-reinforcement-learning.

[27] NATARAJAN S, TADEPALLI P. Dynamic preferences in multi-criteria reinforcement learning[C/OL]// RAEDT L D, WROBEL S. Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005. Ed. by RAEDT L D, WROBEL S. Vol. 119. ACM International Conference Proceeding Series. [S.l.]: ACM, 2005: 601-608. http://doi.acm.org/10.1145/1102351.1102427. DOI: 10.1145/1102351.1102427.

[28] MOFFAERT K V, DRUGAN M M, NOWÉ A. Scalarized multi-objective reinforcement learning: Novel design techniques[C/OL]// Proceedings of the 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL 2013, IEEE Symposium Series on Computational Intelligence (SSCI), 16-19 April 2013, Singapore. [S.l.]: IEEE, 2013: 191-199. https://doi.org/10.1109/ADPRL.2013.6615007. DOI: 10.1109/ADPRL.2013.6615007.

[29] MOSSALAM H, ASSAEL Y M, ROIJERS D M, et al. Multi-Objective Deep Reinforcement Learning[J/OL]. CoRR, 2016, abs/1610.02707arXiv: 1610.02707. http://arxiv.org/abs/1610.02707.

[30] BARRETT L, NARAYANAN S. Learning all optimal policies with multiple criteria[C/OL]// COHEN W W, MCCALLUM A, ROWEIS S T. Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008. Ed. by COHEN W W, MCCALLUM A, ROWEIS S T. Vol. 307. ACM International Conference Proceeding Series. [S.l.]: ACM, 2008: 41-47. http://doi.acm.org/10.1145/1390156.1390162. DOI: 10.1145/1390156.1390162.

[31] HIRAOKA K, YOSHIDA M, MISHIMA T. Parallel Reinforcement Learning for Weighted Multi-criteria Model with Adaptive Margin[C/OL]// ISHIKAWA M, DOYA K, MIYAMOTO H, et al. Neural Information Processing, 14th International Conference, ICONIP 2007, Kitakyushu, Japan, November 13-16, 2007, Revised Selected Papers, Part I. Ed. by ISHIKAWA M, DOYA K, MIYAMOTO H, et al. Vol. 4984. Lecture Notes in Computer Science. [S.l.]: Springer, 2007: 487-496. https://doi.org/10.1007/978-3-540-69158-7_51. DOI: 10.1007/978-3-540-69158-7_51.

[32] IIMA H, KUROE Y. Multi-objective reinforcement learning for acquiring all Pareto optimal policies simultaneously - Method of determining scalarization weights[C/OL]// 2014 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2014, San Diego, CA, USA, October 5-8, 2014. [S.l.]: IEEE, 2014: 876-881. https://doi.org/10.1109/SMC.2014.6974022. DOI: 10.1109/SMC.2014.6974022.

[33] PIROTTA M, PARISI S, RESTELLI M. Multi-Objective Reinforcement Learning with Continuous Pareto Frontier Approximation[C/OL]// BONET B, KOENIG S. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. Ed. by BONET B, KOENIG S. [S.l.]: AAAI Press, 2015: 2928-2934. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9798.

[34] PARISI S, PIROTTA M, PETERS J. Manifold-based multi-objective policy search with sample reuse[J/OL]. Neurocomputing, 2017, 263: 3-14. https://doi.org/10.1016/j.neucom.2016.11.094. DOI: 10.1016/j.neucom.2016.11.094.

[35] CASTELLETTI A, PIANOSI F, RESTELLI M. Multi-objective fitted Q-iteration: Pareto frontier approximation in one single run[C/OL]// Proceedings of the IEEE International Conference on Networking, Sensing and Control, ICNSC 2011, Delft, The Netherlands, 11-13 April 2011. [S.l.]: IEEE, 2011: 260-265. https://doi.org/10.1109/ICNSC.2011.5874921. DOI: 10.1109/ICNSC.2011.5874921.

[36] CASTELLETTI A, PIANOSI F, RESTELLI M. Tree-based Fitted Q-iteration for Multi-Objective Markov Decision problems[C/OL]// The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June 10-15, 2012. [S.l.]: IEEE, 2012: 1-8. https://doi.org/10.1109/IJCNN.2012.6252759. DOI: 10.1109/IJCNN.2012.6252759.

[37] YANG R. How Does Value-Based Reinforcement Learning Find the Optimal Policy? A General Explanation from Topological Point of View[EB/OL]. (2017) [2017-10-04]. https://runzhe-yang.science/2017-10-04-contraction/.

[38] BERTSEKAS D P. Regular Policies in Abstract Dynamic Programming[J/OL]. SIAM Journal on Optimization, 2017, 27(3): 1694-1727. https://doi.org/10.1137/16M1090946. DOI: 10.1137/16M1090946.

[39] BERTSEKAS D P. Abstract dynamic programming[M]. [S.l.]: Athena Scientific Belmont, MA, 2018.

[40] BELLEMARE M G, DABNEY W, MUNOS R. A Distributional Perspective on Reinforcement Learning[C/OL]// Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. [S.l.]: [s.n.], 2017: 449-458. http://proceedings.mlr.press/v70/bellemare17a.html.

[41] ANDRYCHOWICZ M, CROW D, RAY A, et al. Hindsight Experience Replay[C/OL]// Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. [S.l.]: [s.n.], 2017: 5055-5065. http://papers.nips.cc/paper/7090-hindsight-experience-replay.

[42] LILLICRAP T P, HUNT J J, PRITZEL A, et al. Continuous control with deep reinforcement learning[J/OL]. CoRR, 2015, abs/1509.02971arXiv: 1509.02971. http://arxiv.org/abs/1509.02971.

[43] GU S, LILLICRAP T P, SUTSKEVER I, et al. Continuous Deep Q-Learning with Model-based Acceleration[C/OL]// Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. [S.l.]: [s.n.], 2016: 2829-2838. http://jmlr.org/proceedings/papers/v48/gu16.html.

[44] METZ L, IBARZ J, JAITLY N, et al. Discrete Sequential Prediction of Continuous Actions for Deep RL[J/OL]. CoRR, 2017, abs/1705.05035arXiv: 1705.05035. http://arxiv.org/abs/1705.05035.

[45] Van HASSELT H, GUEZ A, SILVER D. Deep Reinforcement Learning with Double Q-Learning[C/OL]// SCHUURMANS D, WELLMAN M P. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. Ed. by SCHUURMANS D, WELLMAN M P. [S.l.]: AAAI Press, 2016: 2094-2100. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

[46] SCHAUL T, QUAN J, ANTONOGLOU I, et al. Prioritized Experience Replay[J/OL]. CoRR (Published at ICLR 2016), 2015, abs/1511.05952arXiv: 1511.05952. http://arxiv.org/abs/1511.05952.

[47] WANG Z, SCHAUL T, HESSEL M, et al. Dueling Network Architectures for Deep Reinforcement Learning[C/OL]// Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. [S.l.]: [s.n.], 2016: 1995-2003. http://jmlr.org/proceedings/papers/v48/wangf16.html.

[48] WATSON L T, HAFTKA R T. Modern homotopy methods in optimization[J]. Computer Methods in Applied Mechanics and Engineering, 1989, 74(3): 289-305.

[49] HAARNOJA T, TANG H, ABBEEL P, et al. Reinforcement Learning with Deep Energy-Based Policies[C/OL]// Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. [S.l.]: [s.n.], 2017: 1352-1361. http://proceedings.mlr.press/v70/haarnoja17a.html.

[50] Van der MAATEN L, HINTON G. Visualizing High-Dimensional Data Using t-SNE[J/OL]. Journal of Mahcine Learning Research, Nov 2008. http://www.cs.toronto.edu/~hinton/absps/tsne.pdf.

[51] SENEFF S, HURLEY E, LAU R, et al. GALAXY-II: a reference architecture for conversational system development[C/OL]// The 5th International Conference on Spoken Language Processing, Incorporating The 7th Australian International Speech Science and Technology Conference, Sydney Convention Centre, Sydney, Australia, 30th November - 4th December 1998. [S.l.]: [s.n.], 1998. http://www.isca-speech.org/archive/icslp_1998/i98_1153.html.

[52] WALKER M A. An Application of Reinforcement Learning to Dialogue Strategy Selection in a Spoken Dialogue System for Email[J/OL]. J. Artif. Intell. Res., 2000, 12: 387-416. https://doi.org/10.1613/jair.713. DOI: 10.1613/jair.713.

[53] SINGH S P, LITMAN D J, KEARNS M J, et al. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System[J/OL]. J. Artif. Intell. Res., 2002, 16: 105-133. https://doi.org/10.1613/jair.859. DOI: 10.1613/jair.859.

[54] YOUNG S J. Statistical Spoken Dialogue Systems and the Challenges for Machine Learning[C/OL]// Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017. [S.l.]: [s.n.], 2017: 577. http://dl.acm.org/citation.cfm?id=3022746.

[55] MURALIDHAR S, MAST M S, GATICA-PEREZ D. How may I help you? behavior and impressions in hospitality service encounters[C/OL]// Proceedings of the 19th ACM International Conference on Multimodal Interaction, ICMI 2017, Glasgow, United Kingdom, November 13 - 17, 2017. [S.l.]: [s.n.], 2017: 312-320. http://doi.acm.org/10.1145/3136755.3136771. DOI: 10.1145/3136755.3136771.

[56] DURSTON P J, FARRELL M, ATTWATER D, et al. OASIS natural language call steering trial[C/OL]// EUROSPEECH 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event, Aalborg, Denmark, September 3-7, 2001. [S.l.]: [s.n.], 2001: 1323-1326. http://www.isca-speech.org/archive/eurospeech_2001/e01_1323.html.

[57] YAN B, WENG F, FENG Z, et al. A Conversational In-Car Dialog System[C/OL]// Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, April 22-27, 2007, Rochester, New York, USA. [S.l.]: [s.n.], 2007: 23-24. http://www.aclweb.org/anthology/N07-4012.

[58] WILLIAMS J D, YOUNG S J. Partially observable Markov decision processes for spoken dialog systems[J/OL]. Computer Speech & Language, 2007, 21(2): 393-422. https://doi.org/10.1016/j.csl.2006.06.008. DOI: 10.1016/j.csl.2006.06.008.

[59] WANG Z, LEMON O. A Simple and Generic Belief Tracking Mechanism for the Dialog State Tracking Challenge: On the believability of observed information[C/OL]// Proceedings of the SIGDIAL 2013 Conference, The 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 22-24 August 2013, SUPELEC, Metz, France. [S.l.]: [s.n.], 2013: 423-432. http://aclweb.org/anthology/W/W13/W13-4067.pdf.

[60] KAELBLING L P, LITTMAN M L, CASSANDRA A R. Planning and Acting in Partially Observable Stochastic Domains[J/OL]. Artif. Intell., 1998, 101(1-2): 99-134. https://doi.org/10.1016/S0004-3702(98)00023-X. DOI: 10.1016/S0004-3702(98)00023-X.

[61] GAI M, JURIEK F, KEIZER S, et al. Gaussian Processes for Fast Policy Optimisation of POMDP-based Dialogue Managers[J]. 2010: 201-204.

[62] FATEMI M, ASRI L E, SCHULZ H, et al. Policy Networks with Two-Stage Training for Dialogue Systems[J]. ArXiv.org, 2016arXiv: 1606.03152.

[63] TORREY L, TAYLOR M. Teaching on a Budget: Agents Advising Agents in Reinforcement Learning[C]// Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. International Foundation for Autonomous Agents, Multiagent Systems. [S.l.]: [s.n.], 2013: 1053-1060.

[64] WILLIAMS J D, ZWEIG G. End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning.[J]. CoRR, 2016.

[65] AMIR O, KAMAR E, KOLOBOV A, et al. Interactive Teaching Strategies for Agent Training[C]// IJCAI. International Joint Conferences on Artificial Intelligence. [S.l.]: [s.n.], 2016.

[66] CLOUSE J A. On Integrating Apprentice Learning and Reinforcement Learning[C]//. [S.l.]: University of Massachusetts, 1996.

[67] CHEN L, YANG R, CHANG C, et al. On-line Dialogue Policy Learning with Companion Teaching[C/OL]// Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers. [S.l.]: [s.n.], 2017: 198-204. https://aclanthology.info/papers/E17-2032/e17-2032.

[68] CHANG C, YANG R, CHEN L, et al. Affordable On-line Dialogue Policy Learning[C/OL]// Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017. [S.l.]: [s.n.], 2017: 2200-2209. https://aclanthology.info/papers/D17-1234/d17-1234.

[69] THOMAZ A L, BREAZEAL C. Teachable Robots: Understanding Human Teaching Behavior to Build More Effective Robot Learners[J]. Artificial Intelligence, 2008, 172(6): 716-737.

[70] JUDAH K, ROY S, FERN A, et al. Reinforcement learning via practice and critique advice[C]// Twenty-Fourth AAAI Conference on Artificial Intelligence. [S.l.]: [s.n.], 2010: 481-486.

[71] CHEN L, ZHOU X, CHANG C, et al. Agent-Aware Dropout DQN for Safe and Efficient On-line Dialogue Policy Learning[C/OL]// Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017. [S.l.]: [s.n.], 2017: 2454-2464. https://aclanthology.info/papers/D17-1260/d17-1260.

[72] ULTES S, BUDZIANOWSKI P, CASANUEVA I, et al. Reward-Balancing for Statistical Spoken Dialogue Systems using Multi-objective Reinforcement Learning[C/OL]// Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, August 15-17, 2017. [S.l.]: [s.n.], 2017: 65-70. https://aclanthology.info/papers/W17-5509/w17-5509.

[73] ULTES S, ROJAS-BARAHONA L M, SU P, et al. PyDial: A Multi-domain Statistical Dialogue System Toolkit[C/OL]// Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, System Demonstrations. [S.l.]: [s.n.], 2017: 73-78. https://doi.org/10.18653/v1/P17-4013. DOI: 10.18653/v1/P17-4013.

[74] SCHATZMANN J, YOUNG S J. The Hidden Agenda User Simulation Model[J/OL]. IEEE Trans. Audio, Speech & Language Processing, 2009, 17(4): 733-747. https://doi.org/10.1109/TASL.2008.2012071. DOI: 10.1109/TASL.2008.2012071.

Acknowledgements

At the end of this four-year journey as a member of ACM honors class in Zhiyuan College at Shanghai Jiao Tong University, the author wishes to express his gratitude to many people.

The author would like to thank his supervisor Professor Kai Yu at Shanghai Jiao Tong University for his kind advice and excellent guidance throughout this thesis. Prof. Yu is the author's first research mentor and taught him not only how to conduct research but also how to be a righteous man. The research experience in the SpeechLab is unforgettable. He is always an exemplary scholar to the author.

The author would also like to thank Professor Carla P. Gomes at Cornell University for her helpful instructions and sharing of research philosophy. The author will never forget the nights they revised papers shoulder to shoulder without food and sleep. Prof. Gomes' "fertilizer" story — to always ask profound questions from real problems, and generalize the ideas for solving more — is engraved on the author's mind.

The author is also grateful to Professor Yexiang Xue at Purdue University for the insightful discussion about the scenarios and algorithms of multi-objective reinforcement learning. Prof. Xue is a perfect mentor who cares about his student out of selfless devotion. He is like a neighbor's brother to the author and encourages the author to pursue the Ph.D. degree in computer science.

The author also wants to thank his co-authors Lu Chen, Cheng Chang, Junwen Bai, Brendan Rappazzo, Guillaume Perez, Xiang Zhou, Zihao Ye. It has been great working with these brilliant people, and the author hopes they can continue their collaboration in future. And special thanks to Haoran Cai, who assists the author to run many synthetic experiments in this thesis.

Last but not least, the author would like to thank all his friends and family who have supported him throughout this wonderful journey.