Reinforcement Learning For Robust Decision-Making Under Deep Uncertainty

THE UNIVERSITY OF MELBOURNE

Author: Zhihao Pei
Student number: 1141856

Research Thesis
November 2021

Supervisors:
Dr Fjalar de Haan
Dr Nir Lipovetzky
Dr Enayat A. Moallemi
Angela M. Rojas-Arevalo

Master of Computer Science
School of Computing and Information Systems
The University of Melbourne

Abstract

Decision-making process subject to multiple aspects of uncertainty is known as Decision-Making under Deep Uncertainty (DMDU). To support this process, exploration methods are used to evaluate the performance of candidate policies in possible scenarios, in order to identify a robust plan that performs satisfactorily in any future. These methods show complementarity, hence joint use of them can improve the overall DMDU performance. The main aim of this project is to introduce Reinforcement Learning as a new exploration method into DMDU, and further complement the existing methods with it. This is motivated by the suitability of Reinforcement Learning for handling uncertainty due to its real-time policy adaptability. We first reviewed existing studies and identified three Reinforcement Learning algorithms applicable in DMDU, as well as two baseline Evolutionary Algorithms that have already been applied in DMDU. Then, we constructed a common environment to compare these algorithms in an uncertainty problem and two deep uncertainty problems. Our experiments empirically showed the viability of this introduction. Meanwhile, they reflected the complementarity between Reinforcement Learning and Evolutionary Algorithm. The former generally provided higher efficiency and robustness to parameter uncertainty, and could handle random initial states. The latter performed better in dealing with objective uncertainty, and was less sensitive to the randomness of the exploration process. Moreover, we also demonstrated the application of these methods in a real-world problem in the last experiment, and investigated how the high complexity of the problem affected their performance. Overall, this project also revealed the differences in characteristics of domain problems and algorithms in the Reinforcement Learning area and DMDU field, which contributed to inspiring possible directions for future robust planning research in them.

Declaration

I certify that:

- this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text;
- the thesis is 29368 words in length (excluding text in images, tables, bibliographies and appendices).
- 01/11/2021

Acknowledgements

I would like to express my sincere thanks to my supervisors, Doctor Fjalar de Haan, Doctor Nir Lipovetzky, Doctor Enayat A. Moallemi, and Angela M. Rojas-Arevalo. All of them went beyond expectations throughout this entire research project. Their ongoing guidance, support, encouragement, and sharing of their knowledge were paramount to my work in this project and the expansion of my knowledge.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

In the context of decision-making, uncertainty implies the gap between available information and that required by decision-makers to make the optimal policy decisions [1]. Many real-world planning applications, such as Energy Transitions [2], Societal Aging [3], Water Management [4, 5], and Climate Change [6], involve such gaps in aspects of model, parameter, and objective, which can hardly be eliminated until the future unfolds. These conditions are characterized as *deep uncertainty*, where decision-makers need to develop robust plans that not only maximize the goals, but also perform satisfactorily in any possible future [7].

Nowadays, many Decision-Making under Deep Uncertainty (DMDU) approaches have been proposed, which specify appropriate planning processes to develop robust plans in deep uncertainty problems [6]. These approaches are centered on exploration to identify a set of static policies, which are fixed sequences of actions, to deal with various possible futures. Then, decision-makers can adjust their plans accordingly during deployment. Existing DMDU exploration methods include Multi-Objective Robust Evolutionary Algorithm (MOREA) [4, 8], Info-Gap Analysis [9], and Scenario Discovery [10]. Each method has its own strengths and weaknesses. MOREA works by systematically exploring the performance of candidate policies in possible futures. Theoretically, this thoroughness guarantees identifying highly robust policies, but also brings a high computational cost [11]. Info-Gap Analysis reduces this cost by only considering a set of progressively more opportune and dire possible futures [12]. In contrast, Scenario Discovery derives a signpost (signal that reality deviates from the assumption) to guide the policy design by analyzing the vulnerabilities of current policies. This method is also less computationally intensive, but cannot provide the same robustness as MOREA either [11]. Previous studies have argued the complementarity between these methods [11, 12]. First, the results from MOREA and Scenario Discovery provide complementary information on the performance of candidate policies in

less extreme futures for Info-Gap Analysis. Second, since Info-Gap Analysis employs the most lightweight exploration, it can be applied to streamline the candidate policy space in MOREA and Scenario Discovery to promote their exploration efficiency. The robustness curve produced in Info-Gap Analysis also complements the others with useful visualizations. Lastly, the signpost derived by Scenario Discovery provides insights into candidate policies, which can in turn guide the exploration of MOREA and Info-Gap Analysis. For these reasons, joint use of these methods is recommended in DMDU approaches to improve their efficiency and the robustness of resulting plans in different aspects. Given this finding, we believe that introducing a new exploration method can be beneficial by further complementing the existing ones.

Reinforcement Learning (RL) is another typical exploration method for decision-making, which produces policies through agents learning from interactions with environments [13]. Compared with the existing DMDU exploration methods, the main difference of RL is that its output policy is a function that maps actual observations of the environment to actions in real-time, rather than a fixed sequence of actions. To distinguish between these two types of policies, we will refer to them as dynamic and static policies respectively in the rest of this thesis. Researchers believe that this inherited real-time control allows the RL policy to automatically adapt to the actual situation during deployment, thereby providing higher efficiency and robustness in dealing with uncertainty [14]. Therefore, we think that RL may be able to complement the existing DMDU exploration methods.

However, to the best of our knowledge, no existing research has studied the application of RL in DMDU. This might be because traditional RL algorithms are built on deterministic models, and mainly designed for optimization rather than robustness problems [13, 15, 16]. With more applications of RL in complex, high-stakes problems, their robustness has been emphasized and drawn more attention from the research community. Recently, great progress has been made in the areas of Multi-Objective RL (MORL) [17] and Robust RL [16]. With these algorithms shown well-performed in decision-making under different uncertainties, we decide to introduce their integration, Multi-Objective Robust Reinforcement Learning (MORRL), as a new exploration method into DMDU, and evaluate its complementarity with the existing methods.

1.2 Aims

There are two general research questions to be answered in this project:

- Can MORRL be applied as an alternative exploration method in DMDU approaches?
- What are the strengths, weaknesses, and complementarity of MORRL in terms of efficiency and robustness to deep uncertainty, compared with the existing DMDU exploration methods?

To answer these questions, we implemented two MOREAs as baselines, representing the existing DMDU exploration methods, and three MORRL algorithms as candidates. Then, we compared their performance in an uncertainty problem (one uncertainty aspect) and two deep uncertainty problems (multiple uncertainty aspects) with the following aims guiding our investigations:

- Design a general model interface for model implementations of MOREA and MORRL, and a general DMDU process for their applications in deep uncertainty problems. These tools also serve as the common environment for comparison. This step not only enhances the fairness of comparison, but also demonstrates how to apply MORRL in DMDU.
- Empirically show the viability of applying MORRL as an alternative exploration method in DMDU approaches. To achieve this, the MORRL algorithms must be able to support the exploration processes in our experimental deep uncertainty problems, while satisfying the two requirements for DMDU: (1) accept, understand and manage uncertainties; (2) adjust plans adaptively as the future unfolds.
- Compare the efficiency and robustness to deep uncertainty of the MORRL algorithms and MOREAs to investigate their complementarity.
- Investigate the differences in characteristics of domain problems and algorithms in the MORRL area and DMDU field, in order to inspire possible directions for future robust planning research in them.

Additionally, we also set three secondary aims:

- Demonstrate how to apply the general model interface and DMDU process we design to a real-world deep uncertainty problem.
- Investigate how the high complexity of this real-world problem affects the performance of the MORRL algorithms and MOREAs in DMDU.
- Compare the performance of different MORRL algorithms in uncertainty and deep uncertainty problems. Since MORL and Robust RL are novel research areas, no previous study has compared these algorithms before.

1.3 Case Studies

To answer our research questions, we used the following three problems and their variants as case studies to compare our algorithms: robust Cartpole problem, Lake problem, and Electricity Market

problem. All these problems have their publicly available open-source implementations, which we refer to as their source models. This greatly facilitated our experiments.

The robust Cartpole problem is a benchmark problem in the Robust RL area, which describes an inverted pendulum system where decision-makers need to decide forces applied to it to prevent the pendulum from falling over as long as possible [18]. This problem involves parameter uncertainty only, but puts a high requirement on the real-time policy adaptability. We used it to compare the algorithm performance in dealing with parameter uncertainty and study the difference between dynamic and static policies. Meanwhile, a visualization tool for this problem is available, which helped us explain our findings.

The Lake problem is a domain problem in the DMDU field, where decision-makers need to decide annual pollution emissions into a lake to maximize economic benefits while avoiding lake eutrophication [19, 20]. This problem involves both parameter and objective uncertainty, which we used to investigate the algorithm performance in handling deep uncertainty. Additionally, we also compared this problem with the robust Cartpole problem to study the difference between standard domain problems in the Robust RL area and DMDU field, in order to identify the gap between their robust planning research.

The Electricity Market problem is a real-world deep uncertainty problem, where decision-makers need to decide policies to simultaneously minimize electricity prices and greenhouse gas emissions in the Australian National Electricity Market. This problem also involves both parameter and objective uncertainty. We used it to demonstrate the application of our algorithms in such a real-world problem with high complexity and deep uncertainty, and investigate their performance in this case.

The rest of this thesis is organized as follows: Chapter 2 reviews current advances on decision-making under uncertainty in the DMDU field and RL area. Based on this review, we determined the MOREAs and MORRL algorithms to be compared in this project. Chapter 3 describes the overall plan of our experiments, which reflects the general DMDU process we designed for MOREA and MORRL. This chapter also provides sufficient implementation details to support the reproduction of our experiments. Chapters 4, 5, and 6 respectively present our experimental results and analyses in the robust Cartpole problem, Lake problem and Electricity Market problem. Finally, Chapter 7 summarizes our findings, discusses the strengths and weaknesses of this project, and proposes our suggested future directions for this research area.

Chapter 2

Literature Review

Extensive research has been conducted in the DMDU field and RL area to support decision-making under uncertainty. In this chapter, we first give background information about DMDU and introduce the MOREAs used in the existing DMDU approaches. Then, we introduce RL, review various ideas and corresponding implementations of MORL and Robust RL, and discuss their applicability in the context of DMDU. Based on this review, we eventually selected two MOREAs as baselines and three MORRL algorithms as candidates for comparison in our experiments to answer our research questions. Figure 2.1 outlines the important concepts of exploration methods covered in this chapter.

2.1 Decision-Making Under Deep Uncertainty

Decision-making problems under deep uncertainty refer to conditions where stakeholders cannot determine

> '(1) the appropriate conceptual models that describe the relationships among the key driving forces that will shape the long-term future, (2) the probability distributions used to represent uncertainty about key variables and parameters in the mathematical representations of these conceptual models, and/or (3) how to value the desirability of alternative outcomes [21].'

In this thesis, we refer to these uncertainties as model uncertainty, parameter uncertainty, and objective uncertainty respectively. The traditional approach to solve these problems is to predict the most probable future through gathering more information, such as statistical analysis of historical

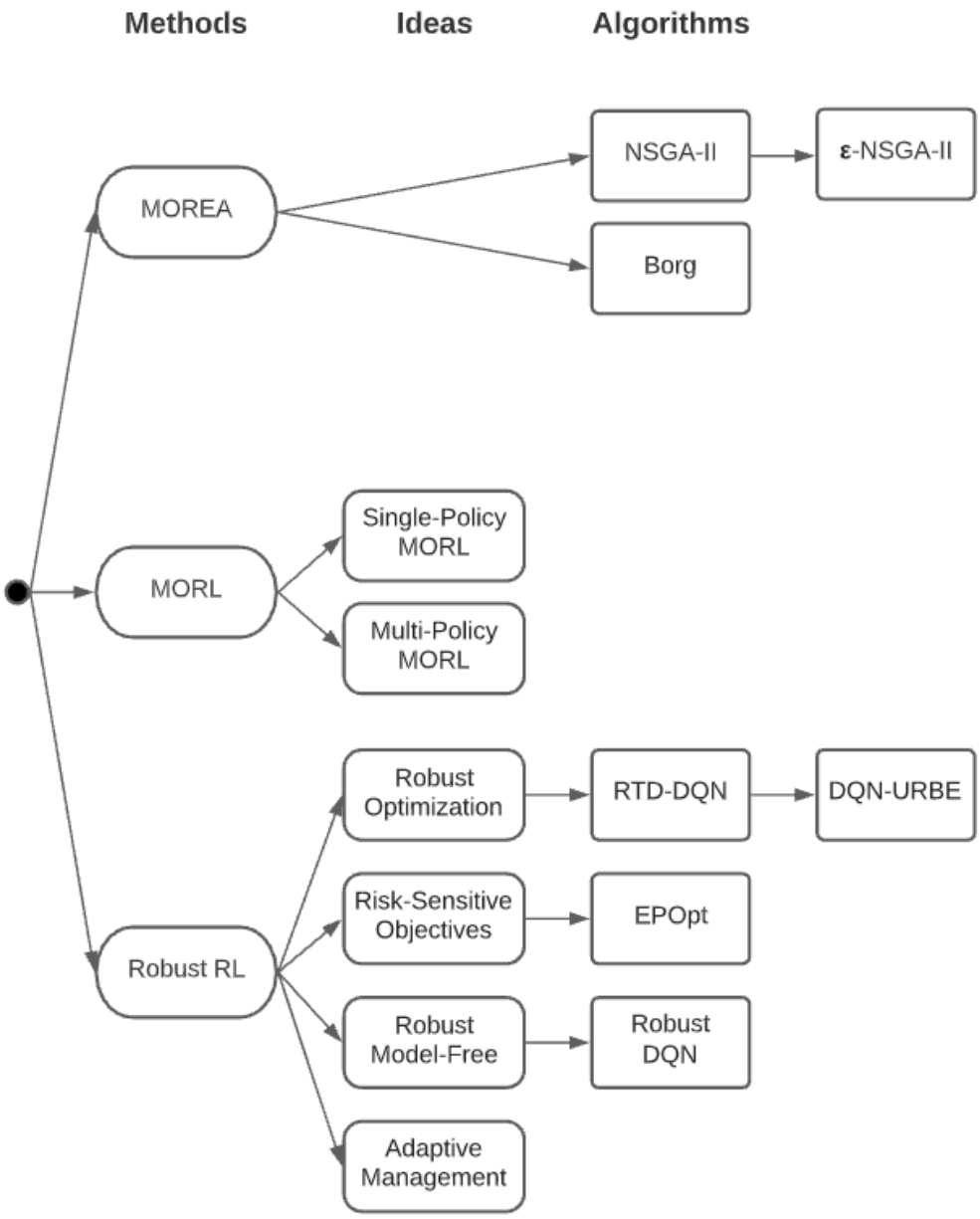


Figure 2.1: Important concepts of exploration methods covered in Literature Review.

data, and develop a static optimal plan accordingly. This approach is usually termed 'predict-then-act' [22–24]. By reducing non-deterministic problems to deterministic ones, this approach allows applying regular optimization planning processes to solve deep uncertainty problems. However, this approach has been criticized for three limitations [7]. First, the assumption that the future closely relates to the past is not always correct. Many uncertain factors cannot be predicted until the future unfolds [6]. Second, its static plan cannot adapt to new situations. If the actual future differs from the predicted one, this plan may still fail [25, 26]. This risk can be unacceptable in many high-stakes problems [27]. Lastly, this static plan may also miss unseen opportunities, and thus

fail to maximize its actual performance. For these reasons, two requirements for making robust plans under deep uncertainty have been drawn that decision-makers should (1) accept, understand and manage uncertainties [7], and (2) adjust their plans adaptively as the future unfolds [6, 28]. These requirements are also considered as the major challenges in DMDU due to its large, complex uncertainty space.

2.1.1 Decision-Making Under Deep Uncertainty Approaches

Due to the nature of deep uncertainty problems, there is often a myriad of assumptions and models consistent with known facts when decision-making [7]. Researchers argue that exploratory model-based approaches are well suited to support this process [29]. These approaches provide robustness by exploring the performance of candidate policies across the uncertainty space and adjusting plans accordingly during deployment. Existing DMDU approaches generally adopt this idea, and are empirically shown valuable for planning in systems with deep uncertainty, high complexity and large policy space [6], such as Robust Decision Making [10, 30], Dynamic Adaptive Planning [31, 32], Dynamic Adaptive Policy Pathways [28], and Info-Gap Decision Theory [9, 33]. Figure 2.2 shows their general framework. In this project, we chose to mainly focus on the exploration process in this framework because it plays an essential role in meeting the two requirements for DMDU and thereby providing robustness. We also excluded the cycling process due to the time limit. As mentioned previously, many complementary DMDU exploration methods exist, among which MOREA is the most widely used. It is also mainly computer-based, hence requiring less involvement of expert knowledge [11]. For these reasons, we selected MOREA as the baseline, representing the existing DMDU exploration methods to be contrasted with MORRL in our experiments.

2.2 Multi-Objective Robust Evolutionary Algorithms

2.2.1 Evolutionary Algorithms

Evolutionary Algorithm (EA) is a general optimization method inspired by Darwinian's natural evolution [34]. EA generally works as follows. First, an objective function is defined to evaluate the quality of each solution, called 'fitness function'. Then, EA randomly samples a set of candidate solutions from the solution space, which forms the first-generation population P_0. After that, for each generation population P_t, some solutions with the highest fitness are *selected*, and then operated to produce the next-generation population P_{t+1} using some recombination operators, such as *mutation*, *recombination*, and *inheritance*. This process continues until the fitness of the last

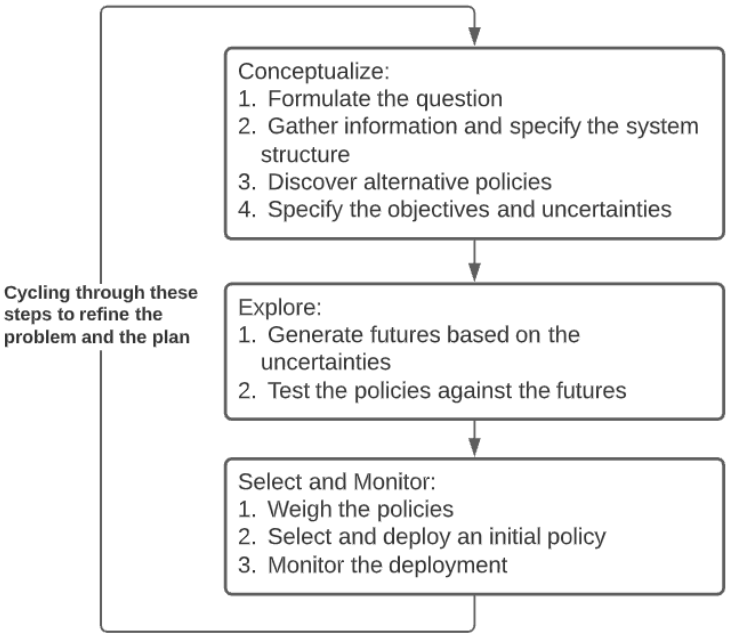


Figure 2.2: Framework of DMDU approaches, adapted from [6]. These approaches generally consist of three iterative steps. The first step is the conceptualization of the decision-making problem. At this stage, decision-makers cooperate with other stakeholders to define the context, system structure, alternative policies, objectives and uncertainties of the problem based on their expert knowledge. The second step is exploration. The performance of the policies is assessed across various possible futures arising from the uncertainties to evaluate their robustness. This process is often supported by exploratory analysis using computational experiments. Finally, decision-makers select an initial optimal robust policy based on the exploration results and monitor its deployment so that they can adaptively adjust the plan as the plan progresses. Moreover, according to the nature of deep uncertainty, more information can be gathered to reduce the uncertainty as the future unfolds. Therefore, the practical application of these approaches often involves cycling through these steps to refine the problem and the plan as new opportunities or vulnerabilities appear [6, 28].

generation solutions is good enough. Recently, EA has become very popular because of its several strengths [35]. First, EA is conceptually simple and can be used with little expert knowledge. Second, most processes in EA perfectly support parallelization, which enhances its efficiency. Finally, EA provides high flexibility, hence applicable to various optimization problems, especially for those where available information is imperfect.

2.2.2 Multi-Objective Evolutionary Algorithms

The problem under objective uncertainty, also known as the multi-objective problem, involves multiple objectives, where the true weight of each objective is unknown [36]. Therefore, there is a set of solutions that outperform all others regarding at least one objective. These solutions are

known as Pareto-optimal or non-dominated solutions. Since the dominance of these solutions cannot be determined without further information, multi-objective optimization methods are expected to provide decision-makers with as many Pareto-optimal solutions as possible for reference [37]. To deal with this uncertainty, two Multi-Objective Evolutionary Algorithms (MOEAs), namely NSGA-II [37] and Borg [38], have been proposed and used in the existing DMDU approaches [8, 11]. Compared with other traditional methods, the main advantage of MOEAs is that they can identify multiple Pareto-optimal solutions in one run, thereby providing higher efficiency [37].

Non-Dominated Sorted Genetic Algorithm II

Non-Dominated Sorted Genetic Algorithm II (NSGA-II) is one of the most famous MOEAs [8]. On the basis of ordinary EA, NSGA-II mainly makes two improvements [37]. First, it introduces a fast non-dominated sorting into its selection process, where the solution in each generation population P_t is first ranked based on its fitness on each objective, and then sorted based on all its ranks for selection. In this way, NSGA-II can simultaneously maintain and evolve multiple non-dominated solutions, providing various preferences between the objectives in one run. Second, NSGA-II employs elitism, which is a strategy that allows the traits of the most fitting solutions to be copied to the next generation. Elitism has been shown able to promote the evolutionary process [39].

Deb *et al.* compared the performance of NSGA-II with two other MOEAs, namely Pareto-archived Evolution Strategy [40] and Strength-Pareto EA [41], in nine testing problems [37]. Their experimental results showed that NSGA-II provided the best convergence to the Pareto-optimal set in seven out of the nine problems. NSGA-II also outperformed the other two in terms of the number of identified Pareto-optimal solutions and the diversity of these solutions in all the problems.

Epsilon-Dominance Non-Dominated Sorted Genetic Algorithm II

To further improve the reliability, efficiency and convenience of NSGA-II, follow-up studies proposed Epsilon-Dominance Non-Dominated Sorted Genetic Algorithm II (ε-NSGA-II) [36]. This algorithm further makes three improvements on the basis of NSGA-II. First, it introduces the concept of ε-dominance [42], which allows users to define precision for each objective of the problem. Users first need to specify an ε-value for each objective. After that, an ε-grid is generated, where each ε-value determines the grid width in the corresponding objective dimension. Solutions are then ranked and sorted according to this grid during evolution. In this way, ε-NSGA-II promotes solution diversity and simplifies the way of handling problems with objectives of different scales. Second, ε-NSGA-II employs adaptive population resizing. This algorithm starts with a small population, and automatically adapts the population size to the problem complexity to enhance the efficiency. This adaptation is achieved through a 25% injection scheme. For every generation, the

identified ε-non-dominated solutions are stored in an ε-dominance archive. Then, these solutions will form 25% of the subsequent population to direct the search, while the remaining 75% of solutions will be randomly generated to encourage exploration. This scheme achieves both elitism and a balance between exploration and exploitation. Finally, if there is no obvious improvement between the last two consecutive generations, ε-NSGA-II will automatically terminate to minimize the computational cost.

Kollat *et al.* compared the performance of ε-NSGA-II and NSGA-II in long-term groundwater monitoring test cases with four or two objectives, respectively [36]. Their experimental results in the four-objective case showed that ε-NSGA-II outperformed NSGA-II in terms of accuracy, convergence, and stability to randomness. More importantly, ε-NSGA-II maintained this superiority throughout the entire exploration process, which meant it was also more efficient. Nevertheless, these two algorithms provided comparable solution diversity in this four-objective case, as well as similar performance in all aspects in the two-objective case.

Due to these advantages of NSGA-II and ε-NSGA-II, Kwakkel *et al.* successfully applied NSGA-II as the exploration method in an existing DMDU approach [8]. In addition, Kwakkel also implemented ε-NSGA-II as the default exploration method in Exploratory Modelling and Analysis (EMA) Workbench [43], which provides a toolkit to support exploration in DMDU and will be introduced in detail in Chapter 3. However, an existing study has shown that the performance of NSGA-II can be compromised in specific problem classes [44]. Therefore, Kwakkel *et al.* suggested a more modern MOEA, Borg, for providing better performance characteristics in DMDU approaches [8].

Borg

Borg is a state-of-the-art MOEA, which has been shown to be one of the best genetic algorithms for multi-objective problems [38, 44]. It has also been applied in the same DMDU approach as NSGA-II [11], and implemented as an alternative exploration method in EMA Workbench.

Borg (Algorithm 1, Figure 2.3) [38] is mainly built on ε-MOEA [45], and makes the following improvements based on ordinary EA: (1) Borg adopts ε-dominance as ε-NSGA-II, and deals with objective uncertainty by evolving the population P_t and the ε-dominance archive E_t simultaneously; (2) Borg also adopts adaptive population resizing to maintain a fixed population-to-archive proportion γ (for example, γ=4 in ε-NSGA-II), which has been observed helpful for avoiding local minima in multi-modal problems [46]; (3) Borg uses ε-progress to indicate evolutionary stagnation; (4) Borg adopts auto-adaptive recombination, which automates the selection of optimal recombination operators for different problems. To achieve this, Borg holds a group of candidate operators, and maintains a feedback loop to monitor and reward those producing more accepted

child solutions. These rewarded operators will be used to produce more children in subsequent generations. Borg is highly efficient, because it does not require expensive sorting, and its working process also supports parallelization well [38].

Algorithm 1 Borg, adapted from [38]

Require: Updating period T, Population-to-archive proportion γ

Randomly sample the first generation population P_0 from the solution space, and generate an ε-dominance archive E_0 with the ε-non-dominated solutions from P_0, where the size of E_0 is determined by γ
for $t = 0$,...N **do**
    Select a recombination operator O according to the auto-adaptive recombination procedure
    Assume O requires k parents, select $k-1$ parents from P_t and one parent from E_t
    Recombine these k parents using O to produce a child solution s_t
    if s_t dominates any solution in P_t **then**
        Replace one of these solutions in P_t with s_t
    else
        if s_t is not dominated by any solution in P_t **then**
            Replace a random solution in P_t with s_t
        end if
    end if
    Remove all solutions dominated by s_t from E_t
    if s_t is not dominated by any solution in E_t **then**
        Add s_t to E_t
        Increase the occurrence of ε-progress by 1
    end if
    if t is divisible by T **then**
        if the counter of ε-progress has not changed from the last check OR the current population-to-archive proportion deviates γ more than 25%) **then**
            Resize P_t by refilling it with the solutions in E_t and their mutations
        end if
    end if
end for

2.2.3 Multi-Objective Robust Evolutionary Algorithms

To support exploration in deep uncertainty problems, which involve a mixture of model, parameter, or objective uncertainties, previous studies have extended MOEAs to MOREAs as the exploration methods by using a computational scenario-based approach [8, 11]. This approach first requires decision-makers to generate an ensemble of possible scenarios that covers the model and parameter uncertainty space of the problem. Each scenario describes a possible assumption about all the uncertain factors. In this case, the robustness of a policy is measured as to its performance over

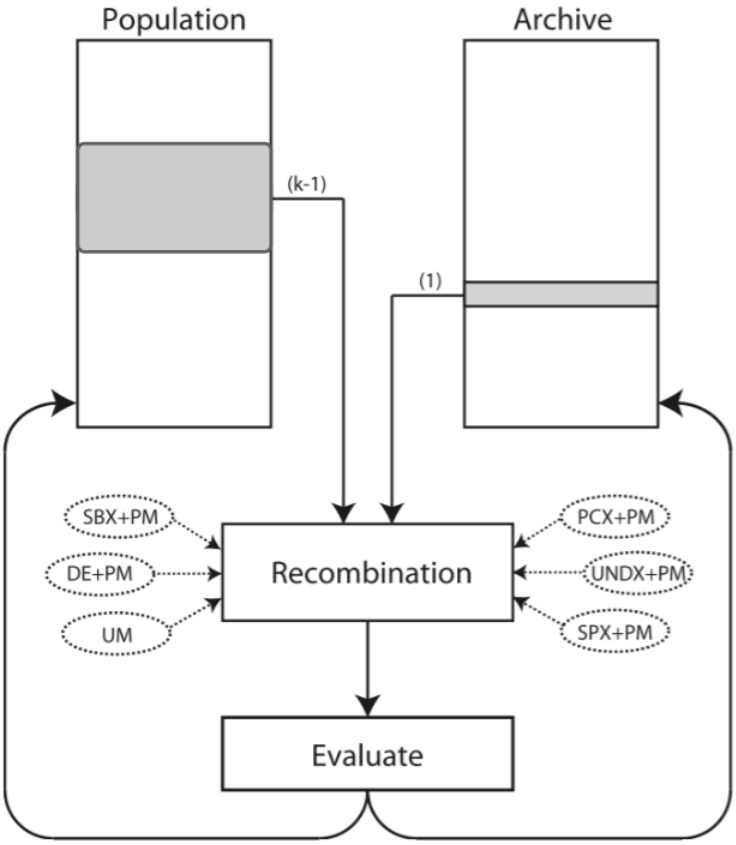


Figure 2.3: Borg flowchart, adapted from [38]. The bubbles around *Recombination* refer to different recombination operators.

the entire ensemble, and is often operationalized as some robustness metrics (for example, 10^{th} percentile performance). Then, decision-makers measure the fitness of each policy according to its robustness on each objective arising from the objective uncertainty independently, and use this result to guide the MOEAs. In this way, ordinary NSGA-II, ε-NSGA-II and Borg can be directly used as MOREAs to screen out a set of robust Pareto-optimal static policies in one run.

2.3 Reinforcement Learning

RL is another typical optimization method. Like EA inspired by biological evolution, RL is inspired by the most common learning process for humans or animals, which is learning by interacting with the environment [13]. RL is mainly composed of two components, agent and environment. Agent represents the decision-makers in decision-making problems, who have some actions to apply. Environment represents the system with which the agent interacts. Additionally, the learning process of RL is based on a standard problem formalization, named *Markov Decision Process (MDP)* [15, 47]. We propose Figure 2.4 to show the process of decision-making with offline RL.

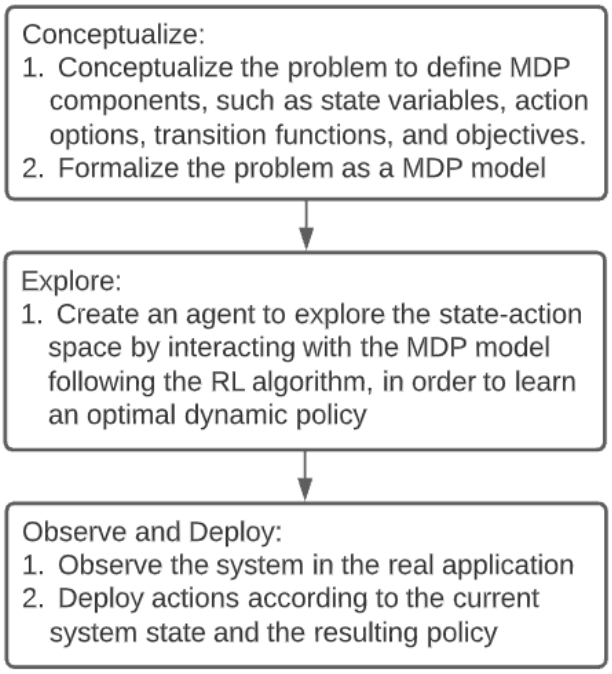


Figure 2.4: Framework of decision-making with offline RL. This framework consists of three steps: (1) Conceptualize the problem as a MDP model; (2) Learn an optimal dynamic policy through exploration following the RL algorithm; (3) Adaptively deploy actions according to the current system state and the resulting dynamic policy.

2.3.1 Markov Decision Process

MDP is a mathematical formalization of a stochastic process of sequential decision-making [13]. The MDP of a problem can be represented as a tuple $\{S, A, P(\cdot|s,a), r\}$. S is a finite set of model states, called state space. Each state s is described by a set of state variables, whose corresponding values form the *observation* of s. $A(s)$ is a finite set of actions available from s, where taking an action will transfer the agent to another state. $P(\cdot|s,a)$ is the transition probability distribution of taking action a from s, and arriving at other states. These distributions deterministically specify the stochasticity of state transitions in the problem, which is conceptually different from uncertainty. Each transition (s,a,s') of the agent taking a from s to s' is associated with a reward $r(s,a,s')$, where r is called reward function. These rewards can be positive or negative real values, and are designed to reflect the objective of the problem to guide the learning process.

2.3.2 Learning Process

The learning process of RL involves many episodes, through which the agent gradually learns a value function that estimates the expected return of applying each action a in each model state s. These estimates are called Q-values $Q(s,a)$ and this function is known as Q-function $Q(\cdot)$ [13].

The Q-function starts with random values. In each *episode*, the agent takes a sequence of actions to move from the initial model state to a terminal state. Taking an action a will transfer the agent from its current state s_t to the next s_{t+1} according to $P(\cdot|s_t,a)$, and give it a reward $r(s_t,a,s_{t+1})$. Then, the agent will treat the weighted sum of $r(s_t,a,s_{t+1})$ and $max(Q(s_{t+1},\cdot))$ as a more accurate estimate of $Q(s_t,a)$ to update the Q-function. Moreover, the sequence of collected observations, applied actions and received rewards in the episode forms the corresponding *trajectory*. During this learning process, the agent both exploits what it has experienced to try actions with high expected returns more frequently, and explores alternative actions to avoid stuck with local minima. This balance is achieved with Multi-Armed Bandit Algorithms [48–50]. This process continues until the Q-function converges to optimal or pre-defined computational budgets are used up [13]. Then, the final dynamic policy is extracted from the resulting Q-function, by mapping each model state to the action with the highest Q-value. In this sense, the essence of the learning process in RL is to learn the optimal estimate of the true expected return of each state-action pair.

As another typical optimization method in recent years, RL has its own strengths. First, model-free RL can learn from interactions with simulation models instead of true models. This feature makes RL very useful when the system is too complex to model, but a great amount of data on its historical interactions is available [51]. Second, RL can implicitly provide a certain degree of robustness to uncertainty due to the inherited real-time control of dynamic policies, which naturally adapt their actions to actual environmental states in real-time. These policies can also deal with unseen futures, if some similar states have been visited during the learning processes [13].

2.4 Multi-Objective Reinforcement Learning

As introduced in Section 2.2.2, the main feature of the problem under objective uncertainty is that a set of Pareto-optimal solutions exists, and multi-objective optimization methods are required to find as many of these solutions as possible. To achieve this, there are currently two ideas for MORL, which are *Single-Policy MORL* and *Multi-Policy MORL*.

2.4.1 Single-Policy Multi-Objective Reinforcement Learning

For a multi-objective problem, the single-policy MORL algorithm requires a specific preference between the objectives as input, and learns a single optimal policy accordingly [52]. This preference must be pre-defined based on decision-makers' expert knowledge of the domain. In this way, this algorithm actually reduces the multi-objective problem to a single-objective one by 'predicting the most probable future'. Most MORL algorithms adopt this idea. To reduce the objective

uncertainty space, a common approach is to define the preference through a linear scalarization of the objectives. Decision-makers first predict the weight of each objective, and then define the reward function for RL as the weighted sum of rewards gained on each objective [17, 53]. However, as a 'predict-then-act' approach, we argue that single-policy MORL is not suitable for supporting DMDU, because it suffers from the three limitations (Section 2.1).

2.4.2 Multi-Policy Multi-Objective Reinforcement Learning

To overcome this shortcoming, multi-policy MORL has been proposed to produce multiple Pareto-optimal solutions simultaneously [17, 54]. To achieve this, decision-makers first need to identify all possible preferences between the objectives arising from the objective uncertainty of the problem. Then, the multi-policy MORL algorithm will learn multiple policies targeted at each of these preferences in parallel. This idea can be considered as the parallelization of multiple single-policy MORL algorithms with different objective preferences. Although multi-policy MORL has been shown able to approximate the true Pareto-optimal set efficiently, it still requires prior prediction of possible objective preferences from the decision-maker [17].

2.5 Robust Reinforcement Learning

With more applications of RL in decision-making under uncertainty, the real-time adaptability of dynamic policies alone can no longer provide sufficient robustness to uncertainty. Recently, many ideas for Robust RL have been proposed [18]. In this section, we review four of these ideas and their corresponding implementations, which we think are applicable to deep uncertainty problems.

2.5.1 Robust Optimization

Dietterich [55] summarizes eight ideas for Robust Artificial Intelligence. Two ideas suitable for applying Robust RL for exploration in DMDU are *Robust Optimization* and *Risk-Sensitive Objectives*.

The main idea of Robust Optimization is to provide robustness by maximizing the worst-case policy performance [56]. This idea is conceptually simple, but Dietterich argues that it often leads to overly conservative policies, which may perform poorly in other possible cases [55]. Therefore, decision-makers should take measures to mitigate this conservativeness.

The implementation of this idea with RL is mainly built on an extension of MDP, called Robust Markov Decision Process (RMDP) [57]. RMDP is extended to describe the model and parameter uncertainty. Compared with standard MDP, the transition distribution $P(\cdot|s,a)$ for each state-action

pair (s,a) is not deterministically defined in RMDP. Instead, a set of possible distributions arising from the problem uncertainty is given $\mathcal{P}(s,a) = \{P_0(\cdot|s,a), P_1(\cdot|s,a), ...\}$. This definition implies that RMDP describes uncertainty for each state-action pair independently. This characteristic is called state-action rectangularity [18].

Many algorithms have been proposed to solve RMDP. The earliest examples are Value Iteration and Policy Iteration [56, 58–60]. The Q-functions in these algorithms record Q-values in the form of a lookup table, where the agent must try each action in each state enough times to estimate the Q-value of each state-action pair explicitly during the learning process. Therefore, these algorithms have poor scalability and do not support continuous state space, hence unsuitable for deep uncertainty problems that often have high complexity. For the same reason, other poorly scalable algorithms were also excluded from our experiments [61–64]. Robust Temporal Difference Deep Q-Network (RTD-DQN) employs a Q-function approximator to provide scalability, but it takes no measure to mitigate the conservativeness [65]. Recently, two algorithms have been proposed to overcome these shortcomings by integrating RTD-DQN with other algorithms, namely Deep-RoK [65] and DQN-URBE [18]. Since the implementation of Deep-RoK is unavailable, we chose Deep Q-Network Uncertainty Robust Bellman Equation (DQN-URBE) as our first Robust RL algorithm for comparison in this project.

Deep Q-Network

Compared with the lookup table used in regular RL, Deep Q-Network (DQN) uses a deep neural network as an approximator of the Q-function (Figure 2.5) [66]. This network contains an input layer, an output layer, and several hidden layers. Each node in the input layer corresponds to a state variable, and each node in the output layer corresponds to an action option. Moreover, every node and connection in this network is associated with a weight. For each model state s, the observation of s is used as the input signal, and processed through the network according to the node and connection weights until reaching the output layer. Then, each output-layer node n_i would produce the estimated Q-value $Q(s,a_i)$ of applying the corresponding action a_i in s as output. Unlike regular RL learning the Q-value of each state-action pair, DQN learns the weights of the network components and uses the final network as its resulting policy. In this case, the DQN policy is applicable in unseen states if the agent has learned from some similar ones. It also supports continuous state space because the network accepts real-number inputs. Hence, DQN provides high scalability. A shortcoming of standard DQN is that it does not support continuous action space, because its network has finite output nodes, each of which corresponds to one action option.

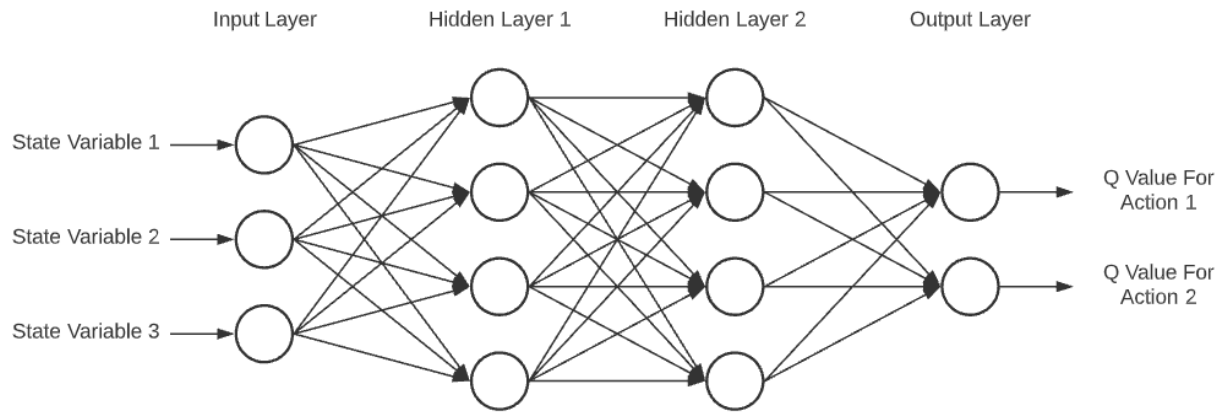


Figure 2.5: Neural network architecture in DQN.

Robust Temporal Difference Deep Q-Network

RTD-DQN maintains its neural network as an approximator of the robust Q-function, which estimates the worst-case return of each state-action pair in the uncertainty problem [65]. This network is updated by learning from the worst independently at each step based on the underlying RMDP. During the learning process, whenever the agent takes an action a_t in a state s_t, it will move to the next state s_{t+1} and receive a reward r_t as in standard MDP. Here, both s_{t+1} and r_t are derived from the unknown true MDP model. Meanwhile, the agent will also consider all possible transition distributions $\mathbb{P}(s_t, a_t) = \{P_i(s^i_{t+1}|s_t, a_t)\}$, and their corresponding next states $\{s^i_{t+1}\}$. Finally, the agent will update the network based on the weighted sum of r_t and $max(Q(s^{min}_{t+1}, \cdot))$, where $s^{min}_{t+1} = argmin_{s^i \in \{s^i_{t+1}\}}(max(Q(s^i, \cdot)))$, hence considered as the worst case.

Uncertainty Bellman Equation

Uncertainty Bellman Equation (UBE) is built on the concept of Bayesian RL, where the Q-function estimates the posterior distribution over each Q-value rather than the value itself [67]. O'Donoghue *et al.* proved that the variances of these distributions are bounded, whose boundaries can be learned in the same way as the Q-values [68]. Since true posterior distributions are often intractable to learn, they designed the UBE algorithm to learn to approximate them. For each state-action pair (s,a), the agent simultaneously learns the standard Q-value $Q(s,a)$ and the variance boundary $u(s,a)$ of the posterior distribution $D(s,a)$, and approximates $D(s,a)$ as a Gaussian distribution $\mathcal{N}(Q(s,a), diag(u(s,a)))$. These approximated distributions reflect the potential true Q-value of each state-action pair, so O'Donoghue *et al.* suggested them as a good exploration heuristic to guide the learning of RL. Their experimental results showed that DQN with UBE (DQN-UBE) outperformed DQN with ε-greedy [13], which is the most common Multi-Armed Bandit Algorithm, in 51 out of 57 test cases. They explained that the former provided more accurate estimates

of variances in the Q-function, thus guiding more reasonable exploration trajectories during the learning process.

Deep Q-Network Uncertainty Robust Bellman Equation

To mitigate the conservativeness of RTD-DQN, Derman *et al.* proposed DQN-URBE by integrating RTD-DQN with UBE [18]. Figure 2.6 shows the neural network architecture in DQN-URBE. This network contains one input head and two output heads. The upper output head learns to

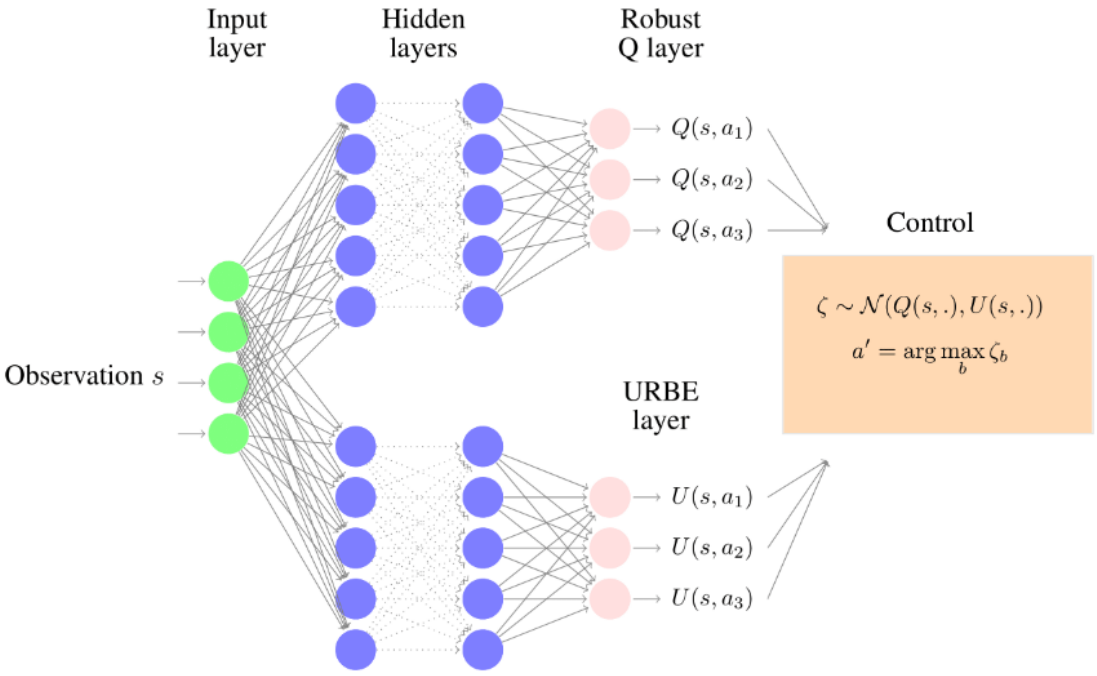


Figure 2.6: Neural network architecture in DQN-URBE [18].

approximate the robust Q-function as RTD-DQN, while the lower head learns to approximate variance boundaries in this robust Q-function as UBE. Algorithm 2 describes the learning process of DQN-URBE. At each step t of an episode, the agent in state s_t feeds its observation o_t into the network N to estimate the robust Q-value $Q(s_t,a_i)$ and its posterior variance boundary approximation $U(s_t,a_i)$ for each action a_i available from s_t. Then, for each state-action pair (s_t,a_i), the agent approximates its true posterior distribution as $\mathcal{N}(Q(s_t,a_i),U(s_t,a_i))$, and applies Thompson sampling [69] over it to approximate the potential true Q-value of (s_t,a_i). By selecting the next action a' greedily accordingly, the agent is expected to maximize the potential gain in the next step. In this way, the DQN-URBE agent is encouraged to explore less extreme scenarios so that a less conservative policy can be produced. As in RTD-DQN, taking a' will also transfer the DQN-URBE agent to the next state s_{t+1} and give it a reward r_t deterministically according to the unknown true

MDP model, which DQN-URBE assumes as the nominal one, and estimate the worst-case next state s_{t+1}^{min} for updating. The network update process is the same as in RTD-DQN and UBE.

Algorithm 2 DQN-URBE, adapted from [18]

Require: Neural network N, Thompson sampling hyperparameter β

for episode k = 1,... **do**
  Initial model state s_0 and its observation o_0
  for t = 0,...T-1 **do** ▷ For each step
    Feed current o_t of s_t into N to compute $Q(s_t, a_i)$ and $U(s_t, a_i)$ for each action a_i available from s_t
    Sample $\zeta_{a_i} \sim \mathcal{N}(0,1)$ for all a_i
    $a' = \text{argmax}_{a_i} \left(Q(s_t, a_i) + \beta \zeta_{a_i} \sqrt{U(s_t, a_i)} \right)$
    Take a' and observe $(s_{t+1}, r_t, s_{t+1}^{min})$
    Update N by using RTD-DQN and UBE
  end for
end for

The experimental results of Derman *et al.* showed that DQN-URBE outperformed standard DQN and RTD-DQN in three simple uncertainty problems [18]. In these problems, standard DQN followed the 'predict-then-act' approach and provided the worst robustness. It worked the best only if the actual scenario was sufficiently similar to the predicted one. Otherwise, its performance was greatly compromised. RTD-DQN was the most stable to uncertainty, but its conservativeness prevented it from performing well in any scenario. Compared with them, DQN-URBE achieved the best trade-off between average performance and stability to uncertainty, thus providing the highest robustness. Nevertheless, a shortcoming of their experiments was that they did not test these algorithms in more complex problems.

The main shortcoming of DQN-URBE is that its computational tractability relies on the state-action rectangularity assumption in RMDP [18], where the uncertainty for each state-action pair is described independently [57]. This assumption allows the agent to learn from trajectories of incompatible worst cases, which can result in overly conservative policies. Although some existing algorithms that implement Robust Optimization have circumvented this assumption, they are inapplicable in deep uncertainty problems because of their poor scalability [63, 64].

2.5.2 Risk-Sensitive Objectives

The second idea introduced by Dietterich is *Risk-Sensitive Objectives* [55]. This idea is to take a risk-measure of the policy performance over an ensemble of possible scenarios as the objective for RL to learn robust policies [70], which is similar to the scenario-based approach introduced in Section 2.2.3. Common risk-measures include Value-at-Risk, Conditional-Value-at-Risk (CVaR),

and variance [71], among which CVaR is one of the best and most popular measures for decision-making problems with MDPs [55]. Therefore, we only focused on this measure in this project. CVaR is defined as follows. Given a policy performance distribution over the scenario ensemble arising from the model and parameter uncertainty of the problem Z, $CVaR(\alpha)$ equals to the expected value of Z, conditioned on α^{th} percentile v_α of Z [72].

$$CVaR(\alpha) = E\left[R \,|\, R \leq v_\alpha\right]$$

Compared with Robust Optimization, Robust RL with a CVaR objective is often less conservative because it aims to maximize the expected policy performance in a range of worst cases. Nevertheless, the main shortcoming of this idea is that it cannot handle unquantified uncertainties [73].

Previous studies have proposed many algorithms to support Robust RL with a CVaR objective. Chow *et al.* investigated Value Iteration [71]. However, this algorithm is inapplicable in deep uncertainty problems because of its poor scalability, as we discussed before. Other works also studied Policy Gradient or Distributional RL [72, 74, 75]. Since these algorithms all assume that there is a single true MDP model describing all uncertainties, we argue they are also inapplicable in DMDU. Sharma *et al.* relaxed this assumption by building their algorithm on a distribution over possible scenarios [73]. However, this distribution must be discrete to guarantee the computational tractability of their algorithm. In contrast, EPOpt by Rajeswaran *et al.* only requires a scenario ensemble as input, therefore more suitable for supporting exploration in DMDU [76]. Moreover, Pinto *et al.* eliminated the need for these inputs by introducing the concept of adversarial RL [77]. During the learning process of their algorithm, a regular agent and an adversarial agent are trained simultaneously. The latter learns to control all uncertain factors of the problem to generate percentage worst cases, from which the former can learn a robust policy. However, this algorithm also suffers from the rectangularity like DQN-URBE, because its adversarial agent makes decisions independently in each state. Based on this review, we eventually chose Ensemble Policy Optimization (EPOpt) as our second Robust RL algorithm for comparison in this project.

Ensemble Policy Optimization

EPOpt employs a round-based learning protocol [76]. Its learning process consists of multiple rounds. Each round further includes two steps: (1) Robust Policy Search, and (2) Source Domain Ensemble Adaptation. In Step (1), the EPOpt agent learns a robust policy by interacting with the current ensemble of possible scenarios, namely source domain ensemble. In Step (2), EPOpt interacts with the real-world environment following the policy learned in Step (1), and collects data to update the source domain ensemble so that it approximates the real-world environment more accurately. Then, this ensemble will be used to produce a better policy in the next round.

It can be observed that this round-based learning process matches the iterative DMDU framework (Figure 2.2). Since this project did not consider the cycling process in DMDU, we only considered the single-round Robust Policy Search of EPOpt in our experiments.

EPOpt - Robust Policy Search

Algorithm 3 describes the single-round Robust Policy Search process in EPOpt(α) with a $CVaR(\alpha)$ objective [76]. This process starts with an input source domain ensemble S_ϕ, which covers the model and parameter uncertainty space of the problem, and a random policy π_0. This policy is updated to optimal robustness through multiple iterations. In each iteration i, EPOpt first samples N possible scenarios $\{s_n\}_{n=1}^N$ from S_ϕ. Then, a set of possible trajectories $\{\tau_n\}_{n=1}^N$ is collected by letting the agent interact with the model in each scenario s_n following the current policy π_i in each episode n. Unlike ordinary RL, although the EPOpt agent still receives rewards, no step-wise policy update happens during this trajectory collection process. Instead, the accumulated rewards gained in these trajectories are computed $\{R(\tau_n)\}_{n=1}^N$, based on which a batch of the $\alpha\%$ worst trajectories Υ can be found. Eventually, π_i is updated based on Υ by using a policy gradient based batch policy optimization algorithm. For example, Rajeswaran *et al.* used Trust Region Policy Optimization (TRPO) [78] by default. In this way, EPOpt describes the uncertainty in the form of the entire trajectory, rather than independently for each state-action pair as DQN-URBE. Therefore, EPOpt circumvents the rectangularity assumption and mitigates unnecessary conservativeness. Finally, since the EPOpt policy is represented as the Gaussian distribution or categorical distribution, EPOpt supports both continuous and discrete action spaces.

Algorithm 3 EPOpt(α) - Robust Policy Search, adapted from [76]

Require: Source domain ensemble S_ϕ, Initial policy π_0, Sample size N, Hyperparameter α

for iteration i = 0,... **do**
  for episode n = 1,...N **do**
    Sample a possible scenario $s_n \sim S_\phi$
    Collect a trajectory $\tau_n = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T-1}$ by interacting with the model in s_n following π_i
    Compute the accumulated reward $R(\tau_n)$ gained in τ_n
  end for
  Compute $Q_\alpha = \alpha^{th}$ percentile of $\{R(\tau_n)\}_{n=1}^N$
  Select subset $\Upsilon = \{\tau_n | R(\tau_n) \leq Q_\alpha\}$
  Update policy: $\pi_{i+1} = BatchPolOpt(\pi_i, \Upsilon)$
end for

Unconstrained Initial Exploration

The sub-sampling step in Robust Policy Search of EPOpt (line 8 in Algorithm 3) contributes to learning robust policies by emphasizing penalizing poor trajectories. For EPOpt(α) with a small α, taking this step from the beginning of its learning process may prevent the agent from performing the necessary exploration to find good trajectories, and thereby result in unstable learning. Therefore, this Robust Policy Search process often starts with $\alpha = 100$ for some iterations before switching α to the desired value [76].

Rajeswaran *et al.* compared the performance of standard TRPO and EPOpt(α) with different α values in their experiments [76]. Similar to the standard DQN tested in [18], their standard TRPO also followed the 'predict-then-act' approach, and only worked well when the actual scenario was similar enough to the predicted one, hence providing poor robustness. In contrast, all EPOpt generally performed satisfactorily in most scenarios. With α decreasing from 100 to 5, the average performance of EPOpt(α) decreased slightly, but its performance variance was halved. Therefore, Rajeswaran *et al.* took EPOpt(10) as the default algorithm, which achieved the best trade-off between performance and stability to uncertainty.

2.5.3 Robust Model-Free Reinforcement Learning

In addition to the two ideas for Robust RL introduced above, ordinary model-free RL has also been widely applied to handle uncertainty in practical decision-making problems [14, 51, 79–82]. RL can be further categorized into model-based RL and model-free RL [13, 16]. The former requires a complete MDP model of the problem as input. The latter simply learns through trial and error in the real environment or its simulator, where the agent cannot access the internal mechanisms of the system, such as the transition probability distributions P or the reward function r. Existing studies have experimentally shown that if there is an environment that can produce the system behavior under uncertainty, model-free RL can be directly used as Robust RL to learn a robust policy from sufficient interactions with the environment [79, 80, 82]. This idea is conceptually simple and easy to implement, but no theoretical robustness guarantee has been provided in the studies cited above.

Common model-free RL algorithms include model-free Q-learning [83], SARSA [84], Monte-Carlo methods, and actor-critic methods [16]. Among these algorithms, Q-learning and its variants, such as DQN, are the predominant ones applied in existing studies, due to their simplicity and guarantee of optimality [14, 16]. In this project, we chose robust model-free DQN as our third Robust RL algorithm for comparison, because DQN further improves the stability of Q-learning and supports continuous state space [16]. For simplicity, we will abbreviate robust model-free DQN as Robust DQN in the rest of this thesis.

2.5.4 Adaptive Management

Unlike the previous three ideas that provide robustness by learning robust policies, *Adaptive Management* achieves Robust RL by improving the process of decision-making with RL [85–87]. The main idea of adaptive management is to adjust the decision-making practice based on policy deployment outcomes [86]. The two main types of adaptive management are active and passive adaptive management, which respectively correspond to the adaptive policy deployment process and the cycling process in the general DMDU framework (box 2,3 and loop in Figure 2.2).

Active Adaptive Management

Active adaptive management requires 'thinking ahead' [87]. Decision-makers should develop a policy for every possible scenario they predict beforehand so that they can quickly adjust their plan according to the actual situation during deployment. The main disadvantage of this idea is that planning for all possible futures is often computationally intractable, especially for deep uncertainty problems where the uncertainty is unpredictable. Hence, we argue this idea is inapplicable in DMDU.

Passive Adaptive Management

Passive adaptive management requires decision-makers to first develop and deploy a plan based on their preliminary knowledge of the problem [87]. As the plan progresses, differences between their knowledge and the fact should be identified and used to improve their knowledge and rework a better plan. This idea matches the cycling process in the DMDU framework (Figure 2.2), hence applicable in DMDU.

2.6 Multi-Objective Robust Reinforcement Learning

Based on our review, the ideal way to construct MORRL for exploration in DMDU is to integrate multi-policy MORL with the Robust RL algorithm, and embed it into the passive adaptive management framework. Meanwhile, although active adaptive management is not applied, the policy deployment process in this framework is still adaptive due to the nature of the dynamic policy. We propose Figure 2.7 to show the framework of decision-making with MORRL and passive adaptive management. Through this way, the DMDU approach (Figure 2.2) and decision-making with MORRL (Figure 2.7) almost follow the same flow, so MORRL should also suit the general DMDU framework. This discussion theoretically demonstrates the viability of applying MORRL as an alternative exploration method in DMDU approaches with regard to the flow.

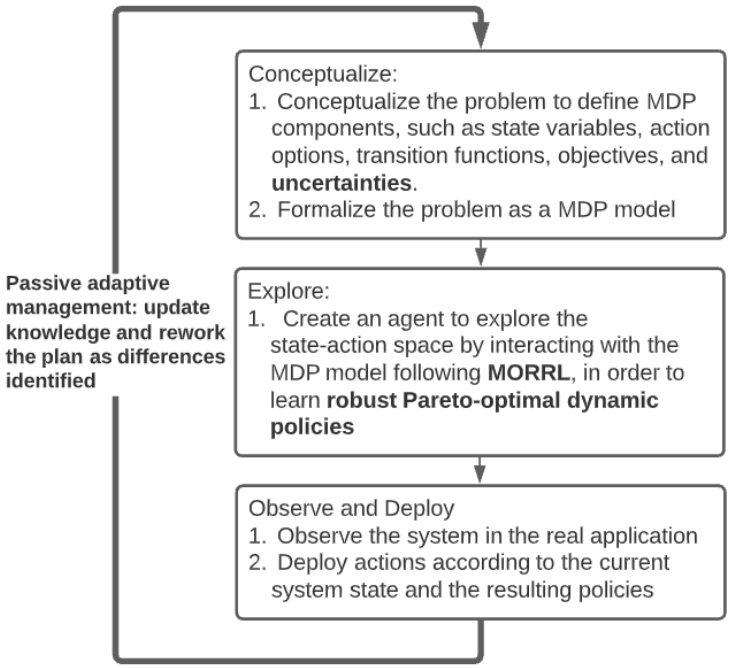


Figure 2.7: Framework of decision-making with MORRL and passive adaptive management.

However, to the best of our knowledge, no existing work has studied the integration of multi-policy MORL and Robust RL. Given this knowledge gap and the time limit of this project, we eventually decided to adopt 'naive' multi-policy MORRL in our multi-objective case studies. It worked by only considering a small number of possible objective preferences and applying the Robust RL algorithm to learn policies targeted at each of them sequentially. Additionally, we also excluded passive adaptive management from our experiments because this project did not consider the cycling process in DMDU (loop in Figure 2.2).

2.7 Conclusion

Based on these reviews, we eventually selected two MOREAs as baselines, which were ε-NSGA-II and Borg, and three MORRL algorithms as candidates, which were Robust DQN, DQN-URBE, and EPOpt, for comparison in our experiments to answer our research questions. We chose these algorithms mainly because they are not only suitable for exploration in DMDU, but also have available implementations. Moreover, since Robust RL is a novel research area, previous experiments on these Robust RL algorithms mainly compared them with non-robust ones, but no existing research has studied the comparison between them. This is another knowledge gap that we tried to fill in this project.

Chapter 3

Experimental Plan

We compared the performance of MOREA and MORRL in three problems to answer our research questions. This chapter introduces the overall plan of these experiments, including sufficient implementation details to support their reproduction. Moreover, these contents also reflect the general model interface and DMDU process we designed for MOREA and MORRL as the common environment for fair comparisons. Our plan consists of the three steps (Problem Conceptualization, Exploration, and Deployment and Evaluation) in the general DMDU framework (Figure 2.2). Unlike most practical applications, our experiments did not involve the cycling process due to the time constraint practical within a one-year project. Similar designs have been applied in many past studies of DMDU approaches [8, 11, 20, 28]. This chapter proceeds as follows. Section 3.1 introduces how we formalized the problem as corresponding models for our algorithms. Section 3.2 details how each algorithm was implemented for exploration. And Section 3.3 discusses how we evaluated and analyzed the algorithms' performance for comparison.

3.1 Problem Conceptualization

3.1.1 Problem Formalization

In the practical deep uncertainty problem, decision-makers often first frame the question and specify possible system structures based on their observations of the real world, in order to design a simulation model to support exploration [6]. To facilitate this process, we used three previously studied problems with available source models, based on which we identified the following components to construct our models: problem parameters, time horizon of the episode, time-step between two consecutive model states, performance indicators characterizing these states-namely state variables, initial and terminal model states, action options, underlying transition function, as well as the parameter and objective uncertainties involved in the problem. We did not consider model

uncertainty in these problems, because we assumed their source models were accurate (Section 2.1).

3.1.2 Model Implementation

To establish a common environment to fairly compare our algorithms, we used MDP as the underlying formal model to build all the models needed in our experiments. These MDPs were implemented as OpenAI Gym Environments [88] in different ways to describe parameter uncertainty, in order to support explorations using different algorithms. Furthermore, the exploration processes of the MOREAs and our evaluations of policy performance also relied on another Python package, namely EMA Workbench [43].

OpenAI Gym Environment

OpenAI Gym is a convenient toolkit for implementing and comparing RL algorithms [88]. It also provides a standard interface to programmatically specify a MDP to model the deterministic problem (OpenAI Gym Environment). According to this interface, implementing a MDP requires defining the following three functions: (1) *'init'* initializes the model, including specifying the parameter values and the ranges of state variables and action options involved in the model; (2) *'reset'* resets the model to its initial state; (3) *'step'* performs an input action to transit the model to its next state based on the transition function, and returns the observation and reward received at that state, as well as a flag indicating whether the terminal state is reached. These functions can be implemented based on the components identified in Section 3.1.1.

Usually, *'init'* is only called once at the beginning of the RL learning process. In each episode, the agent first *'reset'* the model to its initial state and iteratively *'step'* until reaching a terminal state. During this period, the feedback returned by *'step'* forms the corresponding trajectory, which is used to update the current policy of the agent.

Exploratory Modelling and Analysis

Exploratory Modelling and Analysis (EMA) is a research methodology that infers and analyses the performance of systems under uncertainty by using computational experiments [89, 90]. EMA considers uncertainties surrounding the problem, exploits available knowledge to establish an ensemble of plausible scenarios, and samples or searches over this scenario set for analysis purposes. Specifically, in the context of DMDU, these uncertainties may include a broad range of model structures and parameter values [7]. EMA is mainly used in the exploration process of DMDU approaches to evaluate the performance of alternative policies across various uncertainties [6–8,

11, 91]. EMA Workbench provides programmers with EMA functionalities via application programming interfaces (APIs) [43].

Robust DQN

Model-free DQN learns through trial and error in the real environment or its simulator without knowledge of the internal mechanisms of the system [13]. The underlying MDP can act as such a simulator, where *'step'* encapsulates the transition function and hides it from the agent. To further model the parameter uncertainty, a common practice is to assign different parameter values to the model in every episode, in order to simulate different possible scenarios for the agent to learn from during exploration [79]. Therefore, our *Robust MDP Simulator* extended the underlying MDP by requiring a set of possible scenarios S as extra input for *'init'*. Whenever an episode starts, *'reset'* reads a random scenario $s \in S$, and assigns its parameter values to the model before initializing the model state. Then, the episode proceeds with these assignments until it terminates (Figure 3.1). In this way, the agent learns to plan for one possible scenario in each episode. With a sufficient number of episodes, the agent should be able to experience all possible situations enough times and learn a policy robust to them [79, 80, 82].

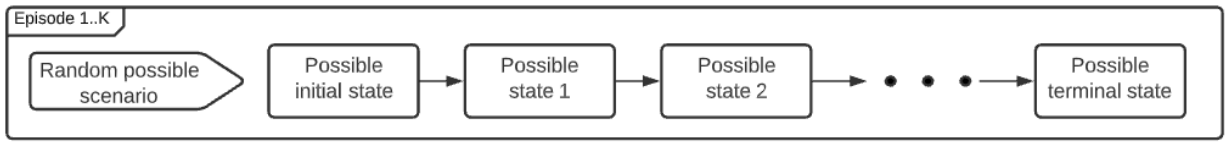


Figure 3.1: Flowchart of the exploration process of Robust DQN.

DQN-URBE

DQN-URBE is a planning algorithm for RMDP. For a deep uncertainty problem, RMDP samples the trajectory according to its nominal scenario, and also considers other possible scenarios independently at each step to describe the parameter uncertainty [57]. Our implementation of RMDP was adapted from [18] (Figure 3.2). On the basis of the underlying MDP, *'init'* of our RMDP reads both a nominal scenario $s_{nominal}$ and a set of possible scenarios S as extra input. Unlike Robust MDP Simulator, *'reset'* of RMDP always initializes the model according to $s_{nominal}$. Whenever *'step'* is called, RMDP not only transits to the next state and computes the corresponding reward according to $s_{nominal}$, but also predicts other possible next states according to each scenario $s \in S$. Then, the nominal reward and observations of both nominal and possible next states are collected for the agent to plan for the worst. Since the state sampling undertaken in different steps is independent, DQN-URBE suffers from the rectangularity assumption [18].

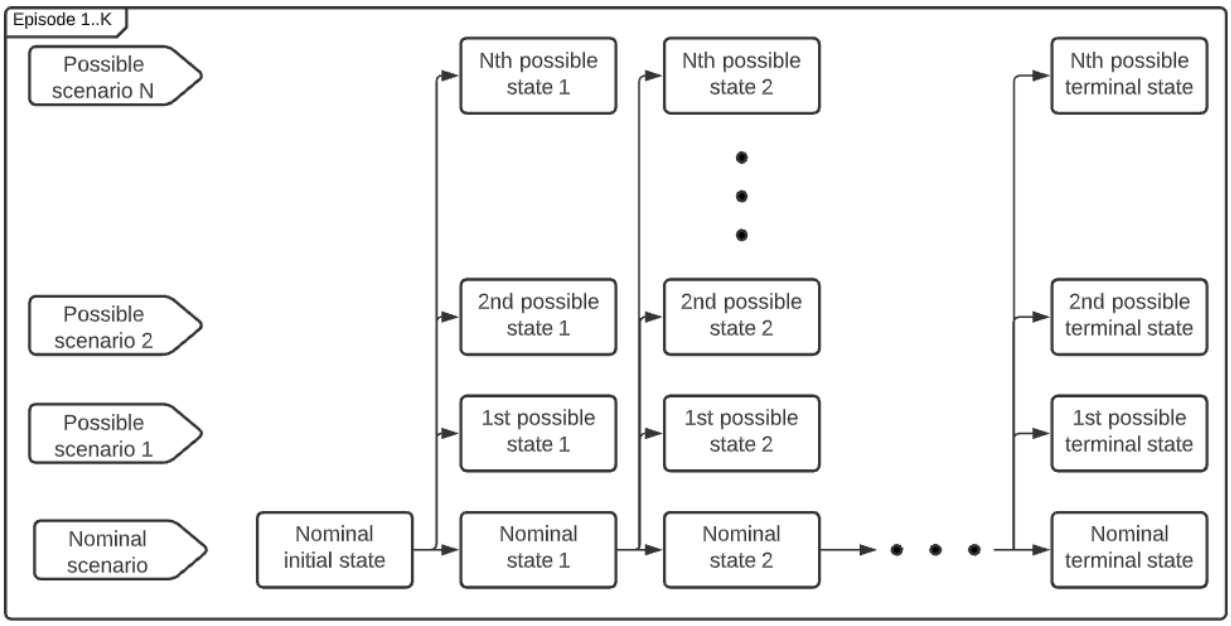


Figure 3.2: Flowchart of the exploration process of DQN-URBE, corresponding to Algorithm 2.

EPOpt

To learn a robust policy while circumventing the rectangularity assumption, EPOpt(α) describes the parameter uncertainty by sampling N complete possible trajectories $\{\tau_n\}_{n=1}^{N}$ according to N possible scenarios $\{s_n\}_{n=1}^{N} \subseteq S$, and plans for the $\alpha\%$ worst trajectories in every iteration [76]. Our Robust MDP Simulator can support this process. Moreover, we also let the EPOpt agent load the n^{th} scenario from S in episode n in every iteration, so that the agent can always iterate over entire S in every iteration (Figure 3.3).

ε-NSGA-II and Borg

Previous studies of MOREAs in DMDU approaches mainly used EMA Workbench to support their experiments [8, 11], and we also followed this way. EMA Workbench provides the implementation of ε-NSGA-II and Borg, evaluators that measure policy fitness in given scenarios, and optimizers that combine these tools to identify robust Pareto-optimal static policies for the problem. An EMA optimizer requires the following five inputs: an EMA model, a set of possible scenarios S, the epsilon value for each objective, the number of function evaluations used for exploration, and a robustness function. Here, the EMA model is an encapsulation of a simulation model M and the specification of the problem uncertainty. The robustness function specifies the robustness metric for each objective used to measure the policy robustness. During the exploration process, the optimizer first samples some candidate policies randomly, and then evolves them through many

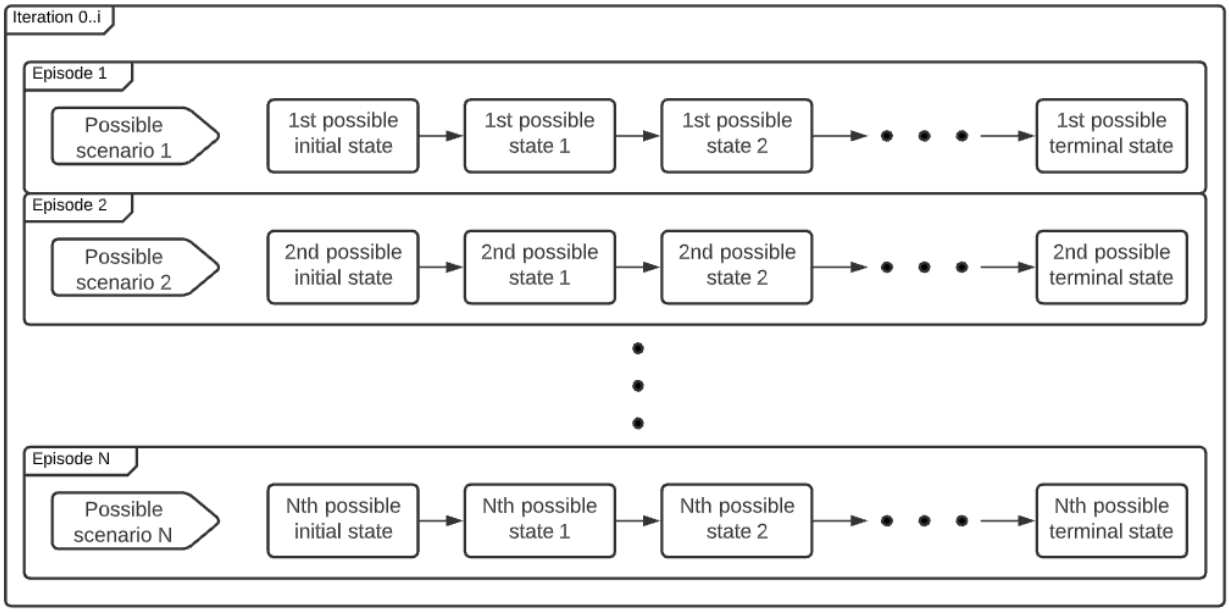


Figure 3.3: Flowchart of the exploration process of EPOpt, corresponding to Algorithm 3.

function evaluations following the MOREA. In each of these evaluations, the robustness of a candidate policy is evaluated over S as its fitness for selection, by using M and an EMA evaluator. Here, M is a function that reads a scenario and a static policy as input, and samples a trajectory accordingly to measure the policy performance in that scenario. Since this model is conceptually similar to the simulator used by model-free DQN, we also implemented it based on Robust MDP Simulator: Whenever M is invoked, it will call Robust MDP Simulator to *'init'* and *'reset'* according to the input scenario; Then, M will iteratively call *'step'* of the simulator following the input policy until the simulator terminates; Finally, the observations collected in this process will form the trajectory used to measure the policy performance.

3.1.3 Parametric Model Misspecification

We further considered another source of parameter uncertainty that cannot be described by the model, called parametric model misspecification. It refers to the case where true values of uncertain parameters fall outside the ranges estimated for decision-making [18]. Therefore, in our experiments, we defined two value ranges for each uncertain parameter: a training range and a testing range, while the training range is a subset of the testing range. We let our algorithms explore only possible scenarios sampled from the training ranges to generate robust policies during exploration, and used scenarios sampled from the testing ranges to assess their policy performance during evaluation.

3.1.4 Stabilization Analysis

In previous studies of Robust RL, their MDPs often described the parameter uncertainty by using a random generator to produce random parameter values in every episode to simulate different possible scenarios for learning [18, 76, 79]. However, Kwakkel *et al.* argued that this strategy would introduce extra noise because scenarios generated in different episodes varied slightly [8]. Instead, they recommended using the same set of scenarios for all explorations in the same experiment. To determine how many scenarios were needed to describe the uncertainty stably, we adopted the same method used by Kwakkel *et al.*, called stabilization analysis. In each experiment, we first evaluated the performance of 10 static policies in 1,000 scenarios to find out the number of scenarios when the robustness metrics for the objective indicators stabilized. These testing policies and scenarios were randomly sampled from the action space and training parameter value ranges of the problem, by using Latin Hypercube sampling [92] provided by EMA Workbench and random seed α (Table 8.1). We also used the product of the median and the interquartile distance plus one as the robustness metric in this analysis. After determining the number, we again used the same way to generate a set of possible scenarios S of that size, called training scenario set, and used it for all model implementations introduced in Section 3.1.2.

3.1.5 Objective Uncertainty

As mentioned in Section 2.6, we adopted 'naive' multi-policy MORRL to deal with the objective uncertainty, which sequentially learned multiple policies targeted at different pre-defined objective preferences. Therefore, in each of our multi-objective case studies, we actually built multiple MDP models, each of which had one of these preferences defined in its reward function. By interacting with one of these models per run, our MORRL algorithms were able to learn multiple robust Pareto-optimal policies for the problem through many runs.

In contrast, the MOREAs could sort their candidate policies non-dominantly for evolution during exploration, thereby generating multiple Pareto-optimal policies in one run. To support this, their corresponding simulation models measured the policy performance regarding each objective independently, rather than associating it with the single reward function in Robust MDP Simulator. In this case, the MOREAs only required one EMA model for each multi-objective problem, and did not require decision-makers to predict possible objective preferences in advance.

3.2 Exploration

3.2.1 Hyperparameters

The selection of hyperparameter values has a great impact on the algorithm performance. Therefore, the same amount of hyperparameter tuning should always be performed for all algorithms to guarantee a fair comparison between them [93]. Ideally, these parameters should be tweaked to obtain optimal algorithm performance in every case study, but we found the required computational budgets were impractical for this one-year project. Eventually, we decided to always use the default hyperparameter values of our algorithms in the experiments.

3.2.2 Robust DQN

We adopted the implementation and default hyperparameter configuration of DQN from Ray RLlib [94]. Ray is a Python package that provides general APIs for distributed applications, while RLlib is a Ray-based library, providing scalable APIs for RL applications. To apply this DQN implementation to a deep uncertainty problem, we first registered the corresponding Robust MDP Simulator as a Ray environment and initialized a Ray server. Then, we instantiated a DQN trainer agent from RLlib and associated it with the environment. Finally, we iteratively called the *'train'* method of the agent to learn a robust policy. This method encapsulates all logic necessary for the learning process, and executes multiple episodes with a fixed total number of model steps at each iteration. The multiprocessing of the learning process was also fully supported by the Ray server.

3.2.3 DQN-URBE

Our implementation and hyperparameter values (Table 3.1) of DQN-URBE were adapted from [18]. To execute the learning process, the program first initialized a neural network as the policy, and used it to interact with the RMDP through many episodes, from which the program learned to optimize the network (policy). This network was constructed and parallelly updated by using TensorFlow [95], which is a general and extensible platform for machine learning, providing tools for neural network optimization and its multiprocessing. However, unlike this parallel network update subroutine, Derman *et al.* designed the RMDP to work sequentially in DQN-URBE [18]. This is because the step-wise sampling in RMDP occurs much too frequently, and thus parallelizing this subroutine would in turn slow it down in simple problems.

Hyperparameters	Descriptions	Values
GAMMA	Discount factor of Q-learning	0.9
LR	Learning rate	10^{-4}
MAX EXPERIENCE	Size of experience replay memory	2000
BATCH SIZE	Batch size in batch optimization	256
H1 SIZE	Size of hidden layer 1 for robust Q-function in the nerual network	128
H2 SIZE	Size of hidden layer 2 for robust Q-function in the nerual network	128
H3 SIZE	Size of hidden layer 3 for robust Q-function in the nerual network	128
TARGET UPDATE INTERVAL	How often to update the target network in terms of the number of episodes	10
MU_VARIANCE	Scaling variance for uncertainty in UBE	0.01
H_SIZE_U	Size of hidden layer for posterior variance boundary approximation in the nerual network	100
BETA_TS	Thompson Sampling parameter	0.5

Table 3.1: Hyperparameter values of DQN-URBE [18].

3.2.4 EPOpt

We adopted the implementation and hyperparameter values (Table 3.2) of EPOpt from [76]. To execute the learning process, the program first instantiated a multilayer perceptron as the initial policy. Then, it sampled trajectories by interacting with all possible scenarios following its current policy, and applied TRPO as the batch policy optimization algorithm to update the policy according to the percentage worst trajectories through many iterations. Here, the trajectory collection subroutine was fully parallelized with the Python multiprocessing package. The multilayer perceptron and TRPO were adapted from a RL development framework, called rllab [96]. Finally, it is worth noting that EPOpt returned the policy that maximized the average performance over the training scenario set S during its learning process as its result, rather than the policy produced in the end like the other algorithms.

3.2.5 ε-NSGA-II and Borg

As introduced in Section 3.1.2, we used the optimizer from EMA Workbench to apply ε-NSGA-II and Borg for exploration in our experiments. Meanwhile, we also adopted their hyperparameter

Hyperparameters	Descriptions	Values
hidden1 size	Size of hidden layer 1 in the multilayer perceptron	64
hidden2 size	Size of hidden layer 2 in the multilayer perceptron	64
max_kl	Maximum KL divergence in TRPO	0.01
gamma	Discount factor of Q-learning	0.995
init_explore_iter	Number of iterations for the unconstrained initial exploration	100
alpha	Value of α for EPOpt(α) with a $CVaR(\alpha)$ objective	10

Table 3.2: Hyperparameter values of EPOpt [76].

settings from the documentation of EMA Workbench (Table 3.3). Since all logic and multiprocessing of their exploration processes are encapsulated in the *'robust_optimize'* method of the EMA optimizer, we only needed to call it once with the required inputs to produce the resulting policies in each experiment.

Hyperparameters	Descriptions	Values
robustness function	Specifying the robustness metric for each objective indicator used to measure the policy robustness over the training scenario set S as its fitness for selection	10^{th} percentile performance
epsilons	Epsilon value for each objective	0.01

Table 3.3: Hyperparameter values of ε-NSGA-II and Borg [43].

3.3 Deployment and Evaluation

Eventually, we evaluated the performance of our MORRL algorithms and MOREAs regarding their computational efficiency and policy robustness to deep uncertainty for comparison.

3.3.1 Efficiency Evaluation

We measured the computational efficiency of the algorithms in terms of how quickly they identified their final policies during exploration. The data were directly collected from their exploration processes. The metrics we used were learning curve [13] and ε-progress curve [38]. A learning

curve plots the accumulated reward obtained by a MORRL agent in its current episode as a function of time. An ϵ-progress curve plots the cumulative number of times a MOREA finds a substantially better policy as a function of time. These two metrics are widely used in the RL and EA areas respectively [11, 18, 72, 73, 75]. An alternative efficiency metric is the number of episodes required for each algorithm to converge [11, 12]. However, since our algorithms spent quite different time on policy updates in each episode, this metric could not accurately reflect their computational efficiency. For simplicity, we will abbreviate computational efficiency as efficiency in the rest of this thesis.

3.3.2 Robustness Evaluation

We measured the robustness of the algorithms in terms of how stably their policies provided satisfactory performance across the uncertainty space of the problem. These evaluations were also supported by EMA Workbench. In each experiment, we took the simulation model used by the MOREAs for exploration as input, and used an EMA evaluator to evaluate the performance of our algorithms' resulting policies across a testing scenario set. For the same reason discussed in Section 3.1.4, the same scenario set was used for the evaluation of each algorithm. This set was randomly sampled using Latin Hypercube sampling and random seed α from the testing parameter value ranges to simulate model misspecification. We used sets of size 5,000 in the Cartpole and Lake problems and a set of size 1,000 in the Electricity Market problem. Finally, random seed η was used to control the randomness of all evaluation processes. Similar robustness evaluation approaches have been used in previous studies of DMDU approaches [4, 6, 8, 11].

These evaluations used four metrics to reflect policy robustness, including (1) **policy performance distribution** over the uncertainty space, (2) **Maximin** [97]: the policy performance in the worst scenario, (3) **Starr's domain criterion** [98]: the fraction of scenarios in which the policy performance meets a pre-defined threshold, and (4) **10^{th} percentile performance**: the policy performance in the 10^{th} percentile worst case. These metrics come from different robustness metrics families, hence reflecting different robustness aspects [27]. Simultaneous use of these metrics to evaluate our algorithms helped us draw a comprehensive view of their policy robustness from these aspects, and therefore revealed their complementarity [99].

3.4 Interpretation

3.4.1 Efficiency

In each experiment, we plotted a line chart of learning curves for the MORRL algorithms, a line chart of ε-progress curves for the MOREAs, and their 95% confidence intervals for efficiency analysis (Figure 3.4). When comparing two MORRL algorithms (*C* and *D*), or a MORRL algorithm and a MOREA (*C* and *A*), we only compared their convergence time and argue that *C* was more efficient than *D* and *A* because *C* converged faster. Their policy performance during exploration was not comparable, because these algorithms were built on different models. In contrast, ε-NSGA-II and Borg were built on the same model (*A* and *B*), so their comparisons involved both the time and the number of ε-progresses taken to converge. We argue that *B* was more efficient than *A*, because *B* took a shorter time and fewer ε-progresses to converge, which indicated that it took less effort to identify its final policies [38]. Moreover, in experiments where each algorithm was run multiple times with the use of different random seeds, the curve was averaged across all runs. In this case, the confidence interval of the curve reflected the performance difference of the algorithm when using different random seeds. Therefore, we argue that *C* was less sensitive to the randomness of the exploration process than *D*, because it had a smaller interval. Finally, it is worth noting that these curves only reflected the algorithms' exploration processes in one run. However, to identify multiple Pareto-optimal solutions in a multi-objective problem, each 'naive' multi-policy MORRL algorithm took multiple runs, while each MOREA only took one run, so the former was actually less efficient than shown in the figure (Section 3.1.5).

3.4.2 Robustness

Policy Performance Distribution

In each single-objective problem, we plotted the policy performance distribution and its standard deviation of each algorithm over the parameter uncertainty space (Figure 3.5). When comparing two algorithms *A* and *B*, we argue that *A* provided higher average performance because its distribution had a higher mean. *A* also provided higher stability to the parameter uncertainty, because its performance was more densely distributed over the uncertainty space. Based on these two points, we argue that *A* provided higher robustness to the parameter uncertainty. Additionally, the policy performance of *A* was not obviously compromised outside the training range but that of *B* was, so *A* provided robustness to the model misspecification but *B* did not. In experiments involving multiple random seeds, these distributions were also averaged across all runs, and their standard deviations reflected the performance difference of the algorithms when using different seeds. Therefore, we argue that *A* was less sensitive to the exploration randomness than *B*, because of its smaller standard

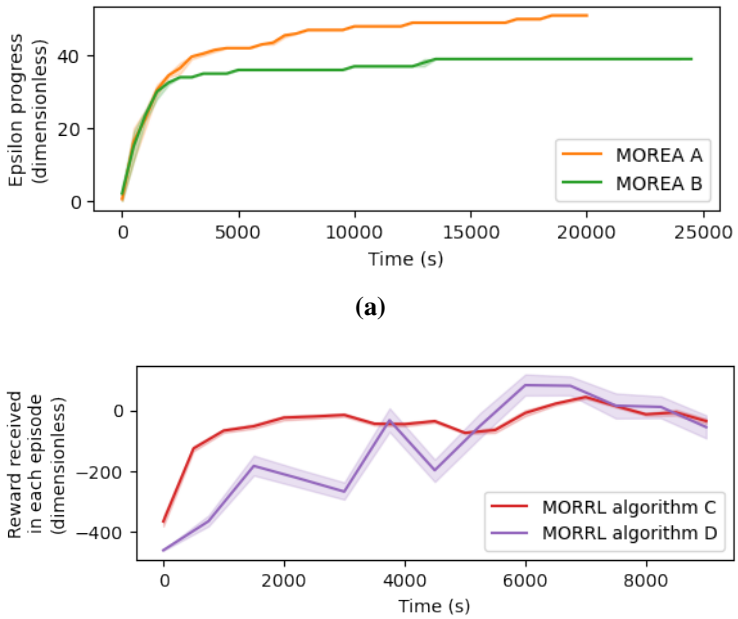


Figure 3.4: Sample line charts for an efficiency evaluation, including ε-progress curves for the MOREAs (a), learning curves for the MORRL algorithms (b), and their 95% confidence intervals. Each curve of the algorithm is averaged over its multiple runs with different random seeds. Each point on the curve represents the value averaged over 500 seconds.

deviation. Moreover, in these experiments, we also plotted the performance distributions generated using different random seeds separately for each algorithm, which more intuitively reflected this sensitivity of the algorithms.

In each multi-objective problem, we plotted the policy trade-off distribution and its interquartile range of each algorithm over the objective uncertainty space (Figure 3.6). For two policies a and b, if a averagely outperformed b on both objectives, a was said to dominate b. We compared the average policy performance of the algorithms based on their policy dominance relationship. For example, the policies of A and B were mutually non-dominated, so they provided similar average performance. Some policies of C dominated some policies of A, and no C policy was dominated by the A policies, so we argue that C provided higher average performance. Meanwhile, C also provided higher stability to the parameter uncertainty, because its interquartile ranges were smaller, which indicated its policy performance was more densely distributed over the parameter uncertainty space. Therefore, C provided higher robustness to the parameter uncertainty. On the other hand, we evaluated the algorithms' robustness to the objective uncertainty according to the ranges and densities of their trade-off distributions. Since the distribution range of B was larger than that of A, B was theoretically able to provide more possible preferences between the two objectives. The distribution density of A was greater than that of B, so A would allow decision-

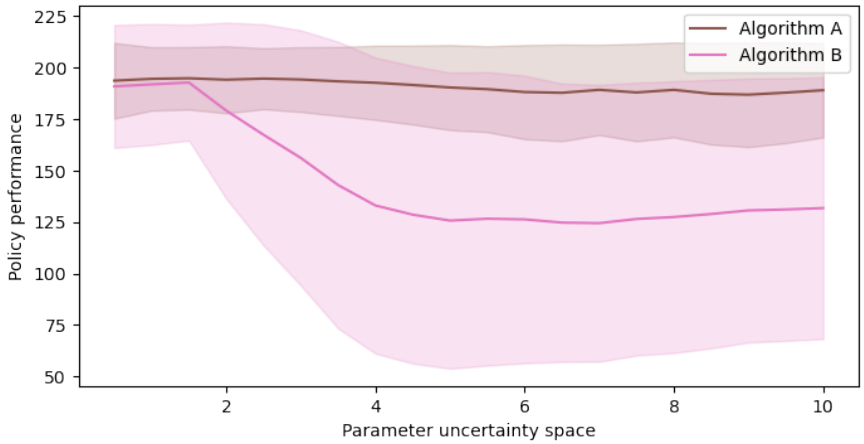


Figure 3.5: Sample average policy performance distributions. Each line refers to the policy performance distribution and its standard deviation of an algorithm over the parameter uncertainty space, which averages the performance of the algorithm generated using different random seeds. In this experiment, the training parameter value range is [0.5, 2], while the testing value range is [0.5, 10].

makers to finer-adjust their expected preferences during policy deployment. Thus, we argue that *B* provided higher robustness in the first aspect, and *A* provided higher robustness in the second aspect.

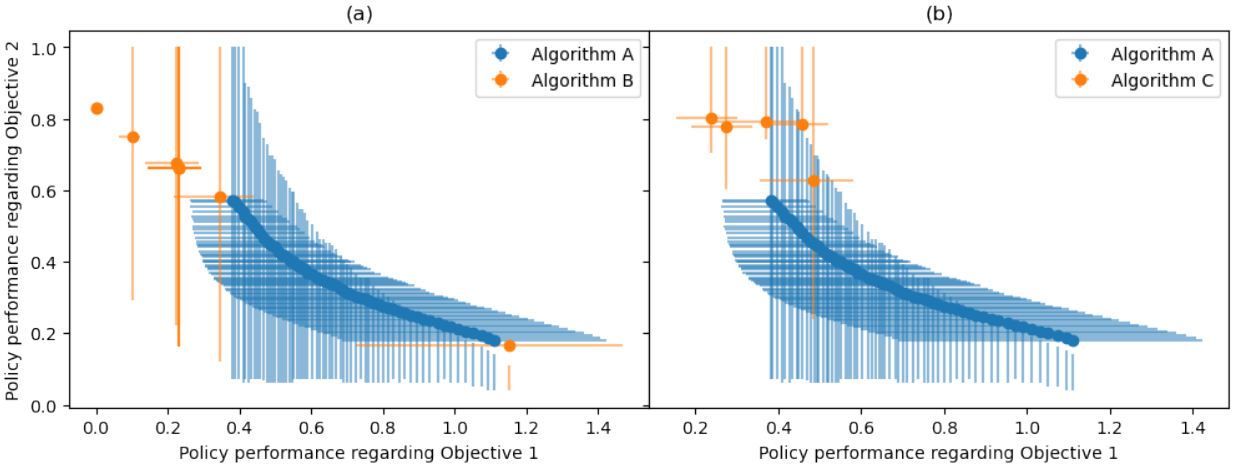


Figure 3.6: Policy trade-off distributions and their performance variances. Each point refers to the average performance of a policy of an algorithm over the parameter uncertainty space concerning Objectives 1 and 2, and each error bar connected with the point refers to the corresponding interquartile range concerning Objective 1 or 2.

Other Robustness Metrics

Maximin, Starr's domain criterion, and 10^{th} percentile performance reflect the policy robustness of the algorithms in different aspects, and they are not necessarily consistent with each other [27]. Therefore, we compared the algorithm performance regarding each of them independently. Generally, algorithms with higher Maximin provided better performance in the worst case; algorithms with higher Starr's domain criterion met the pre-defined threshold in more scenarios; algorithms with higher 10^{th} percentile performance provided better performance in relatively dire scenarios.

3.4.3 Conclusion

We drew conclusions about these comparisons based on these aspects altogether. When comparing two algorithms A and B, if A outperformed B in all these aspects, we argue that A dominated B. Otherwise, A and B were complementary, so joint use of them could improve the performance of the DMDU approach. According to our research aims, we mainly focused on comparisons between MORRL and MOREA in our experiments.

Chapter 4

Cartpole Problem

4.1 Introduction

4.1.1 Background

Cartpole problem describes an inverted pendulum system where a pole is attached to a cart by an un-actuated joint, and the cart moves along a two-dimensional frictionless track (Figure 4.1) [100]. The pole starts upright, but it is affected by gravity continuously. The objective of this problem is to prevent the pole from falling over as long as possible. The solution policy can achieve this by applying a sequence of left, right or no forces to the cart to control its movement.

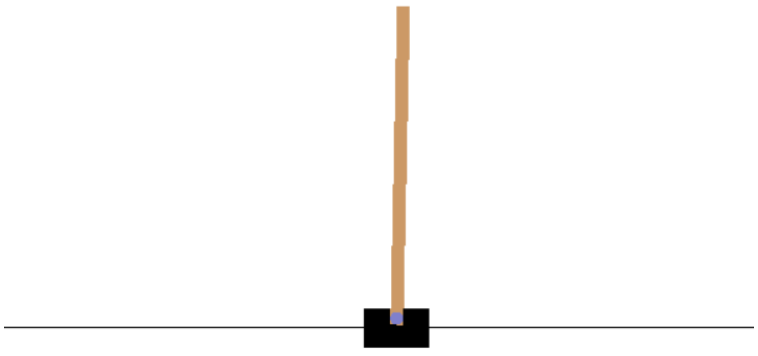


Figure 4.1: Cartpole problem [88].

4.1.2 Introduction

This chapter presents our experiments in three variants of the Cartpole problem, including a single-objective robust Cartpole problem, a single-objective problem with fixed initial state, and a complex-objective problem with fixed initial state. Existing studies of Robust RL generally involved the first one only, because they did not consider objective uncertainty or non-RL algorithms that poorly support the real-time control required to handle random initial states, such as EAs [18, 65, 77]. Here, we start with the first problem to connect our experiments to those previous ones, and reflect the difference between dynamic and static policies. Then, we use the second problem to investigate the algorithms' robustness to pure parameter uncertainty by removing the factor of random initial state that is hardly seen in the deep uncertainty problem. Eventually, we take the third problem as a transition from single-objective problems to multi-objective ones, and investigate how the algorithm performance changes when the objective becomes complex.

4.1.3 Motivation

Cartpole problem is a widely applied benchmark problem in the fields of RL and Robust RL [18, 65, 77]. Using it as our first experimental environment has the following four benefits. First, Cartpole problem is simple and its source model from OpenAI Gym [88] provides a visualization of agent-environment interactions, which helped us gain more insights into the performance of the algorithms and connect it with their characteristics. Second, Cartpole problem puts a high requirement on the real-time policy adaptability, where a good agent must be able to make decisions according to the current model state. This nature reflected the main difference between the dynamic policy and the static policy provided by RL and EA respectively. Third, using a previously studied problem helped fit our works into the current research context of Robust RL by complementing previous findings with ours. For example, both studies of DQN-URBE and EPOpt have independently tested their algorithms in variants of the robust Cartpole problem [18, 76], and we reproduced their experiments to compare these two algorithms, which has not been studied before. Lastly, Cartpole problem also served as a representative of RL domain problems. Comparing it with the deep uncertainty problems used in this project revealed the difference between robust planning research in the RL area and DMDU field.

4.2 Single-Objective Robust Cartpole Problem

4.2.1 Problem Conceptualization

Problem Formalization

We formalized this problem based on its source model from OpenAI Gym, which contains the following parameters (Table 4.1). The time horizon of each episode is 4 seconds, or until the

Parameters	Descriptions	Nominal Values
gravity	Gravitational acceleration constant	9.8 (m/s^2)
masscart	Mass of the cart	2.0 (kg)
masspole	Mass of the pole	0.1 (kg)
length	Length of the pole	0.75 (m)
force_mag	Force magnitude	10 (N)
angle_threshold	Maximum angle between the pole and the vertical axis before the pole falls over	12 $(degree)$

Table 4.1: Parameters and their nominal values in the Cartpole problem [18].

pole falls over (when the angle between the pole and the vertical axis exceeds 12 degrees). The time-step between two consecutive model states is 0.02 seconds. In other words, each episode contains at most 200 steps. The model states are described by four variables with continuous values, $(x, \dot{x}, \theta, \dot{\theta})$, which represent the cart position, the cart velocity, the pole angle, and the pole angular velocity respectively. The model starts from random initial states, where each variable value is normally sampled from $[-0.05, 0.05]$. At each step, the agent can take one of the three actions, which are applying a left force (force = -force_mag), a right force (force = force_mag), or no force (force = 0) to the cart. The transition function between states is also coded in the source model, referring to the physical principles described in [101].

To formulate a single-objective robust Cartpole problem, we chose the pole length as the only uncertain parameter. We set its training value range to $[0.5, 2.5]$ and its testing value range to $[0.5, 10]$. This problem did not involve objective or model uncertainty. The only objective was to keep the pole upright for as long as possible. We used 'the number of steps the pole remains upright' as the objective indicator, which we referred to as *reliability*. To guide the exploration of the algorithms, we defined the fitness function and reward function for EA and RL as follows: at each step, the agent receives a reward of 1 if the pole has not fallen over, otherwise, it receives a -500 and the episode terminates immediately. These settings referred to the experiment in [18].

Model Implementation

The source model of the Cartpole problem is already implemented as an OpenAI Gym Environment. Therefore, we extended it following the processes described in Section 3.1.2 to construct our models to describe the parameter uncertainty and support the exploration processes of our algorithms.

4.2.2 Experimental Setup

In this experiment, we applied five algorithms for exploration to identify optimal robust policies for this problem for comparison. They were ε-NSGA-II, Robust DQN, DQN-URBE, EPOpt and Deterministic DQN. Here, Deterministic DQN refers to the standard model-free DQN. It handled the uncertainty by following the 'predict-then-act' approach and deterministically planning for the nominal scenario. We did not consider Borg in this experiment because its implementation provided in EMA Workbench [43] does not support discrete action space. Table 4.2 details the experimental setups.

4.2.3 Results

Robustness Evaluation

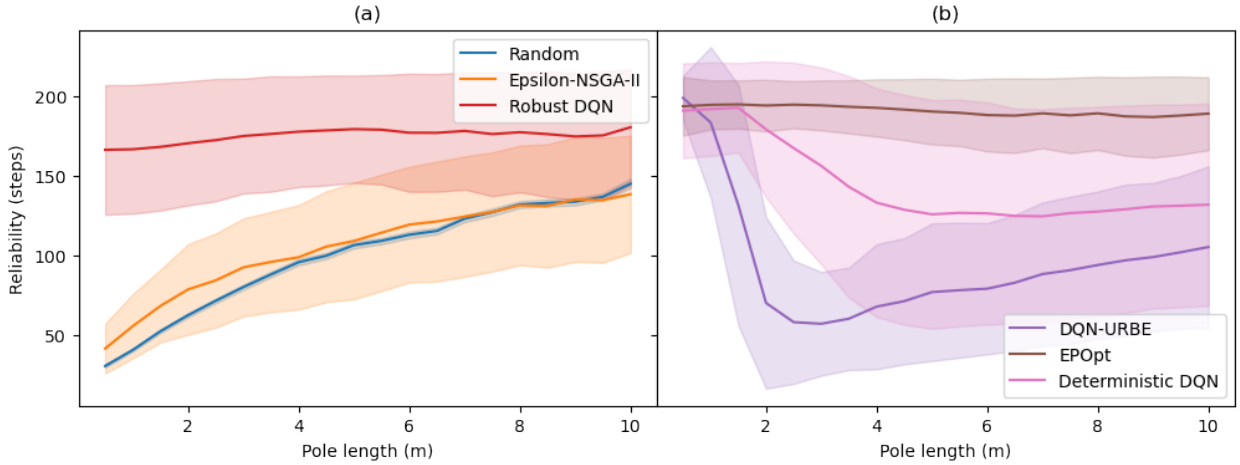


Figure 4.2: Average policy performance distributions in the single-objective robust Cartpole problem. Each line refers to the performance distribution and its corresponding standard deviation of an algorithm over the parameter uncertainty space, which averages the policy performance of the algorithm generated using the five random seeds.

Figure 4.2 shows the average performance distributions of the algorithms over the parameter uncertainty space and their corresponding variances. In this experiment, these variances were caused

Setups	Descriptions	Values
Training scenario set size	Size of the training scenario set (Section 3.1.4)	100 (Figure 8.1)
Random seeds	Random seeds used to control the randomness of the exploration processes	α, β, γ, δ, and ζ (Table 8.1)
Number of CPUs	Number of CPUs used to support the exploration processes of the algorithms	75
ε-NSGA-II	Number of function evaluations taken in the exploration processes of ε-NSGA-II	100,000 function evaluations
Robust DQN	Number of iterations taken in the exploration processes of Robust DQN	8,000 iterations
DQN-URBE	Number of episodes taken in the exploration processes of DQN-URBE	4,000 episodes
EPOpt	Number of iterations taken in the exploration processes of EPOpt	4,000 iterations
Deterministic DQN	Number of iterations taken in the exploration processes of Deterministic DQN	8,000 iterations

Table 4.2: Experimental setups for the single-objective robust Cartpole problem. The table specifies five random seeds, which means that we ran each algorithm five times, each time using a different seed from the table to control the randomness of the exploration process. Moreover, we allocated a sufficient computational budget for each algorithm to converge its exploration process, in order to yield a fair comparison.

by the performance difference of the policies generated by using different random seeds and the randomness of the initial model state. Here, *Random* refers to the policies randomly generated using the five seeds (Table 4.2). Their *reliability* greatly increased with the pole length, and averaged only 100 over the uncertainty space. The average performance of ε-NSGA-II was similarly distributed, but its variance was greater. In contrast, the Robust DQN policies performed much better. For poles of any length, they robustly provided average *reliability* of about 175, but their performance variance was also mediocre. DQN-URBE outperformed the other algorithms (providing average *reliability* of around 190) only when the actual pole length was close to the nominal one (0.75). Otherwise, its overall performance was greatly compromised, which was even worse than *Random*. In contrast, EPOpt robustly achieved the highest average performance and the smallest variance over the entire uncertainty space in this experiment. Lastly, the performance distribution of Deterministic DQN shared a similar trend with DQN-URBE, except that its average *reliability* was relatively less compromised when the actual and nominal pole lengths were different. Deterministic DQN also had the largest performance variance compared with all other algorithms.

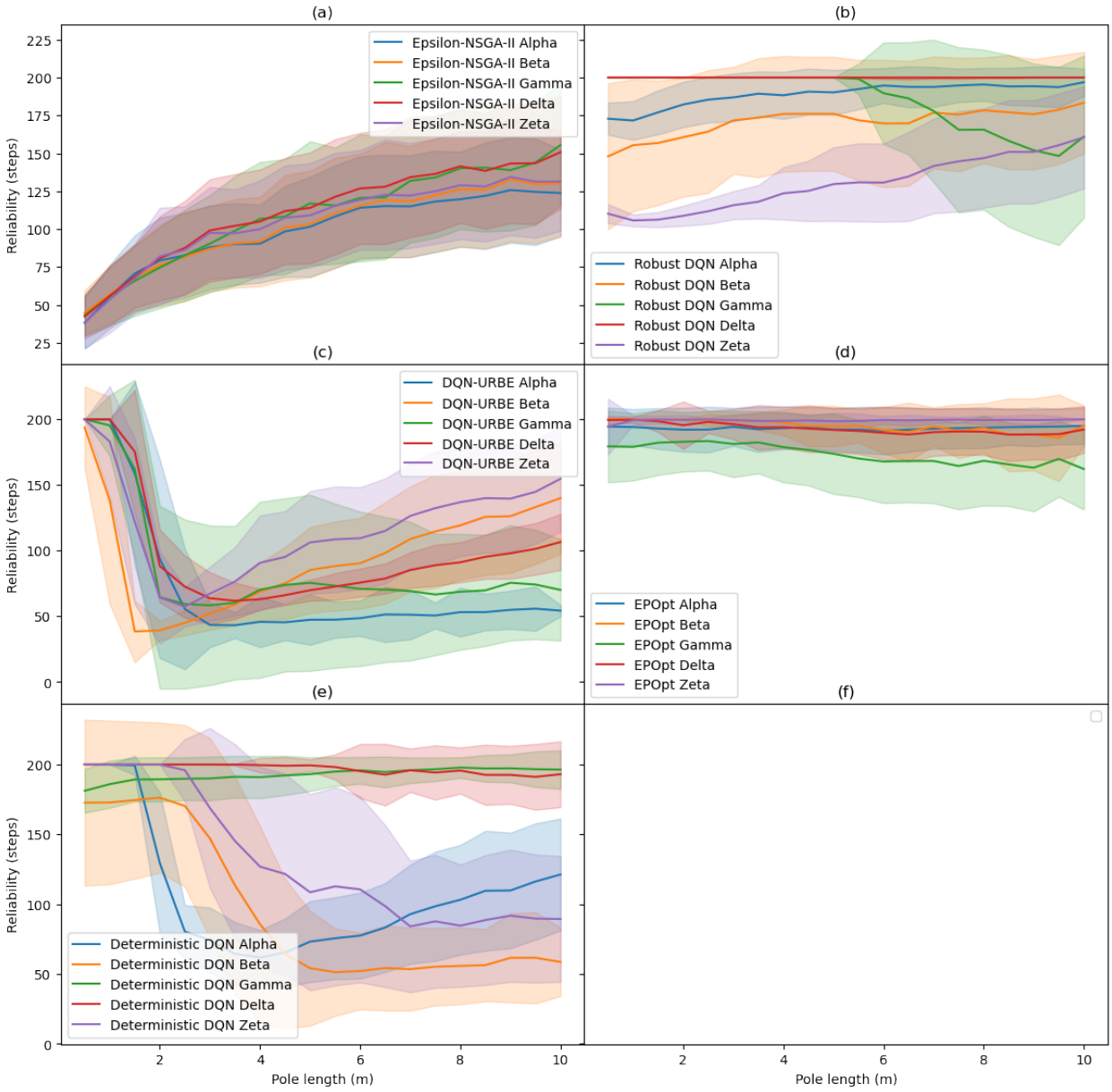


Figure 4.3: Individual policy performance distributions in the single-objective robust Cartpole problem. Each subplot in this figure corresponds to one algorithm, where each line refers to the performance distribution and its corresponding standard deviation of a policy over the parameter uncertainty space. These policies were generated by the algorithm using different random seeds. Therefore, for each algorithm, the performance difference of its different policies reflects the impact of the exploration randomness on its average performance, while these performance variances reflect the impact of the random initial state on its average performance.

Figure 4.3 shows policy performance generated using different random seeds individually for each algorithm. In this way, we separated the impact of the exploration randomness from that of the random initial state on the average performance of the algorithms (Figure 4.2). It can be

observed that different ε-NSGA-II and EPOpt policies had similar performance distributions, so their overall performance variances were mainly caused by the random initial state (Figure 4.2 (a,d)). Meanwhile, EPOpt also had the smallest individual performance variance on average. In contrast, the selection of random seeds had a great impact on the policy performance of Robust and especially Deterministic DQN (Figure 4.2 (b,e)). Some of their policies provided nearly the maximum *reliability* robustly in any case (for example, the policies learned by using random seed δ), while the others performed very unstably when dealing with various pole lengths or initial states. Finally, different DQN-URBE policies performed similarly only when the actual pole length was close to 0.75 (Figure 4.2 (c)). Otherwise, their performance was different, but all greatly compromised in terms of both *reliability* and variance.

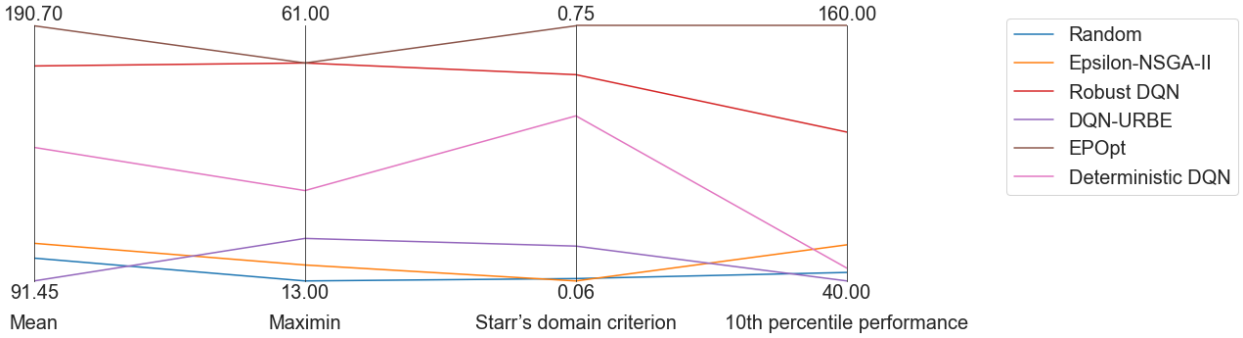
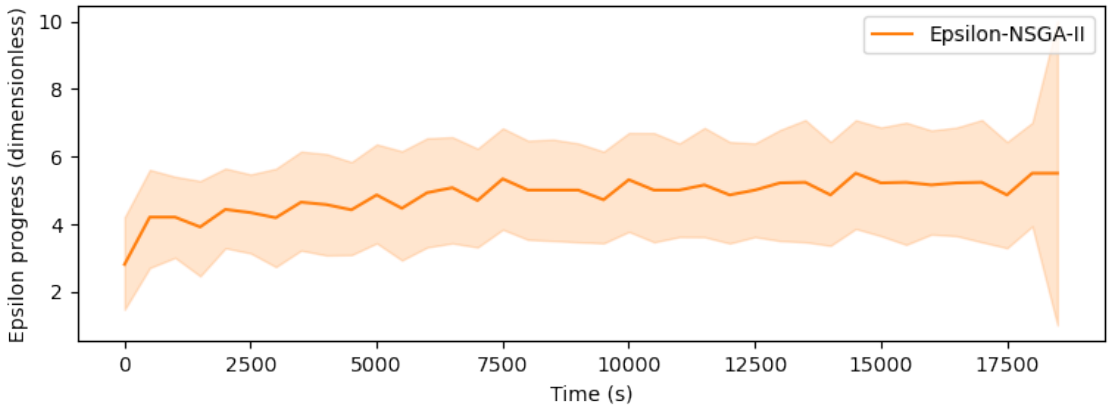


Figure 4.4: Parallel coordinate figure that shows different robustness metrics for *reliability* of each algorithm in the single-objective robust Cartpole problem. In this figure, we used *reliability* = 200, which was also the maximum *reliability* that could be provided in an episode, as the pre-defined threshold for Starr's domain criterion. Moreover, Mean is not a robustness metric and used for reference only.

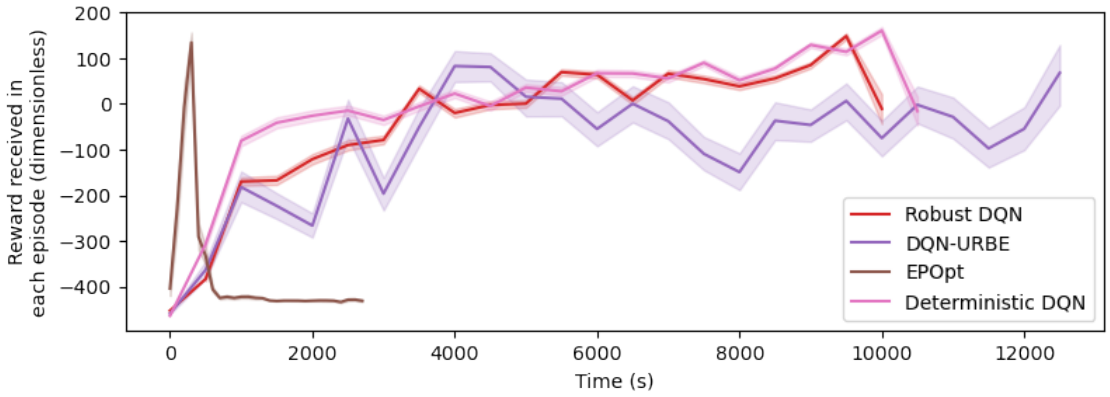
In terms of the other three robustness metrics, EPOpt also obviously outperformed the other algorithms (Figure 4.4). Although Robust DQN provided similar worst-case policy performance, it achieved the maximum *reliability* in fewer scenarios and had lower 10^{th} percentile performance. This was because the performance variance of Robust DQN was greater than EPOpt. In addition, the performance of ε-NSGA-II, DQN-URBE and Deterministic DQN regarding these metrics was consistent with their average performance distributions, and was much worse than that of Robust DQN and EPOpt.

Efficiency Evaluation

In this experiment, we observed that ε-NSGA-II converged very fast with only a few ε-progresses made during its exploration process (Figure 4.5 (a)). In contrast, all the RL algorithms learned to obviously improve their policy performance (Figure 4.5 (b)). Robust, Deterministic DQNs and DQN-URBE generally took a similar time to converge, but the policy performance of DQN-URBE



(a)



(b)

Figure 4.5: ε-progress curve for ε-NSGA-II (a), learning curves for the RL algorithms (b), and their 95% confidence intervals in the single-objective robust Cartpole problem. Each curve of the algorithm is averaged over its multiple runs with different random seeds. Each point on the curve represents the value averaged over 500 seconds.

varied the most during this process. Moreover, the learning curve trend of EPOpt was distinct from the others, which rose the fastest at the beginning, and then sharply dropped after reaching the peak. According to the implementation of EPOpt (Section 3.2.4), its resulting policies were identified at the peak, which only took around 500 seconds. This observation indicated that EPOpt has identified its resulting policies before the other algorithms converged in this experiment.

4.2.4 Discussion

Based on our experimental results and interpretation methods (Section 3.4), we argue that EPOpt was the best algorithm for this single-objective robust Cartpole problem, and it was not complemented by the others. EPOpt not only achieved the highest average policy performance, but also provided the highest robustness to the parameter uncertainty, model misspecification and random

initial state in the robustness aspects reflected by the four robustness metrics. Meanwhile, it was also the most efficient algorithm.

Robustness

Algorithms	Average performance	Robustness	Stability
ε-NSGA-II	Reliability = 106	Parameter uncertainty (*NO*) Model misspecification (*NO*) Random initial state (*NO*)	Insensitive to the exploration randomness
Robust DQN	Reliability = 175	Parameter uncertainty (*YES*) Model misspecification (*YES*) Random initial state (-)	Sensitive to the exploration randomness
DQN-URBE	Reliability = 91.5	Parameter uncertainty (*NO*) Model misspecification (*NO*) Random initial state (*NO*)	Sensitive to the exploration randomness
EPOpt	Reliability = 188	Parameter uncertainty (*YES*) Model misspecification (*YES*) Random initial state (*YES*)	Insensitive to the exploration randomness
Deterministic DQN	Reliability = 143	Parameter uncertainty (-) Model misspecification (-) Random initial state (-)	Most sensitive to the exploration randomness

Table 4.3: Summary of the robustness evaluation results in the single-objective robust Cartpole problem. In this table, *YES* indicates that we argue the algorithm provides robustness in the corresponding aspect, *NO* indicates the algorithm does not provide robustness, and - means the algorithm performance depends on the use of random seeds.

The performance of baseline *Random* reflected that this problem became easier as the pole length increased. This was because applying the same force (including the force applied by the agent and the gravitational force) to a Cartpole system with the same state but a longer pole caused a smaller angular acceleration. Therefore, taking the same sequence of random actions could prevent the pole from falling over for a longer time.

In this experiment, ε-NSGA-II performed no better than *Random*. Its policy performance mainly depended on the problem difficulty, so we argue that ε-NSGA-II failed to provide robustness. This was mainly because its static policies provided no real-time control required to handle the random initial state. Applying the same sequence of actions in models with only different initial states might still lead the agent to very different final states. Hence, the agent must be able to adapt its actions to the actual model state in real-time (like the dynamic policy) to guarantee

good performance. This shortcoming also contributed to the majority performance variance of ε-NSGA-II (Figure 4.3 (a)). Finally, although the performance of ε-NSGA-II was insensitive to the exploration randomness, we think it was mainly because ε-NSGA-II failed to make obvious progress on improving its policy performance during the exploration process.

Since the Robust DQN policies generally provided high *reliability* in most evaluation scenarios, we argue it provided robustness to the parameter uncertainty and model misspecification. Its success empirically proved the intuition employed by some previous studies of Robust RL that model-free RL can learn a robust policy from sufficient interactions with a simulator that produces the system performance under uncertainty. The main shortcoming of Robust DQN was its sensitivity to the exploration randomness. By using appropriate random seeds, Robust DQN has learned a perfect policy, outperforming all the EPOpt ones. But the others performed less satisfactorily, mainly regarding their average performance and robustness to the random initial state (Figure 4.3 (b)). This shortcoming caused the overall performance of Robust DQN to be lower than EPOpt in all aspects. This observation might indicate that (1) TRPO used by EPOpt was less sensitive to randomness compared with model-free DQN used by Robust DQN in this problem; (2) Robust DQN should be able to outperform EPOpt and become the best algorithm for this problem with appropriate use of random seeds; (3) Randomness plays a vital role in the effectiveness of model-free DQN, so decision-makers should always choose random seeds carefully in future applications of Robust DQN. Although this project did not further investigate these indications, they still inspired directions for future research.

All the DQN-URBE policies only performed well when the actual pole length was close to its nominal value, even though the problem was more challenging in these cases. This showed a main disadvantage of DQN-URBE that it is very likely to overfit the nominal scenario, where its policies provide no robustness to parameter uncertainty or model misspecification. In this case, the performance of DQN-URBE might also be sensitive to the exploration randomness (Figure 4.3 (c)). This disadvantage is caused by the nature of DQN-URBE that it samples trajectories and computes rewards solely based on the nominal scenario that it assumes as the true one, while other possible scenarios are only used to collect possible observations. However, this finding contradicted the previous experiments on DQN-URBE, where DQN-URBE was shown free from this overfitting issue and provided high robustness [18]. This inconsistency might be caused by the difference in our experimental models, such as the choice of the action space, training scenario set size, scenario sampling method, random seed, and evaluation approach. Further investigations are needed to explain this inconsistency.

Deterministic DQN was the most sensitive to the exploration randomness. By using different random seeds, it could learn policies that provided high robustness or overfitted the scenario it predicted (Figure 4.3 (e)). Therefore, it generally failed to provide robustness, but still outperformed

DQN-URBE in this experiment. However, contradicting the observation obtained from previous studies that the 'predict-then-act' approach often failed if the reality differed from its prediction [18, 25, 76], we observed Deterministic DQN was able to learn a policy that provided high robustness to the entire uncertainty space and even random initial state. This robustness was mainly achieved through the real-time adaptability of the dynamic policy and the application of a strategy applicable in any case. We will discuss this in detail in a latter section.

Efficiency

Although ε-NSGA-II was found to converge quickly in this experiment, that was because it failed to make obvious improvements to its randomly generated initial policies, even though the computational budgets were sufficient. This observation also showed that static policies could not do better than random ones when dealing with random initial states in this problem, because of the lack of real-time adaptability. We argue that Robust, Deterministic DQNs and DQN-URBE had similar efficiency because their convergence time was similar. However, DQN-URBE had the greatest learning curve variance, which meant its efficiency was also sensitive to the exploration randomness. EPOpt was the most efficient algorithm for this problem, because it took the shortest time to identify its final policy. The learning curve trend of EPOpt showed that by using TRPO, EPOpt improved its policy performance to the best in the first few hundred episodes of its learning process at the fastest speed. After that, although the agent was encouraged for more explorations, no better policy was found. This phenomenon might indicate that TRPO was more efficient in solving such simple problems compared with DQN, which was used by Robust, Deterministic DQNs and DQN-URBE.

4.3 Single-Objective Robust Cartpole Problem With Fixed Initial State

In the last experiment, the uncertainty of the model state comes from two sources, the uncertain pole length and the random initial state. The latter is rare in practical deep uncertainty problems, where decision-makers often treat the present as a fixed initial state and plan for the future. This randomness also puts a high requirement on the real-time policy adaptability, which prevents ε-NSGA-II from performing well. Therefore, we designed the second experiment to compare our algorithms under pure parameter uncertainty by fixing the initial model state.

4.3.1 Problem Conceptualization And Experimental Setup

In this experiment, we used the same models and experimental setups as the previous one, except that these models always started from a fixed initial state (Table 4.4).

State variables	Descriptions	Initial values
x	Cart position	$-0.042\ (m)$
$\dot{x}$	Cart velocity	$0.045\ (m/s)$
θ	Pole angle	$-0.025\ (degrees)$
$\dot{\theta}$	Pole angular velocity	$0.020\ (degrees/s)$

Table 4.4: The fixed initial state for all models in the single-objective robust Cartpole problem with fixed initial state. This is an arbitrary assumption.

4.3.2 Results

Robustness Evaluation

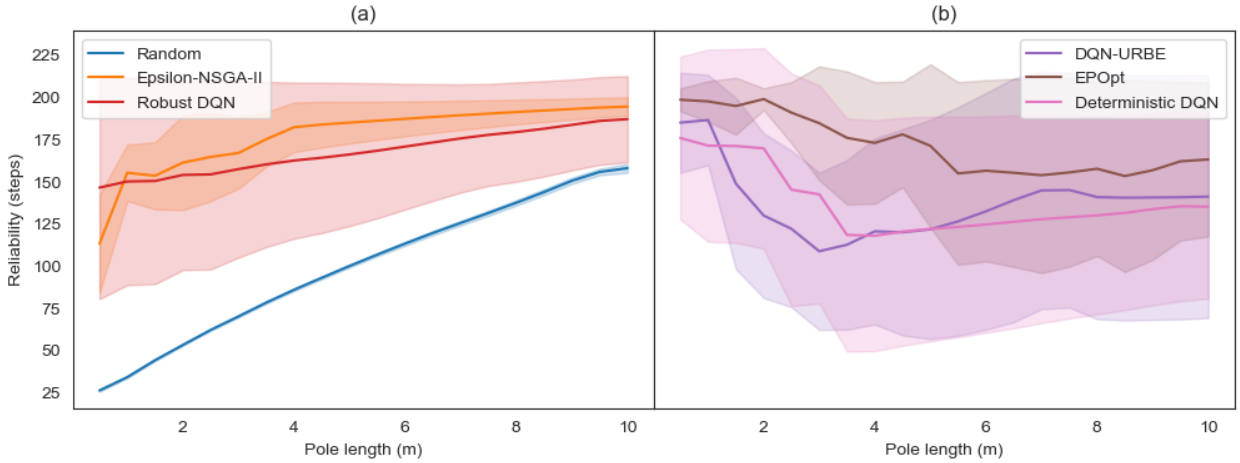


Figure 4.6: Average policy performance distributions in the single-objective robust Cartpole problem with fixed initial state. Each line refers to the performance distribution and its corresponding standard deviation of an algorithm over the parameter uncertainty space, which averages the policy performance of the algorithm generated using the five random seeds.

By comparing these distributions (Figure 4.6) with those in the last experiment (Figure 4.2), we found that the average *reliability* provided by *Random* was lower for short poles but higher for long poles. The performance of ε-NSGA-II was significantly improved regarding both average *reliability* and performance variance for poles of any length. Although its static policies still behaved less satisfactorily for short poles, they averagely outperformed all the RL policies in both

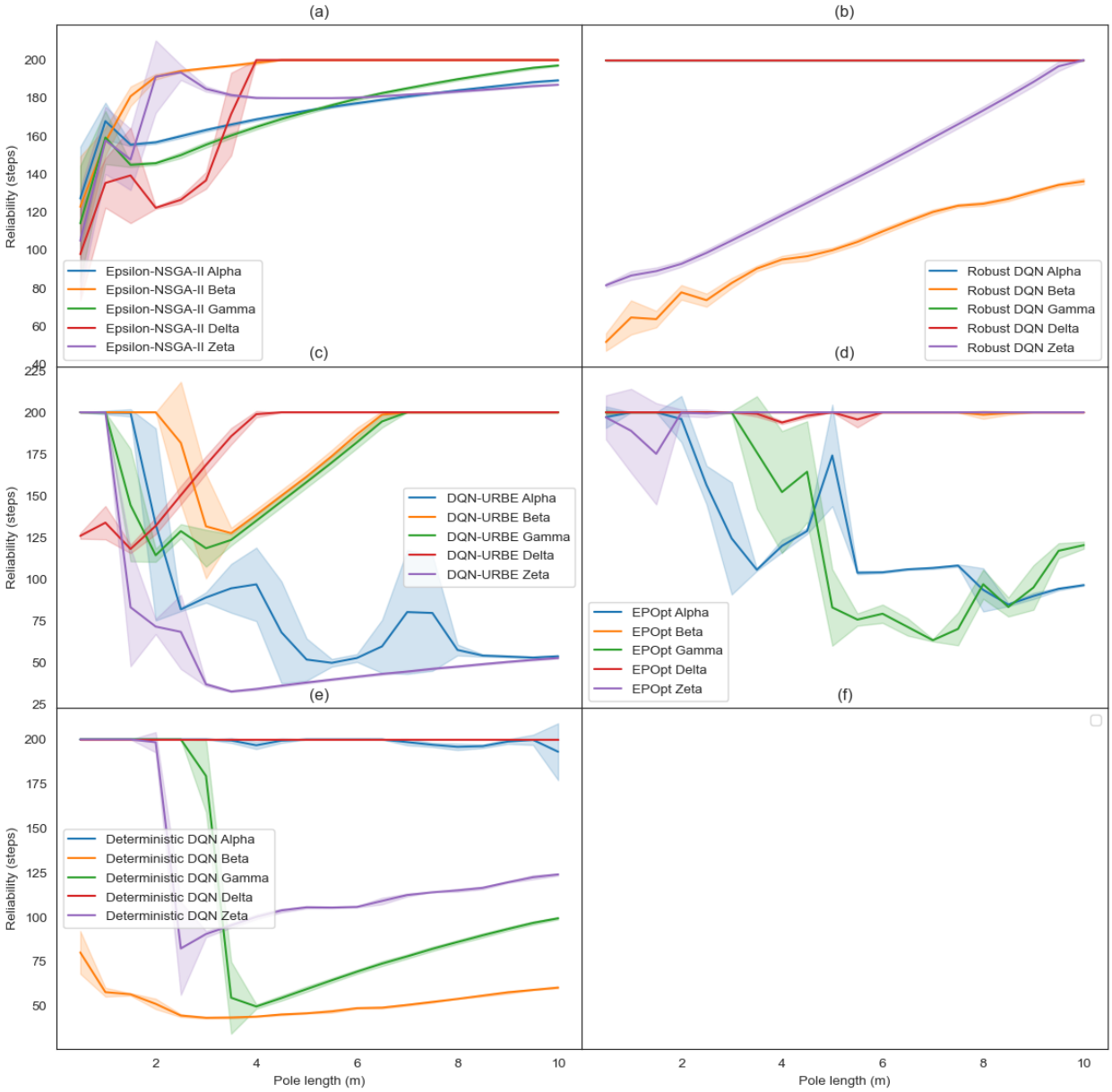


Figure 4.7: Individual policy performance distributions in the single-objective robust Cartpole problem with fixed initial state. Each subplot in this figure corresponds to one algorithm, where each line refers to the performance distribution and its corresponding standard deviation of a policy over the parameter uncertainty space. These policies were generated by the algorithm using different random seeds. Therefore, for each algorithm, the performance difference of its different policies reflects the impact of the exploration randomness on its average performance.

aspects when the actual pole length exceeded 4 meters. In contrast, the performance of the RL algorithms was compromised. Robust and Deterministic DQNs generally had lower average *reliability* and higher performance variances in those short-pole scenarios, while similar changes occurred to EPOpt in the long-pole scenarios. Additionally, the overall performance variance of

DQN-URBE also increased. Although its average *reliability* for long poles was improved, that was still the second-to-last in this experiment.

Individual policy performance distributions provided more insights into these changes (Figure 4.7). Compared with the previous results (Figure 4.3), the most obvious difference was that the variances of these distributions were greatly reduced, which meant the algorithm performance was more similar in scenarios with similar pole lengths. This was because this experiment excluded the randomness of the initial model state and the parameter uncertainty became the only uncertain factor. Individual performance distributions of ε-NSGA-II were generally consistent with its average distribution (Figure 4.7 (a)). The performance difference of its different policies increased compared with the last experiment, which meant ε-NSGA-II became more sensitive to the exploration randomness. Nevertheless, the RL algorithms also became more sensitive and still more sensitive than ε-NSGA-II. This sensitivity of Robust, Deterministic DQNs and EPOpt was also the main reason for the compromisation of their overall performance (Figure 4.7 (b,d,e)). Although they were more likely to learn near-perfect policies by using appropriate random seeds, their other policies performed worse than those they learned with inappropriate seeds in the last experiment. For example, some Robust and Deterministic DQN policies performed even worse than *Random*, and some EPOpt policies had the overfitting issue now. In contrast, DQN-URBE slightly alleviated its overfitting issue, and now three of its five policies could provide *reliability* of 200 in the long-pole scenarios (Figure 4.7 (c)). Nevertheless, it still did not identify a policy robust to the entire parameter uncertainty space like the other RL algorithms.

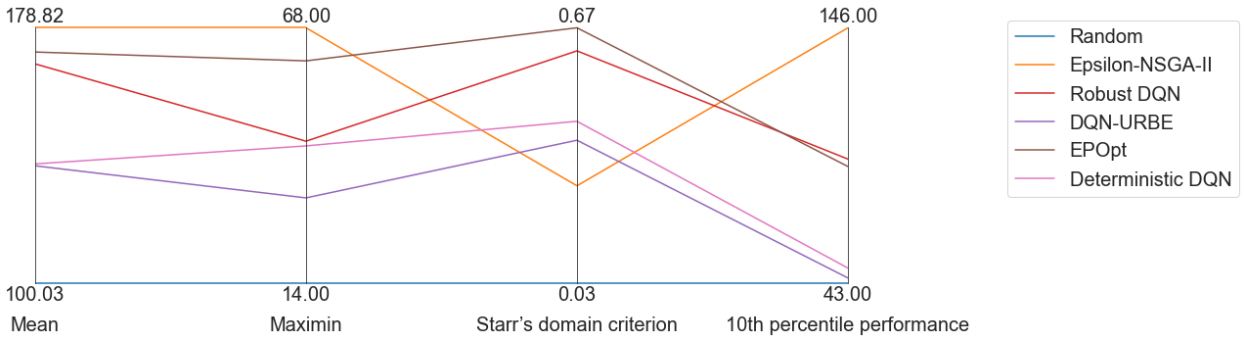


Figure 4.8: Parallel coordinate figure that shows different robustness metrics for *reliability* of each algorithm in the single-objective robust Cartpole problem with fixed initial state. In this figure, we used *reliability* = 200, which was also the maximum *reliability* that could be provided in an episode, as the pre-defined threshold for Starr's domain criterion. Moreover, Mean is not a robustness metric and used for reference only.

Similar changes also took place in the other robustness metrics for *reliability* of the algorithms (Figure 4.8). In this experiment, ε-NSGA-II outperformed the RL algorithms in terms of Maximin and 10^{th} percentile performance, but underperformed them in terms of Starr's domain criterion.

This meant its policy performance was more robust in relatively bad scenarios, but less robust in relatively good ones. Besides, the overall rankings of the RL algorithms concerning these metrics were EPOpt, Robust DQN, Deterministic DQN and DQN-URBE, where the former outperformed the latter in all aspects except that Robust DQN had slightly better 10^{th} percentile performance than EPOpt.

Efficiency Evaluation

In Table 4.5, we observed that the quotients of the RL algorithms were significantly greater than that of ε-NSGA-II. Since their models were all extended from the same underlying MDP, we assumed they spent similar time exploring a scenario. Thus, this observation indicated that the RL algorithms spent much longer on policy update than scenario exploration compared with ε-NSGA-II.

ε-NSGA-II	Robust DQN	DQN-URBE	EPOpt	Deterministic DQN
0.002 (s)	0.04 (s)	0.029 (s)	0.008 (s)	0.041 (s)

Table 4.5: For each algorithm, the quotient of its total running time and the number of times T it explored its current policy performance in a scenario, during its exploration process. For example, in this problem, $T = 1$ in each episode for Robust, Deterministic DQNs and EPOpt, which explored one scenario per episode; $T = 100$ in each function evaluation for ε-NSGA-II or each episode for DQN-URBE, which explored the entire training scenario set per evaluation/episode (Section 3.1.2).

4.3.3 Discussion

We argue that ε-NSGA-II and EPOpt were mutually complementary algorithms best for this problem. If the decision-maker placed more emphasis on policy performance in general or dire scenarios, ε-NSGA-II should be applied to provide higher average *reliability* robustly. If the decision-maker was more concerned with performance in opportune scenarios, EPOpt was preferable because its policies were more likely to provide the maximum *reliability*. Moreover, it is also worth noting that Robust, Deterministic DQNs and EPOpt were able to identify policies providing optimal performance and robustness over the entire uncertainty space, with the use of appropriate random seeds,

Algorithms	Average performance	Robustness	Stability
ε-NSGA-II	Reliability = 179	Parameter uncertainty (*NO*) Model misspecification (*YES*)	Least sensitive to the exploration randomness
Robust DQN	Reliability = 168	Parameter uncertainty (-) Model misspecification (*YES*)	Sensitive to the exploration randomness
DQN-URBE	Reliability = 136	Parameter uncertainty (*NO*) Model misspecification (*NO*)	Sensitive to the exploration randomness
EPOpt	Reliability = 171	Parameter uncertainty (-) Model misspecification (-)	Sensitive to the exploration randomness
Deterministic DQN	Reliability = 137	Parameter uncertainty (-) Model misspecification (-)	Sensitive to the exploration randomness

Table 4.6: Summary of the robustness evaluation results in the single-objective robust Cartpole problem with fixed initial state. In this table, *YES* indicates that we argue the algorithm provides robustness in the corresponding aspect, *NO* indicates the algorithm does not provide robustness, and - means the algorithm performance depends on the use of random seeds.

Robustness

Random was the same as in the last experiment. By comparing its performance, we found the initial state we set in this experiment averagely made the problem harder for short poles but easier for long poles.

By fixing the initial model state in this experiment, ε-NSGA-II obviously improved its policy performance through exploration, and identified solutions that worked well in long-pole scenarios (Figure 4.6). However, none of them could handle short poles satisfactorily. This was still because static policies provided no real-time adaptability. In this problem, the angles of shorter poles were more likely to be changed by force, so good policies should adjust their actions more frequently. In this case, no fixed sequence of actions could be well generalized to both short-pole and long-pole scenarios. Practical applications of DMDU approaches overcome this shortcoming of EA by cycling through the planning steps to rework new policies to adapt to the actual future, but this process is inapplicable in the Cartpole problem that has a short time horizon. Therefore, we argue ε-NSGA-II was not robust to the parameter uncertainty. Its policy performance was not compromised outside the training parameter value range, so it had no overfitting issue and provided robustness to the model misspecification (Figure 4.7).

Intuitively, eliminating an uncertain factor should simplify the problem, but the average performance of Robust, Deterministic DQNs and EPOpt was actually compromised. We found it was because they became more sensitive to the exploration randomness, and their bad policies

performed worse than those in the last experiment (Figure 4.7 and 4.3). We explained this by the number of policy updates that occurred during their exploration processes. These updates involved randomness, and those with inappropriate random seeds sometimes negatively affected the current policy performance, which we called *bad updates*. The more bad updates occurred, the worse the final policy performance would be. Since this problem was simpler than the previous one, the RL algorithms actually needed fewer good updates to identify the optimal policies. Training their agents with the same computational budgets actually created more opportunities for the bad updates, leading to worse policy performance with the use of inappropriate random seeds. For the same reason, since ε-NSGA-II made more policy updates during exploration in this experiment, more bad updates might also occur. Hence, the performance difference of its policies increased. Moreover, since policy updates occurred more often in the exploration process of RL compared with ε-NSGA-II (Table 4.5), the RL algorithms were generally more sensitive to the exploration randomness and provided poorer average performance in those relatively bad scenarios (Figure 4.8). This hypothesis still needs further experiments to prove it. Finally, we also noticed that Robust DQN was the only RL algorithm free from the overfitting issue and thereby robust to the model misspecification (Figure 4.7 (b)). We considered this as one of its advantages.

In contrast, the average performance of DQN-URBE was generally improved (Figure 4.7 (c)). This was mainly because it performed the worst due to the overfitting issue in the last experiment, and fixing the initial model state slightly alleviated this issue. This observation at least showed that DQN-URBE did not handle the random initial state well when running under our experimental setups, which made it more likely to overfit the nominal scenario it assumed. However, we need to conduct further experiments to investigate whether this shortcoming can be resolved by using the original experimental setups from [18] and how to explain it by the learning logic of DQN-URBE. Moreover, since DQN-URBE is also built on DQN, we think its sensitivity to the exploration randomness could be explained in the same way.

As mentioned before, no static policy could be well generalized to the entire parameter uncertainty space in this problem. Nevertheless, some RL policies achieved this, relying on their real-time adaptability. Robust, Deterministic DQNs and EPOpt have learned policies that provided nearly the maximum *reliability* for poles of any length. That meant, these policies could adaptively adjust their actions more frequently in short-pole scenarios and less frequently in long-pole scenarios. This observation indicated that the dynamic policy by RL provided higher adaptability than the static policy by EA. When dealing with the same deep uncertainty problem, the former should be able to adapt to more actual scenarios before a new policy needs to be reworked, thereby also improving the overall efficiency of the DMDU approach. This also explained why all the RL algorithms outperformed ε-NSGA-II in terms of Starr's domain criterion (Figure 4.8).

4.4 Complex-Objective Robust Cartpole Problem With Fixed Initial State

In the last experiment of this chapter, we introduced a complex objective to the robust Cartpole problem as a transition from single-objective robust problems to multi-objective ones. This problem both revealed how our algorithms balanced multiple goals, and was simple enough to be expressed by a single reward function and solved by our 'naive' multi-policy MORRL algorithms in one run.

4.4.1 Problem Conceptualization And Experimental Setup

To formulate a complex-objective problem, we reframed the original robust Cartpole problem to include an objective and a hard constraint. The objective was to minimize external interference to the Cartpole system, while the hard constraint was still that the pole could not fall over. These two goals were not adversarial, and we referred to them as *utility* and *reliability*. We used 'the number of steps the agent applies no force to the cart' as the *utility* indicator and the same *reliability* indicator as in the previous experiments. We adopted the same models as in Section 4.3, but used a different fitness and reward function, defined as follows: At each step, the agent receives a reward of 1 if it applies no force. Whenever the pole falls over, the agent receives a -500 and the episode terminates immediately. Finally, we also used the same experimental setups, including the same training scenario set because we found its appropriate size was also 100 in this problem (Figure 8.2).

4.4.2 Experimental Results

Robustness Evaluation

We used the same *Random* as in the last experiment, so its average *reliability* distribution was identical (Figure 4.9). Its average *utility* distribution showed a similar trend, but much lower values. Meanwhile, ε-NSGA-II also had a similar average *reliability* distribution as in the last experiment, but its performance variance increased. It provided constant and mediocre *utility* for poles of almost any length, due to the nature of its static policies. Unlike them, the RL algorithms generally provided similar distributions of *utility* and *reliability*. The overall performance of Robust and Deterministic DQNs obviously increased with the pole length. Their policies performed mediocrely only when the pole was very short, but provided nearly the maximum *reliability* and *utility*, and the minimum variances when the actual pole length exceeded 5 meters. Moreover, Deterministic DQN outperformed Robust DQN in all these aspects, and it was free from the overfitting issue in this

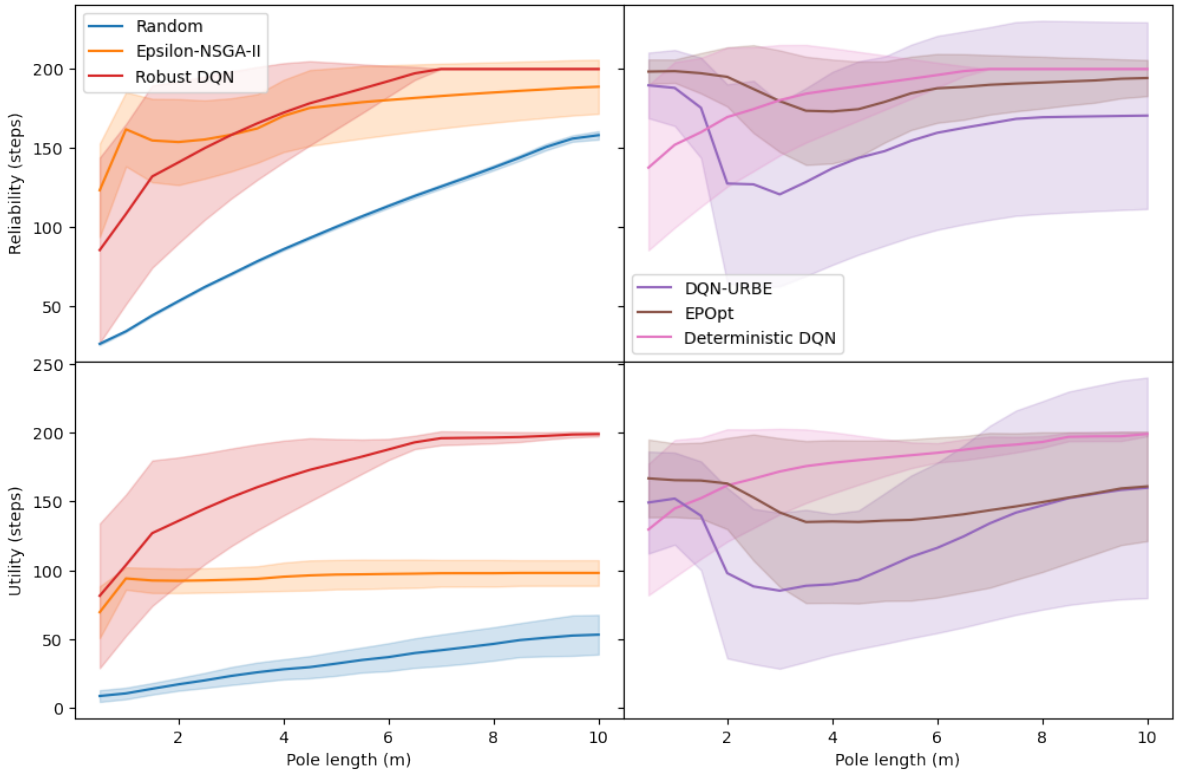


Figure 4.9: Average policy performance distributions regarding *reliability* and *utility* in the complex-objective robust Cartpole problem with fixed initial state. Each line refers to the performance distribution and its corresponding standard deviation of an algorithm over the parameter uncertainty space, which averages the policy performance of the algorithm generated using the five random seeds.

experiment. In contrast, DQN-URBE and EPOpt still performed best in the short-pole scenarios. Although they provided higher average *reliability* and lower *reliability* variances for long poles compared with the last experiment, those were incomparable to Robust and Deterministic DQNs because of their overfitting issue. DQN-URBE and EPOpt also provided a worse balance between *reliability* and *utility*, because they performed relatively worse in *utility* than *reliability*, compared with Robust and Deterministic DQNs. Nevertheless, EPOpt still outperformed DQN-URBE in all aspects.

We obtained similar observations from their individual policy performance distributions (Figure 4.10 and Figure 4.11). It is worth noting that the *reliability* performance of ε-NSGA-II was now more sensitive to the exploration randomness, while its *utility* performance was insensitive to that. On the contrary, the *reliability* performance of the RL algorithms became less sensitive to this randomness and the overfitting issue. But their sensitivity was relatively more severe regarding their *utility* performance. Finally, we also found the best individual policy for this problem was learned by Deterministic DQN by using random seed α (Table 8.1), which provided the maximum

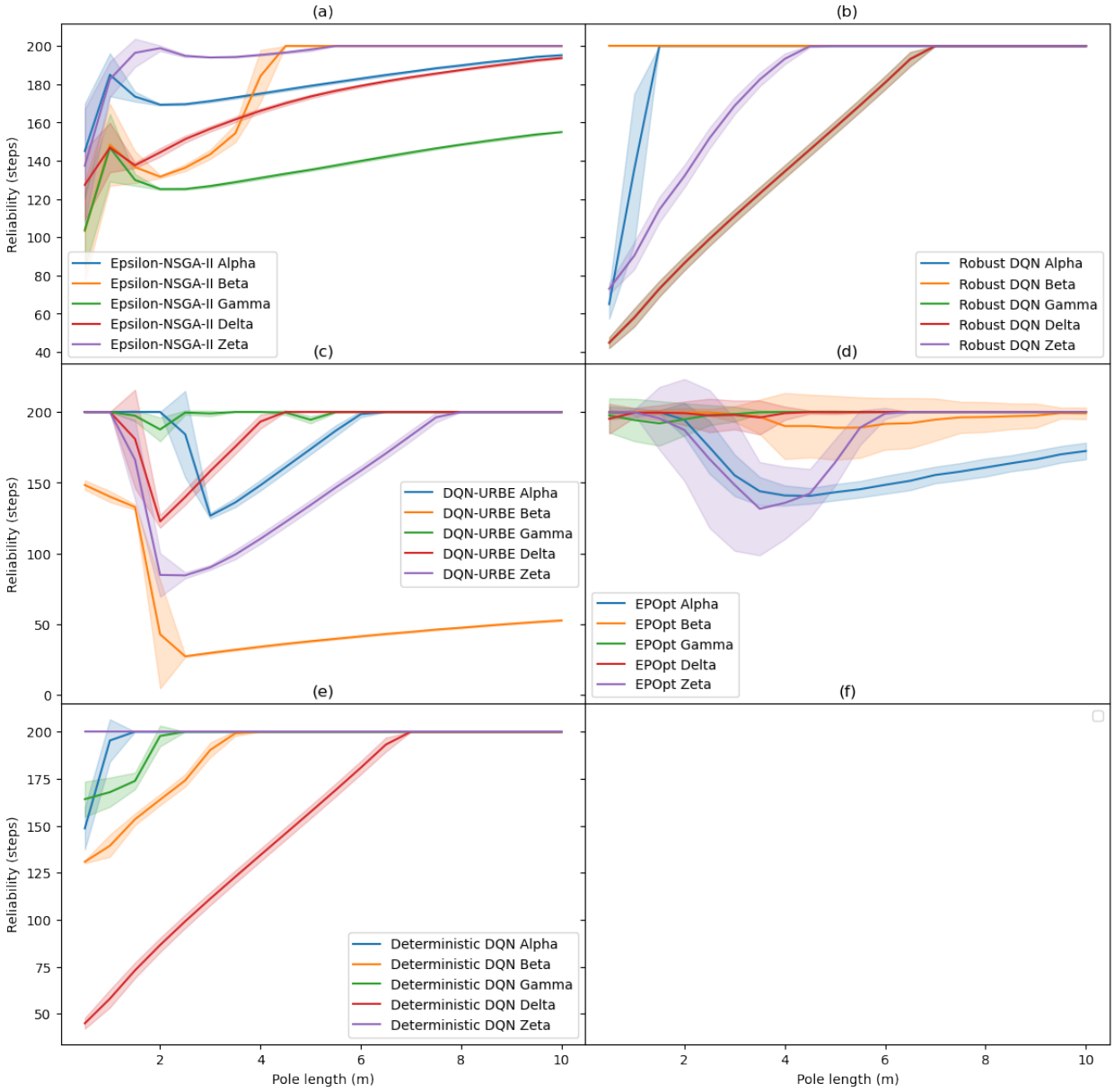


Figure 4.10: Individual policy performance distributions in the complex-objective robust Cart-pole problem with fixed initial state. Each subplot in this figure corresponds to an algorithm, where each line refers to the *reliability* distribution and its corresponding standard deviation of a policy over the parameter uncertainty space. These policies were generated by the algorithm using different random seeds. Therefore, for each algorithm, the performance difference of its different policies reflects the impact of the exploration randomness on its average performance.

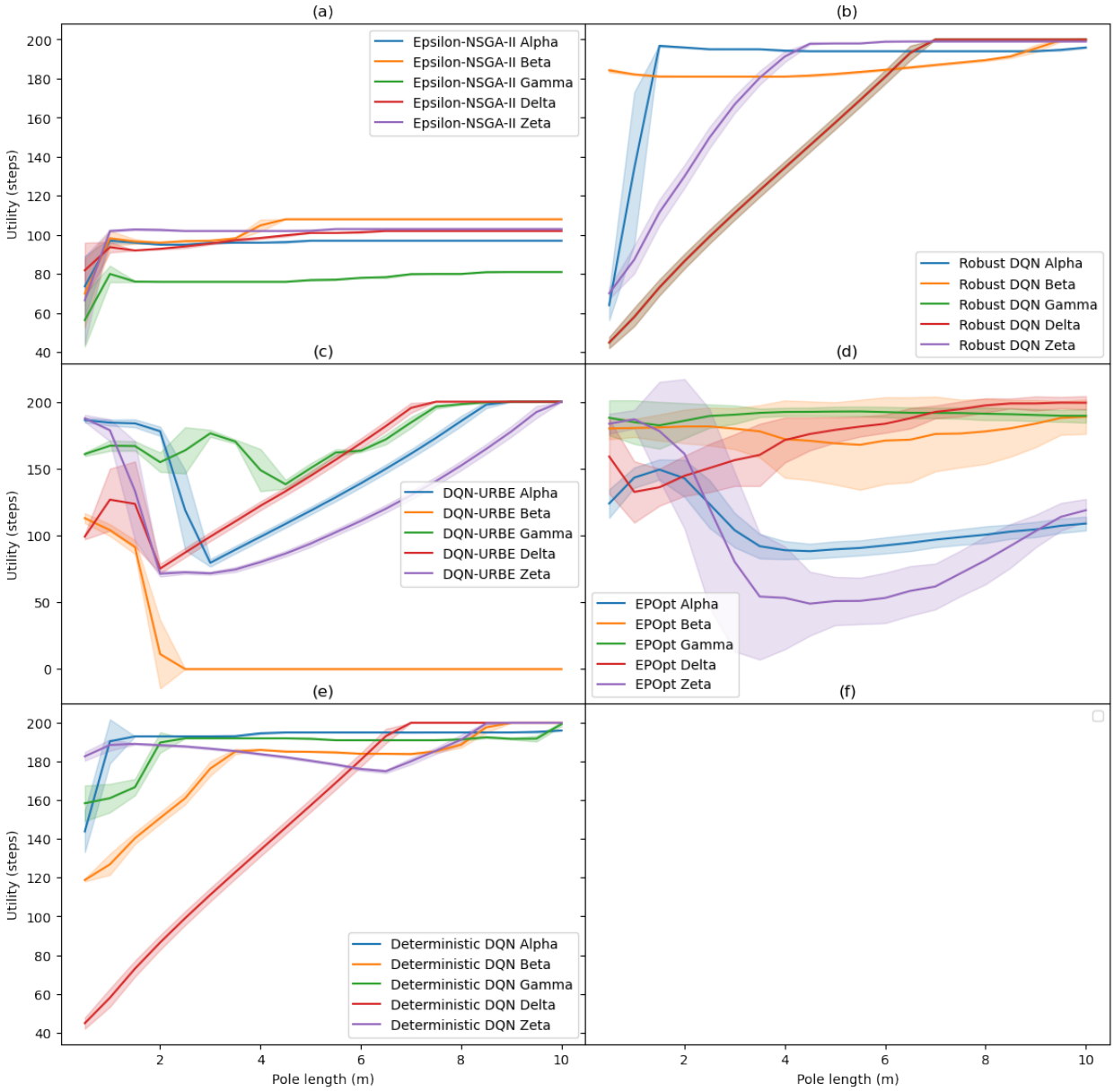
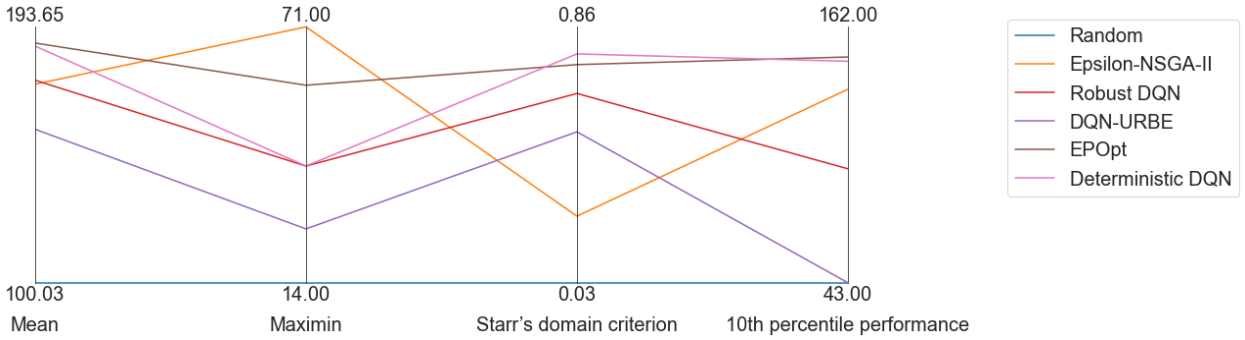
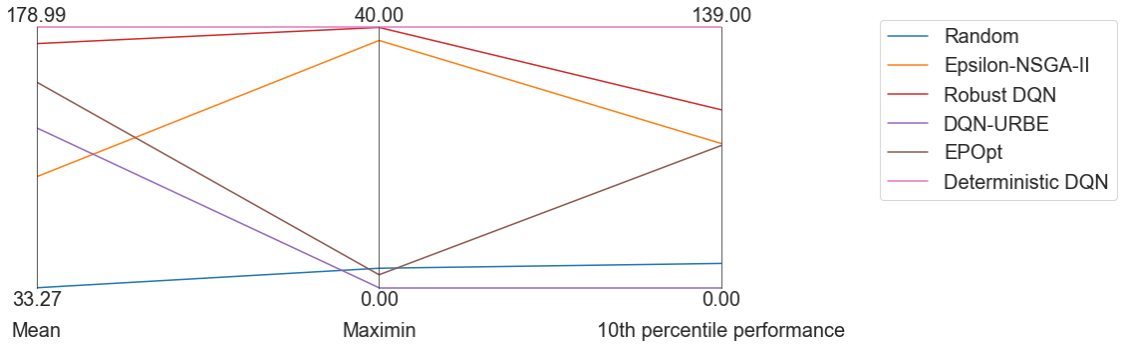


Figure 4.11: Individual policy performance distributions in the complex-objective robust Cart-pole problem with fixed initial state. Each subplot in this figure corresponds to an algorithm, where each line refers to the *utility* distribution and its corresponding standard deviation of a policy over the parameter uncertainty space. These policies were generated by the algorithm using different random seeds. Therefore, for each algorithm, the performance difference of its different policies reflects the impact of the exploration randomness on its average performance.



(a)

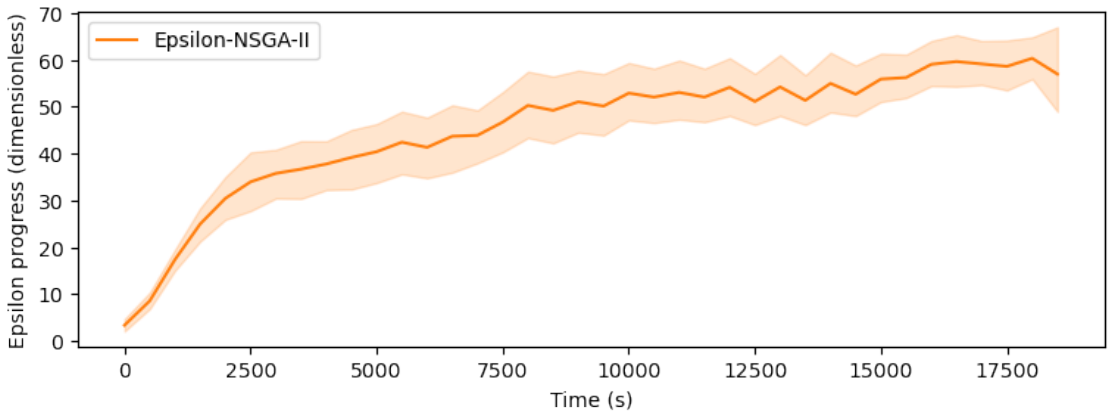


(b)

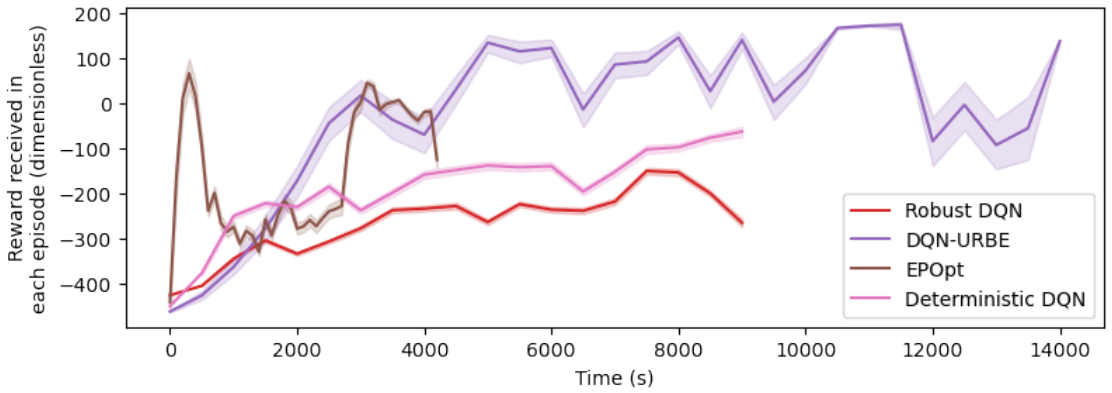
Figure 4.12: Parallel coordinate figures that show different robustness metrics for *reliability* (a) and *utility* (b) of each algorithm in the complex-objective robust Cartpole problem with fixed initial state. In Figure (a), we used *reliability* = 200, which was also the maximum *reliability* that could be provided in an episode, as the pre-defined threshold for Starr's domain criterion. Moreover, Mean is not a robustness metric and used for reference only.

utility and *reliability* for poles of almost any length.

By comparing Figure 4.12 (a) with Figure 4.8, we also observed obvious improvements in the *reliability* performance of the RL algorithms in terms of the other robustness metrics, especially for EPOpt and Deterministic DQN. These two algorithms provided similar Mean (188 and 187), Starr's domain criteria (0.74 and 0.77) and 10^{th} percentile performance (148 and 146) in this experiment, outperforming all the other algorithms. However, ε-NSGA-II still provided the highest *reliability* in the worst case, and outperformed Robust DQN and DQN-URBE concerning 10^{th} percentile performance. With regard to *utility* (Figure 4.12 (b)), Deterministic DQN also provided the best performance regarding all the robustness metrics. It was followed by Robust DQN, ε-NSGA-II, EPOpt and DQN-URBE, where the former outperformed the latter in terms of both Maximin and 10^{th} percentile performance.



(a)



(b)

Figure 4.13: ε-progress curve for ε-NSGA-II (a), learning curves for the RL algorithms (b), and their 95% confidence intervals in the complex-objective robust Cartpole problem with fixed initial state. Each curve of the algorithm is averaged over its multiple runs with different random seeds. Each point on the curve represents the value averaged over 500 seconds.

Efficiency Evaluation

Compared with the Cartpole problem with random initial state (Figure 4.5), ε-NSGA-II was now able to make more ε-progresses during its exploration processes, thereby identifying better final policies (Figure 4.13 (a)). However, this also made it take longer to converge than the RL algorithms. The convergence process of DQN-URBE was also extended, and its policy performance still varied the most during this process (Figure 4.13 (b)). In contrast, Robust and Deterministic DQNs had similar convergence time, which was also similar to that in the first experiment. EPOpt still took the shortest time to identify its resulting policies, because these policies were identified at the peak of its learning curve, as we explained in Section 4.2.3. This process only took around 400 seconds.

4.4.3 Discussion

Averagely, we argue that Deterministic DQN achieved the best trade-off between *utility* and *reliability* in this problem, but it was still complemented by ε-NSGA-II and EPOpt. ε-NSGA-II provided higher *reliability* in the worst case, while EPOpt outperformed Deterministic DQN in all aspects for short poles and was more computationally efficient. Moreover, with the use of appropriate random seeds, both Deterministic DQN and EPOpt learned policies that outperformed all the ε-NSGA-II ones, and Deterministic DQN also identified the best policy for this problem.

Robustness

Algorithms	Average performance	Robustness	Stability
ε-NSGA-II	Utility = 95 Reliability = 173	Parameter uncertainty (*NO*) Model misspecification (*NO*)	Sensitive to the exploration randomness
Robust DQN	Utility = 170 Reliability = 174	Parameter uncertainty (-) Model misspecification (*YES*)	Sensitive to the exploration randomness
DQN-URBE	Utility = 123 Reliability = 156	Parameter uncertainty (-) Model misspecification (-)	Sensitive to the exploration randomness
EPOpt	Utility = 148 Reliability = 188	Parameter uncertainty (-) Model misspecification (-)	Sensitive to the exploration randomness
Deterministic DQN	Utility = 179 Reliability = 187	Parameter uncertainty (-) Model misspecification (*YES*)	Sensitive to the exploration randomness

Table 4.7: Summary of the robustness evaluation results in the complex-objective robust Cartpole problem with fixed initial state. In this table, *YES* indicates that we argue the algorithm provides robustness in the corresponding aspect, *NO* indicates the algorithm does not provide robustness, and - means the algorithm performance depends on the use of random seeds.

As the objective became complex in this experiment, the *Random* performance reflected that it was harder to achieve high *utility* than *reliability* (Figure 4.9). Therefore, the overall performance of ε-NSGA-II was compromised. Compared with the last experiment, ε-NSGA-II provided extra *utility* by slightly costing its *reliability* performance. To identify good trade-offs between these two goals, it also made more ε-progresses during its exploration processes, resulting in its higher sensitivity to the exploration randomness (Figure 4.10). However, it still could only provide constant and

mediocre *utility* in most evaluation scenarios (Figure 4.11 (a)). This was caused by its lack of real-time policy adaptability. Since the static policies by ε-NSGA-II were fixed action sequences, the number of times they applied no force was also fixed. Without cycling through the planning steps to rework new policies, which we have argued inapplicable in the Cartpole problem, these ε-NSGA-II policies could not seize the opportunities that were not predicted to obtain more benefits. For example, when the actual pole was longer than those in the training scenarios and less likely to fall over, the agent should apply no force more frequently to increase its *utility*, but the static policy did not support this automatic adaptation in real-time. Therefore, we also argue that ε-NSGA-II was not robust to the model misspecification in this problem.

In contrast, the real-time policy adaptability allowed the RL algorithms to seize these opportunities and obtain more benefits without reworking new policies, so they generally performed better on *utility*. Robust and Deterministic DQNs showed the same trend in their performance distributions (Figure 4.9), because they learned the same dynamic strategy for their policies to solve this problem: the agent should (1) push the cart until the pole angle is small enough; (2) stop applying force until this angle exceeds a threshold; and (3) loop through these steps. In most cases, this strategy allowed the agent to keep the pole upright for 200 steps without applying force most of the time. Therefore, Robust and Deterministic DQNs achieved nearly the maximum *utility* and *reliability* simultaneously. Moreover, this strategy was applicable for poles of any length, so Deterministic DQN could also learn it, even though planning for the nominal scenario only. In this case, the simplicity of its deterministic learning process minimized its sensitivity to the exploration randomness, and this strategy guaranteed its robustness to the parameter uncertainty and model misspecification. Thus, Deterministic DQN provided the best performance in this experiment. This finding contradicted the previous belief that the 'predict-then-act' approach often fails to provide robustness because they cannot adapt to unseen scenarios [18, 25, 76], and showed how much the policy adaptability contributed to the robustness of RL. The only shortcoming of this strategy was that a good threshold was hard to learn, especially for short-pole scenarios, where the pole angle changed too quickly and the pole was more likely to fall over once the angle exceeded the threshold. Hence, some Robust and Deterministic DQN policies performed poorly in these cases (Figure 4.10 and 4.11 (b,e)).

In the previous two experiments, Deterministic DQN was likely to learn a different strategy, which was to interfere with the system frequently to keep the pole as vertical as possible. This strategy worked for short poles, but often overreacted in the long-pole scenarios. Therefore, Deterministic DQN was more sensitive to the overfitting issue and exploration randomness, which also compromised its overall performance (Figure 4.7 (e)). This distinction was caused by the difference in the reward functions used in these experiments. The action of applying no force was rewarded in this experiment but not in the previous ones. The same reason might also explain why

the *reliability* performance of DQN-URBE and EPOpt was less sensitive, too (Figure 4.10 (c,d)). Therefore, unlike ε-NSGA-II, the overall performance of the RL algorithms was improved even though the objective became complex in this experiment. Nevertheless, we found that DQN-URBE and EPOpt did not follow the same strategy exactly like Robust and Deterministic DQNs, so they did not avoid the overfitting issue completely, and provided a worse balance between *utility* and *reliability* (Figure 4.9). Finally, it is worth noting that this complex objective did not conflict with the single objective we used before. So in the single-objective Cartpole problem, we could also define the reward function in the same way to encourage the RL algorithms to learn this 'minimal-interference' strategy and thereby improve their overall *reliability* performance.

In conclusion, we argue that the robustness of RL heavily depends on the generalizability of the strategy it learns, and its real-time policy adaptability. Properly guiding the algorithm to learn generalizable strategies plays a vital role in the application of Robust RL in decision-making under uncertainty.

Efficiency

In this experiment, Robust and Deterministic DQNs were similarly efficient because they had similar convergence time (Figure 4.13 (b)). EPOpt still provided the highest computational efficiency because it identified its final policies before the others converged. Moreover, the efficiency of these algorithms did not change obviously compared with the first experiment. In contrast, the efficiency of DQN-URBE was compromised and was still sensitive to the exploration randomness. This might indicate that DQN-URBE was less efficient than the other RL algorithms in handling complex objectives. After excluding the randomness of the initial model state, ε-NSGA-II was now able to make more policy updates during exploration (Figure 4.13 (a)). However, its convergence process was also greatly extended, which made ε-NSGA-II the least efficient algorithm for this problem.

4.5 Conclusion

In this chapter, we compared the MOREAs and MORRL algorithms in three variants of the robust Cartpole problem. The main characteristics of the Cartpole problem are (1) it starts from random initial states, and (2) its time horizon and time-step are short. So real-time policy adaptability is necessary to guarantee good performance in it.

We found that static policies by ε-NSGA-II could not provide this adaptability as dynamic policies by the RL algorithms. Therefore, they performed poorly in scenarios with random initial states or short poles, hence providing poor robustness to parameter uncertainty in the single-

objective problem. However, ε-NSGA-II was still robust to model misspecification because it did not overfit the training scenarios. In contrast, Robust, Deterministic DQNs and EPOpt averagely provided higher robustness to parameter uncertainty. However, their performance was more sensitive to exploration randomness. These RL algorithms performed perfectly with the appropriate use of random seeds but worse in other cases. Due to the nature of DQN-URBE, it was severely affected by the overfitting issue, and failed to provide robustness in any case.

In the complex-objective problem, we found the RL algorithms generally provided better trade-offs between *utility* and *reliability*, and robustness to parameter uncertainty than ε-NSGA-II, due to their policy adaptability. Robust and Deterministic DQNs simultaneously maximized these two goals even though the latter is a 'predict-then-act' approach, because they learned a strategy well generalized to the entire parameter uncertainty space. This finding further emphasized the contribution of policy adaptability to the robustness of RL. Eventually, the RL algorithms were also more efficient than ε-NSGA-II in solving all these problems.

In conclusion, we argue that the MOREAs and MORRL algorithms were complementary in the Cartpole problems. The RL algorithms generally provided higher policy adaptability, robustness and efficiency, while ε-NSGA-II was less sensitive to exploration randomness and overfitting issues, thus always providing better worst-case performance.

Chapter 5

Lake Problem

5.1 Introduction

5.1.1 Background

Lake problem describes a situation where inhabitants of a town intend to increase their economic benefits through developing industry and agriculture [19, 20]. These activities will emit more pollution (in the form of phosphorus) into a lake nearby. Once the lake pollution exceeds a threshold, irreversible lake eutrophication will occur and cause huge economic losses. As the phosphorus in the lake will gradually decrease due to natural removal, appropriate control of pollution emissions can achieve sustainable development. This decision-making problem requires decision-makers to decide the annual phosphorus emissions into the lake for the next 99 years, and the main target is to maximize the economic benefit of the town while avoiding eutrophication.

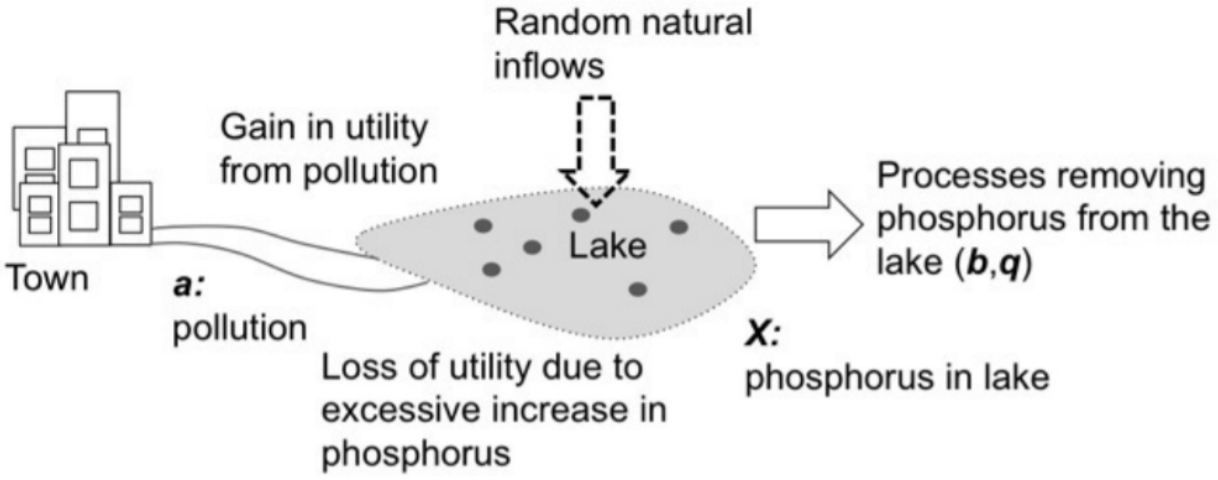


Figure 5.1: Lake problem [20].

5.1.2 Introduction

As a deep uncertainty problem, Lake problem involves both parameter and objective uncertainties. The original Lake problem has four objectives, whose actual desirabilities are unknown for decision-making. However, due to the existing knowledge gap, our MORRL algorithms can only identify one Pareto-optimal solution at a time, hence inefficient in solving problems with such a large objective uncertainty space. To facilitate our experiments, we decided to simplify the original problem. In this chapter, we consider two variants of the Lake problem, including a complex-objective Lake problem and a multi-objective Lake problem. The former involves an objective and a hard constraint, while the latter involves two objectives without assumptions about their preferences. We start with the first problem to study the robustness of our algorithms to the parameter uncertainty in the Lake problem, and then use the second to investigate their robustness to the deep uncertainty.

5.1.3 Motivation

Lake problem is a hypothetical and stylized deep uncertainty problem widely used in studies of DMDU approaches [20, 102, 103]. We took it as our second experimental environment for the following three motivations. First, Lake problem involves deep uncertainty, which we used to compare the performance of MOREA and MORRL in dealing with deep uncertainty. Second, Lake problem is relatively simple. Its state space, action space and uncertainty space are small so that the time taken to solve it was acceptable for this one-year project. Its simple internal mechanisms also allowed us to better understand the performance and characteristics of our algorithms. Finally, Lake problem is a well-known domain problem in the DMDU field. We used it to demonstrate the application of MORRL in DMDU, thereby introducing MORRL to this new field. Meanwhile, we also contrasted it with the Cartpole problem to study the difference between robust planning research in the DMDU field and RL area.

5.2 Complex-Objective Lake Problem

In the first experiment of this chapter, we considered the complex-objective Lake problem, which only involved the parameter uncertainty from the original problem. We excluded the complexity introduced by the objective uncertainty, so that we could gain more insights into how the parameter uncertainty affected the behavior of the Lake system, and what strategies our algorithms adopted to solve this problem robustly.

5.2.1 Problem Conceptualization

Problem Formalization

We formalized this problem as MDPs based on its source EMA model from EMA Workbench [43]. These MDPs contained the following parameters (Table 5.1). The time horizon of each episode

Parameters	Descriptions	Nominal Values
b	Phosphorus removal rate	0.42 (*dimensionless*)
q	Steepness factor	2.0 (*dimensionless*)
μ	Mean of natural pollution inflow distribution	0.02 (*dimensionless*)
σ	Standard deviation of natural pollution inflow distribution	0.0017 (*dimensionless*)
δ	Discount factor	0.98 (*dimensionless*)
α	Cost multiplier	0.4 (*dimensionless*)

Table 5.1: Parameters and their nominal values in the Lake problem [20].

was 99 years, or until lake eutrophication occurred (when the actual lake pollution exceeded the eutrophication threshold). The time-step between two consecutive model states was one year. Therefore, each episode contained at most 99 steps. The model states were described by four variables, which were the outcomes of interest defined in the source model (Table 5.2). The MDPs

Outcomes of interest	Descriptions
phosphorus	Concentration of phosphor in the lake that year
benefit	Economic benefit obtained that year
Inertia	Yearly change in the anthropogenic pollution rate compared with the last year
eutrophication	Flag indicating whether the lake is eutrophic that year

Table 5.2: Outcomes of interest in the Lake problem [43].

started from a fixed initial state, where all the variable values were 0. In other words, we assumed there was no lake pollution at the beginning of each episode. The action option in each state referred to the amount of phosphorus emitted to the lake that year, whose possible values were real numbers from [0, 0.1]. For those algorithms that support discrete action space only, we instead used 11 discrete values from 0 to 0.1 by an increment of 0.01 as their action options. The transition function between states was also derived from the source model.

The original Lake problem involves five uncertain parameters, whose possible value ranges are specified in the source model. In this experiment, we used these ranges as their testing parameter value ranges, and defined the training ranges ourselves to simulate model misspecification (Table 5.3). Additionally, to introduce a complex objective in this experiment, we assumed the only

Parameters	Training value ranges	Testing value ranges
b	[0.35, 0.45] (*dimensionless*)	[0.1, 0.45] (*dimensionless*)
q	[2, 2.8] (*dimensionless*)	[2.0, 4.5] (*dimensionless*)
μ	[0.01, 0.025] (*dimensionless*)	[0.01, 0.05] (*dimensionless*)
σ	[0.001, 0.002] (*dimensionless*)	[0.001, 0.005] (*dimensionless*)
δ	[0.97, 0.99] (*dimensionless*)	[0.93, 0.99] (*dimensionless*)

Table 5.3: Uncertain parameters and their corresponding value ranges in the Lake problem, adapted from [43]. We defined these training parameter value ranges by setting them to [2/7, 1/2] of the corresponding testing ranges. These two numbers were chosen arbitrarily.

objective was to maximize the economic benefit of the town, while there was a hard constraint that lake eutrophication must be avoided. These two goals were adversarial, and we referred to them as *utility* and *reliability* respectively. In each episode, *utility* was measured as discounted accumulated *benefit* obtained in all years, and *reliability* was measured as a fraction of years when the lake was not eutrophic. To guide the exploration of our algorithms, we defined the fitness and reward functions as follows: at each step, the agent receives a reward of 100 * (*benefit* that year) if the lake is not eutrophic. Otherwise, it receives a -500 and the episode terminates immediately. Finally, this problem did not involve model uncertainty.

Model Implementation

Unlike the Cartpole problem, the source model of the Lake problem is implemented as an EMA model rather than an OpenAI Gym Environment [88]. Therefore, we reconstructed it as our MDPs following the processes described in Section 3.1.2. The source model first reads constant parameter values, a scenario, and a static policy as input and initializes an array to store the annual lake pollution observed during the simulation. Then, it iteratively executes the actions from the policy to drive the simulation, and records the annual lake pollution. Eventually, the static policy and the annual pollution array form the corresponding trajectory, based on which the policy fitness is measured. According to this model, our MDPs were built to specify constant parameter values in *'init'*; read the input scenario and initialize the array in *'reset'*; execute the actions and calculate the step-wise fitness or rewards in *'step'*.

5.2.2 Experimental Setup

In this experiment, we applied six algorithms for exploration to identify optimal robust policies for this problem for comparison. They were ε-NSGA-II, Borg, Robust DQN, DQN-URBE, EPOpt and Deterministic DQN (Section 4.2.2). Due to the time constraint, we ran each algorithm only once, and used random seed α to control the exploration randomness. Table 5.4 details the experimental setups.

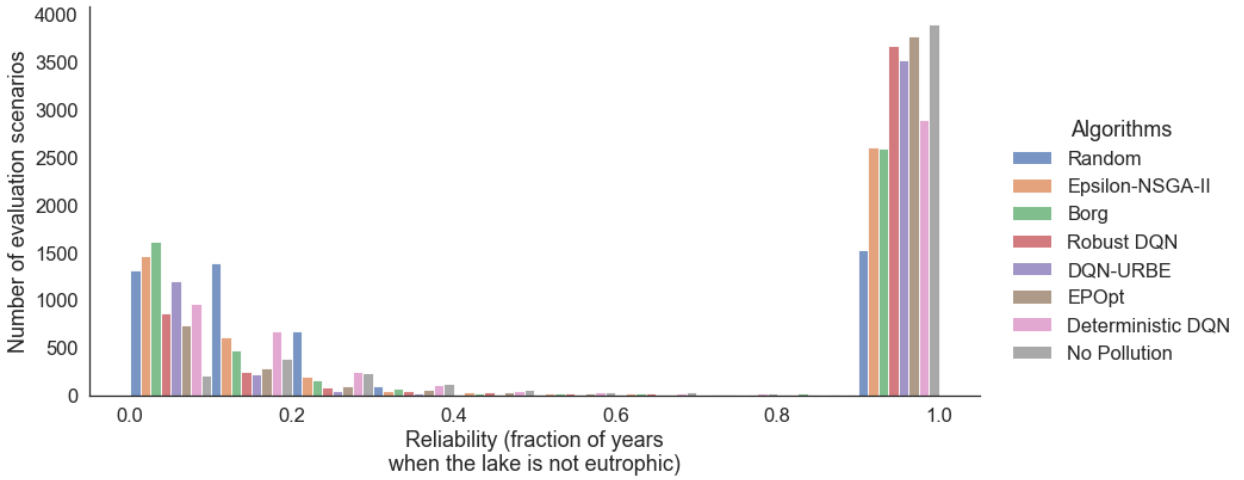
Setups	Descriptions	Values
Training scenario set size	Size of the training scenario set (Section 3.1.4)	200 (Figure 8.3)
Random seeds	Random seeds used to control the randomness of the exploration processes	α (Table 8.1)
Number of CPUs	Number of CPUs used to support the exploration processes of the algorithms	75
ε-NSGA-II	Number of function evaluations taken in the exploration process of ε-NSGA-II	100,000 function evaluations
Borg	Number of function evaluations taken in the exploration process of Borg	100,000 function evaluations
Robust DQN	Number of iterations taken in the exploration process of Robust DQN	8,000 iterations
DQN-URBE	Number of episodes taken in the exploration process of DQN-URBE	4,000 episodes
EPOpt	Number of iterations taken in the exploration process of EPOpt	4,000 iterations
Deterministic DQN	Number of iterations taken in the exploration process of Deterministic DQN	8,000 iterations

Table 5.4: Experimental setups for the complex-objective Lake problem. The table specifies one random seed α, which means that we ran each algorithm once, with α controlling the exploration randomness. Moreover, we allocated a sufficient computational budget for each algorithm to converge its exploration process, in order to yield a fair comparison.

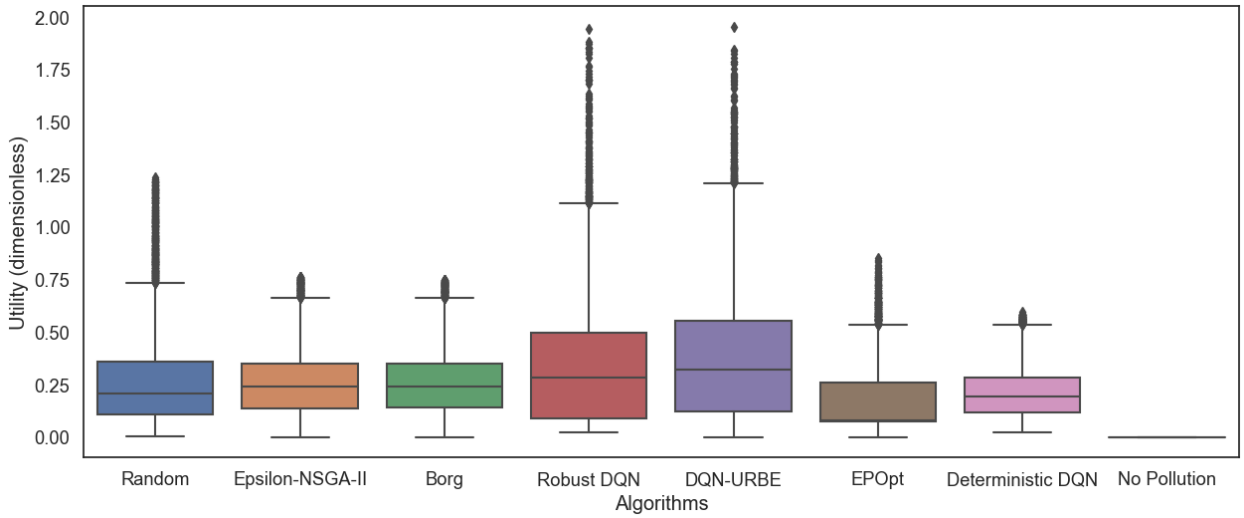
5.2.3 Results

Robustness Evaluation

In this experiment, we set two baseline policies, namely *Random* and *No Pollution*. The former referred to the policy randomly generated using seed α, while the latter referred to the policy of



(a)



(b)

Figure 5.2: *Reliability* (a) and *utility* (b) performance distributions of the algorithms in the 5,000 evaluation scenarios in the complex-objective Lake problem.

emitting no pollution into the lake. *Random* was most prone to lake eutrophication (Figure 5.2, Table 8.2 in the appendix), and it only provided mediocre average *utility* (0.28). *No Pollution* maximized its *reliability* at a cost of all *utility*, but natural pollution inflows alone still caused eutrophication in 1,118 out of 5,000 evaluation scenarios.

The policy performance of ε-NSGA-II and Borg was very similar. Compared with *Random*, they managed to avoid eutrophication in more scenarios and provide average *reliability* of 0.57 over the uncertainty space. Meanwhile, they also provided similar average *utility* (0.27) but smaller *utility* variances (Figure 5.2, Table 8.2 in the appendix). In contrast, Robust DQN and DQN-URBE provided much higher average *reliability* (0.77 and 0.72) and *utility* (0.36 and 0.40). However, al-

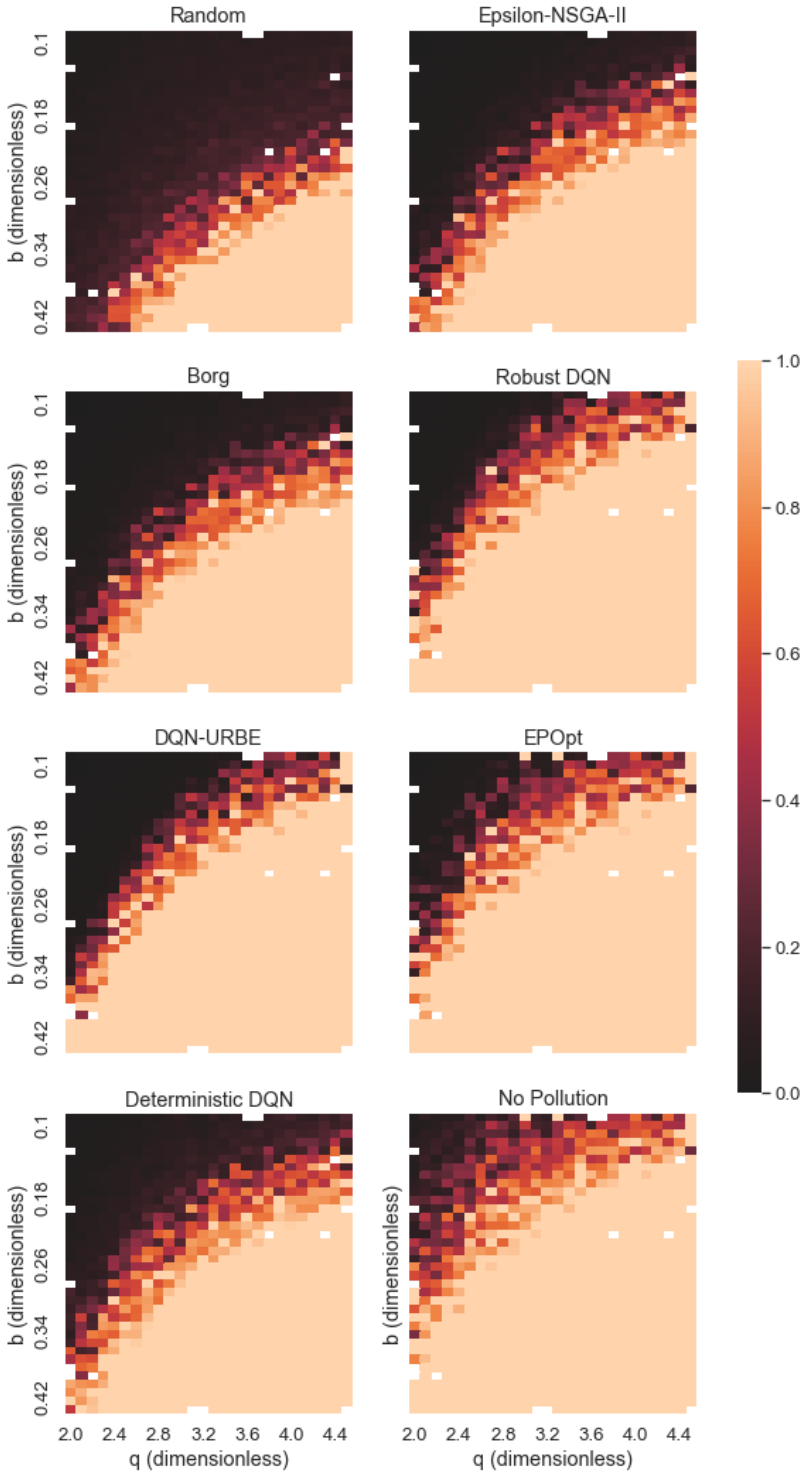


Figure 5.3: *Reliability* heat maps of the algorithms on the parameter uncertainty space in the complex-objective Lake problem. The x-axis of this figure refers to the value range of q, while the y-axis refers to the value range of b. Each subplot in this figure corresponds to an algorithm, where the color reflects the *reliability* provided by the algorithm in the corresponding scenario.

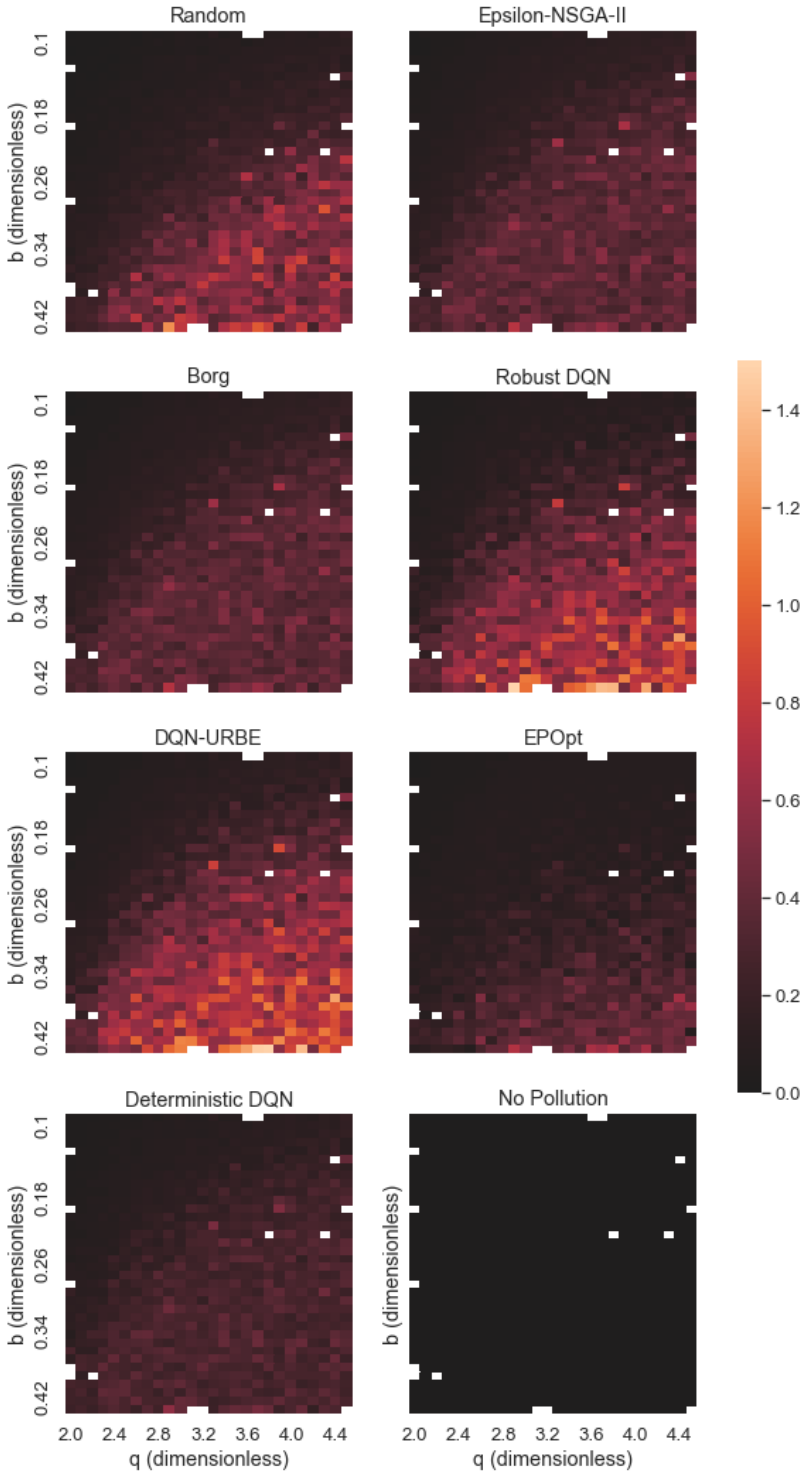
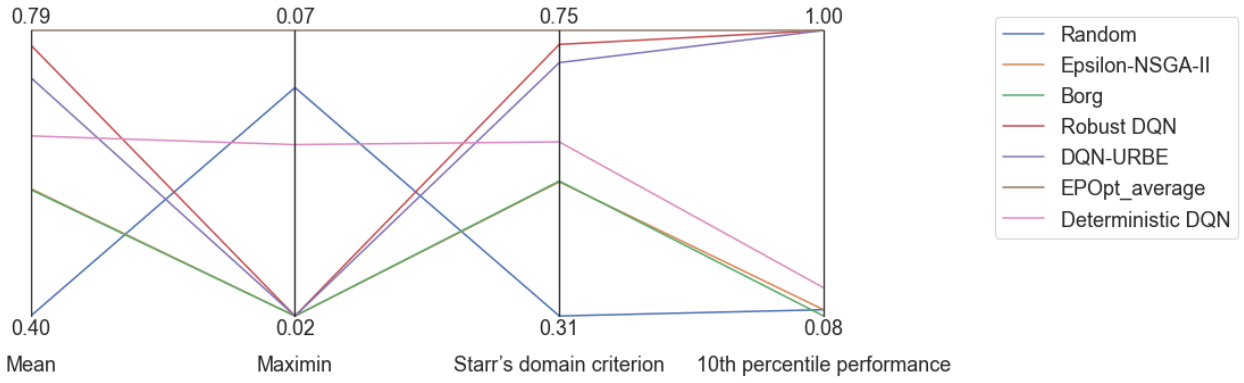


Figure 5.4: *Utility* heat maps of the algorithms on the parameter uncertainty space in the complex-objective Lake problem. The x-axis of this figure refers to the value range of q, while the y-axis refers to the value range of b. Each subplot in this figure corresponds to an algorithm, where the color reflects the *utility* provided by the algorithm in the corresponding scenario.

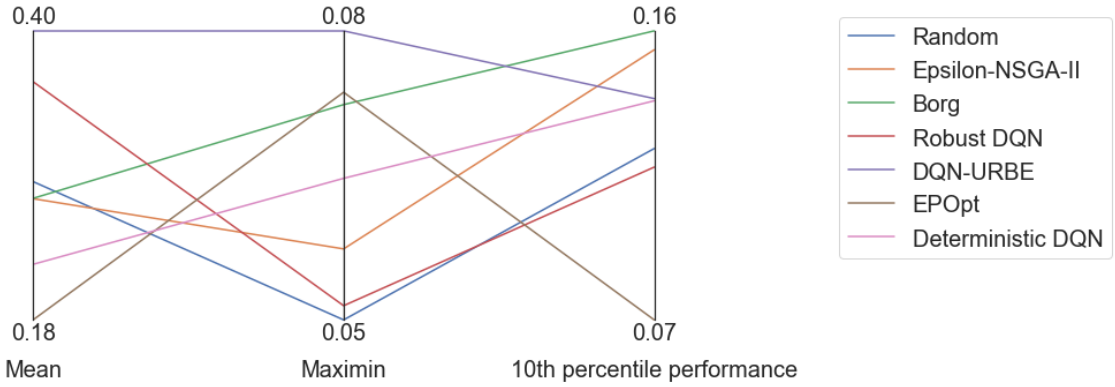
though they robustly avoided eutrophication in most cases, their *utility* distributions had the largest variances. EPOpt provided the best *reliability* distribution but the worst *utility* distribution in this experiment, except for *No Pollution*. Its policy only led to eutrophication in 1,227 scenarios, comparable to 1,118 for *No Pollution*. Finally, the overall performance of Deterministic DQN was close to that of the EAs. Compared with the Robust RL algorithms, its policy was much more likely to cause eutrophication, and also provided densely distributed poor *utility* over the uncertainty space.

Figure 5.3 and 5.4 show the policy performance distributions of these algorithms over the parameter uncertainty space. In each subplot of Figure 5.3, the area of the brightest area reflected the proportion of evaluation scenarios where the algorithm avoided eutrophication. These observations were consistent with Figure 5.2 (a). Moreover, we observed that when both b and q were close to their lower limits, eutrophication always occurred even without anthropogenic pollution inflows (Figure 5.3). It indicated that this problem was unsolvable in these scenarios, which we will ignore in the following discussion. In other cases, the EAs and Deterministic DQN generally provided constant *utility*, because no color gradient was observed in their heat maps (Figure 5.4). In contrast, Robust DQN, DQN-URBE and EPOpt were able to adjust their economic benefits according to different b and q, which explained the large *utility* variances of Robust DQN and DQN-URBE.

Regarding the other robustness metrics for *reliability*, the overall rankings of these algorithms were EPOpt, Robust DQN, DQN-URBE, Deterministic DQN and the EAs, where the former outperformed the latter in all aspects (Figure 5.5 (a)). The only exception was that the Robust DQN and DQN-URBE policies caused eutrophication earlier than the Deterministic DQN one in the worst case. These observations were generally consistent with Figure 5.2 (a). With regard to *utility*, we argue that its Maximin did not accurately reflect the robustness of the algorithms in this experiment. This was because lake eutrophication always occurred in the first several steps in the worst scenarios, so the economic benefits obtained by the algorithms during this period were too similar (Figure 5.5 (b)). In addition, we observed that Robust DQN and DQN-URBE provided low 10^{th} percentile *utility* relative to their high average *utility*. And the EAs outperformed all the RL algorithms concerning this metric.



(a)

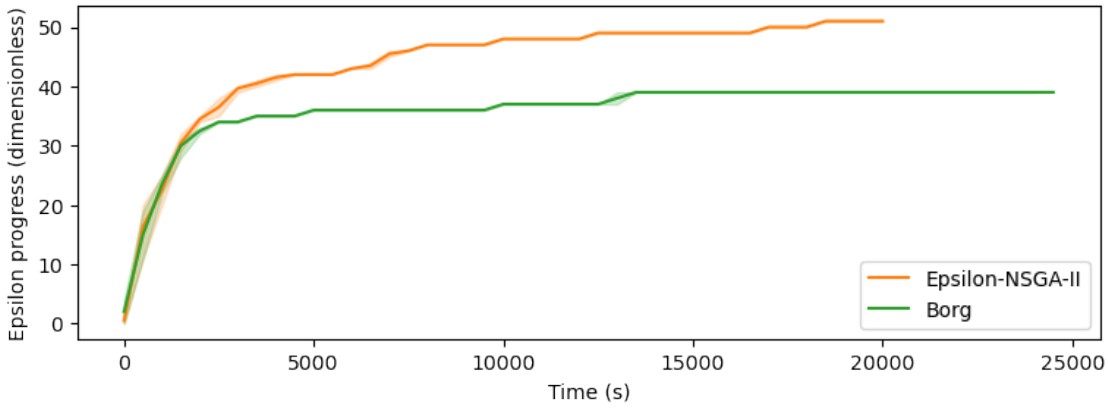


(b)

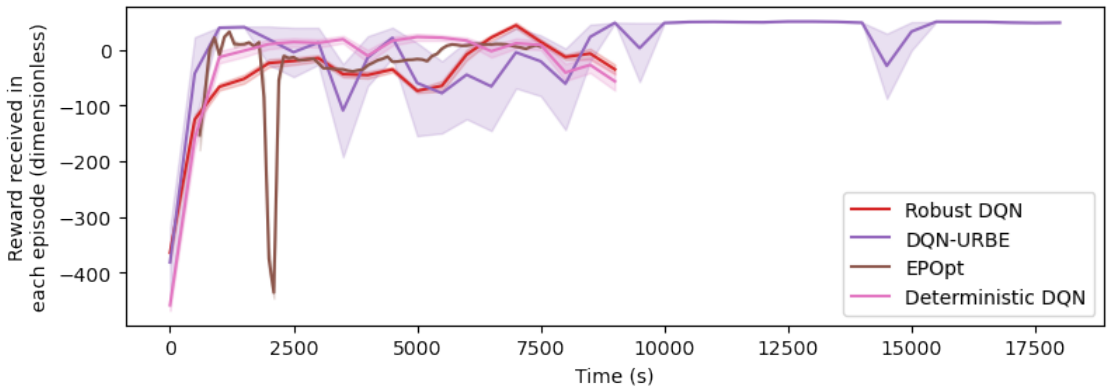
Figure 5.5: Parallel coordinate figures that show different robustness metrics for *reliability* (a) and *utility* (b) of each algorithm in the complex-objective Lake problem. Notice that we ignored the 1,118 unsolvable scenarios in these figures, and used *reliability* = 1, which was also the maximum *reliability* that could be provided in an episode, as the pre-defined threshold for Starr's domain criterion in Figure (a). Moreover, Mean is not a robustness metric and used for reference only.

Efficiency Evaluation

The ε-progress curves of ε-NSGA-II and Borg were very similar in the first 2,000 seconds (Figure 5.6 (a)). After that, Borg converged faster and stopped progressing completely after 14,000 seconds, while the convergence of ε-NSGA-II was slower with more ε-progresses occurring. In contrast, the convergence time of the RL algorithms was shorter and similar to each other (Figure 5.6 (b)). Nevertheless, EPOpt identified its final policy within 1,300 seconds rather than at the end of the exploration process like the others (Section 3.1.2). Finally, although we used the same experimental setups as the robust Cartpole problem in this experiment (Table 4.2 and 5.4), the running time of all algorithms had increased, especially for DQN-URBE and EPOpt.



(a)



(b)

Figure 5.6: ε-progress curves for the EAs (a), learning curves for the RL algorithms (b), and their 95% confidence intervals in the complex-objective Lake problem. Each point on the curves represents the value averaged over 500 seconds.

5.2.4 Discussion

Based on our experimental results and interpretation methods (Section 3.4), we argue that Robust DQN, DQN-URBE and EPOpt provided the best robustness to the parameter uncertainty in this experiment and were not complemented by the others. Since *utility* and *reliability* were adversarial, their policies actually provided the best non-dominated trade-offs between the two goals, and outperformed policies of the other algorithms regarding at least one goal. Additionally, EPOpt was also the most efficient algorithm.

Robustness

The *Random* and *No Pollution* performance reflected the difficulty distribution of the problem over the parameter uncertainty space. Generally, avoiding lake eutrophication was harder than

Algorithms	Robustness
ε-NSGA-II	Parameter uncertainty (*NO*) Model misspecification (*NO*)
Borg	Parameter uncertainty (*NO*) Model misspecification (*NO*)
Robust DQN	Parameter uncertainty (*YES*) Model misspecification (*YES*)
DQN-URBE	Parameter uncertainty (*YES*) Model misspecification (*YES*)
EPOpt	Parameter uncertainty (*YES*) Model misspecification (*YES*)
Deterministic DQN	Parameter uncertainty (*NO*) Model misspecification (*NO*)

Table 5.5: Summary of the robustness evaluation results in the complex-objective Lake problem. In this table, *YES* indicates that we argue the algorithm provides robustness in the corresponding aspect, and *NO* indicates the algorithm does not provide robustness.

obtaining economic benefits (Figure 5.2). The Lake system became more fragile and the problem became more challenging as b and q became smaller (Figure 5.3), because they determined the eutrophication threshold and natural pollution growth rate [20, 43].

To achieve sustainable economic development in this problem, the overall idea was to control annual pollution emissions to allow for natural removal. Different algorithms implemented this idea in different ways. Following the ε-NSGA-II and Borg policies, decision-makers should always discharge large amounts of pollution in the first few years when the lake is less polluted, and discharge moderate pollution periodically afterward. This strategy was applicable in the training scenarios, where b was large but q was small, but was not well generalized to other testing scenarios. When b and q were both small, eutrophication occurred immediately after emitting a medium amount of pollution in any step, so these policies provided low *reliability* (Figure 5.3); When b and q were both large, the system had a strong self-regulation capacity and these policies missed the opportunity to obtain extra *utility* from more pollution (Figure 5.4). Therefore, we argue that ε-NSGA-II and Borg were not robust to the parameter uncertainty or model misspecification in this problem.

In contrast, we observed that the Robust DQN and DQN-URBE policies adaptively decided to emit more pollution in opportune scenarios to obtain more *utility* and reduce emissions in dire scenarios to avoid eutrophication (Figure 5.4). In this way, although they had larger *utility* variances, they actually provided better average *utility* and *reliability*, and robustness compared with the EAs.

Meanwhile, we also observed similar behaviors of the EPOpt policy. Although it was more conservative in its pollution emissions, it still adapted its actions to the actual state to maximize its performance. Therefore, these three algorithms provided the best non-dominated trade-offs between *utility* and *reliability*. And we argue that they provided robustness to the parameter uncertainty and model misspecification in this experiment. This observation again indicated that the real-time policy adaptability greatly promotes the robustness and efficiency of RL as an exploration method in DMDU, by allowing its policy to adapt to more unseen situations and opportunities before a new policy needs to be reworked.

In this problem, the RL agents made decisions based on their imperfect observations of the model state. The natural pollution inflows and eutrophication threshold were essential for them to adjust their actions adaptively, but these parameters were uncertain and inaccessible. Therefore, the RL algorithms learned estimates of these parameters during their exploration processes. However, by learning only from the nominal scenario, Deterministic DQN incorrectly formed fixed estimates of these parameters. In this case, its final policy followed a static strategy and behaved similarly in any evaluation scenario like the static policy (Figure 5.4). This strategy was only applicable in the nominal scenario, so Deterministic DQN failed to provide adaptability or robustness in this experiment. Compared with the dominance of Deterministic DQN in the complex-objective Cartpole problem, this observation again indicated that the robustness of RL heavily depends on whether the strategy it learns can be well generalized to all possible futures.

By taking the parameter uncertainty into consideration, the Robust RL algorithms not only learned such estimates, but also learned to adjust them based on the actual observation during deployment. Hence, Robust DQN, DQN-URBE and EPOpt still achieved adaptability. Nevertheless, in dire scenarios, eutrophication often occurred in the first few steps, where the agent might not collect enough observations for adjustments if the initial estimates greatly differed from the actual ones. Therefore, EPOpt conservatively learned very low estimates of these uncertain parameters, and its policy often stopped emitting pollution even though the lake pollution was far below the actual eutrophication threshold. In contrast, Robust DQN and DQN-URBE were more optimistic about their initial estimates to enhance their *utility*, but more likely to cause eutrophication in those dire scenarios. This explained why EPOpt slightly improved its *reliability* at a cost of much *utility*, compared with the other two (Figure 5.2). Finally, as mentioned before, the main criticism of RTD-DQN is that planning for the worst often leads to overly conservative policies [73], so DQN-URBE was proposed to mitigate this conservativeness by integrating RTD-DQN with UBE [18]. The DQN-URBE performance in this experiment empirically proved that this integration achieved its goal.

These characteristics of the algorithms also explained their performance on the other robustness metrics for *utility* and *reliability* (Figure 5.5). Robust DQN, DQN-URBE and EPOpt adaptively

reduced their pollution emissions in relatively bad scenarios to avoid eutrophication, so they provided higher Starr's domain criterion, the maximum *reliability* in the 10^{th} percentile worst scenario, but lower 10^{th} percentile *utility*. Additionally, Deterministic DQN outperformed Robust DQN and DQN-URBE with regard to the worst-case *reliability*, because the latter two were more optimistic about their initial estimates of the uncertain parameters, but failed to finish their adjustments before eutrophication actually occurred in the worst case.

Efficiency

In this experiment, EPOpt was the most efficient algorithm because it took the shortest time to identify its final policy. We argue that Robust, Deterministic DQNs and DQN-URBE provided similar efficiency and outperformed the EAs, according to their convergence time. Finally, Borg was also more efficient than ε-NSGA-II, because it took less time and fewer ε-progresses to converge.

It is worth noting that function evaluations in this Lake problem and the robust Cartpole problem contained similar numbers of model steps. Each function evaluation in the Lake problem consisted of 200 episodes (one for each scenario in the training scenario set), and each episode contained at most 99 steps; each evaluation in the Cartpole problem consisted of 100 episodes, and each episode contained at most 200 steps. Meanwhile, since we also used the same experimental setups, the exploration processes of each algorithm in different experiments should contain similar total step numbers. Compared with the robust Cartpole problem, we observed that the running time of all algorithms had increased in this experiment, mainly because the transition logic of the Lake problem was more complex and computationally intensive. However, the time of DQN-URBE and EPOpt had increased more due to their implementation characteristics (Section 3.1.2). For DQN-URBE, its RMDP was designed to work sequentially [18]. As the complexity of the transition function increased, the running time of DQN-URBE increased fastest, and would eventually become intractable in complex problems. For EPOpt, it employed batch optimization and updated its policy with the 10% worst scenarios in each iteration [76]. As the size of the training scenario set had doubled in this experiment, the number of scenarios used by EPOpt for updating had also doubled, so its running time had also greatly increased. Compared with them, the other algorithms were free from these issues, so they were less affected.

5.3 Multi-Objective Lake Problem

With a deeper understanding of the behavior of the Lake system and the robustness of the algorithms to its parameter uncertainty, we now switch to the multi-objective Lake problem. This is also the first deep uncertainty problem considered in this project, involving both parameter and

objective uncertainties. We used this problem to investigate the robustness of the algorithms to deep uncertainty.

5.3.1 Problem Conceptualization And Experimental Setup

Setups	Descriptions	Values
Training scenario set size	Size of the training scenario set (Section 3.1.4)	200 (Figure 8.4)
Random seeds	Random seeds used to control the randomness of the exploration processes	α (Table 8.1)
Weight vectors	Weight vectors used in the MDP reward function of the RL algorithms	(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7), (1,8), (1,9), (1,10)
Number of CPUs	Number of CPUs used to support the exploration processes of the algorithms	75
ε-NSGA-II	Number of function evaluations taken in the exploration process of ε-NSGA-II	200,000 function evaluations
Borg	Number of function evaluations taken in the exploration process of Borg	200,000 function evaluations
Robust DQN	Number of iterations taken in the exploration processes of Robust DQN	4,000 iterations
DQN-URBE	Number of episodes taken in the exploration processes of DQN-URBE	4,000 episodes
EPOpt	Number of iterations taken in the exploration processes of EPOpt	2,000 iterations
Deterministic DQN	Number of iterations taken in the exploration processes of Deterministic DQN	4,000 iterations

Table 5.6: Experimental setups for the multi-objective Lake problem. In this experiment, we ran ε-NSGA-II and Borg once but each RL algorithms 10 times with different weight vectors. In each of these vector, the first value is the weight for *utility* and the second value is the weight for *reliability* in the MDP reward function of the RL algorithms. Moreover, we allocated a sufficient computational budget for each algorithm to converge its exploration process, in order to yield a fair comparison.

To introduce objective uncertainty, we reframed the Lake problem from the last experiment, taking *utility* and *reliability* as two independent objectives without assuming their preferences. We also adopted the same models but redefined their fitness and reward functions to describe the objective uncertainty. The fitness function was defined as taking the values of *utility* and *reliability* obtained

by the agent in the entire episode as two independent objective indicators, while the reward function was defined as the weighted sum of *utility* and *reliability* obtained at each step. As this experiment did not involve hard constraints, each episode always contained 99 steps. These settings referred to the source model of the Lake problem [43]. Additionally, according to the nature of MOREA and 'naive' multi-policy MORRL, we ran ε-NSGA-II and Borg once, but each RL algorithm 10 times with different weight vectors, in order to identify multiple Pareto-optimal solutions using each algorithm. Finally, we also used the same training scenario set because we found its appropriate size was also 200 in this problem (Figure 8.4). Table 5.6 details the experimental setups.

5.3.2 Results

Robustness Evaluation

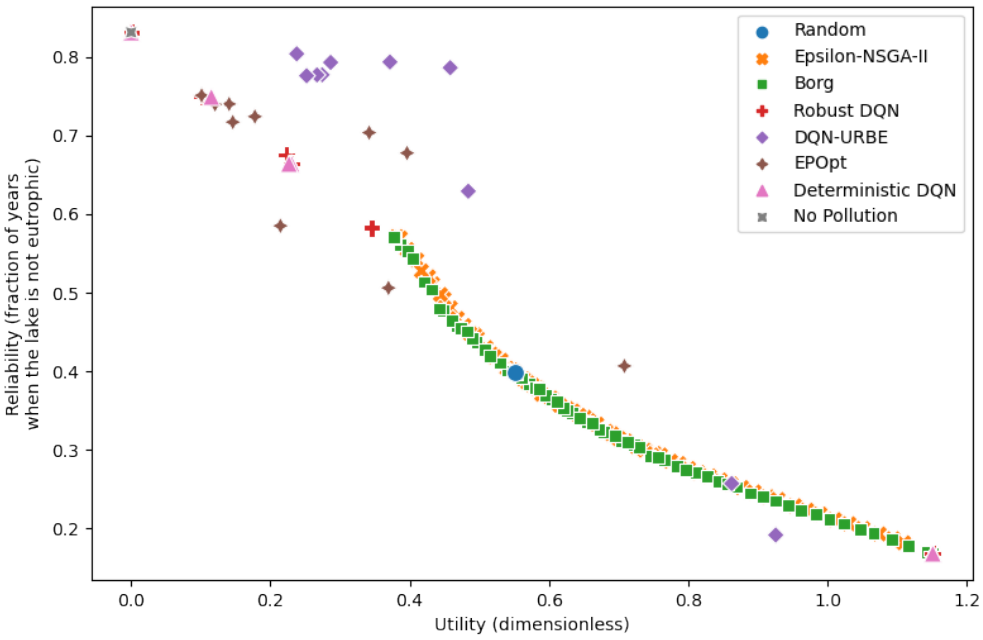


Figure 5.7: Policy trade-off distributions in the multi-objective Lake problem. Each point refers to the average performance of a policy of an algorithm over the parameter uncertainty space, concerning *utility* and *reliability*.

We used the same *Random* and *No Pollution* as in the last experiment. The former provided mediocre average *utility* (0.55) and *reliability* (0.40), while the latter cost all *utility* to maximize its average *reliability* (0.83) (Figure 5.7). The overall performance of ε-NSGA-II and Borg was similar. They both identified many policies (63 and 65) in one run, providing various trade-offs between the two objectives, including the one provided by *Random*. These policies were mutually non-dominated, because they averagely outperformed each other regarding one of the objectives.

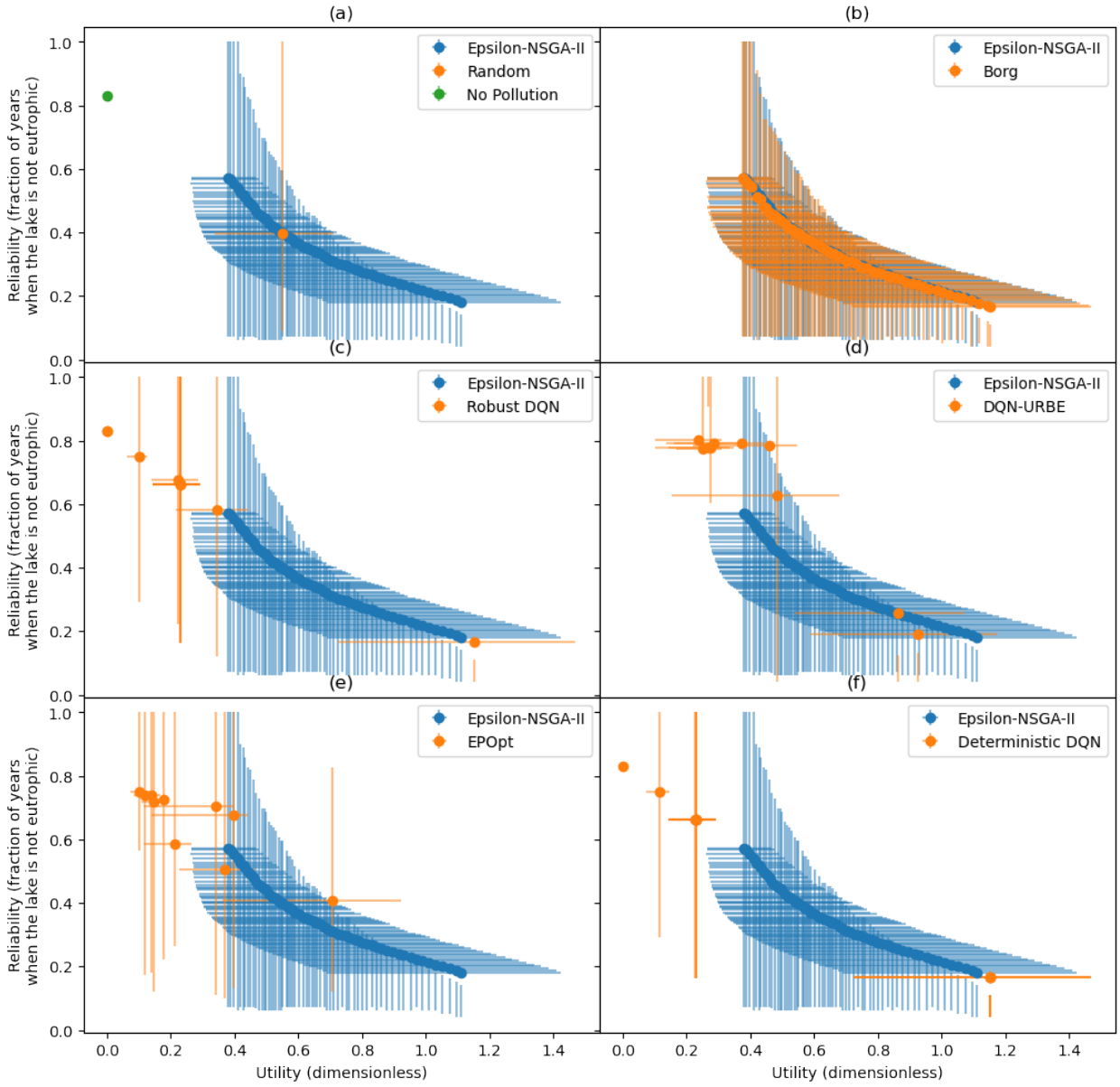


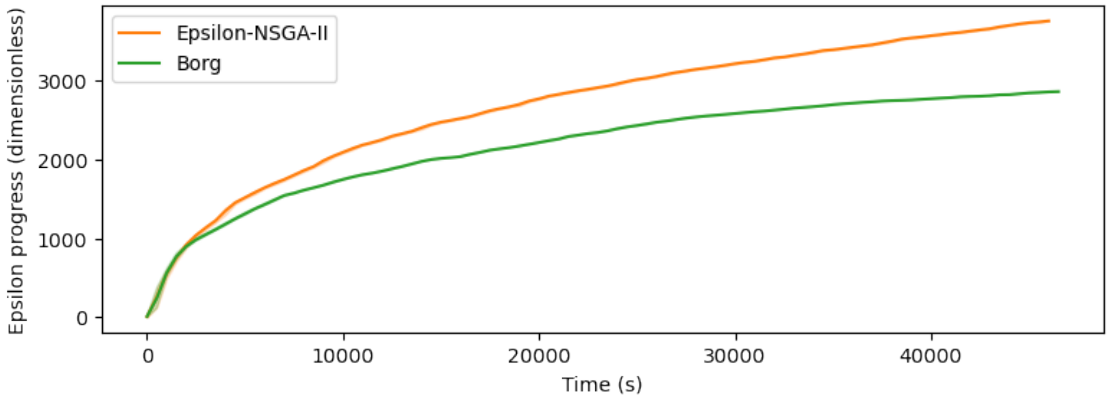
Figure 5.8: Policy trade-off distributions and their performance variances in the multi-objective Lake problem. Each point refers to the average performance of a policy of an algorithm over the parameter uncertainty space concerning *utility* and *reliability*, and each error bar connected with the point refers to the corresponding interquartile range concerning *utility* or *reliability*. Some points are not connected by vertical error bars because their corresponding policies provided the maximum *reliability* in more than 75% of the evaluation scenarios, so the corresponding interquartile ranges were 0.

For the EA policies that provided the same average trade-off as *Random*, their *reliability* distributions were more positively skewed (Figure 5.8 (a)). Finally, although the trade-offs provided by ε-NSGA-II and Borg were densely distributed, neither of them identified a policy providing average *reliability* higher than 0.6, even though such a static policy existed (*No Pollution*).

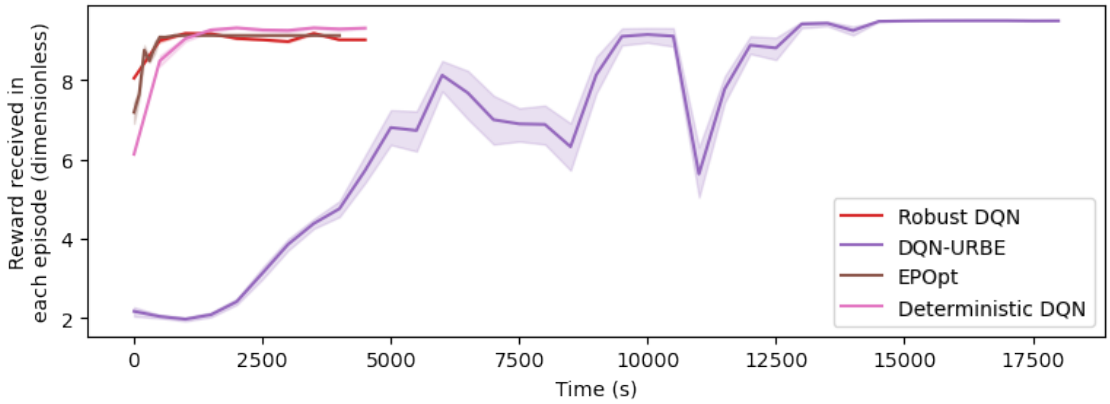
Robust and Deterministic DQNs only identified policies that provided either high *utility* or high *reliability* (Figure 5.8 (c,f)). These policies were also non-dominated by the EA ones, and had similar performance variances if their average performance was close to that of the EA policies. However, the different trade-offs provided by them were fewest (6 and 4) and least densely distributed. The DQN-URBE policies performed best in providing high *reliability*, where they were not mutually non-dominated, and the best of them outperformed the similar policies of Robust, Deterministic DQNs and the EAs regarding both objectives (Figure 5.8 (d)). These DQN-URBE policies also had their first and third quartile *reliability* close to the maximum. Nevertheless, DQN-URBE did not identify a policy that maximized *utility* in this experiment. Its policies that provided relatively high *utility* could be dominated by the EA ones, and had more positively skewed *reliability* distributions. DQN-URBE also provided a denser trade-off distribution than Robust and Deterministic DQNs, but that was still incomparable to the EAs. Finally, the EPOpt policies generated from different runs performed unstably in terms of their average performance and variances (Figure 5.8 (e)). They either outperformed, underperformed, or non-dominated the similar Robust, Deterministic DQN and EA policies in these aspects, where no general pattern was observed. Nevertheless, we still found that the trade-off distribution of EPOpt had a similar density and range to that of DQN-URBE.

Efficiency Evaluation

We observed that the ε-progress curves of ε-NSGA-II and Borg showed similar trends to the last experiment (Figure 5.9 (a) and Figure 5.6 (a)), where ε-NSGA-II took more ε-progresses and longer to converge than Borg during their exploration processes. Meanwhile, both ε-NSGA-II and Borg also took much more ε-progresses and longer to converge compared with themselves in the last experiment. The learning curves of Robust, Deterministic DQNs and EPOpt were also similar to those in Figure 5.6 (b), in terms of both trend and convergence time (Figure 5.9 (b)). Robust DQN and EPOpt still had similar convergence time, but that of Deterministic DQN was slightly longer compared with them. In contrast, the convergence time and running time of DQN-URBE had greatly increased. In this experiment, DQN-URBE took more than triple the time to identify its final policy compared with the other RL algorithms.



(a)



(b)

Figure 5.9: ε-progress curves for the EAs (a), learning curves for the RL algorithms when using a weight vector of (1, 9) (b), and their 95% confidence intervals in the multi-objective Lake problem. Each point on the curve represents the value averaged over 500 seconds.

5.3.3 Discussion

In this multi-objective Lake problem, ε-NSGA-II and Borg performed similarly. They simultaneously provided the best average policy performance and robustness to the parameter and objective uncertainties regarding policy trade-off with medium or high *utility*. They were also more efficient than the RL algorithms in dealing with the objective uncertainty. Their only shortcoming was that they could not identify policies providing high *reliability*, which was complemented by the RL algorithms. In this respect, DQN-URBE outperformed all the other algorithms regarding both average performance and robustness.

Average Policy Performance And Robustness To Parameter Uncertainty

As introduced in Section 3.4.2, we compared the average policy performance of the algorithms based on their policy dominance relationship in Figure 5.7 and 5.8. However, their robustness to the parameter uncertainty was only reflected by their *reliability* variances in this problem, as discussed in Section 5.2.4.

Based on Figure 5.8, the EAs and *Random* provided comparable average policy performance because their policies were mutually non-dominated. However, the *reliability* distributions of the EAs were more positively skewed, indicating that they provided lower *reliability* in more evaluation scenarios. This observation reflected the EAs suffered from the overfitting issue in this problem. Their policies might only perform well in the scenarios used for training but much worse in those unseen evaluation ones. Moreover, this shortcoming also prevented the EAs from identifying policies that provided high *reliability*, as discussed in Section 5.2.4. Therefore, we argue that the EAs were not robust to the model misspecification in this problem.

For the same reason, we argue that Robust and Deterministic DQNs provided similar average performance and robustness to the parameter uncertainty regarding policy trade-off with high *utility*, compared to the EAs (Figure 5.8). DQN-URBE outperformed the other algorithms in all aspects when providing high *reliability*, because its policies dominated the others, and robustly provided nearly the maximum *reliability* in most cases. However, DQN-URBE was bad at providing high *utility*. Under this condition, its policies overfit the training scenarios more severely, and provided lower *utility* and *reliability* in those unseen evaluation scenarios compared with the EAs. Therefore, we found that DQN-URBE had poorer average performance and robustness to the parameter uncertainty in this condition. Finally, due to the nature of the dynamic policy, all the RL algorithms learned policies providing high *reliability*, which meant they provided robustness to the model misspecification regarding policy trade-off with high *reliability*.

There were also some observations that required further research to explain. First, the learning processes of DQN-URBE and EPOpt were unstable. Some of their policies outperformed their other policies regarding both *utility* and *reliability*, rather than being mutually non-dominated like the Robust, Deterministic DQN and EA ones (Figure 5.8). This issue prevented us from drawing a settled conclusion about the policy performance of EPOpt. We speculate this unstable learning of EPOpt was due to the fact that its default unconstrained initial exploration was not enough for such a complex problem [76]. Nevertheless, further investigations are needed to verify this hypothesis and explain the unstable learning of DQN-URBE. Second, unlike the complex-objective Lake problem, Robust DQN and EPOpt lost their dominance over average performance and robustness to parameter uncertainty compared with the EAs and Deterministic DQN, but DQN-URBE did not. By further studying the strategies adopted by these algorithms to solve this problem, these changes could be explained by the characteristics of the algorithms. Lastly, the RL algorithms rarely gen-

erated policies providing medium or high *utility*, so we could not evaluate their performance under this condition until further experiments.

Robustness To Objective Uncertainty

As well-known multi-objective optimization methods, ε-NSGA-II and Borg provided the best robustness to the objective uncertainty regarding their trade-off distribution density, but failed to achieve the largest distribution range because of their poor robustness to the model misspecification (Figure 5.7). In contrast, all the RL algorithms could theoretically provide the largest range by using appropriate weights (like Robust and Deterministic DQNs), because they were free from this issue. However, this observation still needs further experiments to verify. The main shortcoming of the RL algorithms was that their trade-off distributions were far sparser than the EAs, especially for Robust and Deterministic DQNs. This was mainly caused by their two weaknesses in handling objective uncertainty: (1) the nature of 'naive' multi-policy MORRL, which generates one policy to provide one trade-off per run; and (2) the unpredictable mapping from the objective weights used in the MDP reward function to the final policy performance of the RL algorithms.

In this experiment, we used a fixed weight for *utility* and weights increased from 1 to 10 for *reliability* to guide the RL algorithms to learn policies providing different trade-offs between the two objectives. Intuitively, we expected these policies to gradually provide slightly decreasing *utility* but increasing *reliability* with the weight for *reliability* increased. But in fact, for the same algorithm, its policy performance A generated using a weight vector $(1, X)$ might unpredictably differ from its policy performance B generated using $(1, X-1)$ (Figure 5.7). Based on our observations, A might: (1) be identical to B (Robust or Deterministic DQN learned the same policy even with some different weight vectors); (2) dominate B or be dominated by B (DQN-URBE or EPOpt learned policies that dominated its other policies); (3) provide higher *utility*, but lower *reliability* compared with B; (4) greatly differ from B (no Robust or Deterministic DQN policy provided medium *utility* or *reliability*). This unpredictability is mainly caused by the high complexity and sensitivity to randomness of the RL learning process. It greatly challenges the design of input weight vectors for the RL algorithm to learn policies that provide the desired trade-off distribution. Decision-makers might need to try a much larger number of different vectors so that their RL algorithm can produce the same trade-off distribution as the EAs. Moreover, this weakness is also exacerbated by the nature of 'naive' multi-policy MORRL, which might make it intractable for our RL algorithms to provide satisfactory robustness to objective uncertainty in many problems.

Efficiency

In this experiment, Borg was still more efficient than ε-NSGA-II, because it took a shorter time and fewer ε-progresses to converge (Figure 5.9 (a)). Additionally, the computational efficiency of both ε-NSGA-II and Borg was greatly compromised after introducing the objective uncertainty. This was because these MOREAs simultaneously maintained and evolved a set of non-dominated solutions in their exploration processes, in order to generate multiple Pareto-optimal solutions at the end of one run.

In contrast, our 'naive' multi-policy MORRL algorithms still learned one policy per run, so the efficiency of Robust, Deterministic DQNs and EPOpt was not obviously affected (Figure 5.9 (b)). We argue that EPOpt was the most efficient algorithm among them, because it only took around 830 seconds to identify its final policy, instead of at the end of its learning process. Deterministic DQN was slightly less efficient than the other two, because it took longer to converge even if it learned from a deterministic model. This might be caused by the exploration randomness. The only exception was DQN-URBE. Although it also learned one policy per run, its convergence time had greatly increased. This observation indicated that DQN-URBE was the least efficient in dealing with the objective uncertainty compared with the other RL algorithms.

Finally, it is worth noting that Figure 5.9 only reflected the efficiency of the algorithms in one run. Theoretically, ε-NSGA-II and Borg were able to identify all Pareto-optimal solutions during this process, but each 'naive' multi-policy MORRL algorithm could identify one of them only. Overall, we argue that the EAs were much more efficient than our RL algorithms in dealing with the objective uncertainty.

5.4 Conclusion

In this chapter, we compared the MOREAs and MORRL algorithms in two variants of the Lake problems. Compared with the Cartpole problem, the Lake problem has the following characteristics: (1) it starts from a fixed initial state; (2) its time horizon and time-step are long enough for decision-makers to cycle through the three DMDU steps (Figure 2.2) to rework new plans to adapt to the future; (3) it does not explicitly specify state variables to describe the system state. Specifically, (1) and (2) indicate the Lake problem puts lower requirements on real-time policy adaptability, while (3) indicates it poorly supports the real-time control of the dynamic policy.

Nevertheless, our experimental results showed that this adaptability still contributed much to the performance of RL in DMDU. In the complex-objective problem, Robust DQN, DQN-URBE and EPOpt provided the best robustness to parameter uncertainty and model misspecification. This was because their policies could adapt to the actual scenario in real-time, to simultaneously max-

imize *utility* and *reliability*, and avoid overfitting the training scenarios. In contrast, ε-NSGA-II and Borg could not achieve this without reworking new policies adaptively. Deterministic DQN also provided poor robustness, because it failed to learn a strategy well generalized to the entire uncertainty space due to the overfitting issue. Finally, since our Robust RL algorithms were also more efficient than the EAs, we argue they dominated the EAs in this problem.

In the multi-objective problem, the EAs and RL algorithms generally provided similar robustness to parameter uncertainty, but the EAs still could not handle model misspecification. On the other hand, the EAs and RL algorithms had different strengths with regard to their robustness to objective uncertainty. ε-NSGA-II and Borg provided highly dense trade-off distributions, but failed to identify policies providing high *reliability* due to the overfitting issue. The RL algorithms could provide the largest range distributions theoretically, but it might be intractable for them to provide the same distribution density as the EAs. Moreover, we argue that the EAs were much more efficient than the RL algorithms in handling objective uncertainty, because they identified many Pareto-optimal solutions in one run. In conclusion, we argue that the MOREAs and MORRL algorithms were complementary in this problem.

Chapter 6

Electricity Market Problem

6.1 Introduction

6.1.1 Background

Australian Energy Market Operator (AEMO) operates the National Electricity Market (NEM) to support commodity exchange for electricity across five eastern and south-eastern states in Australia [104, 105]. To overcome the difficulty of electricity storage, and control the market electricity cost, the NEM acts as a wholesale spot market that matches electricity supply and demand through a centralized dispatch process in real-time. Specifically, this process divides each day into a sequence of 5-minute trading intervals. For each set interval, electricity generators first submit their bids, specifying the amounts of electricity they can supply at corresponding prices. Then, AEMO collects and orders these bids from low to high according to their offered prices and dispatches the successful bidders according to this order. Finally, the electricity price for this interval is settled to the highest price offered by these successful bidders. This process is known as merit order and aims to minimize the electricity price in the NEM.

In 2016, Australia adopted an international treaty on climate change, namely Paris Agreement. This treaty requires Australia to reduce its greenhouse gas emissions (GHGE) by 26-28% from 2005 levels by 2030 [106]. Since the electricity sector in Australia is one of the major contributors to GHGE [107], looking at how to transit this sector into a low-carbon future presents important GHGE reduction opportunities. This brings us to Electricity Market problem, where decision-makers need to make a plan for rule-makers for the next 10-20 years, to minimize GHGE in the NEM while keeping its electricity price as low as possible. Meanwhile, this plan should also provide robustness, to handle a variety of possible circumstances in reality. In our experiment on this problem, we only focused on the participation of the state of Victoria, because it was the place where this project was developed, facilitating access to data and expert knowledge.

6.1.2 Introduction

This chapter presents one experiment that investigated the performance of our algorithms in the Electricity Market problem. However, due to the complexity of this problem, we did more preprocessing for simplification to reduce the time taken to solve it, and construct our MDPs based on its source model. This chapter is organized as follows. We first introduce the source model we used, the GR4SP simulation model [108]. Then, we present the preprocessing steps we took, including how we reduced the complexity of the source model and the problem. After that, we introduce our formalization of the simplified problem, and how we implemented our MDPs based on the simplified source model. Eventually, we also present our experimental results, as well as the analysis and discussion on them.

6.1.3 Motivation

Electricity Market problem is a real-world decision-making problem with high complexity and deep uncertainty. We took it as our last experimental environment for the following three motivations. First, we used this problem to investigate the impact of high complexity on the performance of MOREA and MORRL in DMDU, including (1) how they formalized this complex problem; and (2) how complexity affected the policies they generated and their robustness to deep uncertainty. Second, this experiment served as an example, demonstrating how to apply the general model interface and DMDU process we designed to solve a real-world problem. Third, we also contrasted this problem with the Cartpole problem to further study the difference between practical robust planning research in the DMDU field and RL area.

6.2 Problem Conceptualization

Like our previous experiments, we conceptualized the Electricity Market problem based on an existing system model of the NEM. In this way, this problem also involved parameter and objective uncertainties only. Although many models of the Australian electricity market have been proposed in previous studies, they either focus on retrospective effects of renewable energy in merit-order markets [109], or investment decisions on new generation [110, 111], which are dynamics irrelevant for our specific focus. Another model that could be useful for this problem is the NEMSIM model [112], but it cannot be accessed. In this experiment, we mainly referred to the GR4SP simulation model as our source model [108]. This model simulates the NEM with data from the Victorian Market only, which captures the problem dynamics we are interested in at the right level of complexity.

6.2.1 GR4SP Simulation Model

GR4SP is a package of software components that provides tools to simulate the socio-technical layout of any electricity system, in order to analyze its long-term development and performance [113]. On this basis, Rojas also implemented a simulation model of the NEM in Victoria, Australia in Java language as a case study for demonstration [108]. This GR4SP simulation model is an agent-based model that reflects the system behavior emerging from interactions between individual agents, such as electricity generators. The bidding mechanism within the NEM forms the core of these agent interactions. Specifically, each run of the GR4SP simulation model involves many rounds, and each round corresponds to a trading interval in the Victorian Market. In each round, the generators first bid for the current trading interval according to their costs and expected profits. Then, these generators are dispatched following a merit order of prices specified in their bids, until the current electricity demand is met. Eventually, this dispatch further determines the current electricity price, GHGE, generators' profits and thus their next-round bidding in the model. Different from the real NEM where the bids are received every 5 minutes, bidding happens every 30 minutes in the GR4SP simulation model. This is because the historical demand data is only available for this time interval.

The GR4SP simulation model relies on both historical data and simulation settings to project the market performance into the future. Specifically, the former are data on generation assets, population, electricity consumption and demand, forecasts for consumption, solar uptake, energy efficiency and onsite generation of the Victorian Market for almost 140 years [114]. These data were collected from Australian official sources and stored in a PostgreSQL database [115–118]. These data are common to all simulation runs. The latter is a set of parameters that characterize different assumptions about the future market behavior, such as the simulation date range, inflation rate and costs of different types of electricity generations. These values are specified in an input YAML file. By varying these parameters, the model can be used to explore the market performance in different scenarios. Moreover, many of these parameter values cannot be accurately predicted until the future unfolds (Table 8.3 and 8.4), which also constitute the parameter uncertainty of the Electricity Market problem.

Finally, the outputs of the GR4SP simulation model are monthly and yearly time series of performance indicators of the electricity system between the simulation start and end dates (Table 8.5 and 8.6).

Nominal Trajectory

In Rojas's work, the simulation results generated by the original GR4SP simulation model using random seed θ (Table 8.1) and the nominal parameter values are considered as the nominal tra-

jectory of the model [108]. This is because this trajectory best matches the historical records, and assumes a business-as-usual scenario where the status-quo will be maintained and the uncertainty will be minimized.

6.2.2 Preprocessing

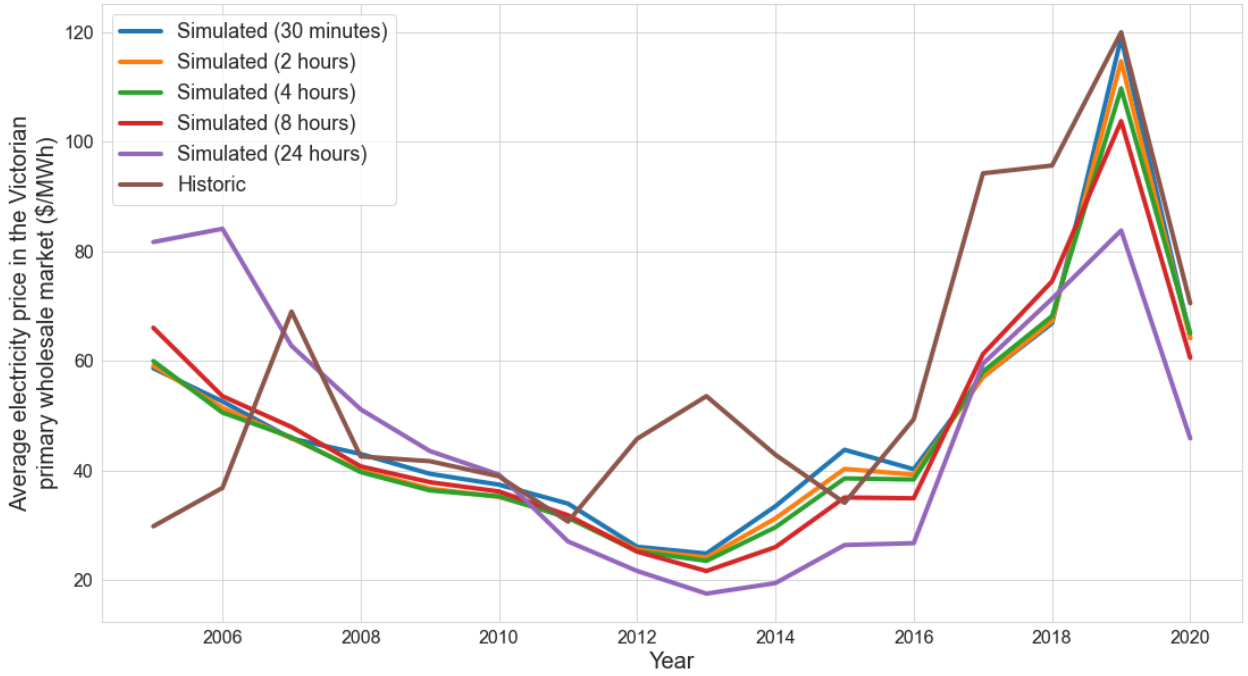
Given the complexity of the GR4SP simulation model, we decided to embed it directly into our MDPs as the transition function. However, according to our tests, executing the original model with one CPU to project the market performance between 01/01/1998 and 01/01/2050 took about 30 seconds. Considering the time limit of this one-year project, it was unaffordable to use such a computationally intensive transition function for data-hungry algorithms such as EA and RL, whose exploration processes often involve millions of episodes. Therefore, we took several steps to sufficiently reduce their running time. These steps can be divided into two categories, model simplification and problem simplification. The former includes trading interval extension and data preloading, which aimed to reduce the running time of the GR4SP model. The latter includes uncertainty space reduction and Principal Component Analysis, which aimed to reduce the number of episodes required for the algorithms to identify their final policies.

Trading Interval Extension

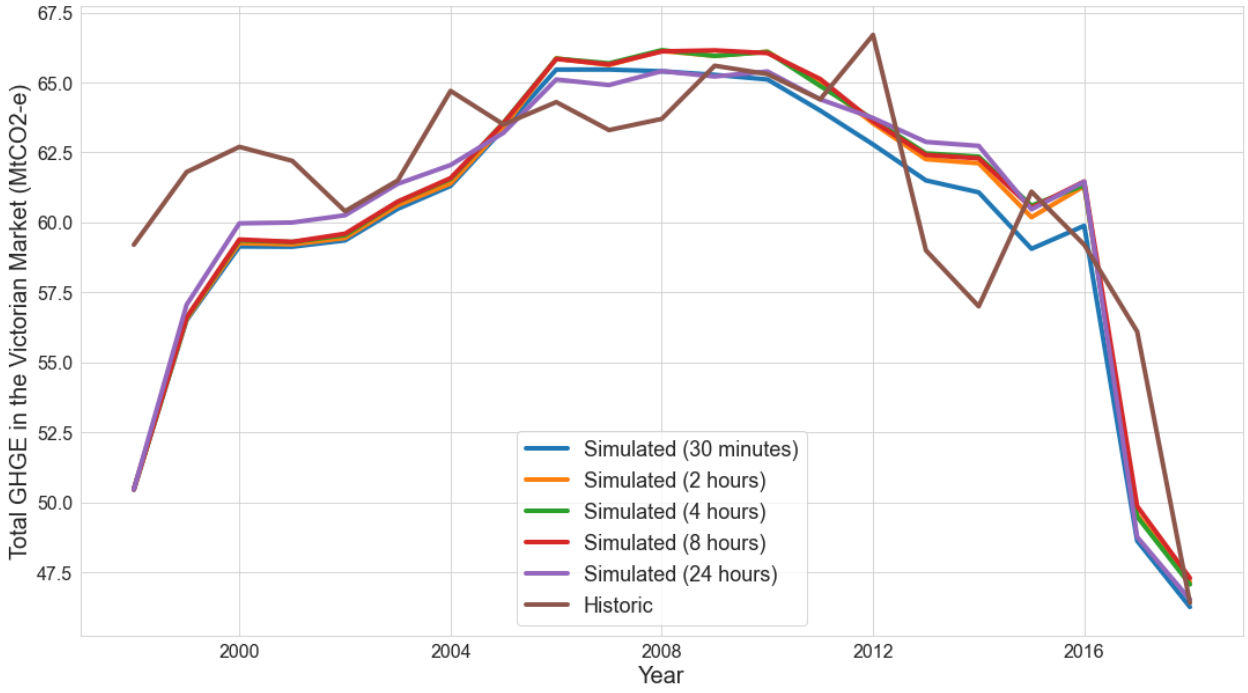
As mentioned in Section 6.2.1, each trading interval is 5 minutes in the real NEM and 30 minutes in the original GR4SP simulation model [108]. In this experiment, we decided to further extend this interval to an appropriate length so that the simulation could be sped up without unduly affecting its accuracy. Here, we considered four other interval lengths, which were 2, 4, 8 and 24 hours. To validate their performance, we used the GR4SP model to simulate the performance of the markets with these intervals, and compared the results with the historical records.

It can be observed that using a larger trading interval in the model increased the error of simulating the annual average electricity price in the market (Figure 6.1 (a)). The larger the interval, the greater the GR4SP model overestimated the prices before 2006 and underestimated the prices between 2012 and 2020. Specifically, the historical mean of annual average electricity prices between 2005 and 2020 is 55.98 ($/MWh) [114], and Table 6.1 shows the errors of the models with different trading intervals. Nevertheless, the historical increase in prices from 2012 to 2014 was due to the *Carbon Pricing* scheme issued by the Australian government to discourage the use of high-GHGE generators, which was not reflected in the GR4SP model. This fact indicated that the actual error of the model should be smaller than that shown in the figure.

In contrast, we found the GR4SP model was more accurate in simulating the annual total GHGE in the market (Figure 6.1 (b)). Moreover, extending the trading interval in the model also



(a)



(b)

Figure 6.1: Yearly time series of outcomes from the historical records, or simulation results from the GR4SP simulation model using different trading intervals, in terms of annual average electricity price in the primary wholesale market (a), and annual total GHGE (b) in the Victorian Market.

Interval Lengths	Mean Absolute Errors	Mean Absolute Percentage Errors
Original (30 minutes)	14.08 ($/MWh)	25.2%
2 hours	14.64 ($/MWh)	26.2%
4 hours	14.88 ($/MWh)	26.6%
8 hours	15.54 ($/MWh)	27.8%
24 hours	22.10 ($/MWh)	39.5%

Table 6.1: Mean absolute errors and mean absolute percentage errors of the models with different trading intervals compared with the historical records, in terms of the annual average electricity prices between 2005 and 2020.

had little impact on this simulation accuracy. For example, the historical mean of annual total GHGE between 2005 and 2020 is 61.34 (MtCO2-e) [114], and Table 6.2 shows the errors of the models with different trading intervals.

Interval Lengths	Mean Absolute Errors	Mean Absolute Percentage Errors
Original (30 minutes)	2.53 (MtCO2-e)	4.1%
2 hours	2.65 (MtCO2-e)	4.3%
4 hours	2.61 (MtCO2-e)	4.3%
8 hours	2.62 (MtCO2-e)	4.3%
24 hours	2.34 (MtCO2-e)	3.8%

Table 6.2: Mean absolute errors and mean absolute percentage errors of the models with different trading intervals compared with the historical records, in terms of the annual total GHGE between 2005 and 2020.

The choice of the trading interval length in the GR4SP simulation model in this experiment led to a trade-off between simulation accuracy and running time. Based on our results, we decided to set this length to 24 hours, and consider the corresponding trajectory in Figure 6.1 as the nominal trajectory and baseline. We believe the error caused by this approximation was acceptable compared with that of the original model, because this experiment mainly focused on the performance of our algorithms in DMDU rather than the simulation itself. However, we still argue that rerunning this experiment with the original model is valuable for obtaining more accurate results if time permits in future studies.

Data Preloading

After reducing the bidding frequency in the GR4SP simulation model, it still took about 12 seconds to run once, and its data loading process occupied about 70% of this running time. This process is responsible for loading the historical data from the PostgreSQL database and predicting their future values within the simulation date range. This process was identical in all simulations, because the historical data are deterministic [114] and we fixed all the parameter values involved in this process. Therefore, we created a checkpoint for the model after it completed the data loading process, and let all simulations start from this checkpoint to avoid repeating the same process. This data-preloading strategy helped us reduce the running time of the GR4SP model to about 3.5 seconds.

Uncertainty Space Reduction

In DMDU, the dimensionality of the uncertainty space of the problem has a great impact on the difficulty of exploration [6]. The more complex the uncertainty space is, on the one hand, the more scenarios are required to describe it (as suggested by stabilization analysis in Section 3.1.4); on the other hand, the harder it is to find a robust policy to all uncertainties. This often results in a longer convergence time for the exploration method. Therefore, we also reduced the time taken to solve the Electricity Market problem by reducing its objective and parameter uncertainty space.

The GR4SP simulation model simulates the market performance concerning a variety of indicators (Table 8.5 and 8.6). Although many of them are concerned by real rule-makers when making plans to deliver Australia's obligation under Paris Agreement, we only focused on the two core indicators to define the objectives in this experiment, which were the annual average electricity price in the primary wholesale market (*'Primary Wholesale ($/MWh)'*) and the annual GHGE per household (*'GHG Emissions (tCO2-e) per household'*) in the Victorian Market.

Additionally, the GR4SP model also involves 28 uncertain parameters (Table 8.3 and 8.4), constituting the parameter uncertainty space in this experiment. We reduced this space by removing those parameters that have trivial impacts on the two indicators we chose. To quantify these impacts, we performed a global sensitivity analysis, namely SOBOL indices [119]. SOBOL indices work through decomposing the variances of the indicators into fractions that are attributed to all uncertain parameters. For each indicator I and each uncertain parameter P, the fraction of the variance of I caused by the variance of P (first-order index), and the fraction caused by P together with interactions between P and other parameters (total-order index) are computed. The larger these indices, the greater impact P has on I. In this experiment, we adopted the implementation of SOBOL indices from EMA Workbench [43]. To perform this analysis, we ran the GR4SP simulation model 12,600 times and randomly sampled parameter values from their possible ranges in

each run, in order to simulate possible market performance between 01/01/1998 and 01/01/2050. During this process, the first-order indices and total-order indices for each year were collected for analysis. These analysis settings referred to [108].

With regard to *'Primary Wholesale ($/MWh)'*, it can be observed that all the uncertain parameters had small median S1 values (Figure 6.2). However, the median ST values of *'consumption'*, *'generationRolloutPeriod'*, *'includePublicallyAnnouncedGen'*, *'nameplateCapacityChangeBrownCoal'*, *'nameplateCapacityChangeWind'* and *'scheduleMinCapMarketGen'* were much larger than the other parameters. These parameters together with *'generatorRetirement'* also provided large maximum ST values, while *'nameplateCapacityChangeBrownCoal'* was the only parameter that provided a large maximum S1 value. Therefore, we argue that only the uncertainty of these parameters had a non-trivial impact on the variance of *'Primary Wholesale ($/MWh)'*. For similar reasons, we also identified the parameters that contributed to the majority variance of *'GHG Emissions (tCO2-e) per household'* were *'domesticConsumptionPercentage'*, *'includePublicallyAnnouncedGen'*, *'learningCurve'*, *'nameplateCapacityChangeBrownCoal'*, *'priceChangePercentageBrownCoal'*, and *'priceChangePercentageWind'* (Figure 6.3).

Based on these results, we decided to only consider the following 7 uncertain parameters in our experiment: *'generationRolloutPeriod'*, *'generatorRetirement'*, *'includePublicallyAnnouncedGen'*, *'nameplateCapacityChangeBrownCoal'*, *'nameplateCapacityChangeWind'*, *'priceChangePercentageBrownCoal'*, and *'priceChangePercentageWind'*. The uncertainty of these parameters has been shown to have a great impact on at least one of the two indicators we cared about. Some parameters that also provided great impacts, such as *'consumption'* and *'domesticConsumptionPercentage'*, were excluded mainly because they were involved in the data loading process of the GR4SP simulation model, and unfixing their values would fail our data-preloading strategy.

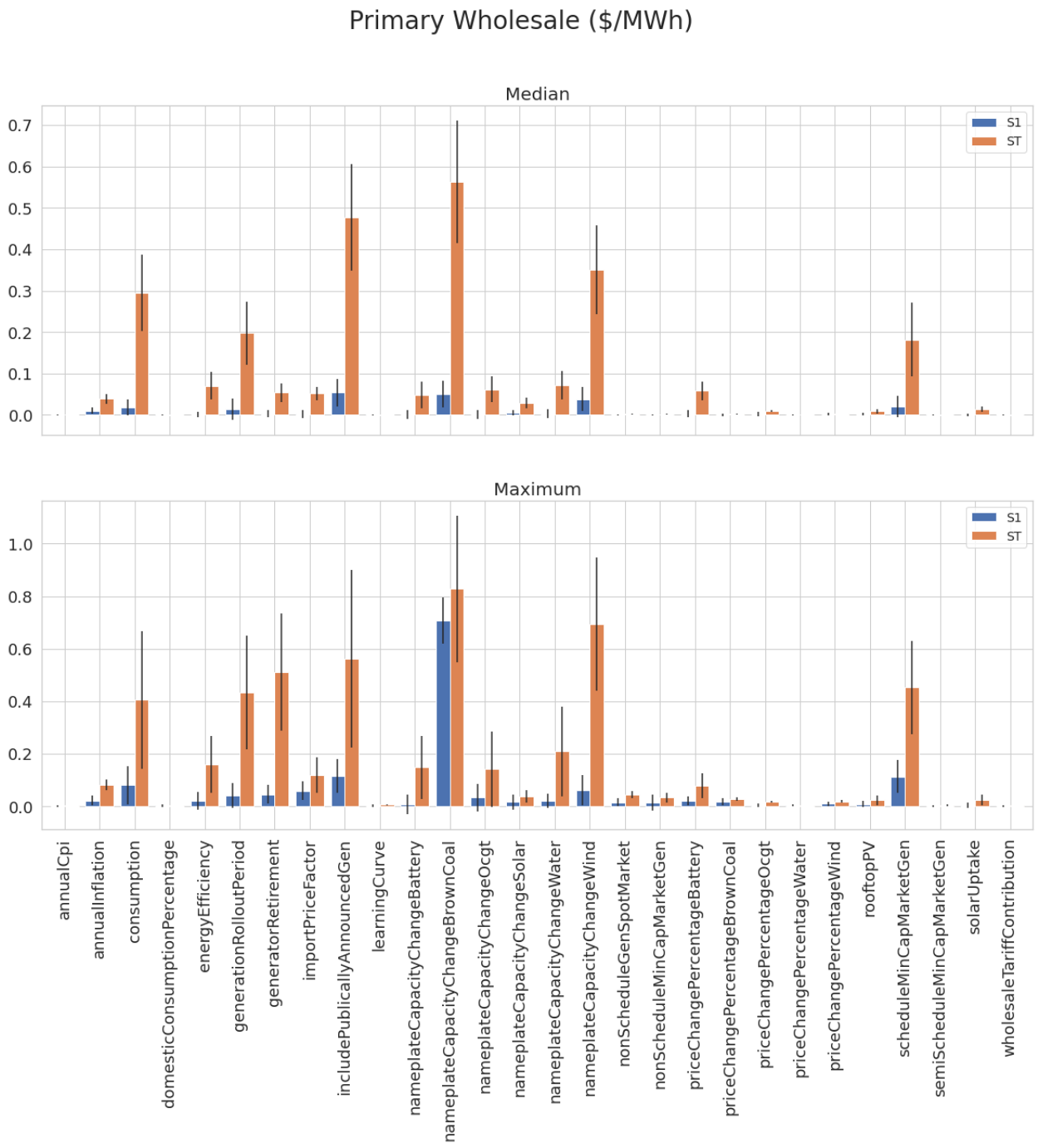


Figure 6.2: SOBOL indices concerning *'Primary Wholesale ($/MWh)'* in the Electricity Market problem. The upper subplot shows the median of S1 (first-order index) and ST (total-order index) of each uncertain parameter between 01/01/1998 and 01/01/2050. The lower subplot shows the maximum of S1 and ST of each uncertain parameter during this period.

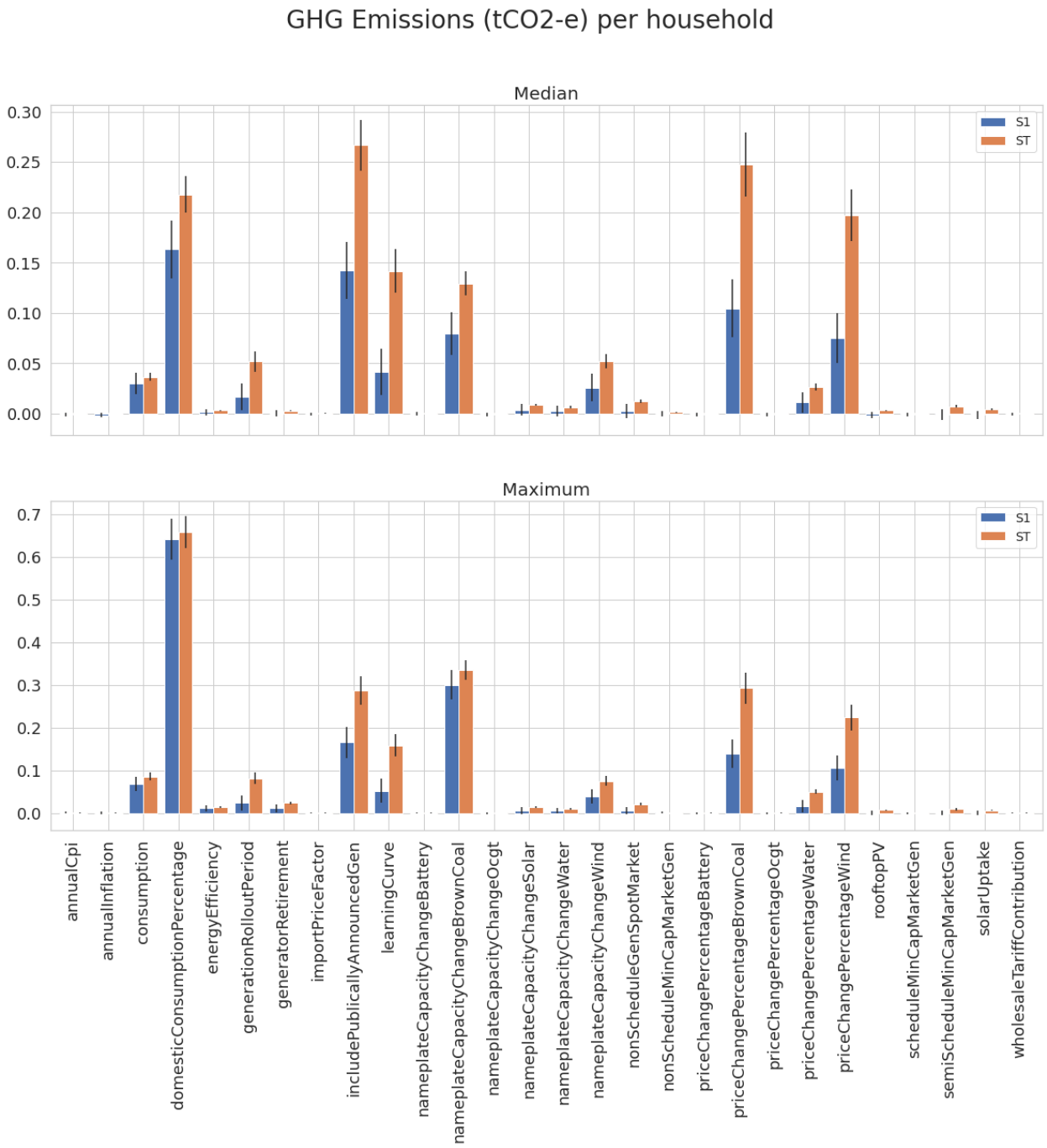


Figure 6.3: SOBOL indices concerning *'GHG Emissions (tCO2-e) per household'* in the Electricity Market problem. The upper subplot shows the median of S1 (first-order index) and ST (total-order index) of each uncertain parameter between 01/01/1998 and 01/01/2050. The lower subplot shows the maximum of S1 and ST of each uncertain parameter during this period.

Principal Component Analysis

To build our MDPs, we also needed to specify state variables to characterize the model state (Section 3.1). The yearly performance indicators produced by the GR4SP simulation model (Table 8.5 and 8.6) were suitable for this purpose. We excluded the 4 indicators relevant to the hourly market performance, because they were meaningless after we extended the trading interval to 24 hours. However, there were still $21 \times 5 = 105$ indicators associated with each state, as we set the time-step between two consecutive states to 5 years. In this case, these indicators could not be used as the state variables directly, because such a high-dimensional state space would greatly compromise the efficiency of the RL algorithms [120].

To overcome this issue, we applied Principal Component Analysis (PCA) [121, 122]. Given a dataset D involving N attributes, its data entities can be represented as points in a N-dimensional coordinate space. The principal components of D are a sequence of N unit vectors P, where the line attributed by the k^{th} vector ($v_k, v_k \in P$) minimizes its average squared distance to all the data points and is also orthogonal to the previous $k-1$ vectors. In this way, the first principal component always guarantees to explain as much as possible the variance in D, while each subsequent component explains as much as possible the variance that is uncorrelated with the previously explained variance [120]. By computing these principal components and using the most effective ones as the new attributes to characterize D, PCA reduces the dimensionality of D while minimizing the information loss.

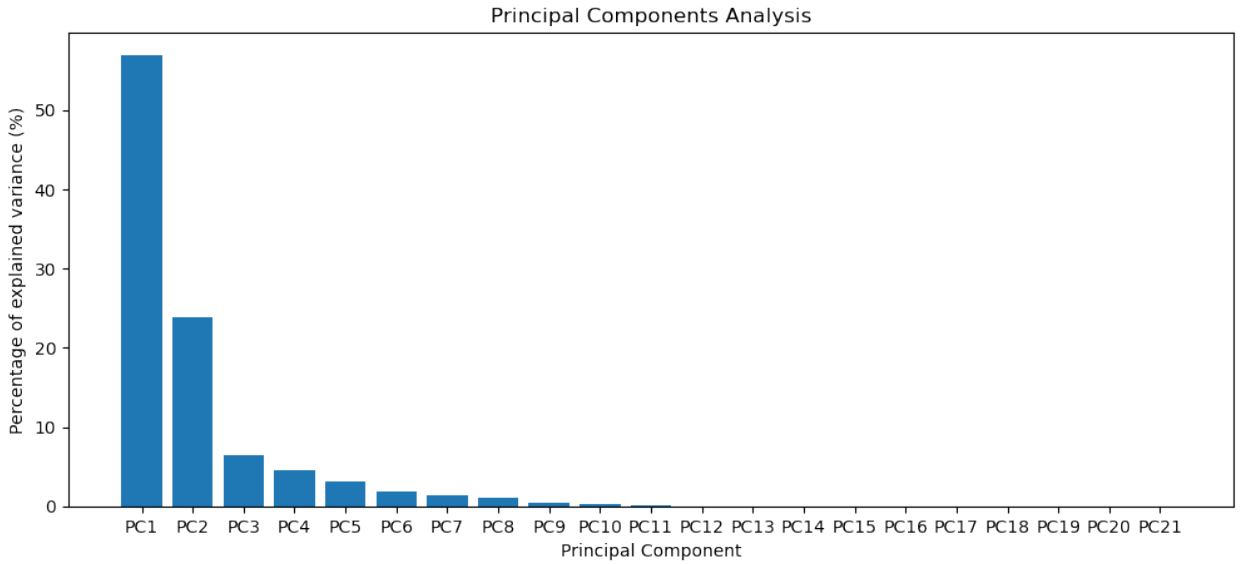


Figure 6.4: Principal Components Analysis results in the Electricity Market problem. Each bar shows the percentage of variance in the original dataset that is explained by each principal component.

To apply PCA to reduce the state space of the Electricity Market problem, we followed the

flow introduced in [120]. We first ran the GR4SP simulation model in all scenarios from the training scenario set to collect demo trajectories. Then, we characterized each model state in these trajectories by using the average of each performance indicator over the last 5 years. In this way, each state was now described by 21 average performance indicators. After that, we treated all these states as the data entities in a dataset and the 21 average indicators as their attributes, and performed PCA on them, which ended up with 21 principal components. Figure 6.4 shows the percentage of variance that was explained by each of these components. Based on the results, we decided to use the first 7 components as the final state variables because they explained more than 98% variance in those demo trajectories. To achieve this, we extracted the transformer used in our PCA to transform the 21 indicators into the first 7 components. And in every step of the RL learning process, the agent used this transformer to transform the performance indicators produced by the GR4SP simulation model into the 7 state variables in the same way, and used them to characterize the current model state.

6.2.3 Problem Formalization

Finally, we formalized our simplified Electricity Market problem as MDPs based on the simplified GR4SP simulation model as follows. The simulation settings specified in the YAML file were considered as the parameters in our MDPs, except for the simulation date range. They were all set to their nominal values, except for the 7 uncertain parameters. The time horizon of each episode in this problem was 24 years, starting on 01/01/2019 and ending on 01/01/2043. We chose this start date because the database used by the GR4SP model contained historical data before 2019 [114], so we treated them as the known past and planned for the future. We chose this end date because the existing data might not be used to predict the farther future accurately, where the installation of new generators was unpredictable. We assumed the time-step between two consecutive model states was 5 years. Therefore, each episode always contained 5 steps, corresponding to 2019-2023, 2024-2028, 2029-2033, 2034-2038, and 2039-2042, and the agent took action at the beginning of each step. Moreover, the uncertain parameters were fixed to their nominal values before 2019 so that the initial model state was fixed. Each state was described by the 7 principal components, which were transformed from the 21 average performance indicators over the corresponding step. Finally, we embedded the simplified GR4SP simulation model directly into our MDPs as the transition function. We will discuss this in more detail in Section 6.2.4.

In this experiment, the GR4SP model served the simulation purpose, providing a platform for evaluating the performance of policy actions in achieving the objectives of the Electricity Market problem. To define these action options, we referred to two existing schemes issued by the Australian government:

- **Carbon Pricing** requires electricity generators to pay extra taxes on the carbon dioxide they produce [123]. This tax price was $23 per tonne of carbon dioxide equivalent (tCO2-e) in 2012-2013, and increased in subsequent years due to inflation. This scheme increases the cost of electricity generations with carbon-intensive energy, thereby increasing their bid prices and reducing their competitiveness in the NEM, to achieve emission reductions.
- **Renewable Energy Target** requires electricity buyers to buy a target percentage of electricity generated from renewable energy sources every year [124]. The Australian Victoria government has set this target to 25% by 2020 and 50% by 2030 [125]. This scheme achieves emission reductions by promoting the competitiveness of renewable energy electricity generations in the NEM.

In addition to these schemes, we also designed another one according to the nature of the NEM:

- **Emission-Order Based Dispatch** requires AEMO to organize bids and dispatch generators following a merit order of emission factors rather than prices for each trading interval in the NEM. This scheme promotes the competitiveness of low-emission electricity generations, to achieve emission reductions.

According to these schemes, we designed the following 11 action options for our Electricity Market problem (Table 6.3).

Indices	Action Option Descriptions
1	Set the carbon tax price to $23 per tCO2-e
2	Set the carbon tax price to $13 per tCO2-e
3	Set the carbon tax price to $3 per tCO2-e
4	Set the carbon tax price to $0 per tCO2-e
5	Organize bids and dispatch generators following a merit order of prices
6	Organize bids and dispatch generators following a merit order of emission factors
7	Set the renewable energy target to 30%
8	Set the renewable energy target to 50%
9	Set the renewable energy target to 70%
10	Set the renewable energy target to 0%
11	No Action

Table 6.3: Action options in our Electricity Market problem.

The parameter uncertainty of this problem was constituted by the 7 uncertain parameters. We used their possible value ranges specified in the source model as the testing parameter value ranges and self-defined narrower ranges for training to simulate model misspecification (Table 6.4). We

Parameters	Training value ranges	Testing value ranges
includePublicallyAnnouncedGen	[0..1]	[0..1]
generationRolloutPeriod	[1..5]	[1..10]
generatorRetirement	[-3..3]	[-5..5]
priceChangePercentageBrownCoal	[-25..25]	[-50..50]
priceChangePercentageWind	[-25..25]	[-50..50]
nameplateCapacityChangeBrownCoal	[-25..25]	[-50..50]
nameplateCapacityChangeWind	[-25..25]	[-50..50]

Table 6.4: Uncertain parameters and their corresponding value ranges in the Electricity Market problem, adapted from [108]. We defined these training parameter value ranges by setting them to about 1/2 of the corresponding testing ranges. This number was chosen arbitrarily.

also defined two objectives with no assumption about their preferences, which we referred to as *price-reduction* and *emission-reduction*. These objectives were measured as accumulated reductions of *'Primary Wholesale ($/MWh)'* and *'GHG Emissions (tCO2-e) per household'* in the actual trajectory between 01/01/2019 and 01/01/2043, compared with the nominal trajectory. The nominal trajectory referred to the simulation outcomes generated by the GR4SP model using random seed θ, nominal parameter values and 24-hour intervals, as introduced in Section 6.2.2. The actual trajectory referred to the outcomes of the interaction between the algorithm policy and its corresponding model in possible scenarios, during the exploration or evaluation process. Given these definitions, the fitness function of the EAs was defined as

$$\textit{price-reduction} = \sum_{i=2019}^{2042} \big(\textit{NominalPrice}(i) - \textit{ActualPrice}(i)\big)$$

$$\textit{emission-reduction} = \sum_{i=2019}^{2042} \big(\textit{NominalEmission}(i) - \textit{ActualEmission}(i)\big)$$

where $\textit{NominalPrice}(i)$ and $\textit{ActualPrice}(i)$ refer to *'Primary Wholesale ($/MWh)'* in year i in the nominal and actual trajectories, while $\textit{NominalEmission}(i)$ and $\textit{ActualEmission}(i)$ refer to *'GHG Emissions (tCO2-e) per household'* in year i in these two trajectories respectively. Similarly, the reward function of the RL algorithms was defined as the weighted sum of *price-reduction* and *emission-reduction* over each step. Finally, this problem did not involve model uncertainty.

6.2.4 Model Implementation

To embed the GR4SP simulation model as the transition function into our MDPs, we constructed the latter as interfaces of the former. To achieve this, we first implemented two of the three functions required by MDP (*'reset'* and *'step'*, Section 3.1.2) in the GR4SP model, which we referred to as *'gr4sp-reset'* and *'gr4sp-step'*. These two functions were defined as follows: (1) *'gr4sp-reset'* reads a scenario as input, restores the model from the data-preloading checkpoint (Section 6.2.2), executes the simulation until 01/01/2019, and then sets the uncertain parameter values according to the input scenario; (2) *'gr4sp-step'* reads and applies an input action to the model, and executes the simulation for another 5 years or until 01/01/2043. Eventually, both functions return the 21 average performance indicators over the last 5 years before their termination. Then, the MDP was implemented as follows: (1) *'init'* reads the nominal trajectory (Section 6.2.2), PCA transformer (Section 6.2.2), training scenario set, and a YAML file that specifies the nominal values of all the parameters into the model (Section 6.2.1); (2) *'reset'* initializes the GR4SP simulation model program with the YAML file, calls its *'gr4sp-reset'* with the scenario chosen by the algorithm, and transforms the return of *'gr4sp-reset'* into the 7 principal components to describe the initial state; (3) *'step'* calls *'gr4sp-step'* with the input action, and computes the state variables, reward and termination flag based on the return of *'gr4sp-step'*. It is also worth noting that our MDPs were implemented in Python but the GR4SP model was implemented in Java. The integration of these two programs was supported by a Python package, namely JPype [126].

6.3 Experimental Setup

In this experiment, we applied four algorithms to solve the Electricity Market problem for comparison, which were ε-NSGA-II, Robust DQN, EPOpt and Deterministic DQN (Section 4.2.2). We excluded Borg because its implementation in EMA Workbench does not support discrete action space, and our previous experiments had showed that its performance was similar to that of ε-NSGA-II. We excluded DQN-URBE because of its intractability in such a complex problem, which we will discuss in detail later. Furthermore, due to the time limit and the characteristics of our MOREA and MORRL algorithms, we still ran ε-NSGA-II once and each RL algorithm 10 times with different weights for the two objectives. Table 6.5 details the experimental setups.

Setups	Descriptions	Values
Training scenario set size	Size of the training scenario set (Section 3.1.4)	200 (Figure 8.5)
Random seeds	Random seeds used to control the randomness of the exploration processes	α (Table 8.1)
Weight vectors	Weight vectors used in the MDP reward function of the RL algorithms	(1,10), (1,20), (1,30), (1,40), (1,50), (1,60), (1,70), (1,80), (1,90), (1,100)
Number of CPUs	Number of CPUs used to support the exploration processes of the algorithms	75
ε-NSGA-II	Number of function evaluations taken in the exploration process of ε-NSGA-II	21,000 function evaluations
Robust DQN	Number of iterations taken in the exploration processes of Robust DQN	500 iterations
EPOpt	Number of iterations taken in the exploration processes of EPOpt	500 iterations
Deterministic DQN	Number of iterations taken in the exploration processes of Deterministic DQN	500 iterations

Table 6.5: Experimental setups for the Electricity Market problem. In this experiment, we ran ε-NSGA-II once but each RL algorithms 10 times with different weight vectors. In each of these vector, the first value is the weight for *price-reduction* and the second value is the weight for *emission-reduction* in the MDP reward function of the RL algorithms. Moreover, we allocated a sufficient computational budget for each algorithm to converge its exploration process, in order to yield a fair comparison.

6.4 Results

6.4.1 Robustness Evaluation

In this experiment, we used *No Action* as the baseline policy, which referred to applying no action to the market. Compared with the other algorithm policies, *No Action* simultaneously maximized average *price-reduction* and minimized average *emission-reduction* (Figure 6.5). However, we observed that the performance of *No Action* greatly differed from that of *Nominal Trajectory* regarding both objectives. This difference was mainly caused by the parameter uncertainty, especially for *'includePubliccallyAnnouncedGen'* and *'generationRolloutPeriod'*, and the use of different random seeds for *Nominal Trajectory* and our evaluation. For example, the nominal value of *'includePubliccallyAnnouncedGen'* was 0, but its possible value range was [0..1]. Setting *'includePubliccallyAn-*

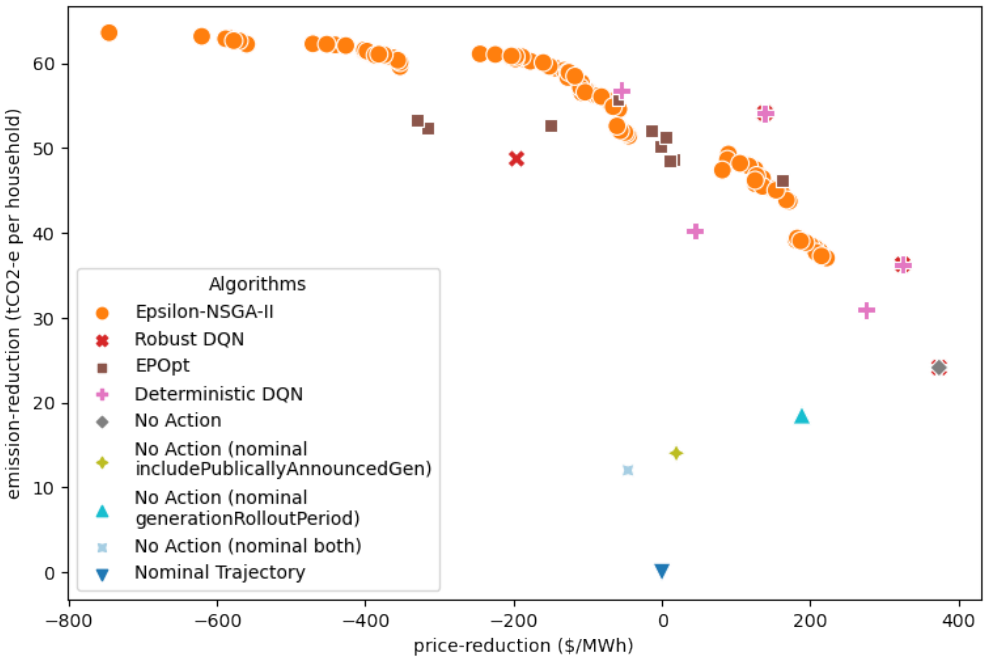


Figure 6.5: Policy trade-off distributions in the Electricity Market problem. Each point refers to the average performance of a policy of an algorithm over the parameter uncertainty space, concerning *price-reduction* and *emission-reduction*. Note that negative *price-reduction* means the actual electricity price is higher than the nominal price. Moreover, *Nominal Trajectory* refers to the 24-hour nominal trajectory (Section 6.2.2). Therefore, its *price-reduction* and *emission-reduction* are both 0.

nouncedGen' to 1 was more conducive to reducing electricity prices and GHGE in the market. Therefore, although no action was applied, *No Action* still provided a higher average performance in evaluation compared with *Nominal Trajectory*. It can be observed that by fixing these two uncertain parameters to their nominal values, the performance of *No Action* moved significantly towards that of *Nominal Trajectory*.

ε-NSGA-II identified many policies (105) in one run, providing various trade-offs between the two objectives (Figure 6.5). Most of these policies were mutually non-dominated, which averagely outperformed each other regarding one of the objectives. Moreover, they also provided similar *emission-reduction* variances but much smaller *price-reduction* variances compared with all other policies (Figure 6.6). The trade-off distribution of ε-NSGA-II had the largest range but consisted of several discrete clusters. The trade-offs in each cluster were densely distributed, but relatively different from those in different clusters. Nevertheless, ε-NSGA-II identified no policy providing average *price-reduction* higher than 300 ($/MWh), although such a static policy existed (*No Action*).

Robust and Deterministic DQNs performed similarly. Like ε-NSGA-II, not all of their policies were mutually non-dominated (Figure 6.5). Some of them also dominated some EA policies,

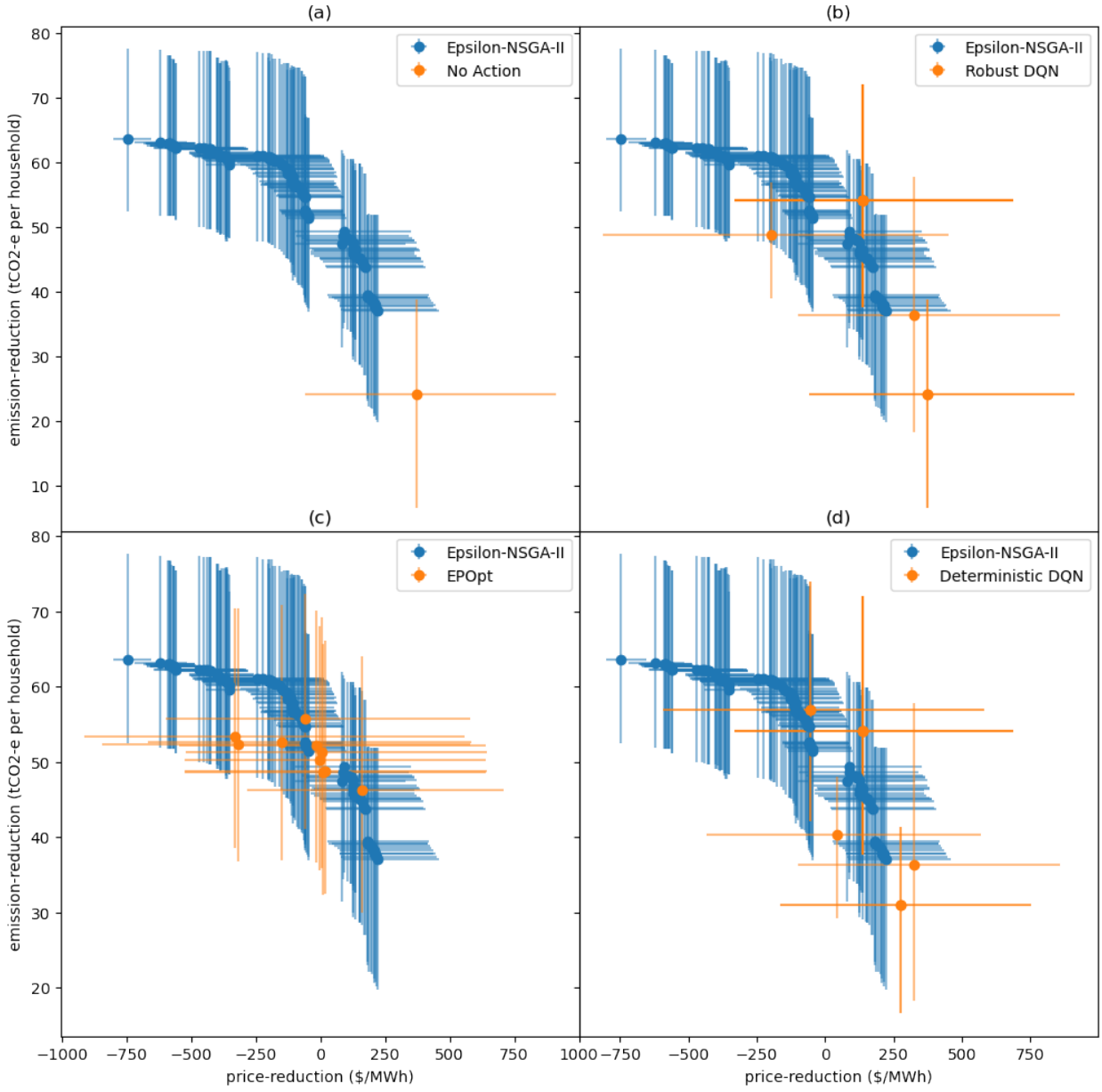


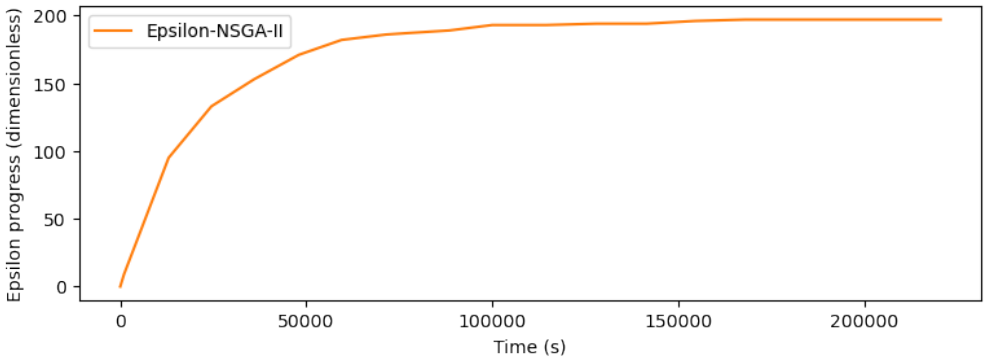
Figure 6.6: Policy trade-off distributions and their performance variances in the Electricity Market problem. Each point refers to the average performance of a policy of an algorithm over the parameter uncertainty space concerning *price-reduction* and *emission-reduction*, and each error bar connected with the point refers to the corresponding interquartile range concerning *price-reduction* or *emission-reduction*. Note that negative *price-reduction* means the actual electricity price is higher than the nominal price.

while the others might be dominated. Meanwhile, these policies performed very differently in their performance variances (Figure 6.6 (b,d)). Robust and Deterministic DQNs also provided the fewest and most sparsely distributed different trade-offs. They both identified policies providing average *price-reduction* higher than 400 ($/MWh), but only Robust DQN learned the policy of doing nothing to maximize *price-reduction* and minimize *emission-reduction*. The EPOpt policies were not always mutually non-dominated either (Figure 6.5). Some of them were dominated by some EA policies, while the others were non-dominated. Different EPOpt policies had similar *emission-reduction* variances, but relatively different *price-reduction* variances (Figure 6.6 (c)). The trade-off distribution of EPOpt had the smallest range and a mediocre density, which mainly included the trade-offs between the two clusters in the trade-off distribution of ε-NSGA-II.

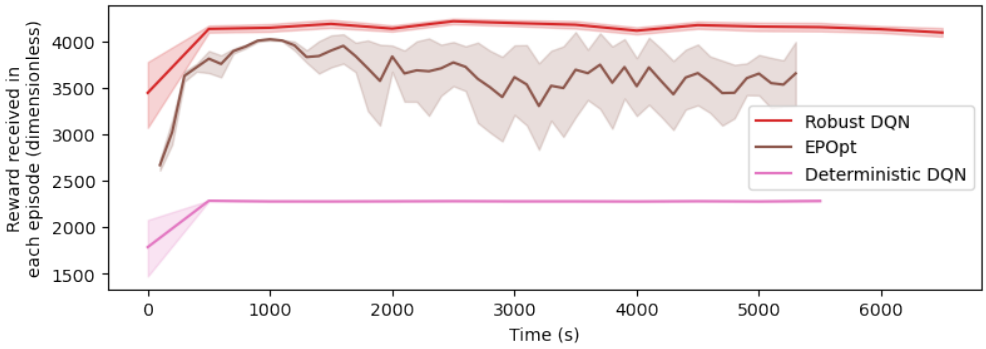
Next, we assumed the true weights for *price-reduction* and *emission-reduction* were 1 and 70 respectively, and compared the corresponding policies of our algorithms (Table 6.6). We chose these numbers because Robust and Deterministic DQNs learned the same policy in this case, so we could focus on the comparison between other algorithms. Here, we observed that the Robust and Deterministic DQN policy statically took Action 1 in each step in any scenario, which dominated the EPOpt policy that made decisions adaptively. The ε-NSGA-II policy was similar to this model-free DQN policy, except that it took Action 5 in the first step and Action 4 in the third step. Nevertheless, these two non-adaptive policies were still mutually non-dominated. In fact, we also noticed that Action 5 was always taken in the first step in any ε-NSGA-II policy, but never taken in the RL policies.

Algorithms	Policies	Average *price-reduction*	Average *emission-reduction*
ε-NSGA-II	[5,1,4,1,1]	-188.8	60.8
Robust DQN	[1,1,1,1,1]	138.8	54.1
EPOpt	Adaptive policy	-148.4	52.3
Deterministic DQN	[1,1,1,1,1]	138.8	54.1

Table 6.6: Corresponding policies of our algorithms and the average trade-offs they provided when the true weights for *price-reduction* and *emission-reduction* were 1 and 70 respectively. Here, most policies are represented as sequences of numbers. These numbers refer to the actions in Table 6.3. The EPOpt policy is an adaptive policy, which adapts its actions to the actual model state in real-time. Note that negative *price-reduction* means the actual electricity price is higher than the nominal price.



(a)



(b)

Figure 6.7: ε-progress curve for ε-NSGA-II (a), learning curves for the RL algorithms when using a weight vector of (1, 70) (b), and their 95% confidence intervals in the Electricity Market problem. Each point on the curve represents the value averaged over 500 seconds.

6.4.2 Efficiency Evaluation

In this experiment, Robust and Deterministic DQNs provided the shortest convergence time, which was slightly shorter than that of EPOpt and significantly shorter than that of ε-NSGA-II (Figure 6.7). Additionally, compared with the multi-objective Lake problem (Figure 5.9), ε-NSGA-II made much fewer ε-progresses but took much longer to converge in this experiment, but the convergence time of the RL algorithms did not change obviously.

6.5 Discussion

Based on our experimental results and interpretation methods (Section 3.4), we argue that ε-NSGA-II generally provided the best average policy performance, robustness to the parameter and objective uncertainties, and computational efficiency in this experiment. The only shortcoming of ε-NSGA-II was that its policies did not provide all possible trade-offs between the objectives, which was complemented by the RL algorithms to some extent.

6.5.1 Average Policy Performance And Robustness To Parameter Uncertainty

According to the policy dominance relationship of the algorithms (Figure 6.5), we argue that Robust, Deterministic DQNs and ε-NSGA-II provided similar average policy performance, while EPOpt underperformed them in this aspect. Additionally, ε-NSGA-II provided the best robustness to the parameter uncertainty, because it also had the smallest performance variances, especially for *price-reduction* (Figure 6.6). This might be because the EA policies always started with Action 5, but the others did not.

It is worth noting that simply introducing the uncertainty of *'includePublicallyAnnouncedGen'* and *'generationRolloutPeriod'* significantly improved the average performance of *No Action*. This observation indicated that the parameter uncertainty of this problem had a great impact on the model performance, which might be greater than those of some action options. This fact greatly challenged the exploration method to provide robustness in this deep uncertainty problem. One of its effects was that it exacerbated the impact of the model misspecification on the policy performance. When the actual parameter value fell outside the range estimated for exploration, the actual scenario might be very different from the training scenarios. In this case, the policy was less likely to perform satisfactorily. For example, some ε-NSGA-II policies were dominated by the others (Figure 6.5). This meant ε-NSGA-II wasted time identifying non-Pareto-optimal solutions in this experiment, which compromised its overall efficiency. This issue could be alleviated by eliminating the model misspecification (Figure 6.8). However, all the ε-NSGA-II policies were mutually non-dominated in the multi-objective Lake problem, even considering the model misspecification (Figure 5.7). This was mainly because the parameter uncertainty had less impact on the model performance in that problem.

6.5.2 Robustness To Objective Uncertainty

ε-NSGA-II also provided the best robustness to the objective uncertainty, because its trade-off distribution simultaneously had the largest range and density (Figure 6.5). However, this distribution was clustered, whereas the distributions of the EAs were more spread out along a curve in the multi-objective Lake problem (Figure 5.7). This was mainly because the action options in the Electricity Market problem were discrete and had very different impacts on the model performance. EPOpt complemented ε-NSGA-II in this respect with its dynamic policies, which were shown able to provide trade-offs between two clusters in the trade-off distribution of ε-NSGA-II. Additionally, ε-NSGA-II also failed to identify all static Pareto-optimal solutions in this problem (for example, *No Action* or [1,1,1,1,1], Table 6.6), and this was not caused by the model misspecification (Figure 6.8). This might indicate that, compared with this experiment, ε-NSGA-II would

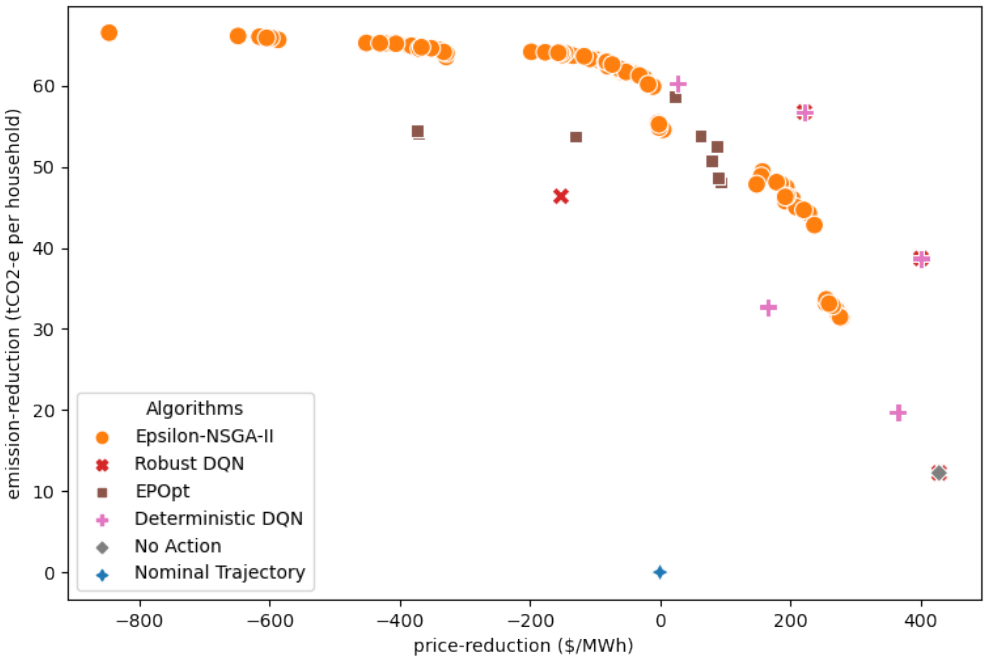


Figure 6.8: Policy trade-off distributions in the Electricity Market problem without model misspecification. Each point refers to the average performance of a policy of an algorithm over the parameter uncertainty space, concerning *price-reduction* and *emission-reduction*. Note that negative *price-reduction* means the actual electricity price is higher than the nominal price. It can be observed that almost all ε-NSGA-II policies are mutually non-dominated. Nevertheless, they still did not provide trade-offs similar to that of *No Action* or [1,1,1,1,1]. Therefore, we argue that ε-NSGA-II was not able to identify all static Pareto-optimal solutions in this experiment even without model misspecification.

actually take longer to fully converge in this Electricity Market problem, but further experiments are needed to verify this. Robust and Deterministic DQNs complemented ε-NSGA-II in this respect. For example, they learned the policy of [1,1,1,1,1], which even dominated some ε-NSGA-II policies. Eventually, the poor robustness of the RL algorithms to the objective uncertainty was also explained by their two weaknesses discussed in Section 5.3.3.

6.5.3 Policy

As introduced in Section 1.1, EAs produce static policies, which are fixed sequences of actions. RL algorithms produce dynamic policies, which are functions that map actual states to actions in real-time. However, in this experiment, Robust and Deterministic DQNs learned some policies performing statically regardless of the actual model state, for which there were two possible reasons. First, Robust and Deterministic DQNs did not form a comprehensive view of the uncertainty space to correctly guide their learning processes. Robust DQN learned a robust policy by interacting with a scenario randomly sampled from the training scenario set in each episode (Section 3.1.2). Since

the parameter uncertainty of this problem had a great impact on the model performance, Robust DQN might try a bad policy in an opportune scenario and receive a good reward, and try a good policy in a dire scenario and receive a bad reward. In this case, the experiences gained by its agent in different episodes might conflict with each other, which prevented Robust DQN from learning policies performing adaptively. Deterministic DQN did not hold a view of the uncertainty space at all because it 'predict-then-act', hence more likely to learn policies that performed statically, as we discussed in the complex-objective Lake problem. In contrast, EPOpt was free from this issue, because it always evaluated its current policy over the entire training scenario set and used the 10% worst scenarios for the policy update. This observation indicated a great weakness of the *Robust Model-Free* idea (Section 2.5.3). Second, the GR4SP simulation model is highly complex, so the state variables we used might not be able to accurately characterize its states. If there were two states with similar state variable values but different internal attributes, the RL agent would not be able to distinguish them and also gain inconsistent experiences from interactions with them. This prevented our RL algorithms from learning to map these model states to correct actions in this experiment. This might also explain why EPOpt performed the worst in this problem.

6.5.4 Efficiency

In this experiment, ε-NSGA-II provided the highest computational efficiency. Although it converged the slowest in one run (Figure 6.7 (a)), it identified many Pareto-optimal solutions (105) in this process, while each RL algorithm identified one solution only. Robust and Deterministic DQNs were more efficient than EPOpt, because they had shorter convergence time (Figure 6.7 (b)). Additionally, compared with the model of the multi-objective Lake problem, the running time of the GR4SP simulation model was much longer. This explained why ε-NSGA-II took fewer ε-progresses but longer to converge in this experiment. Meanwhile, this also indicated that the RL algorithms failed to obviously improve their policy performance during their learning processes, for the two reasons we discussed in the last section.

We excluded DQN-URBE from this experiment because its application in this problem was intractable. As mentioned in Section 3.1.2, RMDP in DQN-URBE describes parameter uncertainty by sampling possible scenarios in every model step [18], so this process occurs much more frequently than those in the other algorithms. Hence, in the original implementation of DQN-URBE, this process was designed to work sequentially to avoid wasting time in frequent parallelization [18]. This strategy is applicable in simple problems such as the Cartpole problem, where the model transition is much less computationally intensive than the policy update in DQN-URBE, but not in this Electricity Market problem. Moreover, we found that parallelization could not facilitate this process either. When using the multiprocessing package in Python to parallelize the RMDP of

this problem, we needed to create a copy of the GR4SP model for each processor to work independently at each step. However, this copying process was also extremely time-consuming for such a complex model. In conclusion, we argue that DQN-URBE is only applicable in simple problems until a more efficient RMDP implementation is proposed.

6.6 Reflections On MORRL Versus MOREA for Complex Problems

Overall, this experiment revealed the applicability of MORRL and MOREA in the highly complex deep uncertainty problem, for which we mainly made the following two observations.

First, MORRL is less flexible than MOREA. ε-NSGA-II evaluates the fitness of its candidate policies for evolution based on their complete trajectories, which are collected from their interactions with the model from the initial state to the terminal state. In contrast, the MORRL agents update their policies by interacting with the model step by step to obtain rewards during their learning processes. Hence, in this Electricity Market problem, we could have directly embedded the original GR4SP simulation model in an EMA model to support the exploration process of ε-NSGA-II, but we had to modify the GR4SP model to make it work step-wisely to support the MORRL algorithms. This fact compromises the problem conceptualization efficiency and fault tolerance of MORRL in many problems, where decision-makers must have a good understanding of the source system model and put extra effort to reconstruct it correctly before applying MORRL.

Second, the problem complexity might greatly increase the difficulty of producing good dynamic policies. Compared with MOREA, MORRL further requires the specification of state variables that accurately characterize the system state as input for learning the dynamic policy. However, it is often difficult to find such appropriate variables without adding too much unnecessary complexity in the complex problem. For example, we used 7 principal components as the state variables in this experiment. However, it turned out these components did not characterize the model state accurately enough, which compromised the average policy performance and robustness to parameter uncertainty of our MORRL algorithms. To overcome this issue, we might have to use the original 21 performance indicators, or even some internal attributes that are not included in the outputs of the GR4SP model as the state variables. This also requires decision-makers to put much more effort in the problem conceptualization. Moreover, if too many state variables must be used to characterize the system state accurately, the computational efficiency of MORRL will also be compromised.

In conclusion, given these issues and the nature of 'naive' multi-policy MORRL that provides poor efficiency and robustness to objective uncertainty, we argue that MOREA is more applica-

ble in decision-making problems with high complex and deep uncertainty. Nevertheless, these discussions should inspire future directions for robust planning research in the MORRL area.

Chapter 7

Conclusion

7.1 Conclusion

Exploratory model-based DMDU approaches have been proposed to support the decision-making process in deep uncertainty problems. These approaches are centered on exploration, so their performance is closely related to their used exploration methods. In this research project, we successfully introduced MORRL as an alternative exploration method into the field of DMDU. The viability of this application was empirically proved, by showing that MORRL could be used for exploration to solve deep uncertainty problems, while meeting the two requirements for DMDU: (1) accept, understand and manage uncertainties; (2) adjust its plan adaptively as the future unfolds. Moreover, we also showed that MORRL was complementary to some existing methods, because they provided different strengths and weaknesses in terms of efficiency and robustness. Hence, joint use of them is recommended in the DMDU approach to promote its overall performance. To achieve this, we constructed a common environment to compare the performance of MORRL and MOREA, which served as a representative of existing DMDU exploration methods, in one uncertainty and two deep uncertainty problems.

In Chapter 4, we investigated the main difference between MORRL and MOREA where MORRL provided real-time policy adaptability in the Cartpole problem. Relying on this adaptability, MORRL generally provided much higher efficiency and robustness to parameter uncertainty than MOREA. MORRL was also able to handle random initial states that MOREA could not deal with. The only advantage of MOREA was that its performance was less sensitive to the exploration randomness, so its policies could perform better in the worst case.

In Chapter 5, we compared the performance of MORRL and MOREA in DMDU by using the Lake problem. We found that the policy adaptability also made MORRL more robust to parameter uncertainty and model misspecification in this problem. Additionally, these two methods provided different advantages in terms of robustness to objective uncertainty. The main advantage

of MOREA was that it was more efficient in handling multiple objectives, because it could identify multiple Pareto-optimal solutions in one run.

In Chapter 6, we demonstrated their applications and compared their performance in a real-world decision-making problem with high complexity and deep uncertainty, namely the Electricity Market problem. MORRL was still applicable in this problem, but the high complexity compromised its problem conceptualization efficiency and policy adaptability. In contrast, MOREA was more flexible, and provided higher efficiency and robustness to parameter uncertainty. Nevertheless, MORRL and MOREA still showed different advantages in handling objective uncertainty. This experiment proposed important directions for future research in the MORRL area. Table 7.1 summaries the key findings drawn from these experiments.

Aspects	MORRL	MOREA
Model	(1) Simulation model (2) Works step by step (3) Specifies state variables (4) Specifies possible objective preferences	(1) Simulation model (2) Works from start to end (3) No need for state variables (4) Objective function without preferences
Output policy	(1) Dynamic policy (2) A function that adapts actions to current states in real-time	(1) Static policy (2) A fixed sequence of actions
Robustness to parameter uncertainty	(1) High (2) Relies on real-time policy adaptability (3) Sensitive to exploration randomness	(1) Low (2) Relies on cycling through the planning process to adapt to new scenarios
Efficiency in handling parameter uncertainty	(1) High (2) Shorter convergence time (3) Adapts to more scenarios before a new policy needs to be reworked	(1) Low (2) Longer convergence time (3) Adapts to fewer scenarios before a new policy needs to be reworked
Robustness to objective uncertainty (Section 3.4)	(1) Large trade-off distribution range (2) Sparse trade-off distribution	(1) Small trade-off distribution range (2) Dense trade-off distribution
Efficiency in handling objective uncertainty	(1) Low (2) At most one Pareto-optimal solution per run	(1) High (2) Multiple Pareto-optimal solutions per run
Sensitive to exploration randomness	High	Low

Table 7.1: Summary of the key findings of the comparison between MORRL and MOREA.

7.2 Strengths

This project has the following five strengths. First, compared with previous studies of Robust RL, we introduced the concept of stabilization analysis into our experiments (Section 3.1.4). This tool provides an appropriate way to describe the model and parameter uncertainties of the problem without introducing extra noise into this process. Second, we constructed a common environment for the implementation and application of MORRL and MOREA in DMDU, which enhanced the fairness of the comparison. Third, we fully controlled the exploration randomness in our experiments. This not only guaranteed the reproducibility of these experiments, but also reflected the impact of randomness on the algorithm performance. Fourth, we tested MORRL and MOREA in problems that come from different research fields, and involve various characteristics and uncertainties. This helped to make a comprehensive evaluation of their performance under different conditions. Finally, we also evaluated their robustness using multiple robustness metrics, which helped to comprehensively reflect their robustness from different aspects.

7.3 Limitations

The time limit of this project also leads to the following three limitations. First, we used the default hyperparameter values of the algorithms in our experiments, but it might still bring some algorithms unfair disadvantages. Ideally, we should tweak the hyperparameters of each algorithm to obtain its optimal performance for comparison in every case study. Second, due to the nature of 'naive' multi-policy MORRL, we were only able to apply each RL algorithm to produce a small number of policies in the multi-objective case studies. This prevented us from drawing a comprehensive conclusion on their robustness to the entire uncertainty space. Finally, we also noticed the robustness to parameter uncertainty of MORRL was generally compromised in multi-objective problems, but we have not conducted further experiments to explain this change.

7.4 Future Directions

This project revealed the differences in characteristics of domain problems and algorithms in the MORRL area and DMDU field. On this basis, we identified the following three directions for future research. First, to maintain a common environment to compare our algorithms, this project only investigated offline planning (MORRL). However, existing studies have also proposed online planning algorithms for decision-making under various uncertainties. It is worth studying the performance of these online algorithms in DMDU in future research, in order to introduce them into the DMDU field and further complement the existing exploration methods. Second, as we

mentioned before, no previous work has studied the integration of multi-policy MORL and Robust RL to deal with deep uncertainty problems, and our 'naive' multi-policy MORRL has been shown inefficient in handling objective uncertainty. Hence, future research is needed to propose a more proper integration of multi-policy MORL and Robust RL to address this knowledge gap. Based on our work, one possible idea is to introduce the non-dominated sorting adopted by MOREA to Robust RL, so that it can identify multiple robust Pareto-optimal solutions in one run, without requiring prior prediction of possible objective preferences from the decision-maker. Lastly, our experiments showed that MORRL was less applicable in practical complex decision-making problems, because it required the source model to work step by step and specify state variables that can accurately characterize the system state to support its learning. Therefore, the overall performance of MORRL might be compromised in those problems without this functionality or information. Future work is needed to design new MORRL algorithms to overcome this shortcoming.

References

[1] W. E. Walker, R. J. Lempert, and J. H. Kwakkel, "Deep uncertainty," in *Encyclopedia of Operations Research and Management Science*, S. I. Gass and M. C. Fu, Eds. Boston, MA: Springer US, 2013, pp. 395–402, ISBN: 978-1-4419-1153-7. DOI: 10.1007/978-1-4419-1153-7_1140. [Online]. Available: https://doi.org/10.1007/978-1-4419-1153-7_1140.

[2] C. Hamarat, J. H. Kwakkel, and E. Pruyt, "Adaptive robust design under deep uncertainty," *Technological Forecasting and Social Change*, vol. 80, no. 3, pp. 408–418, 2013.

[3] W. L. Auping, E. Pruyt, and J. H. Kwakkel, "Societal ageing in the netherlands: A robust system dynamics approach," *Systems Research and Behavioral Science*, vol. 32, no. 4, pp. 485–501, 2015.

[4] J. R. Kasprzyk, S. Nataraj, P. M. Reed, and R. J. Lempert, "Many objective robust decision making for complex environmental systems undergoing change," *Environmental Modelling & Software*, vol. 42, pp. 55–71, 2013.

[5] D. G. Groves, E. Bloom, R. J. Lempert, J. R. Fischbach, J. Nevills, and B. Goshi, "Developing key indicators for adaptive water planning," *Journal of Water Resources Planning and Management*, vol. 141, no. 7, p. 05 014 008, 2015.

[6] V. A. Marchau, W. E. Walker, P. J. Bloemen, and S. W. Popper, *Decision making under deep uncertainty: from theory to practice*. Springer Nature, 2019.

[7] W. E. Walker, M. Haasnoot, and J. H. Kwakkel, "Adapt or perish: A review of planning approaches for adaptation under deep uncertainty," *Sustainability*, vol. 5, no. 3, pp. 955–979, 2013.

[8] J. H. Kwakkel, M. Haasnoot, and W. E. Walker, "Developing dynamic adaptive policy pathways: A computer-assisted approach for developing adaptive strategies for a deeply uncertain world," *Climatic Change*, vol. 132, no. 3, pp. 373–386, 2015.

[9] Y. Ben-Haim, *Info-gap decision theory: decisions under severe uncertainty*. Elsevier, 2006.

[10] B. P. Bryant and R. J. Lempert, "Thinking inside the box: A participatory, computer-assisted approach to scenario discovery," *Technological Forecasting and Social Change*, vol. 77, no. 1, pp. 34–49, 2010.

[11] J. H. Kwakkel, M. Haasnoot, and W. E. Walker, "Comparing robust decision-making and dynamic adaptive policy pathways for model-based decision support under deep uncertainty," *Environmental Modelling & Software*, vol. 86, pp. 168–183, 2016.

[12] E. S. Matrosov, A. M. Woods, and J. J. Harou, "Robust decision making and info-gap decision theory for water resource system planning," *Journal of hydrology*, vol. 494, pp. 43–58, 2013.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] D. Cao, W. Hu, J. Zhao, G. Zhang, B. Zhang, Z. Liu, Z. Chen, and F. Blaabjerg, "Reinforcement learning and its applications in modern power and energy systems: A review," *Journal of Modern Power Systems and Clean Energy*, vol. 8, no. 6, pp. 1029–1042, 2020.

[15] O. Sigaud and O. Buffet, *Markov decision processes in artificial intelligence*. John Wiley & Sons, 2013.

[16] A. Perera and P. Kamalaruban, "Applications of reinforcement learning in energy systems," *Renewable and Sustainable Energy Reviews*, vol. 137, p. 110 618, 2021.

[17] T. T. Nguyen, N. D. Nguyen, P. Vamplew, S. Nahavandi, R. Dazeley, and C. P. Lim, "A multi-objective deep reinforcement learning framework," *Engineering Applications of Artificial Intelligence*, vol. 96, p. 103 915, 2020.

[18] E. Derman, D. Mankowitz, T. Mann, and S. Mannor, "A bayesian approach to robust reinforcement learning," in *Uncertainty in Artificial Intelligence*, PMLR, 2020, pp. 648–658.

[19] S. R. Carpenter, D. Ludwig, and W. A. Brock, "Management of eutrophication for lakes subject to potentially irreversible change," *Ecological applications*, vol. 9, no. 3, pp. 751–771, 1999.

[20] R. Singh, P. M. Reed, and K. Keller, "Many-objective robust decision making for managing an ecosystem with a deeply uncertain threshold response," *Ecology and Society*, vol. 20, no. 3, 2015.

[21] R. J. Lempert, "Shaping the next one hundred years: New methods for quantitative, long-term policy analysis," 2003.

[22] S. Dessai and M. Hulme, "Assessing the robustness of adaptation decisions to climate change uncertainties: A case study on water resources management in the east of england," *Global environmental change*, vol. 17, no. 1, pp. 59–72, 2007.

[23] S. Dessai and J. P. van der Sluijs, *Uncertainty and climate change adaptation: A scoping study*. Copernicus Institute for Sustainable Development and Innovation, Department …, 2007, vol. 2007.

[24] S. Hallegatte, A. Shah, C. Brown, R. Lempert, and S. Gill, "Investment decision making under deep uncertainty–application to climate change," *World Bank Policy Research Working Paper*, no. 6193, 2012.

[25] D. McInerney, R. Lempert, and K. Keller, "What are robust strategies in the face of uncertain climate threshold responses?" *Climatic change*, vol. 112, no. 3, pp. 547–568, 2012.

[26] M. Workman, G. Darch, K. Dooley, G. Lomax, J. Maltby, and H. Pollitt, "Climate policy decision making in contexts of deep uncertainty-from optimisation to robustness," *Environmental Science & Policy*, vol. 120, pp. 127–137, 2021.

[27] C. McPhail, H. Maier, J. Kwakkel, M. Giuliani, A. Castelletti, and S. Westra, "Robustness metrics: How are they calculated, when should they be used and why do they give different results?" *Earth's Future*, vol. 6, no. 2, pp. 169–191, 2018.

[28] M. Haasnoot, J. H. Kwakkel, W. E. Walker, and J. ter Maat, "Dynamic adaptive policy pathways: A method for crafting robust decisions for a deeply uncertain world," *Global environmental change*, vol. 23, no. 2, pp. 485–498, 2013.

[29] C. P. Weaver, R. J. Lempert, C. Brown, J. A. Hall, D. Revell, and D. Sarewitz, "Improving the contribution of climate model information to decision making: The value and demands of robust decision frameworks," *Wiley Interdisciplinary Reviews: Climate Change*, vol. 4, no. 1, pp. 39–60, 2013.

[30] D. G. Groves and R. J. Lempert, "A new analytic method for finding policy-relevant scenarios," *Global Environmental Change*, vol. 17, no. 1, pp. 73–85, 2007.

[31] W. E. Walker, S. A. Rahman, and J. Cave, "Adaptive policies, policy analysis, and policy-making," *European journal of operational Research*, vol. 128, no. 2, pp. 282–289, 2001.

[32] J. H. Kwakkel, W. E. Walker, and V. Marchau, "Adaptive airport strategic planning," *European Journal of Transport and Infrastructure Research*, vol. 10, no. 3, 2010.

[33] Y. Ben-Haim, *Info-gap economics: an operational introduction*. Springer, 2010.

[34] A. Dorin and T. Taylor, "Rise of the self-replicators: Early visions of machines, ai and robots that can reproduce and evolve," 2020.

[35] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC)*, IEEE, 2016, pp. 261–265.

[36] J. B. Kollat and P. M. Reed, "Comparing state-of-the-art evolutionary multi-objective algorithms for long-term groundwater monitoring design," *Advances in Water Resources*, vol. 29, no. 6, pp. 792–807, 2006.

[37] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[38] D. Hadka and P. Reed, "Borg: An auto-adaptive many-objective evolutionary computing framework," *Evolutionary computation*, vol. 21, no. 2, pp. 231–259, 2013.

[39] H. Du, Z. Wang, W. Zhan, and J. Guo, "Elitism and distance strategy for selection of evolutionary algorithms," *IEEE Access*, vol. 6, pp. 44 531–44 541, 2018.

[40] J. Knowles and D. Corne, "The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, IEEE, vol. 1, 1999, pp. 98–105.

[41] E. Zitzler, *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Citeseer, 1999, vol. 63.

[42] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining convergence and diversity in evolutionary multiobjective optimization," *Evolutionary computation*, vol. 10, no. 3, pp. 263–282, 2002.

[43] J. H. Kwakkel, *Ema workbench documentation*, 2020. [Online]. Available: `https://emaworkbench.readthedocs.io/en/latest/`.

[44] P. M. Reed, D. Hadka, J. D. Herman, J. R. Kasprzyk, and J. B. Kollat, "Evolutionary multiobjective optimization in water resources: The past, present, and future," *Advances in water resources*, vol. 51, pp. 438–456, 2013.

[45] K. Deb, M. Mohan, and B. Mishra, "A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions," *KanGAL report*, vol. 2003002, Jan. 2003.

[46] Y. Tang, P. Reed, and T. Wagener, "How effective and efficient are multiobjective evolutionary algorithms at hydrologic model calibration?" *Hydrology and earth system sciences*, vol. 10, no. 2, pp. 289–307, 2006.

[47] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[48] D. A. Berry and B. Fristedt, "Bandit problems: Sequential allocation of experiments (monographs on statistics and applied probability)," *London: Chapman and Hall*, vol. 5, no. 71-87, pp. 7–7, 1985.

[49] J. Gittins, K. Glazebrook, and R. Weber, *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

[50] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *arXiv preprint arXiv:1204.5721*, 2012.

[51] J. R. Vázquez-Canteli and Z. Nagy, "Reinforcement learning for demand response: A review of algorithms and modeling techniques," *Applied energy*, vol. 235, pp. 1072–1089, 2019.

[52] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms," *Machine learning*, vol. 84, no. 1, pp. 51–80, 2011.

[53] A. Castelletti, G. Corani, A. Rizzolli, R. Soncinie-Sessa, and E. Weber, "Reinforcement learning in the operational management of a water system," in *IFAC workshop on modeling and control in environmental issues*, Citeseer, 2002, pp. 325–330.

[54] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 41–47.

[55] T. G. Dietterich, "Steps toward robust artificial intelligence," *AI Magazine*, vol. 38, no. 3, pp. 3–24, 2017.

[56] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton university press, 2009.

[57] W. Wiesemann, D. Kuhn, and B. Rustem, "Robust markov decision processes," *Mathematics of Operations Research*, vol. 38, no. 1, pp. 153–183, 2013.

[58] G. N. Iyengar, "Robust dynamic programming," *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005.

[59] A. Nilim and L. El Ghaoui, "Robust control of markov decision processes with uncertain transition matrices," *Operations Research*, vol. 53, no. 5, pp. 780–798, 2005.

[60] C. Tessler, Y. Efroni, and S. Mannor, "Action robust reinforcement learning and applications in continuous control," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6215–6224.

[61] S. H. Lim, H. Xu, and S. Mannor, "Reinforcement learning in robust markov decision processes," *Advances in Neural Information Processing Systems*, vol. 26, pp. 701–709, 2013.

[62] ——, "Reinforcement learning in robust markov decision processes," *Mathematics of Operations Research*, vol. 41, no. 4, pp. 1325–1353, 2016.

[63] S. Mannor, O. Mebel, and H. Xu, "Robust mdps with k-rectangular uncertainty," *Mathematics of Operations Research*, vol. 41, no. 4, pp. 1484–1509, 2016.

[64] V. Goyal and J. Grand-Clement, "Robust markov decision process: Beyond rectangularity," *arXiv preprint arXiv:1811.00215*, 2018.

[65] S. D.-C. Shashua and S. Mannor, "Deep robust kalman filter," *arXiv preprint arXiv:1703.02310*, 2017.

[66] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[67] N. Vlassis, M. Ghavamzadeh, S. Mannor, and P. Poupart, "Bayesian reinforcement learning," *Reinforcement learning*, pp. 359–386, 2012.

[68] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih, "The uncertainty bellman equation and exploration," in *International Conference on Machine Learning*, 2018, pp. 3836–3845.

[69] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[70] S. Mannor, D. Simester, P. Sun, and J. N. Tsitsiklis, "Bias and variance approximation in value function estimates," *Management Science*, vol. 53, no. 2, pp. 308–322, 2007.

[71] Y. Chow, A. Tamar, S. Mannor, and M. Pavone, "Risk-sensitive and robust decision-making: A cvar optimization approach," *arXiv preprint arXiv:1506.02188*, 2015.

[72] A. Tamar, Y. Glassner, and S. Mannor, "Optimizing the cvar via sampling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

[73] A. Sharma, J. Harrison, M. Tsao, and M. Pavone, "Robust and adaptive planning under model uncertainty," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 410–418.

[74] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, "Risk-constrained reinforcement learning with percentile risk criteria," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6070–6120, 2017.

[75] R. Keramati, C. Dann, A. Tamkin, and E. Brunskill, "Being optimistic to be conservative: Quickly learning a cvar policy," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 4436–4443.

[76] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[77] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2017, pp. 2817–2826.

[78] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[79] M. B. Naghibi-Sistani, M. Akbarzadeh-Tootoonchi, M. J.-D. Bayaz, and H. Rajabi-Mashhadi, "Application of q-learning with temperature variation for bidding strategies in market based power systems," *Energy Conversion and Management*, vol. 47, no. 11-12, pp. 1529–1538, 2006.

[80] G. Li and J. Shi, "Agent-based modeling for trading wind power with uncertainty in the day-ahead wholesale electricity markets of single-sided auctions," *Applied Energy*, vol. 99, pp. 13–22, 2012.

[81] V.-H. Bui, A. Hussain, and H.-M. Kim, "Double deep *q*-learning-based distributed operation of battery energy storage system considering uncertainties," *IEEE Transactions on Smart Grid*, vol. 11, no. 1, pp. 457–469, 2019.

[82] M. Al-Saffar and P. Musilek, "Reinforcement learning-based distributed bess management for mitigating overvoltage issues in systems with high pv penetration," *IEEE Transactions on Smart Grid*, vol. 11, no. 4, pp. 2980–2994, 2020.

[83] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[84] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37.

[85] C. J. Walters and R. Hilborn, "Ecological optimization and adaptive management," *Annual review of Ecology and Systematics*, vol. 9, no. 1, pp. 157–188, 1978.

[86] M. J. Westgate, G. E. Likens, and D. B. Lindenmayer, "Adaptive management of biological systems: A review," *Biological Conservation*, vol. 158, pp. 128–139, 2013.

[87] I. Chadès, S. Nicol, T. M. Rout, M. Péron, Y. Dujardin, J.-B. Pichancourt, A. Hastings, and C. E. Hauser, "Optimization methods to solve adaptive management problems," *Theoretical Ecology*, vol. 10, no. 1, pp. 1–20, 2017.

[88] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *ArXiv*, vol. abs/1606.01540, 2016.

[89] S. Bankes, "Exploratory modeling for policy analysis," *Operations research*, vol. 41, no. 3, pp. 435–449, 1993.

[90] S. Bankes, W. E. Walker, and J. H. Kwakkel, "Exploratory modeling and analysis," *Encyclopedia of operations research and management science*, pp. 532–537, 2013.

[91] J. H. Kwakkel, W. E. Walker, and M. Haasnoot, *Coping with the wickedness of public policy problems: Approaches for decision making under deep uncertainty*, 2016.

[92] M. McKaya, R. Beckmana, and W. Conoverb, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[93] A. Maier, *The seven sins of machine learning*, Jun. 2020. [Online]. Available: https://towardsdatascience.com/the-seven-sins-of-machine-learning-54dbf63fd71d.

[94] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," *arXiv preprint arXiv:1712.09381*, p. 85, 2017.

[95] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[96] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International conference on machine learning*, PMLR, 2016, pp. 1329–1338.

[97] A. Wald, "Statistical decision functions.," 1950.

[98] L. J. Savage, "The theory of statistical decision," *Journal of the American Statistical association*, vol. 46, no. 253, pp. 55–67, 1951.

[99] J. H. Kwakkel, S. Eker, and E. Pruyt, "How robust is a robust policy? comparing alternative robustness metrics for robust decision-making," in *Robustness analysis in decision aiding, optimization, and analytics*, Springer, 2016, pp. 221–237.

[100] S. Geva and J. Sitte, "A cartpole experiment benchmark for trainable controllers," *IEEE Control Systems Magazine*, vol. 13, no. 5, pp. 40–51, 1993.

[101] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," *Center for Cognitive and Neural Studies (Coneural), Romania*, 2007.

[102] D. Hadka, J. Herman, P. Reed, and K. Keller, "An open source framework for many-objective robust decision making," *Environmental Modelling & Software*, vol. 74, pp. 114–129, 2015.

[103] J. H. Kwakkel, "The exploratory modeling workbench: An open source toolkit for exploratory modeling, scenario discovery, and (multi-objective) robust decision making," *Environmental Modelling & Software*, vol. 96, pp. 239–250, 2017.

[104] AEMO, *National electricity market (nem)*, 2021. [Online]. Available: https://www.aemo.com.au/energy-systems/electricity/national-electricity-market-nem.

[105] E. Subramanian, Y. Bichpuriya, A. Achar, S. Bhat, A. P. Singh, V. Sarangan, and A. Natarajan, "Learn: A reinforcement learning based bidding strategy for generators in single sided energy markets," in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, 2019, pp. 121–127.

[106] Department of Industry, Science, Energy and Resources, *International climate change commitments*, 2021. [Online]. Available: https://www.industry.gov.au/policies-and-initiatives/australias-climate-change-strategies/international-climate-change-commitments.

[107] ——, *National greenhouse gas inventory quarterly update: September 2020*, 2021. [Online]. Available: https://www.industry.gov.au/data-and-publications/national-greenhouse-gas-inventory-quarterly-update-september-2020.

[108] A. M. Rojas-Arevalo, "Sustainability transitions modelling and assessment of socio-technical energy systems: An australian case," In Submission., Ph.D. dissertation, The University of Melbourne, 2022.

[109] D. McConnell, P. Hearps, D. Eales, M. Sandiford, R. Dunn, M. Wright, and L. Bateman, "Retrospective modeling of the merit-order effect on wholesale electricity prices from distributed photovoltaic generation in the australian national electricity market," *Energy Policy*, vol. 58, pp. 17–27, 2013.

[110] Climate Change Authority, "Policy options for australia's electricity supply sector: Special review research report," *August 2016*, vol. 19, 2016.

[111] ——, "Modelling illustrative electricity sector emissions reduction policies," Jacobs, Tech. Rep., 2017.

[112] AEMO, *2020 integrated system plan (isp)*, 2020. [Online]. Available: https://www.aemo.com.au/energy-systems/major-publications/integrated-system-plan-isp/2020-integrated-system-plan-isp.

[113] A. Rojas and F. J. de Haan, "Socio-technical representation of electricity provision across scales," *Modelling Transitions: Virtues, Vices, Visions of the Future*, pp. 139–161, 2020.

[114] A. M. Rojas Arevalo, *Gr4sp victoria electricity system*, version 1.0, Zenodo, Apr. 2021. DOI: 10.5281/zenodo.4667997. [Online]. Available: https://doi.org/10.5281/zenodo.4667997.

[115] PROV, *Public record office victoria*, 2021. [Online]. Available: https://prov.vic.gov.au/.

[116] POV, *Parliamentary papers*, 2020. [Online]. Available: https://pov.ent.sirsidynix.net.au/client/en_GB/parl_paper/.

[117] AEMO, *Aggregated price and demand data*, 2020. [Online]. Available: https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data.

[118] AEMO, APVI, BoM, *Opennem: Victoria*, 2021. [Online]. Available: https://opennem.org.au/energy/vic1/?range=all&interval=1M.

[119] I. M. Sobol, "Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates," *Mathematics and computers in simulation*, vol. 55, no. 1-3, pp. 271–280, 2001.

[120] W. Curran, T. Brys, M. Taylor, and W. Smart, "Using pca to efficiently represent state spaces," *arXiv preprint arXiv:1505.00322*, 2015.

[121] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[122] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[123] Clean Energy Regulator, *About the mechanism*, 2021. [Online]. Available: http://www.cleanenergyregulator.gov.au/Infohub/CPM/About-the-mechanism.

[124] ——, *Large-scale renewable energy target*, 2018. [Online]. Available: http://www.cleanenergyregulator.gov.au/RET/About-the-Renewable-Energy-Target/How-the-scheme-works/Large-scale-Renewable-Energy-Target.

[125] Victoria State Government, *Victoria's renewable energy targets*, 2021. [Online]. Available: https://www.energy.vic.gov.au/renewable-energy/victorias-renewable-energy-targets.

[126] S. Menard and L. Nell, *Jpype documentation*, 2021. [Online]. Available: https://jpype.readthedocs.io/en/latest/.

Chapter 8

Appendix

8.1 Random Seeds

We used multiple random seeds to control the randomness in our experiments. Table 8.1 shows the mapping from the variable names we refer to them in this thesis to their actual values.

Random Seed Variable Names	Random Seed Values
α	3186775264
β	3690172787
γ	462638671
δ	1926216712
ζ	3087161096
η	1793476144
θ	1607932481652

Table 8.1: Random seed variable names and their corresponding values.

8.2 Stabilization Analysis

For each of our experiments, we performed a stabilization analysis to determine the appropriate size of its training scenario set, as described in Section 3.1.4. The results are shown below. The x-axes of these plots refer to the numbers of scenarios, and their y-axes refer to the robustness metrics (the product of the median and the interquartile distance plus one) for the objective indicators. Each line in these plots corresponds to the performance of a random policy. We determined the appropriate set sizes according to how many scenarios were taken to stabilize the objective robustness metrics for most policies. Although using more scenarios to describe the parameter uncertainty often increases the stability, Kwakkel suggested balancing it with the computational cost of the experiment [43].

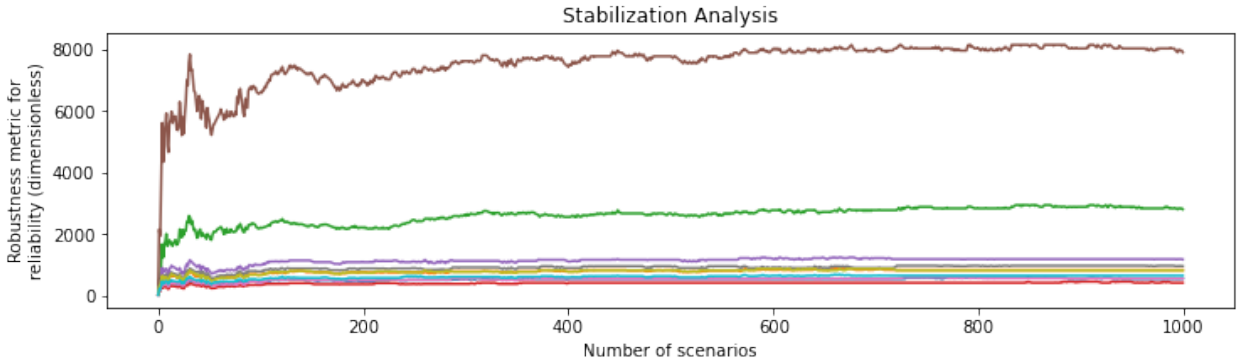


Figure 8.1: Stabilization analysis results for the single-objective robust Cartpole problem. It can be observed that for most policies, the robustness metric for *reliability* stabilized at around 100 scenarios. So we chose this number as the size of the training scenario set in this problem. We believed this set size was reasonable, compared with the set of size 5 used in [65] and the set of size 15 used in [18].

Figure 8.2: Stabilization analysis results for the complex-objective robust Cartpole problem with fixed initial state. It can be observed that for most policies, the robustness metrics for *utility* and *reliability* stabilized at around 100 scenarios. So we chose this number as the size of the training scenario set in this problem.

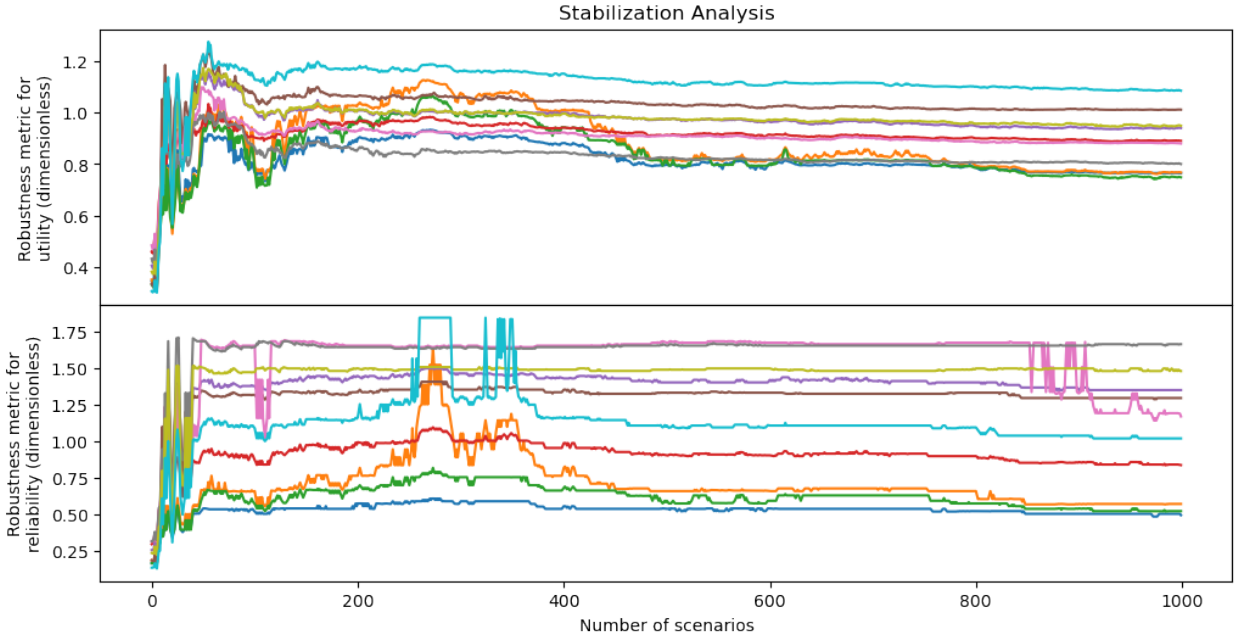


Figure 8.3: Stabilization analysis results for the complex-objective Lake problem. It can be observed that for most policies, the robustness metrics for *utility* and *reliability* stabilized at around 200 scenarios. So we chose this number as the size of the training scenario set in this problem.

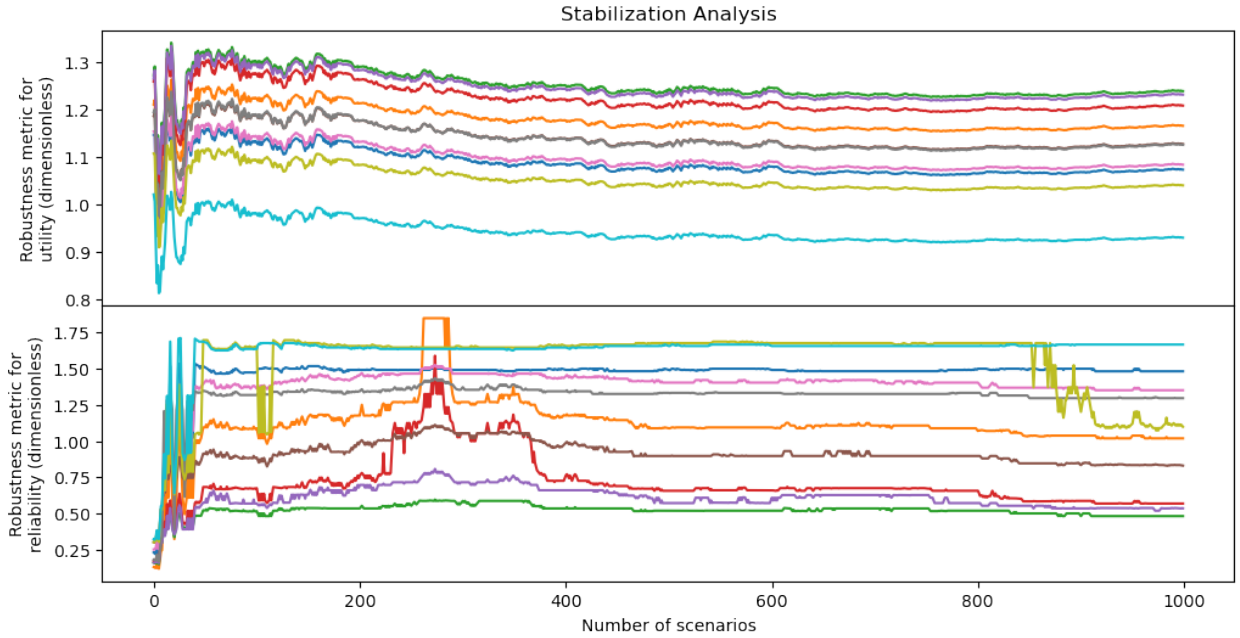


Figure 8.4: Stabilization analysis results for the multi-objective Lake problem. It can be observed that for most policies, the robustness metrics for *utility* and *reliability* stabilized at around 200 scenarios. So we chose this number as the size of the training scenario set in this problem.

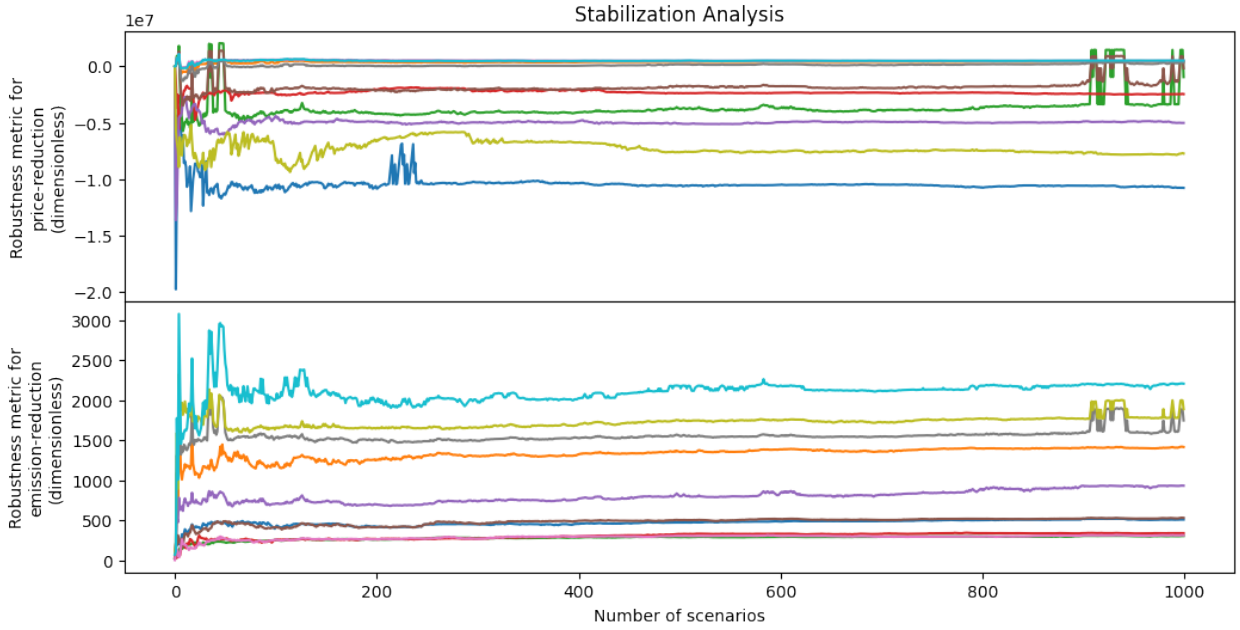


Figure 8.5: Stabilization analysis results for the Electricity Market problem. It can be observed that for most policies, the robustness metrics for *price-reduction* and *emission-reduction* stabilized at around 200 scenarios. So we chose this number as the size of the training scenario set in this problem.

8.3 Supplementary Experimental Results

Algorithms	Average performance	Safe scenario numbers
Random	Utility = 0.28 Reliability = 0.40	1531
ε-NSGA-II	Utility = 0.27 Reliability = 0.57	2584
Borg	Utility = 0.27 Reliability = 0.57	2590
Robust DQN	Utility = 0.36 Reliability = 0.77	3663
DQN-URBE	Utility = 0.40 Reliability = 0.72	3520
EPOpt	Utility = 0.18 Reliability = 0.79	3773
Deterministic DQN	Utility = 0.22 Reliability = 0.64	2698
No Pollution	Utility = 0.28 Reliability = 0.83	3882

Table 8.2: Summary of the experimental results in the complex-objective Lake problem. Here, the safe scenario number of the algorithm refers to the number of evaluation scenarios where the algorithm avoided lake eutrophication (*reliability* = 1).

8.4 Supplementary GR4SP Simulation Model Information

Parameters	Descriptions	Nominal Values	Possible Value Ranges
annualCpi	Consumer price index used to adjust future tariffs to 2019 values.	0.0233	[0.01, 0.05]
annualInflation	Annual inflation that impacts the prices of electricity offered by generators.	0.033	[0.01, 0.05]
rooftopPV	Decision variable to change the ISP forecast on uptake of rooftop photovoltaic systems in residential, business or both sectors.	both	residential, business, both
includePublically--AnnouncedGen	Decision variable to include emerging projects.	false (0)	true (1), false (0)
generation--RolloutPeriod	Number of years to roll out new generation.	1	[1..10]
generatorRetirement	Shift on closure date of brown coal power plants	0	[-5..5]
technological--Improvement	Increase the capacity factors of wind and solar generators by adding a constant factor -not compounded- every year.	0.01	[0.0, 0.1]
Learning curve	Decrement rate of the base price of wind and solar generations.	0.05	[0.0, 0.1]
priceChangePercentageX; X is the generation type	Change percentage of the nominal base electricity price for each technology	0	[-50..50]
nameplateCapacity--ChangeX; X is the generation type.	Change percentage of the nameplate capacity of generators by fuel and technology type.	0	[-50..50]
wholesaleTariff--Contribution	Percentage contributions from wholesale prices to the final electricity tariff.	0.2837	[0.10, 0.45]

Table 8.3: Uncertain parameters in the GR4SP simulation model (1) [108].

Parameters	Descriptions	Nominal Values	Possible Value Ranges
scheduleMin--CapMarketGen	Minimum nameplate capacity in MW required for schedule generation to participate in the market.	30	[0.1, 30]
semiScheduleMin--CapMarketGen	Minimum nameplate capacity in MW required for semi-schedule generation to participate in a market.	30	[0.1, 30]
nonScheduleMin--CapMarketGen	Minimum nameplate capacity in MW required for non-schedule generation to participate in a market.	0.1	[0.1, 15]
nonScheduleGen--SpotMarket	Market in which non-schedule generation can participate. Only generation with a minimum capacity previously defined can be included in the market selected.	none	none, primary, secondary
consumption	Market operator's consumption forecast - ISP 2019-20.	Central	Central, Slow, Fast, High DER, Step
energyEfficiency	Market operator's energy efficiency forecast - ISP 2019-20.	Central	Central, Slow, Step
solarUptake	Market operator's solar uptake forecast - ISP 2019-20.	Central	Central, Slow, Step
domesticConsumption--Percentage	Percentage of residential consumption in Victoria.	30	[20, 50]
importPriceFactor	Premium paid for imported electricity. Applied to the wholesale price when local demand is unmet.	0.29	[-50, 50]

Table 8.4: Uncertain parameters in the GR4SP simulation model (2) [108].

Performance Indicators	Descriptions
Month/Year	Simulated current month/year
Consumption (KWh) per household	Monthly/Yearly average electricity consumption per household
Avg Tariff (c/KWh) per household	Monthly/Yearly average tariff per househole
Primary Wholesale ($/MWh)	Monthly/Yearly average electricity price in the primary wholesale market
GHG Emissions (tCO2-e) per household	Monthly/Yearly average greenhouse gas emission per household
Number of Domestic Consumers (households)	Monthly/Yearly average number of households buying electricity from the market
Percentage Renewable Production	Monthly/Yearly percentage of electricity produced from renewable energy
System Production Primary Spot	Monthly/Yearly electricity supply in the primary wholesale market
System Production Secondary Spot	Monthly/Yearly electricity supply in the secondary wholesale market
System Production Off Spot	Monthly/Yearly electricity supply outside of the wholesale market
System Production Rooftop PV	Monthly/Yearly electricity supply from rooftop photovoltaic systems
System Production Coal	Monthly/Yearly electricity supply from coal-powered generators
System Production Water	Monthly/Yearly electricity supply from water-powered generators
System Production Wind	Monthly/Yearly electricity supply from wind-powered generators
System Production Gas	Monthly/Yearly electricity supply from gas-powered generators
System Production Solar	Monthly/Yearly electricity supply from solar-powered generators

Table 8.5: Performance indicators in the output of the GR4SP simulation model (1) [108].

Performance Indicators	Descriptions
System Production Battery	Monthly/Yearly electricity supply from battery-powered generators
Number of Active Actors	Monthly/Yearly average number of generators participating in the market
Primary Total Unmet Demand (MWh)	Monthly/Yearly unmet electricity demand in the primary wholesale market
Primary Total Unmet Demand (Hours)	Monthly/Yearly number of hours where electricity demand is unmet in the primary wholesale market
Primary Total Unmet Demand (Days)	Monthly/Yearly number of days where electricity demand is unmet in the primary wholesale market
Primary Max Unmet Demand Per Hour (MWh)	Monthly/Yearly maximum unmet electricity demand in any hour in the primary wholesale market
Secondary Total Unmet Demand (MWh)	Monthly/Yearly unmet electricity demand in the secondary wholesale market
Secondary Total Unmet Demand (Hours)	Monthly/Yearly number of hours where electricity demand is unmet in the secondary wholesale market
Secondary Total Unmet Demand (Days)	Monthly/Yearly number of days where electricity demand is unmet in the secondary wholesale market
Secondary Max Unmet Demand Per Hour (MWh)	Monthly/Yearly maximum unmet electricity demand in any hour in the secondary wholesale market

Table 8.6: Performance indicators in the output of the GR4SP simulation model (2) [108].