Towards a Scalable Approach for Risk-Averse Safety Analysis

by

Chuyi Hou

Supervisor: Margaret P. Chapman

April 2022

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Abstract

Risk-averse optimal control is an emerging area of research, where the existing risk-averse control algorithms suffer from the curse of dimensionality limiting their practical utility. We propose an approximate yet tractable method, the Monte Carlo Tree Search, to improve the computational efficiency of a current risk-averse exact method with theoretical guarantees. The computational benefits of our proposed algorithm and the cost improvement property it carries are thoroughly explained. We take steps towards showing that the cost improvement property of stochastic rollout still holds in risk-averse settings and obtain a promising result. We discuss the limitations of our algorithm and the potential directions for future implementations.

Acknowledgements

I am extremely grateful to my thesis supervisor Prof. Margaret Chapman. Her words of encouragement gave me strength, and her stern teachings gave me direction. This project would not have been possible without her advice and support, which I will never forget. I must also thank Prof. Lisa Romkey and Prof. Alan Chong who provided us with lots of care and help during this project. I should also appreciate Nikita Dawe for ensuring the project proceeds smoothly and coordinating the in-person presentations. I would like to express my gratitude to the University of Toronto and the Division of Engineering Science for providing such a precious and memorable undergraduate thesis experience. Finally, I am very grateful to my family for supporting me through this challenging and uncertain time of my life.

Contents

1 Introduction **1**

1.1 Risk-Averse Control . . . 2

1.2 Theoretical Background . . . 4

1.2.1 Stochastic Rollout . . . 4

1.2.2 Monte Carlo Tree Search and Stochastic Rollout . . . 5

1.2.3 Upper Confidence Bounds . . . 6

2 Towards a Scalable Approach **9**

2.1 Examination of the Exact Risk-Averse Method . . . 9

2.2 Approximation in Value Space Using a Base Heuristic . . . 10

2.3 Stochastic Rollout in Risk-Averse Settings . . . 10

2.3.1 Cost Improvement Property of Stochastic Rollout . . . 11

2.4 Simulation Based Approach Using the MCTS . . . 12

2.5 MCTS Pseudocode . . . 14

2.5.1 Different Approaches . . . 17

3 Results **19**

3.1 Proof of the Cost Improvement Property in Risk-Averse Settings . . . 19

3.2 Implications and Potential Applications . . . 22

3.3 Discussion and Limitations . . . 22

4 Conclusion and Vision **25**

A Supplementary Materials **27**

A.1 Definitions and Notations . . . 27

A.2 Stochastic DP Algorithm (Backward) . . . 28

A.3 A General Risk-Averse Formulation[2] . . . 28

A.4 One-step Lookahead Stochastic Rollout Algorithm . . . 28

Bibliography **31**

List of Figures

1.1 An illustration of several risk measures on a probability distribution of the outcome (utility) of a stochastic system at certain stage. 3
1.2 Outline of a Monte-Carlo Tree Search [17] 5

2.1 Schematic illustration of rollout with one-step lookahead for a deterministic problem [23] 11

Chapter 1

Introduction

Problems such as path planning for autonomous vehicles, portfolio optimization in finance, and the design and control of the stormwater infrastructure, all are tied closely to people's daily life. These problems all have one thing in common: they all need dealing with *uncertainty* in the environment. Uncertainty in such problems is also known as *risk*. Hence, it is critical to developing practical and efficient ways to provide safer solutions to these problems.

In the context of autonomous systems with safety goals, the control objectives seek to minimize the cost of interest while ensuring safety. In this research project, our interests lie in risk-sensitive autonomous systems that operate under uncertainty. These systems can often be modelled as a *stochastic system*. Due to the well known "curse of dimensionality" associated with dynamic programs, existing risk-averse control methods for such systems necessitate costly computational resources (time, energy, and memory), limiting their practical utility.

Analyzing safety for risk-aware systems has been a significant challenge in many domains, including traffic control, self-driving vehicles, and energy systems. Therefore, this proposed algorithm will aid in risk-averse safety analysis for stochastic systems, while enhancing the intelligence of autonomous systems by improving their sensitivity to risks in the real world.

The proposed project aims to devise an approximate yet tractable algorithm for risk-averse control using reinforcement learning methods, derive the theoretical guarantees, and justify its use in practice. Therefore, the first objective is to develop the proposed algorithm with theoretical guarantees. The second objective is to investigate its performance and demonstrate the numerical benefits of a grid-free risk-averse algorithm. Our third objective is to evaluate the scalability of the proposed algorithm in practice.

The proposed algorithm is inspired by the stochastic rollout and the Monte Carlo tree search (MCTS) with upper confidence bounds. The use of stochastic rollout is driven by its *cost improvement property* (i.e., the policy or trajectory generated by the rollout algorithm is expected to perform no worse than the one obtained from a base heuristic). The rollout cost improvement property can help with building the theoretical guarantees. Despite this, the simulation effort of stochastic rollout is enormous. Hence, the use of MCTS is motivated because it balances computational efficiency with a hopefully low risk of performance loss (e.g., it may early discard controls deemed to be inferior based on the results of preliminary calculations, and simulate in a limited scope). Thus, stochastic rollout and MCTS together provide an alternative pathway for solving a control problem

approximately with anticipated policy improvements and numerical benefits [1].

To achieve the objectives, first, we will study and write down the pseudocode for Monte Carlo Tree Search with the stochastic rollout and upper confidence bounds. Next, to derive the theoretical guarantee, we will write down stochastic rollout algorithm in the CVaR setting using a general risk-averse problem formulation [2]. Then, we will formulate mathematical proofs to demonstrate the proposed algorithm's cost improvement property and justify that our algorithm is guaranteed to improve upon a base policy. After that, to investigate and demonstrate the advantages of our proposed algorithm, we will compare the accuracy and efficiency of the implementation of our algorithm versus the implementation in Theorem 1 (an exact method) [3]. Finally, we will develop new higher-dimensional examples, such as automated mobile machines, energy systems, and smart cities, to better examine the scalability of the proposed approach in practice.

Various difficulties, for instance, in renewable energy systems, transportation, and finance, require us to make judgments in the presence of uncertainty. Uncertainties such as weather patterns, the behaviour of the vehicle controlled by human drivers, and stock market volatility are usually unpredictable while tying closely to the decision-making process. One of the most difficult aspects of optimizing under uncertainty is dealing with a wide uncertainty space, which typically leads to highly large-scale optimization models. Many difficulties have been highlighted in relevant review papers, in addition to the theory and methodology that has been developed to cope with the complexity of optimization problems under uncertainty [4, 5].

Risk analysis is critical for these real-world stochastic autonomous systems with safety goals. For example, stormwater infrastructure systems must account for flood risk, and self-driving vehicles must account for traffic accidents. Furthermore, these risks not only have the potential to affect the system as a whole, but they are also frequently linked to people's safety. As a result, understanding and developing risk analysis theory and methods is vital to handle these challenges.

The remaining of this chapter will discuss the current gap in risk-averse optimal control for stochastic systems, provide historical evidence to support our proposed algorithm that aims to fill in the gap, and present the essential theoretical background for this project.

1.1 Risk-Averse Control

Traditionally, control policies for stochastic systems are derived from two main paradigms: the risk-neutral paradigm and the worst-case paradigm [6]. Figure 1.1 illustrates several risk measures, each of which correlates to a particular control paradigm. The expected utility or the expected outcome of the system is considered in problems in the risk-neutral paradigm. The worst-case or the maximum possible loss of a system is used as the optimization objectives in problems in the worst-case paradigm. However, limitations exist in using these techniques. The risk-neutral paradigm is concerned with average cost while systems with safety goals may focus on more specific features of a cost distribution (i.e., utilizing expected values to gauge performance may not be appropriate in risk-averse problems). Hence, Ruszczyński has enabled dynamic programming models to be more sensitive to risk by proposing the notion of composition of risk functional to formulate risk-averse control problems [7]. On the other hand, the worst-case paradigm heavily relies on an accurate estimate for the bounds of disturbances and expects such bounds to be predictable during the operation. The *risk-averse paradigm* has been introduced to connect the risk-neutral and worst-case

paradigms, thus establishing a balance between these two paradigms [6].

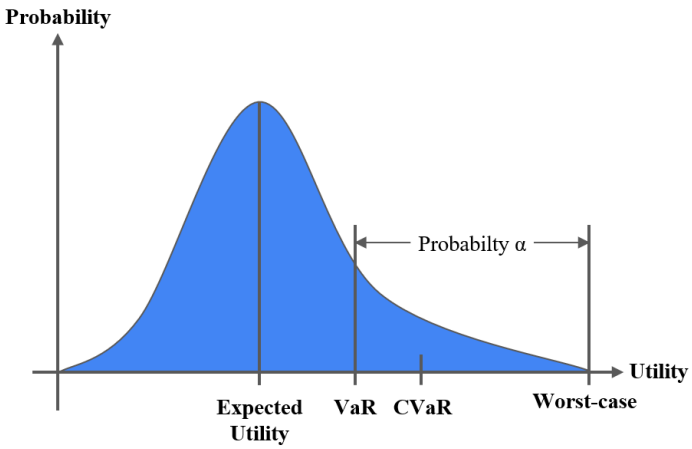

Figure 1.1: An illustration of several risk measures on a probability distribution of the outcome (utility) of a stochastic system at certain stage.

Conditional Value at Risk (CVaR) (a.k.a, *Expected Shortfall* in finance) measures the average risk if the outcomes lie outside of the confidence level. CVaR was initially widely used by financiers. However, it has received more and more attention in research related to risk-sensitive Markov decision processes (MDPs) over the past several years because of people's contribution in the area of risk-sensitive optimal control. Chow et al. developed an approximate value-iteration algorithm to approximate the solution to a CVaR MDP and demonstrated its application on a 2D grid-world terrain planning problem [8]. Ahmadi et al. used path planning under uncertainty to demonstrate the advantage of using risk-averse policy in terms of CVaR [9]. Chapman et al. utilized CVaR to assess the maximum cost of a stormwater infrastructure application with climate uncertainty [3].

Many previous studies on risk-averse controls have emphasized ways to formulate frameworks for optimizing MDPs. Their goals were to improve the control system's sensitivity of risk by improving the objective that was optimized with respect to. However, there was little emphasis on the computation efficiency when dealing with risk-averse problems. Although, many demonstrated the success of their newly developed frameworks through practical examples, such as [3, 8, 9, 10], either the dimension of the problem is small or the computation time of obtaining an exact solution is usually hours of running on high-performance computers.

Currently, the biggest challenge in studying risk-averse problems is the massive amount of computing required due to the unscalability of existing risk-averse control algorithms. Given the recent rapid development of hardware, risk-averse problems have started to be studied and people are able to test their risk-averse solutions in the past decade. Despite this, scalability is still a critical challenge that one needs to overcome. As with current risk-averse control algorithms, it is only applicable in one to two dimensions. Therefore, this shows the importance of this work as it aims towards a scalable approach for analyzing safety in the risk-averse paradigm.

1.2 Theoretical Background

Our proposed algorithm is based on the rollout algorithm and the Monte Carlo Tree Search with upper confidence bounds. In the following subsections, we will discuss the advancement in the rollout algorithm and motivate Monte Carlo Tree Search (MCTS). In addition, we will present the connection between MCTS and rollout. Furthermore, we will discuss the relevant properties of the rollout and breifly describe the procedure of MCTS for a thorough understanding of the building stones for our algorithm. All these components will be thoroughly discussed in Chapter 2.

Appendix A.1 contains definitions and notations to aid in reading the remainder of the report as some notations will be mentioned frequently.

1.2.1 Stochastic Rollout

For high-dimensional stochastic problems, obtaining optimal policies is computationally intractable due to the "curse of dimensionality". Hence, many pieces of research focus on approximate and computationally efficient methods that can produce suboptimal policies. Finding an approximate solution to discrete optimization problems effectively using rollout algorithm and rollout cost improvement property was proposed by Bertsekas et al. in 1997 [11]. Bertsekas and Castañon demonstrated efficient implementations of rollout algorithms with substantial improvement over the base heuristics for stochastic scheduling problems in 1999 [12]. Goodson et al. investigated rollout variants and achieved improved solutions to a large problem instance, the multi-vehicle routing problem with stochastic demand and duration limits, in 2012 [13]. Rollout algorithm has demonstrated its capability of providing approximate solutions to stochastic dynamic problems with a reasonable amount of computation resources. [12, 13, 14].

The theoretical property of rollout that we will make use of is the *cost improvement* property: the cost obtained by rollout using some base heuristic is less or equal to the corresponding cost of the base heuristic. For this property to hold, the base heuristic needs to satisfy some simple conditions, that is, the base heuristic used to generate rollout policy is *sequentially consistent* and *sequentially improving* [1]. Although the choice of base policy is essential for the performance of the rollout approach, experimental evidence has shown that the choice of base policy may not be crucial for many contexts, and in fact, a decent rollout performance may still be attained even with a relatively poor base heuristic, particularly with a long lookahead (e.g., l-step lookahead) [1, 15]. In this project, we will only focus on the one-step lookahead rollout.

In a one-step lookahead form, rollout is approximation in value space with the approximate cost-to-go values $\tilde{J}_{k+1}(x_{k+1})$ calculated by running the base policy, starting from each possible next state x_{k+1}. If the problem involves a long horizon, the run of the base policy may be truncated, i.e., it may be used for a limited number of steps, with some cost function approximation at the end to take into account the cost of the remaining steps. However, Goodson et al.'s work in 2015 made it clear that most rollout implementations are heavily problem-dependent [16]. Hence, one should consider modifying and refining the decision rules based on a general framework and then solve the specific problem.

All that has been said so far about rollout is in deterministic settings. For stochastic rollout, it is simply the rollout algorithm applied in stochastic settings. It has all the properties, the most important, cost improvement property, of the deterministic rollout [1]. The implementation, studied

from [1] and written by us, described in appendix A.4 implicitly assumed that we use all the admissible controls at x_k^j to expand the current state x_k^j with $u_k^i \in U_k(x_k^j)$ for $i = 1, 2, \ldots, m_k^j$. Then, for each trajectory expanded (the number of trajectories is the number of total admissible controls at x_k^j, which is equal to m_k^j), it simulates using the base heuristic to the end of the horizon.

Some of the drawbacks to the stochastic rollout algorithm described above are: 1) huge simulation effort due to many trajectories pointing outward from the current state and these trajectories may also be too long. 2) some of the controls u_k may be inferior to others, and may not be worth as much sampling effort.

These drawbacks have motivated variants that can be implemented and combined with the MCTS approach. A simple remedy for (1) is to limit the length of the rollout trajectories, with some terminal cost approximation at the end. (2) can be solved by deploying MCTS that uses some heuristic or statistical test to discard some of the inferior controls u_k^i. Although solutions to dynamic programming problems are often problem-dependent, the basic idea behind MCTS is to use interim results of the simulation and statistical tests to focus the simulation effort along with the most promising directions. To implement MCTS, one needs to keep track of a lookahead tree that expands as the relevant Q-factors are evaluated through simulation and balances the *competing desires of exploitation and exploration* (generate and evaluate controls that seem most promising in terms of performance versus assessing the potential of inadequately explored controls) [1]. Therefore, MCTS often has the upper confidence bounds (UCB) implemented as its tree policy. Monte Carlo Tree Search, rollout, and upper confidence bounds will be discussed in more detail in sections 1.2.2 and 1.2.3.

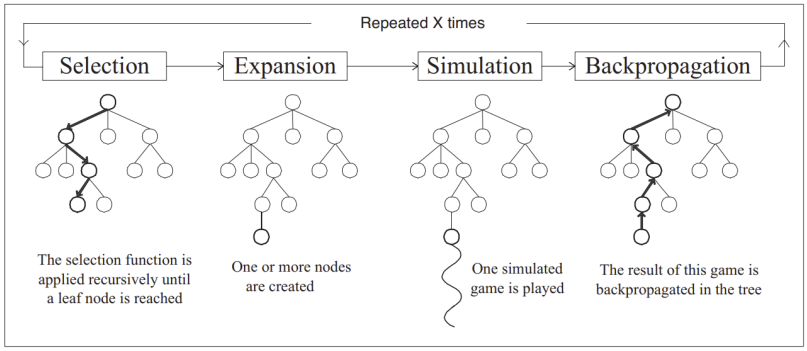

Figure 1.2: Outline of a Monte-Carlo Tree Search [17]

1.2.2 Monte Carlo Tree Search and Stochastic Rollout

Monte Carlo Tree Search (MCTS) is a stochastic simulation-based best-first search technique. Figure 1.2 illustrated an overview of this algorithm. An in-depth explanation of the procedures of MCTS will be given in sections 2.5. MCTS was proposed by Coulom in 2006 [18]. It combines tree search with Monte Carlo simulation, which not only adapts the Monte Carlo method's advantage of dealing with randomness but also inherits computation efficiency from a search tree while increasing the accuracy of Monte Carlo evaluation [18]. Coulom also suggested the potential scalability of MCTS, which enables its applications in larger systems. In the same year, Kocsis and Szepervári formalized a complete MCTS by extending UCB to minimax tree search and named it the Upper Confidence

Bounds for Trees (UCT) method [19]. The success of MCTS has been demonstrated by the AlphaGo computer program [20] which outperforms the best humans in the game of Go. The importance of this success is that it shows the capability of MCTS to deal with high-dimensional problems with large horizons and action spaces. In 2013, Gelly et al. described the connection between the rollout, and MCTS [21]. MCTS will gradually adjust and refine its simulation policy based on Monte Carlo simulation and statistical tests (e.g., upper confidence bounds). The method reverts to a fixed policy (i.e., base policy) when it reaches a leaf node of the current tree. It uses this policy to complete the trajectory. This part of the simulation is known as the rollout.

However, there is a slight difference between the original rollout algorithm and the rollout phase in MCTS. The rollout discussed in the previous section will expand every possible next state, then simulate all trajectories (one trajectory per possible next state) pointing outwards from the current state, and finally, pick the best action based on the simulation results. This approach has drawbacks stated in section 1.2.1. Therefore, MCTS incorporates the rollout in a slightly different way. The expansion phase replaces the one-step lookahead; MCTS will not expand to all possible next states at once. Instead, it will try one control at a time and decide whether other controls will be tried in the future iterations depending on the tree policy (e.g., UCB). To clarify, the tree policy do not pick the control to try at a leaf node, it only picks the nodes to descend to the leaf node. At a leaf node, one control will be tried uniformly randomly. Hence, for nodes that are not selected by the tree policy when descending, the unvisited children of those nodes will not be expanded. Moreover, MCTS will only simulate one trajectory per iteration whereas the rollout algorithm will simulate all possible trajectories from this node at once. Intuitively, MCTS divides the one-step lookahead rollout into pieces and simulates one of them in different iterations. Hence, depending on the tree strategy, not every control will be tested at a state (i.e., some controls deem to be inferior may be discarded), which explains the remedies for disadvantage (2) stated in section 1.2.1.

1.2.3 Upper Confidence Bounds

Upper confidence bounds (UCB) is a statistical test for adaptive sampling. It is often used as the so-called *tree policy* in the selection phase in the MCTS because of the contribution made by Kocsis and Szepervári [19]. The policy in the selection phase of an MCTS will select the control with the best upper confidence bound as follows:

$$u_k \in \underset{u_k^i \in U_k(x_k)}{\arg\min} \{UCB\} \tag{1.1}$$

where u_k^i is one action from a finite control space $U_k(x_k^j) = \{u_k^1, \ldots, u_k^{m_k^j}\}$ at stage k state x_k^j and

$$UCB = \text{exploitation index} + \text{exploration index}$$

The exploitation index emphasizes reward or cost depending on the problem, whereas the exploration index stimulates the investigation of less-visited nodes while reducing the impact of unlucky playouts. When descending the tree, this UCB method seeks to balance exploitation and exploration. The node with the highest UCB value is the most urgent node that will be prioritized during the selection phase (i.e., these nodes will be selected to descend the tree).

Moreover, in stochastic problems, the exploitation index is usually chosen to be the empirical

mean of the Q-factor of control u_k^i assuming that u_k^i has been sampled at least once, and the exploration index is chosen to be $-C_p\sqrt{\frac{2\ln n_j}{s_{u_k^i}}}$ with appropriate constant $C_p > 0$ [1, 19]. The exploitation index is given by

$$\begin{aligned} Q_{i,n} &= \frac{\text{Sum of sampled Q-factor when action } i \text{ taken prior to the } n^{th} \text{ sampling period at a state}}{\text{Number of times the action } i \text{ taken prior to the } n^{th} \text{ sampling period at a state}} \\ &= \frac{\sum_{l=1}^{n} \delta(u_{k,l} = u_k^i) Q_k(x_k, u_k^i)}{\sum_{l=1}^{n} \delta(u_{k,l} = u_k^i)} \end{aligned}$$

$$Q_{i,n} = \frac{\sum_{l=1}^{n} \delta(i_l = i)\tilde{Q}_k(x_k, i_l)}{\sum_{l=1}^{n} \delta(i_l = i)} \tag{1.2}$$

where

$$\delta(u_{k,l} = u_k^i) = \begin{cases} 1 & \text{if } u_{k,l} = u_k^i \\ 0 & \text{if } u_{k,l} \neq u_k^i \end{cases}$$

$u_{k,l} = u_k^i$ means that the control at state k selected at the l^{th} sampling period is equal to the chosen action u_k^i.

Then, UCB is calculated as follows:

$$UCB = Q_{u_k^i, n_j} - C_p\sqrt{\frac{2\ln n_j}{s_{u_k^i}}} \tag{1.3}$$

where

- C_p is the balancing constant that balance these two indices (i.e., is a design parameter).
- n is the n^{th} sampling period of state x_k^j. It is also the number of times the current node's parent has been visited.
- $s_{u_k^i}$ is the number of times u_k^i was chosen up to and include period n, $s_{u_k^i} = \sum_{l=1}^{n} \delta(u_{k,l} = u_k^i)$. It is also the number of times the current node has been visited.

We will give a brief walk-through to help understand how UCB decides the nodes to select when descending the tree. Starting from the first iteration at the root, since no control has been tried yet and there is no other path to descend, the tree will expand by trying out all possible controls at the root. After the simulation results for each leaf are fed back, the root has now been visited for $n = number\_of\_controls\_at\_root$ times and $s_{u_0^i} = 1$ for $i = 1, \ldots, j, \ldots, n$ (assuming $n >= 2, n \in \mathbb{N}$, otherwise if $n = 1$, there is no point of making the decision, similar comment can be made for $n = 0$). Hence, these children have the same exploitation indices but potentially different exploration indices depending on the simulation results. UCB will select the control, let's say u_0^j, that gives the best outcome in the current iteration. In the next iteration, the tree goes from the root to $x_1^j = f_0(x_0, u_0^j, w_0)$ and randomly tries one control $u_1^i \in U_1(x_1^j)$. After the simulation result is backpropagated, $s_{u_0^j} = 2$, $n = n + 1$, and the exploitation index for x_1^j decreases ($\frac{\ln(n+1)}{2} < \frac{\ln n}{1}$ for $n >= 2$). However, the exploration index will change depending on the simulation result. Thus,

u_0^j may or may not give the minimal UCB among other controls in $U_0(x_0)$. If UCB given by u_0^j is still the minimum, the tree will descend to x_1^j and randomly try one untried control. Else if one other control u_0^p produces the minimal UCB at stage 0, although it had larger exploitation index but not it has lower exploitation index (larger in the negative direction) compare to the UCB produced by u_0^j, the child corresponds to u_0^p will be selected to descend and so on. Note, this means not every unvisited node will be expanded. It depends on whether these unvisited nodes' parents will be selected when descending the tree. As a result, it could reduce simulation effort as some unvisited nodes may never be expanded.

To clarify, all n in this section are referring to the number of sampling period of the root node. The sampling period for node x_1^j becomes 2 after the $(n+1)^{th}$ period given that $u_0^j \in U_0(x_0)$ provides the minimal UCB. The sampling period for node x_1^j will be used to compute the UCB of the controls that lead to children of x_1^j, but we did not proceed further in this example.

UCB is an important piece that completes the MCTS algorithm [19]. UCB will be reflected in the pseudocode we have developed in the next chapter. In addition, we will provide evidence that supports the trajectory generated by the UCB algorithm [cf. Eq.(1.1)] can carry the cost improvement property of a one-step lookahead rollout.

Chapter 2

Towards a Scalable Approach

This chapter contained most of our study and understanding of the subject. We will begin by examining an existing exact risk-averse control algorithm identifying the costly operations. Then, to approximate the computationally costly procedures in Eq. (2.1), we will present the approximation in value space, stochastic rollout, and Monte Carlo Tree Search. Following that, we will develop the approximate Q-factor in risk-averse settings, extend the stochastic rollout to risk-averse settings, build the methodology to derive the theoretical guarantee using the rollout's cost improvement property, and delve into the MCTS and its relationship with the rollout, which ties everything together. For future implementations, we will provide pseudocode for the MCTS as well as two alternate implementation scenarios.

2.1 Examination of the Exact Risk-Averse Method

To improve the computational efficiency of the exact risk-averse control algorithm formulated in [2], which is also included in the appendix A.3, we first examine the computational costly operations in this algorithm. Given sufficient background knowledge about dynamic programming, it is not difficult to observe that equation (IA.4), the backward recursive DP algorithm, is the most computational costly. This risk-averse backward recursive DP algorithm is shown in Eq.(2.1) with slight modifications to help communicate which parts are computationally intensive and their remedies.

$$J_k^*(x_k, z_k) = \underbrace{\min_{u_k \in U_k(x_k)}}_{1} \underbrace{E_{w_k}}_{2} \left\{ \underbrace{J_{k+1}^*(f_k(x_k, u_k, w_k), \max\{g_k(x_k, u_k), z_k\})}_{3} \right\} \tag{2.1}$$

where the superscript s for cost J_k^s in Eq.(IA.4) is omitted and * means optimal since this algorithm is an exact method [2].

- A remedy for 1 is to approximate the minimization by not considering all the controls in the finite control space $U_k(x_k)$. There may be controls that are deemed to be inferior using statistical tests such as UCB [1, 19], hence, these controls may be disregarded when performing simulations. An important note, for the minimum to exist when performing the minimization, our proposed algorithm is considering *a finite horizon with finite controls at each stage.*

- A remedy for 2 is to approximate the expectation operation using the empirical mean of simulation results by drawing samples from the distribution of the disturbance.
- A remedy for 3 is to use approximation in value space by replacing the optimal cost-to-go J^*_{k+1} with an approximate cost-to-go $\tilde{J}_{k+1}$. This approximation can be obtained from machine learning methods or a base policy that produces "near-optimal" controls given by human or software "experts" [22]. This is known as *approximation in policy space on top of approximation in value space*. Note, once we choose to use approximation in value space, the approximated cost-to-go at every stage is considered given. This means one can choose to retrieve the suboptimal control sequence by running a forward algorithm OR one can choose to improve this approximation using rollout because of its cost improvement property.

Our methods will be to improve the current exact algorithm around the above three points. All remedies can be provided by a simulation based method, Monte Carlo Tree Search with upper confidence bounds applied [19]. Sections 2.2 to 2.4 will talk about each remedy in details.

2.2 Approximation in Value Space Using a Base Heuristic

Approximation in value space is to replace the exact/optimal cost-to-go J^*_k with an approximate cost-to-go $\tilde{J}_k$. Two ways to provide the approximation are: 1) to use reinforcement learning methods and 2) to provide a base heuristic by human experts. For our work, we do not focus on providing the approximate cost-to-go. However, we assume a base heuristic is given, which satisfies the condition (i.e., sequential consistency or sequential improving) needed for cost improvement property of rollout.

Assume such a base heuristic is given and defined by $H_{k,\pi}$, where k denotes the stage and π denotes the base policy. We can use this base heuristic to replace the exact cost-to-go $J^*_{k+1}(x_{k+1}, z_{k+1})$ in Eq.(IA.5) and obtain the approximate Q-factor in risk-averse settings:

$$\tilde{Q}_k(x_k, z_k, u_k) := E_{w_k}\Big\{H_{k+1,\pi}(f_k(x_k, u_k, w_k), \max\{g_k(x_k, u_k), z_k\})\Big\} \tag{2.2}$$

To emphasize the importance of this step, 1) it provides a way to avoid the large computations required to obtain cost-to-go at stage $k+1$ in Eq.(2.1) and 2) the approximate Q-factor will be an essential component towards the theoretical guarantee which ties closely to the rollout algorithm. In the rollout algorithm, the rollout cost is defined by the minimization of the approximate Q-factors over controls (in our work, we consider the number of controls to be finite). Hence, in the next section, we will talk about stochastic rollout and the theoretical guarantee it provides, the cost improvement property, in details.

2.3 Stochastic Rollout in Risk-Averse Settings

As the first step towards the theoretical guarantee, we need talk about the stochastic rollout algorithm and its cost improvement property in details. Here, we provide an intuitive idea for the rollout algorithm by considering an one-step lookahead rollout for a deterministic problem. Referring to figure 2.1 for visualization, the rollout algorithm is doing the following. Given a base heuristic, first, it tries every feasible control in the control space at the current state x_k and reach all possible

next states x_{k+1}. Next, it applies the base heuristic to descend to terminal states starting from each x_{k+1}. Then, the approximate Q-factors for each simulated trajectory are obtained. Here, the approximate Q-factors are computed by the sum of the stage costs along that trajectory produced by the base heuristic. Finally, the rollout cost at the current state x_k is defined by the minimization of the approximate Q-factors over a set of finite controls at the current state.

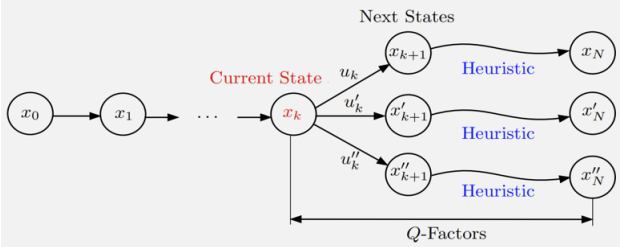

Figure 2.1: Schematic illustration of rollout with one-step lookahead for a deterministic problem [23]

The above is an intuitive explanation of the deterministic rollout. In stochastic rollout, the core difference is one control does not lead to a certain next state, instead, it leads to a distribution of next possible states. Hence, to compute the cost, stochastic rollout uses expectation operation.

Next, we provide a formal definition of the stochastic rollout cost, where the intuition behind the equations should tie closely to the explanation provided above.

$$J_{k,\tilde{\pi}} = \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k) \tag{2.3}$$

where

$$\tilde{Q}_k(x_k, u_k) = E_w \Big\{ g_k(x_k, u_k, w_k) + H_{k+1,\pi}(f_k(x_k, u_k, w_k)) \Big\} \tag{2.4}$$

Now, with the stochastic rollout cost defined, we can provide the stochastic rollout algorithm in risk-averse settings by replacing the approximate Q-factor used in stochastic problems [cf. Eq.(2.4)] with the approximate Q-factor in risk-averse problems [cf. Eq.(2.2)]. Therefore, the rollout cost in risk-averse problems is

$$J_{k,\tilde{\pi}} = \min_{u_k \in U_k(x_k)} E_w \Big\{ H_{k+1,\pi}(f_k(x_k, u_k, w_k), \max\{g_k(x_k, u_k), z_k\}) \Big\} \tag{2.5}$$

2.3.1 Cost Improvement Property of Stochastic Rollout

The cost improvement property of stochastic rollout provides the theoretical guarantee for our algorithm to perform at least as good as a base heuristic does. For the cost improvement property to hold, the base heuristic used by rollout needs to satisfy some condition. That is, the base heuristic should be either sequentially consistent or sequentially improving [1]. The explanations for these two conditions are listed below.

- **Sequential Consistency:** The base heuristic is sequentially consistent if at a given state it chooses control that depends only on that state and not on how we got to that state.

- **Sequential Improving:** Sequential consistency implies sequential improving. Sequential improving states that *the best heuristic Q-factor at x_k is less or equal to the heuristic cost at x_k* (i.e., $\min_{u_k \in U_k(x_k)} \tilde{Q}_{k,\pi} \leq H_{k,\pi}$).

The cost improvement property of stochastic rollout states *the rollout cost is less or equal to the base heuristic cost*. In other words, it means rollout can perform as least as good as the base heuristic does. Three equivalent expressions are listed below to help with the understanding.

$$\begin{aligned} \text{Rollout Cost} &\leq \text{Base Heuristic Cost} \\ \text{Best Base Heuristic (approximate) Q-factor at } x_k &\leq \text{Base Heuristic Cost at } x_k \\ \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + H_{k+1,\pi}(f_k(x_k, u_k, w_k)) \Big\} &\leq E_{w_k}\Big\{ H_{k,\pi}(x_k) \Big\} \end{aligned}$$

where the left hand side of the third inequality is the approximate Q-factor in stochastic problems [cf. Eq.(2.4)].

For showing the cost improvement property holds in risk-averse problems, we will need a base heuristic defined in risk-averse settings and use the approximate Q-factor in risk-averse settings [cf. Eq.(2.2)]. We will prove the cost improvement property of rollout in risk-averse settings (i.e., the above inequality) in Chapter 3.

2.4 Simulation Based Approach Using the MCTS

The computational benefit of the MCTS and its high-level overview has been discussed in the previous chapter. In this section, we focus on explaining why we chose MCTS by providing our understanding and appreciation of this algorithm. In other words, we will talk about the computational efficiency it provides for the risk-averse problems and how it could adopt the cost improvement property from rollout.

The cost improvement property of rollout is meaningful to deriving the theoretical guarantee. However, in practice, rollout still requires lots of simulation effort. Therefore, to hopefully retain this property while making the algorithm tractable, the importance of the MCTS became apparent. As we have discussed in the previous chapter, the expansion phase and simulation phase mimic the rollout algorithm but with less simulation effort because inferior controls may be discarded. In stochastic rollout, the rollout control is defined as

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k) \tag{2.6}$$

where the approximate Q-factor is defined in Eq.(2.4). Finding the rollout control through this minimization is still simulation heavy because for each $\tilde{Q}_k(x_k, u_k)$ we need to perform the expectation operation meanwhile all the controls in the current control space $U_k(x_k)$ need to be simulated. Intuitively, MCTS with upper confidence bounds provides a smarter way (i.e., meaning MCTS does not need to try all controls) to hopefully find the rollout controls with less simulation effort. Instead of computing the exact expectations and simulating for all controls as in Eq.(2.6), MCTS uses empirical means to approximate the expectation and uses upper confidence bounds (UCB) to explore more promising trajectories as in Eq.(1.3). The controls are selected by the minimization of

the sum of the empirical means [cf. Eq.(1.2)] and the UCB value, hence the argmin [cf. Eq.(2.6)] becomes Eq.(1.1).

The reason for the computational efficiency of the MCTS should be covered above. Now, to discuss the potential absorption of the cost improvement property of rollout in the MCTS, we will take a deeper look into the selection phase of the MCTS.

Recall that the expansion phase is selecting a control at a leaf node uniformly randomly. Hence, the phase that is picking the control is the selection phase. The control choices are defined by the minimization of the sum of the exploitation index and exploration index [cf, Eqs.(1.1)-(1.3)]. Now, recall the remedy for 2 in Eq.(2.1). To reduce the simulation effort, instead of computing the exact expectation of the approximate Q-factor, the empirical mean is utilized as the approximation of the approximate Q-factor. We did not plan to show that the cost improvement property holds if we replace the exact expectation with the empirical mean. However, we will proceed to evaluate the influence from the exploration index (i.e., $-C_p\sqrt{\frac{2\ln n_j}{s_{u_k^i}}}$), assuming the cost improvement property would presumably hold with the empirical mean of the sampled Q-factors.

The exploration index is added to the empirical mean to balance the exploitation and exploration of the simulation. We want the algorithm to keep simulating along a promising trajectory (i.e., low empirical mean of the sampled Q-factors), meanwhile, we want to access the potential of inadequately explored controls that have been sampled a small number of times. In other words, although the current trajectory seems to have a small running average Q-factor (i.e., the empirical mean of the sampled Q-factors), the other trajectories might perform better once more trajectories were branched out. Hence, we do not want to give up these possibilities.

Moreover, in the last iteration, when reaching a terminal state in the selection phase, the final trajectory generated by the tree policy should be the trajectory that has been tried the most of the time and has the least running average Q-factor. In other words, initially, the exploration index would dominate the weight that determines the control to explore. Later on, the exploration index for a certain control would become smaller and smaller as it gets tried multiple times by the tree policy (i.e., the UCB algorithm). Also, C_p is a design parameter in the exploration index, and we can design it in a way that the above case is possible. The exploitation index will become the deciding factor for the controls.

For example, we examine the exploration index at a large number of sampling periods n_j given a control u_k^i has been tried very frequently in these n_j sampling periods.

Let n_j be large and the number of the control $s_{u_k^i}$ being selected be close to n_j (i.e., let $s_{u_k^i} = n_j - a$, where $a \in \mathbb{R}$ is a fixed integer). The exploration index when n_j approaches ∞

$$\lim_{n_j\to\infty} -C_p\sqrt{\frac{2\ln n_j}{s_{u_k^i}}} = \lim_{n_j\to\infty} -C_p\sqrt{\frac{2\ln n_j}{n_j - a}} = 0 \text{ (L'Hôpital's rule)}$$

Note: L'Hôpital's rule does not apply for discrete functions because they are not differentiable. To intuitively show the limit equals 0 given a minimal math background, we consider the functions to be continuous.

We note that, over a lengthy sample period, if a certain control is attempted the majority of the time, the exploration index approaches zero. It should be noted that the control was tried the majority of the time since it also results in a low exploitation index in the minimization. [cf.

Eq.(1.1)].

We demonstrated that the exploration index should be a small component in the minimization that produced the final trajectory. On the other hand, the exploitation index, which is the empirical mean of the sampled Q-factors, makes the most significant contribution. As a result, we may conclude that the final trajectory is produced from the minimization of the empirical means of the sampled Q-factors at each stage, providing that we can disregard the exploration index at the last iteration of the MCTS.

To summarise, we discussed the MCTS's computational efficiency and the possible absorption of the cost improvement property of rollout in the MCTS. The expansion and simulation phases work together to produce the one-step lookahead action that rollout does and obtain the simulation result (i.e., the sampled Q-factor at that stage), significantly reducing the simulation effort for conducting rollout. Furthermore, the MCTS resulting trajectory should follow the rollout cost improvement property since the minimization that creates this trajectory is largely dependent on the empirical mean of the sampled Q-factors, given the cost improvement property holds for the empirical mean.

2.5 MCTS Pseudocode

The pseudocode for Monte Carlo Tree Search with the rollout and upper confidence bounds was written down for aiding the future development of our proposed algorithm. Pseudocode is a great approach to communicating algorithms in a succinct and comprehensible manner.

This outline of the pseudocode is learned from [24]. Algorithm 1 states the MCTS algorithm. Algorithm 2 states all the helper functions including the four phases of MCTS (i.e., selection, expansion, simulation, backpropagation) and the UCB that tries to balance the exploration and exploitation in the selection phase.

Algorithm 1 Monte Carlo Tree Search with Rollout and Upper Confidence bounds

Takes in: a current_state, x_k

Provides: a control for the current_state, u_k

```
1: function MCTS(current_state)
2:     root ← current_state          ▷ set the current state x_k as the root node of the search tree
3:     while stopping condition do          ▷ condition such as time or iterations
4:         leaf ← SELECTION(root)          ▷ use the tree policy to descend from the root to a leaf
5:         simulation_result ← ROLLOUT(leaf)          ▷ use the base/default policy to simulate
6:         BACKPROPAGATE(leaf, simulation_result)
7:     end while
8:     return BEST_CHILD_UCB(root, 0)
9: end function
```

Algorithm 2 Helper Functions

```
function SELECTION(node)
    while not node.is_terminal_node do
        if node is not fully expanded then
            return EXPAND(node)          ▷ uniformly randomly choose a control to expand
```

```
        else
            node ← BEST_CHILD_UCB(node, C_p)                    ▷ apply UCB
        end if
    end while
    return node
end function

function EXPAND(node)
    k ← node.level
    pick u ∈ untried actions from U_k(node)
    add new_node to node as its child
    return new_node
end function

function ROLLOUT(node)
    state ← node
    while not state.is_terminal do          ▷ or it can be other stopping conditions
        k ← state.stage
        w_k ← DISTURBANCES()  ▷ meant to take the expected value of the disturbance at stage k
        choose a control u ∈ U_k(state) using a policy          ▷ uniformly random or a base policy
        state ← f(state, u, w_k)
    end while
    return VALUE_FUNCTION(state)        ▷ return the approximate value for this terminal state
end function

function BACKPROPAGATE(node, result)
    while node.has_parent do
        node.visits ← node.visits + 1
        node.stats ← node.stats + result
        n ← n.parent
    end while
end function

function BEST_CHILD_UCB(node, c)
    min ← ∞
    best_child ← node
    for child in node do
        UCB ← child.stats/child.visits + c√(2 ln node.visits / child.visits)
        if UCB <= min then
            min ← UCB
            best_child ← child
        end if
    end for
```

end function

To help understand the pseudocode, the following provides a high-level walk-through of this algorithm (refer to figure 1.2 for an overview of the MCTS):

- Algorithm 1 is the main function that takes a given state x_k^j and provides a control u_k^j for the current state. It sets the input state as the root node of the search tree. Given a stopping condition, such as maximum amount of time or number of iterations, MCTS will run until the stopping condition is met. In the beginning of each iteration, the tree will descend to a leaf node using the tree policy. This is done by the SELECTION($root$) function. Next, starting from the leaf node, a simulation will be rolled out by the ROLLOUT($leaf$) function. The rollout phase will terminate when a terminal state is reached or a certain depth depending on the rollout implementation is reached. Lastly, the simulation result will be backpropagated, BACKPROPAGATE($leaf, simulation\_result$), through the trajectory (i.e., the nodes that were visited in this simulation iteration) obtained by the tree policy (in selection phase) and rollout policy (in the rollout phase) and the number of visits of these nodes will be updated. Finally, this function returns the control that gives the highest UCB at the root. Note, the C_p is set to zero because it is meaningless to compute the exploitation index at the root as the root does not have any parents or siblings.

- Algorithm 2 consists of all the helper functions. These functions are explained one by one below.

  - SELECTION($node$): While the given node is not a leaf of the search tree, it will do one of the two operations: 1) if the node can be expanded using the control that has not yet been used before, it will expand the node using EXPAND($node$) 2) else if all possible controls have been tried for the node, it will pick a child based on UCB using BEST_CHILD_UCB($node, C_p$).
  - EXPAND($node$): It picks one action from the feasible control space at stage k at state x_k while k is given by the level of the node (i.e., level of the tree is the same as the stage number of the system). It performs this action, transitions the node to the next node which is labelled as the child of the current node.
  - ROLLOUT($node$): This function performs the procedure that is similar to the rollout algorithm starting from the state x_k that corresponds to the node. However, since the expand portion of rollout was taken care of by EXPAND($node$), this function will perform a single trajectory simulation. While state is not terminal (or other stopping conditions such as while a certain depth is not reached), it takes the expected value of the disturbance at stage k and uses controls based on the given base policy for the problem to transition. Finally, it will return the outcome of this simulation. This could either be the approximated cost at the terminal state evaluated by a heuristic if it reached the terminal state or a truncated cost with some cost function approximation at the end if the rollout is ran for a limited number of steps.
  - BACKPROPAGATE($node, result$): This function backtraces the nodes that were visited in the current iteration. It updates the number of visits and statistics of these nodes.

– BEST_CHILD_UCB($node, c$): This function deploys the upper confidence bounds to descend the search tree by selecting the nodes that are the most urgent (refer section 1.2.3). Basically, a $[\arg\min_{u_k^i \in U_k(x_k^j)} Q_{u_k^i, n_j} - C_p\sqrt{\frac{2\ln n_j}{s_{u_k^i}}}]$ operation is done here: *child.stats* is the Q-factors accumulated at this child; *child.visits* is the number of action u_k^i taken prior to this iteration (i.e., the number of times this child has been visited); $\frac{child.stats}{child.visit}$ is the empirical mean of the sampled Q-factors $Q_{u_k^i, n_j}$; c (C_p) is the balancing parameter that can be tuned; *node.visits* (n_j) is the number of sampling period counted at the child's parent (i.e., the number of times the child's parent has been visited).

- Note that VALUE_FUNCTION and DISTURBANCE functions were not listed in the pseudocode. This is because they vary in different problems. Similarly, the exact implementation of the ROLLOUT function was not written down either because of the same reason. However, these will be reflected in later in the actual implementation of our algorithm.

2.5.1 Different Approaches

Moreover, based on the MCTS pseudocode developed above, we thought carefully about how to apply it to stochastic control problems. We considered two different approaches to deploying the MCTS: 1) offline approach and 2) online approach. Each has its advantages, disadvantages, and limitations that depend on the specific problem.

Approach 1 An offline approach

1. Run MCTS until a terminal state is reached in the tree selection phase.
2. Then, collect the nodes that were visited and hence collecting the control sequence that produces this trajectory.

This approach can be implemented simply by setting the stopping condition in the MCTS function (algorithm 1) to "until a terminal state is reached".

The advantages of this approach are that 1) a suboptimal control sequence can be obtained offline and 2) actions will be chosen efficiently during online (real-time) planning, given the approximate Q-factor at most of the states were stored and the suboptimal control sequence was generated already. However, the problems with this approach are that 1) it may take a long time to execute because the number of simulations required may be huge and 2) the suboptimal control sequence may fail in certain situations because errors may exist in the simulation and actual disturbances may be different from the the one used in simulation.

Approach 2 An online approach

1. Always starts a fresh tree when a next state is reached in online play (i.e., starts a new tree when the actual system picks the control and reaches its next state).
2. For example, when a decision need to be made (i.e., when a control need to be chosen to transition), the system will run the MCTS given a runtime constraint or stopping condition (refer to algorithm 1), and pick the best one based on the UCB.

In this approach, simulations, decision-makings, and transitions are all done online. This necessitates the use of another function that encapsulates these activities. Algorithm 3 provides the pseudocode for this approach.

Algorithm 3 Approach 2

```
1: function CONTROL(current_state)
2:     state ← current_state
3:     while not state.is_terminal do
4:         u ← MCTS(state)            ▷ MCTS will suggest its "best" move for the current state
5:         w ← DISTURBANCE()          ▷ the observed disturbance in real time or the expected value
6:         state ← f(state, u, w)
7:     end while
8: end function
```

The benefits of this approach are that 1) it may make a more informed decision since it seeks the "best" decision for the current state and 2) it does not rely on a pre-generated single control sequence because it can adapt to changes in the environment. However, the problem with this approach is that it needs a certain amount of computation time at each stage. Also, it requires a computation condition and different conditions may result in differences in performance (e.g., a longer simulation time may provide more information and thus lead to a better choice but may take more time. Vice versa).

Chapter 3

Results

We have shown the absorption of the cost improvement property in the MCTS. In this chapter, we will focus on providing proof of the cost improvement property of rollout in risk-averse settings. Next, we will go through the implications of the result and potential future implementations. Then, we will discuss the limitations and steps that may be taken to improve.

3.1 Proof of the Cost Improvement Property in Risk-Averse Settings

Recall the approximate Q-factor developed from the risk-averse formulation (Appendix A.3) [cf. Eq.(2.2)]. We only need to prove the cost improvement property given a fixed s because different values for s will not affect the algorithm. If the cost improvement property holds for a fixed s, then it holds for $s \in \mathcal{Z}$. Hence, we can drop the s in the proof.

Here, we use proof by induction to demonstrate the cost improvement property can still hold in risk-averse settings.

Let π be the base policy and $\tilde{\pi}$ be the rollout policy. We claim

$$J_{k,\tilde{\pi}}(x_k, z_k) \leq H_{k,\pi}(x_k, z_k) \tag{3.1}$$

In words, the rollout cost will be as least as good as the base heuristic cost.

Proof by Induction

Eq.(3.1) holds for $k = N$ because $J_{N,\tilde{\pi}}(x_N, z_N) = H_{N,\pi}(x_N, z_N) = g_N(x_N)$. Assume it holds for index k + 1, we prove it holds for index k:

$$
\begin{aligned}
J_{k,\tilde{\pi}}(x_k, z_k) &= E_{w_k}\Big\{ J_{k+1,\tilde{\pi}}(f_k(x_k, \tilde{u}_k, w_k), \max\{g_k(x_k, \tilde{u}_k), z_k\}) \Big\} (1) \\
&\leq E_{w_k}\Big\{ H_{k+1,\pi}(f_k(x_k, \tilde{u}_k, w_k), \max\{g_k(x_k, \tilde{u}_k), z_k\}) \Big\} (2) \\
&= \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ H_{k+1,\pi}(f_k(x_k, u_k, w_k), \max\{g_k(x_k, u_k), z_k\}) \Big\} (3) \\
&\leq E_{w_k}\Big\{ H_{k+1,\pi}(f_k(x_k, \bar{u}_k, w_k), \max\{g_k(x_k, \bar{u}_k), z_k\}) \Big\} (4) \\
&= H_{k,\pi}(x_k, z_k) (5)?
\end{aligned}
$$

- **Proof for (1):** This expression is the dynamic programming equation for the rollout policy $\tilde{\pi}$ given the rollout control $\tilde{u}_k$

$$
J_{k,\tilde{\pi}}(x_k, z_k) = E_{w_k}\Big\{ J_{k+1,\tilde{\pi}}(f_k(x_k, \tilde{u}_k, w_k), \max\{g_k(x_k, \tilde{u}_k), z_k\}) \Big\} \tag{3.2}
$$

  where

$$
\tilde{u}_k = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ J_{k+1,\tilde{\pi}}(f_k(x_k, \tilde{u}_k, w_k), \max\{g_k(x_k, \tilde{u}_k), z_k\}) \Big\}
$$

  Since we consider a finite number of controls in the control space $U_k(x_k)$, the minimum exists. $\tilde{u}_k$ produces the least cost, hence, the minimization in Eq.(3.2) can be reduced to expression (1).

- **Proof for (2):** This step holds by the induction hypothesis. We assumed

$$
J_{k+1,\tilde{\pi}}(x_{k+1}, z_{k+1}) \leq H_{k+1,\pi}(x_{k+1}, z_{k+1}), \text{ for all } x_{k+1} \text{ and } z_{k+1}.
$$

  The expectation operation is approximated by the empirical mean which is the case in our proposed algorithm. For fixed x_k and u_k, the next state x_{k+1} depends on the disturbance,

$$
x^i_{k+1} = f_k(x_k, u_k, w^i_k)
$$

  We draw m number of samples from the distribution of the disturbance, the empirical means for the rollout cost and the base heuristic cost are:

$$
\begin{aligned}
E_{w_k}[J_{k+1,\tilde{\pi}}] &= \frac{\sum_{i=1}^{m} J_{k+1,\tilde{\pi}}(x^i_{k+1}, z^i_{k+1})}{m} \\
E_{w_k}[H_{k+1,\pi}] &= \frac{\sum_{j=1}^{m} H_{k+1,\pi}(x^j_{k+1}, z^j_{k+1})}{m}
\end{aligned}
$$

  From the induction hypothesis, we have

$$
J_{k+1,\tilde{\pi}}(x^i_{k+1}, z^i_{k+1}) \leq H_{k+1,\pi}(x^j_{k+1}, z^j_{k+1}), \text{ if i=j}
$$

Therefore,

$$E_{w_k}[J_{k+1,\tilde{\pi}}] \leq E_{w_k}[H_{k+1,\pi}]$$

and hence shows that expression (2) holds.

- **Proof for (3):** Recall the definition of the rollout cost [cf. Eq.(2.3)]. The left hand side of expression (3) is the rollout cost and right hand side is the minimization of the approximate Q-factor over controls. This is exactly the rollout algorithm and hence it holds.

- **Proof for (4):** Similar to (1), this expression is the dynamic programming equation for the base policy that corresponding to the base heuristic. We need the sequential consistency for this step.

  In addition, this step may be skipped if the base heuristic is sequential improving meaning *the best base heuristic Q-factor is less or equal to the base heuristic cost at a state*

$$\min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, z_k, u_k) \leq H_{k+1,\pi}(x_k, z_k)$$

- **Towards showing (5):** To show

$$E_{w_k}\left\{ H_{k+1,\pi}(f_k(x_k, \bar{u}_k, w_k), \max\{g_k(x_k, \bar{u}_k), z_k\}) \right\} = H_{k,\pi}(x_k, z_k)$$

  We can first try showing this equality holds for the exact cost-to-go J_k. From the risk-averse formulation A.3 and also dropping the s, we have:

$$J_k(x_k, z_k) := \min_{u_k \in U_k(x_k)} Q_k(x_k, z_k, u_k)$$

  where the exact Q-factor

$$Q_k(x_k, z_k, u_k) := E_{w_k}\left\{ J_{k+1}(f_k(x_k, u_k, w_k), \max\{g_k(x_k, u_k), z_k\}) \right\}$$

  Let:

$$u_k^* \in \arg\min_{u_k \in U_k(x_k)} Q_k(x_k, z_k, u_k)$$

  hence for any one u_k^* in the set obtained from the above argmin operation

$$\begin{aligned} J_k(x_k, z_k) &= \min_{u_k \in U_k(x_k)} Q_k(x_k, z_k, u_k) \\ &= Q_k(x_k, z_k, u_k^*) \\ &= E_{w_k}\left\{ J_{k+1}(f_k(x_k, u_k^*, w_k), \max\{g_k(x_k, u_k^*), z_k\}) \right\} \end{aligned}$$

  Above showed the exact cost-to-go at stage k can be determined by taking the expectation of the exact cost-to-go at stage $k+1$ given the control u_k^* selected by the policy obtained from

the exact method.

Now, we have shown the equality holds for the exact cost-to-go in risk-averse setting. This provides the reason for why we think (5) is true, despite it requires more steps and deeper understanding of the math to complete the proof. This is why we put a question mark beside expression (5).

3.2 Implications and Potential Applications

Our research made strides in deriving the theoretical guarantees for stochastic rollout in risk-averse applications. Furthermore, we demonstrated in the previous chapter that the cost improvement property may be adopted by the MCTS, which shows a possibility to improve the computational efficiency of the current risk-averse algorithms with theoretical guarantees.

Moreover, a base heuristic is needed for the proposed algorithm. In addition, as mentioned in section 1.2.1, the base heuristic needs to be sequentially consistent or sequential improving to provide the cost improvement property of rollout. As a result, the applications that can potentially make use of our proposed algorithm are but are not limited to:

- **Medical decision making:** human experts such as doctors can provide "near-optimal" suggestions and procedures given the state of their patients; there are also well-established heuristics (i.e., decision strategy) in making medical decisions and a study by Marewski and Gigerenzer suggests that using the heuristic can sometimes be better than considering the entire image of the problem [25].

- **Path planning problems:** depending on the problem, the distance metrics such as Manhattan distance and Euclidean distance can be admissible and consistent provided that the sum of cost-to-come and cost-to-go from the initial state never decreases along any path. Hence, problems such as from [8, 9] could be approximated using our proposed algorithm.

3.3 Discussion and Limitations

Just as with any new approach, there are limitations to our algorithm in its current form. These must be taken into account when implementing it. Here, we discuss the limitations and potential methods to improve the performance of our algorithm and algorithms like it in the future.

- **Base heuristic:** A base heuristic is required for the algorithm to work. To obtain a base heuristic, one can use the reinforcement learning method or human experts' experience. In addition, to maintain the theoretical guarantee, the base heuristic must also satisfy the conditions of sequential consistency or sequential improvement. Although it may be difficult to find a base heuristic that satisfies these conditions, a remedy is to use a multi-step lookahead rollout [1, 15].

- **On-line versus off-line:** We have discussed the trade-off between implementing the algorithm on-line and off-line in section 2.5.1. The trade-off is problem-dependent. If the problem is not time-sensitive and the disturbance can be well modelled, then an off-line approach would be

better. On the other hand, if the problem is time-sensitive and the disturbance is hard to model, then the online approach should be chosen.

- **Approximate Algorithm:** Our algorithm is not exact meaning the solution it generates is not likely to be optimal and sometimes the error can be unacceptable. This problem is often faced by most approximate algorithms. Hence, approximate methods are less useful in applications that require optimal solutions. Hopefully, the approximate algorithms can aid the current research and solve real-life problems. Meanwhile, advancements should be made on the hardware side to make more computational resources available.

Chapter 4

Conclusion and Vision

We proposed an approximate risk-averse control algorithm using the Monte Carlo Tree Search and took steps towards deriving the theoretical guarantee using the cost improvement property of rollout. We began by identifying the current gap in the area of risk-averse optimal control, which is the unscalability of the existing risk-averse control algorithms. Then, we diagnosed the computational costly operations in a general risk-averse formulation [2]. Next, we provided remedies for these operations using dynamic programming methods that could provide theoretical guarantees. To support our hypothesis, we provided mathematical proof and obtained a positive result.

Moreover, risk-averse optimal control is an emerging area of research, and improving the computational efficiency with theoretical guarantees is still challenging. For future work, we will define a risk-averse problem with a given base heuristic and compare the efficiency and accuracy of our proposed algorithm and the exact methods.

Finally, to provide our vision in this field, we believe that in the future, improvements in approximate algorithms and breakthroughs in hardware systems will bring significant progress to the area of risk-averse optimal control. We believe that autonomous stochastic systems will be able to react to uncertainties in the environment efficiently and effectively in real-time. Self-driving cars will be able to make the best judgement to minimize the harm caused by accidents; the control of all stormwater infrastructures in a city will be integrated to react to climate uncertainties together; patients will be able to receive the most useful and effective treatment from an intelligent agent in real-time.

Appendix A

Supplementary Materials

A.1 Definitions and Notations

N is the number of times control is applied (horizon).

k is the time index, $k = 0, \ldots, N-1$.

$S = \{s_0, \ldots, s_N\}$ is a finite set of state space (the horizon is finite). At stage k, there can be m_k possible states x_k. The set of all possible x_k is the *state space* at time k.

$s_k = \{x_k^1, \ldots, x_k^{m_k}\}$ is a finite state space at time k.

x_k is the state of the system, an element of the *state space*.

$A = \{U_0, \ldots, U_{N-1}\}$ is a finite set of action space. Each action space provides a finite set of controls given a state x_k.

$U_k(x_k^j) = \{u_k^1, \ldots, u_k^{m_k^j}\}$ is a finite *control space* for stage k (i.e., in different state x_k^j, number of possible actions can be different). m_k^j is the number of possible controls at state x_k^j.

u_k is the control or action variable, to be selected at time k from some given set $U_k(x_k)$ that depends on x_k.

w_k be the random disturbance (independent random samples) in the stochastic system with a probability distribution $P_k(dw_k|x_k, u_k)$ that may only depend explicitly on x_k and u_k.

f_k in deterministic problems, is a function of (x_k, u_k), in stochastic problems, is a function of (x_k, u_k, w_k). It describes the state transition from time k to time $k+1$.

$G = \{g_0, \ldots, g_N\}$ is the functions that evaluate the stage cost. For $k \neq N$, in deterministic problems, the cost is computed based on x_k and u_k; in stochastic problems, the cost is computed based on x_k, u_k, and w_k. In both cases, the terminal stage cost is just $g_N(x_N)$.

$\Pi = \{\pi_0, \ldots, \pi_n\}$ is the set of admissible policies.

$\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ is given as Base Heuristic/Policy (heuristic that generates a complete trajectory), where $\mu_k : S \to U_k$ for all x_k, and satisfies the control constraints, i.e., $\mu_k(x_k) \in U_k(x_k)$ for all x_k

$J_k(x_k)$ is the cost of the tail subproblem (a.k.a, cost-to-go) that starts at state x_k at time k. $J^*(x_k)$ means the optimal cost-to-go.

A.2 Stochastic DP Algorithm (Backward)

Start with $J_N^*(x_N) = g_N(x_N)$, and for $k = N-1, \ldots, 0$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k} \left\{ g_k(x_k, u_k, w_k) + J_{k+1}^*(f_k(x_k, u_k, w_k)) \right\}, \text{ for all } x_k. \tag{IA.1}$$

A.3 A General Risk-Averse Formulation[2]

The objective is to find the risk-averse safe sets consist of the possible initial states that can produce risk-averse trajectories for the system using a CVaR risk measure. To compute the risk-averse safe sets under Assumption 1 [2]:

$$S_\alpha^r = \left\{ \mathbf{x} \in S : \min_{s \in \mathcal{Z}} (s + \frac{1}{\alpha} J_0^s(\mathbf{x}, a)) \leq r \right\} \tag{IA.2}$$

To perform the minimization above, we need to obtain the cost-to-go J_0^s at an initial state x_0 given different s.

We define the terminal cost $J_N^s : S \times \mathcal{Z} \to \mathbb{R}^*$ by

$$J_N^s(x_N, z_N) := h^s(\max\{g_N(x_N), z_N\}) \tag{IA.3}$$

where

$$h^s(y) := \max\{y - s, 0\}$$

We define the DP algorithm for a risk-averse problem $J_k^s : S \times \mathcal{Z} \to \mathbb{R}^*$ by:

$$J_k^s(x_k, z_k) := \min_{u_k \in U_k(x_k)} E_{w_k} \left\{ J_{k+1}^s(f_k(x_k, u_k, w_k), \max\{g_k(x_k, u_k), z_k\}) \right\} \tag{IA.4}$$

where the Q-factor in a risk-averse setting is defined as

$$Q_k^s(x_k, z_k, u_k) := E_{w_k} \left\{ J_{k+1}^s(f_k(x_k, u_k, w_k), \max\{g_k(x_k, u_k), z_k\}) \right\} \tag{IA.5}$$

and

$$z_{k+1} = \max\{g_k(x_k, u_k), z_k\}$$

A.4 One-step Lookahead Stochastic Rollout Algorithm

Refer back to A.1 for more details about the notations.

Let $x \in S$ be given. Initialize $i = k$ and $x_i = x_k$ ($k = 0$ if starting from the initial stage). For

$i = k+1, \ldots, N-1$, such simulated trajectory has the form

$$x_{i+1}^j = f_i(x_i^j, \mu_i(x_i^j), w_i), \qquad i = k+1, \ldots, N-1,$$

where $\{\mu_{k+1}, \ldots, \mu_{N-1}\}$ is the tail portion of the base policy.
First generated the next state from the current state

$$x_{k+1}^1 = f_k(x_k^j, u_k^1, w_k),$$

The cost of the trajectories corresponding to a pair (x_k^j, u_k^i) can be viewed as samples of the Q-factor

$$Q_k(x_k^j, u_k^i) = E_{w_k}\Big\{g_k(x_k^j, u_k^i, w_k) + H_{k+1,\pi}(f_k(x_k^j, u_k^i, w_k))\Big\},$$

where $J_{k+1,\pi}$ is the cost-to-go function of the base policy. Subscript π is to highlight the policy that is being used to compute the cost-to-go values is the base policy.

Approximate Q-factor $\tilde{Q}_k(x_k^j, u_k^i)$ can be obtained by Monte Carlo averaging the costs of the sample trajectories plus their corresponding terminal costs (if any). We then compute the rollout control $\mu_k(x_k^j)$ with the minimization

$$\mu_k(x_k^j) \in \arg\min_{u_k^i \in U_k(x_k^j)} \tilde{Q}_k(x_k^j, u_k^i)$$

Note, the simulation effort will be huge if we were to explore all possible $u_k^i \in U_k(x_k^j)$. This is the main reason we are adapting the Monte Carlo Tree Search to the stochastic rollout algorithm.

Following is the detailed procedure of one-step lookahead stochastic rollout:

For $k = i, \ldots, N-1$, where i can range from 0 (means the algorithm starts from the initial stage) to $N-1$ (the second last stage), perform the following steps:

1. For each $i = 1, \ldots, m_k$, compute all possible next state from a single current state (initialize $k = i$). All possible next states are

$$x_{i+1}^j = f_i(x_i, u_i^j, w_i)$$

   (a) For each $t = k, \ldots, N-1$, solve the tail subproblem by completing trajectories using the tail portion of the base policy $\{\mu_{k+1}, \ldots, \mu_{N-1}\}$:

$$u_t^j = \mu_t(x_t^j)$$
$$x_{t+1}^j = f_t(x_t^j, u_t^j w_i)$$

For $k =, \ldots, 0$, we repeat the following steps:

1. **One-step lookahead:** for each $j = 1, \ldots, m_k$, compute all possible previous state from a single current state: $x_{k+1}^j = f_k(x_k, u_k^j, w_k)$

For each $t = k, \ldots, N-1$, we complete trajectories by computing:

$$u_t^j = \mu_t(x_t^j)$$
$$x_{t+1}^j = f_t(x_t^j, u_t^j, w_t)$$

2. **Obtain simulation results of each trajectory:** for the current trajectory, define $E_{w_k}\left\{H_{k+1,\pi}(x_{k+1}^j)\right\}$ to be the expected cost of the trajectory generated by the heuristic starting from x_k. (i.e., either it performs the completion of the trajectories from state x_{k+1} and computes the cost of the trajectory or it approximate the cost-to-go at state x_k using a base heuristic).

3. **Pick the "best" action:** choose the action u_k such that the approximate Q-factor is the lowest among all the approximate Q-factors of trajectories generated from state x_k. Denote $\tilde{u}_k$ as the suboptimal control, and $\tilde{\mu}_k$ as the suboptimal rollout policy.

$$\tilde{u}_k \in \arg\min_{j=1,\ldots,m_k} E_{w_k}\left\{g_k(x_k^j, u_k^j, w_k) + H_{k+1,\pi}(f_k(x_k^j, u_k^j, w_k))\right\}$$

This can then be written as:

$$\tilde{u}_k \in \arg\min_{u_k \in U_k} E_{w_k}\left\{g_k(x_k^j, u_k, w_k) + H_{k+1,\pi}(f_k(x_k^j, u_k, w_k))\right\}$$
$$\tilde{u}_k \in \arg\min_{u_k \in U_k} \tilde{Q}(x_k^j, u_k)$$
$$\tilde{\mu}_k(x_k^j) \in \arg\min_{u_k \in U_k} \tilde{Q}(x_k^j, u_k)$$

4. **Transition and repeat:** after obtaining the $\tilde{\mu}_k(x_k^j) = \tilde{u}_k$ which minimize the approximate Q-factors, we apply this control and proceed to the next state $x_{k+1}^j = f_{x+1}(x_k^j, \tilde{\mu}_k(x_k^j), w_k)$ and repeat until we reach the terminal state, x_N^j

   Hence, we have generated the rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ and we can use it to obtain the suboptimal trajectory.

   Note: There are several drawbacks of this algorithm. They are discussed in section 1.2.1. These drawbacks motivate the MCTS algorithm. Instead of obtaining simulation results for all trajectories that pointing outwards from the current state, MCTS selects a more promising action for rollout to explore. Then rollout will complete this single-trajectory simulation. The advantage of this is discussed in section 1.2.2.

Bibliography

[1] Dimitri P. Bertsekas. "2.4. Rollout". In: *Reinforcement learning and optimal control*. Athena Scientific, 2019, pp. 36–53.

[2] Margaret P. Chapman, Michael Fauss, and Kevin M. Smith. "On Optimizing the Conditional Value-at-Risk of a Maximum Cost for Risk-Averse Safety Analysis". In: *arXiv:2106.00776 [cs, eess]* (Dec. 22, 2021). arXiv: 2106.00776. URL: http://arxiv.org/abs/2106.00776 (visited on 02/12/2022).

[3] Margaret P. Chapman et al. "Risk-sensitive safety analysis via state-space augmentation". In: *IEEE Transactions on Automatic Control* (2021).

[4] Nikolaos V. Sahinidis. "Optimization under uncertainty: state-of-the-art and opportunities". In: *Computers & Chemical Engineering* 28.6 (June 2004), pp. 971–983. ISSN: 00981354. DOI: 10.1016/j.compchemeng.2003.09.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0098135403002369 (visited on 01/20/2022).

[5] Ignacio E. Grossmann et al. "Recent advances in mathematical programming techniques for the optimization of process systems under uncertainty". In: *Computers & Chemical Engineering*. 12th International Symposium on Process Systems Engineering & 25th European Symposium of Computer Aided Process Engineering (PSE-2015/ESCAPE-25), 31 May - 4 June 2015, Copenhagen, Denmark 91 (Aug. 4, 2016), pp. 3–14. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2016.03.002. URL: https://www.sciencedirect.com/science/article/pii/S0098135416300540 (visited on 01/20/2022).

[6] Yuheng Wang and Margaret P. Chapman. "Risk-averse autonomous systems:A brief history and recent developments from the perspective of optimal control". In: *Journal of Artificial Intelligence* (2021).

[7] Andrzej Ruszczyński. "Risk-averse dynamic programming for Markov decision processes". In: *Mathematical Programming* 125.2 (Oct. 2010), pp. 235–261. ISSN: 0025-5610, 1436-4646. DOI: 10.1007/s10107-010-0393-3. URL: http://link.springer.com/10.1007/s10107-010-0393-3 (visited on 01/30/2022).

[8] Yinlam Chow et al. "Risk-Sensitive and Robust Decision-Making: a CVaR Optimization Approach". In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015. URL: https://www.researchgate.net/publication/277959194_Risk-Sensitive_and_Robust_Decision-Making_a_CVaR_Optimization_Approach (visited on 01/30/2022).

[9] Mohamadreza Ahmadi et al. “Risk-Averse Planning Under Uncertainty”. In: *2020 American Control Conference (ACC)*. 2020 American Control Conference (ACC). ISSN: 2378-5861. July 2020, pp. 3305–3312. DOI: 10.23919/ACC45564.2020.9147792.

[10] Pantelis Sopasakis et al. “Risk-averse model predictive control”. In: *Automatica* 100 (Feb. 1, 2019), pp. 281–288. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2018.11.022. URL: https://www.sciencedirect.com/science/article/pii/S0005109818305545 (visited on 01/30/2022).

[11] Dimitri P Bertsekas, John N Tsitsiklis, and Cynara Wu. “Rollout Algorithms for Combinatorial Optimization”. In: *Journal of Heuristics* (1997), p. 18.

[12] Dimitri P. Bertsekas and David A. Castanon. “Rollout Algorithms for Stochastic Scheduling Problems”. In: *Journal of Heuristics* 5.1 (Apr. 1, 1999), pp. 89–108. ISSN: 1572-9397. DOI: 10.1023/A:1009634810396. URL: https://doi.org/10.1023/A:1009634810396 (visited on 01/19/2022).

[13] Justin Goodson, Jeffrey Ohlmann, and Barrett Thomas. “Rollout Policies for Dynamic Solutions to the Multivehicle Routing Problem with Stochastic Demand and Duration Limits”. In: *Operations Research* 61.1 (Feb. 1, 2013), pp. 138–154. DOI: 10.1287/opre.1120.1127.

[14] Nicola Secomandi. “A Rollout Policy for the Vehicle Routing Problem with Stochastic Demands”. In: *Operations Research* 49.5 (Oct. 1, 2001). Publisher: INFORMS, pp. 796–802. ISSN: 0030-364X. DOI: 10.1287/opre.49.5.796.10608. URL: https://pubsonline.informs.org/doi/abs/10.1287/opre.49.5.796.10608 (visited on 01/19/2022).

[15] Clara Novoa and Robert Storer. “An approximate dynamic programming approach for the vehicle routing problem with stochastic demands”. In: *European Journal of Operational Research* 196.2 (July 2009), pp. 509–515. ISSN: 03772217. DOI: 10.1016/j.ejor.2008.03.023. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221708003172 (visited on 02/06/2022).

[16] Justin Goodson, Barrett Thomas, and Jeffrey Ohlmann. *A Rollout Algorithm Framework for Heuristic Solutions to Finite-Horizon Stochastic Dynamic Programs*. July 21, 2015. DOI: 10.13140/RG.2.1.3908.3365.

[17] Guillaume Chaslot et al. “Monte-Carlo Tree Search: A New Framework for Game AI.” In: Bijdragen. Jan. 1, 2008.

[18] Rémi Coulom. “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search”. In: *Computers and Games*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 72–83. ISBN: 978-3-540-75538-8. DOI: 10.1007/978-3-540-75538-8_7.

[19] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Machine Learning: ECML 2006*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Red. by David Hutchison et al. Vol. 4212. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293. ISBN: 978-3-540-45375-8 978-3-540-46056-5. DOI: 10.1007/11871842_29. URL: http://link.springer.com/10.1007/11871842_29 (visited on 01/31/2022).

[20] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *arXiv:1712.01815 [cs]* (Dec. 5, 2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815 (visited on 01/20/2022).

[21] Sylvain Gelly et al. “The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions”. In: (2013), p. 10.

[22] Dimitri P. Bertsekas. “2.1.5 Approximation in Policy Space on Top of Approximation in Value Space”. In: *Reinforcement learning and optimal control*. Athena Scientific, 2019, pp. 12–13.

[23] Dimitri P. Bertsekas. “Reinforcement learning and optimal control”. In: *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[24] Cameron B. Browne et al. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (Mar. 2012), pp. 1–43. ISSN: 1943-068X, 1943-0698. DOI: 10.1109/TCIAIG.2012.2186810. URL: http://ieeexplore.ieee.org/document/6145622/ (visited on 01/19/2022).

[25] Julian N. Marewski and Gerd Gigerenzer. “Heuristic decision making in medicine”. In: *Dialogues in Clinical Neuroscience* 14.1 (Mar. 2012), pp. 77–89. ISSN: 1294-8322. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3341653/ (visited on 04/14/2022).