# Risk-Aware and Multi-Objective Decision Making with Distributional Monte Carlo Tree Search[*]

Conor F. Hayes
National University of Ireland Galway
Ireland
c.hayes13@nuigalway.ie

Mathieu Reymond
Vrije Universiteit Brussel
Belgium
mathieu.reymond@vub.be

Diederik M. Roijers
Vrije Universiteit Brussel (BE) &
HU Univ. of Appl. Sci. Utrecht (NL)
diederik.yamamoto-roijers@hu.nl

Enda Howley
National University of Ireland Galway
Ireland
enda.howley@nuigalway.ie

Patrick Mannion
National University of Ireland Galway
Ireland
patrick.mannion@nuigalway.ie

## ABSTRACT

In many risk-aware and multi-objective reinforcement learning settings, the utility of the user is derived from the single execution of a policy. In these settings, making decisions based on the average future returns is not suitable. For example, in a medical setting a patient may only have one opportunity to treat their illness. When making a decision, just the expected return – known in reinforcement learning as the value – cannot account for the potential range of adverse or positive outcomes a decision may have. Our key insight is that we should use the distribution over expected future returns differently to represent the critical information that the agent requires at decision time. In this paper, we propose Distributional Monte Carlo Tree Search, an algorithm that learns a posterior distribution over the utility of the different possible returns attainable from individual policy executions, resulting in good policies for both risk-aware and multi-objective settings. Moreover, our algorithm outperforms the state-of-the-art in multi-objective reinforcement learning for the expected utility of the returns.

## KEYWORDS

Multi-objective; risk-aware; decision making; distributional; reinforcement learning; Monte Carlo tree search

## 1 INTRODUCTION

In reinforcement learning (RL) settings, the expected return is used to make decisions. In many scenarios, the utility of a user is derived from the single execution of a policy [32]. For example, in a medical setting a patient may only have one opportunity to select a treatment. To learn an optimal policy in these scenarios it is important to optimise under the utility of the returns. Therefore, if a policy will only be executed once, making decisions using the expected utility of the returns is not sufficient. For example, imagine we have two choices: bet or don't bet. If we bet there is a 0.5 chance of winning, which gives us a reward of 40, and a 0.5 chance of losing, which returns a reward of -20. If we do not bet, we get a reward of 10. Both of these choices have the same expected reward, with betting potentially returning a negative reward. However, once we consider a human decision maker this story becomes different; if a person receives -20 they would be in debt, and this could have a severe adverse effect on this person's well-being. While getting 40 might be nice, it may not worth the risk of going into debt. Therefore, the decision maker would prefer the non-betting strategy. As such, in order for an agent to have sufficient critical information at decision time, it is crucial to replace the expected return with a posterior distribution over the expected utility of returns. This realisation is key to risk-aware systems and, as we argue, for many multi-objective decision problems as well.

In the aforementioned scenarios, the agent must calculate the returns of a full execution of a policy before deriving the user's utility. To calculate the utility we apply the utility function to the returns where a user's utility function is known a priori. In other words, in the taxonomy of multi-objective sequential decision making by [32], we are in the known-weights non-linear utility setting. When optimising under the expected utility, it is critical to only apply the utility function to the returns of a full execution of a policy [31]. This is a critical step since non-linear utility functions do not distribute across the sum of immediate and future returns [16, 31]. In this case, the agent must know the returns it has already accrued and the future returns before applying the utility function. For example, before the 2008 financial crash, investment bankers were guaranteed their base salaries regardless of their losses, but their bonuses were dependent on their returns from investments. In the case of an investor incurring a loss, the only policy that would result in a bonus would be one that executes an increasingly risky strategy to win back the losses and receive some bonus.

Learning the utility of the returns is thus naturally risk-aware. Optimising the utility of the sum of the accrued and future returns to make decisions enables an agent to avoid certain undesirable outcomes. Without knowing the accrued returns, an agent cannot understand how future actions could affect the cumulative return. To make optimal decisions to maximise the user's utility, the agent must have information about both the accrued and the future returns.

A further complicating factor is that, in the real world, decision-making often involves trade-offs based on multiple conflicting objectives. For example, we may want to maximise the power output of coal-burning electrical generators while minimising $CO_2$ emissions. Many approaches to multi-objective decision-making only consider linear utility functions; this limitation severely restricts the real-world applicability of these methods, given that utility in many real-world problems is derived in a non-linear manner.

[*] This paper extends our AAMAS 2021 extended abstract [15] with additional experimental results and theoretical analysis.

In the multi-objective case, optimising under the expected utility is known as the expected scalarised returns (ESR). For MORL, the utility function expresses the user's preferences over objectives. If the utility function is linear and is known a priori, it is possible to translate a multi-objective decision problem to its single-objective equivalent. Once translated, we can then apply single objective methods to solve the decision problem. However, if the utility function is non-linear, as human preferences often are, strictly multi-objective methods are required to find optimal solutions. We note that the ESR criterion is an understudied challenge in the MORL literature and very few methods that consider the utility of the expected return exist in the current literature.

We propose a novel algorithm, Distributional Monte Carlo Tree Search (DMCTS), which learns a posterior distribution over the expected utility of the returns. DMCTS learns a posterior distribution over the utility of the returns by executing multiple individual policies and calculating the utility of the returns obtained from each policy execution. DMCTS builds upon Monte Carlo Tree Search (MCTS). MCTS is a heuristic search algorithm and has become a highly popular framework [17, 35, 40]. Learning a posterior distribution over the utility of returns overcomes the issues present when making decisions solely with the expected return. Our key insight is that learning a posterior distribution over the utility of the returns is essential when optimising for risk-aware RL and under the MORL ESR criterion. A distribution contains more information about the range of potential negative and positive outcomes at decision time. An added feature of DMCTS is the ability to also optimise under the scalarised expected returns (SER) criterion. We implement and demonstrate DMCTS for both risk-aware and multi-objective problems. DMCTS learns good polices in risk-aware settings. Moreover, DMCTS outperforms the state-of-the-art in MORL under ESR.

# 2 BACKGROUND

## 2.1 Multi-Objective Reinforcement Learning

In multi-objective reinforcement learning (MORL), we deal with decision problems with multiple objectives, often modelled as a multi-objective Markov decision process (MOMDP). An MOMDP represents a tuple, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R})$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a probabilistic transition function, $\gamma$ is a discount factor determining the importance of future rewards and $\mathcal{R}\colon \mathcal{S}\times\mathcal{A}\times\mathcal{S} \rightarrow \mathbb{R}^n$ is an $n$-dimensional vector-valued immediate reward function. In multi-objective reinforcement learning (MORL), $n > 1$.

In MORL, the user's utility derives from the vector-valued outcomes (returns). This is typically modelled as a utility function that needs to be applied to these outcomes in one way or another. For this, there are two choices [16, 32]. Calculating the expected value of the return of a policy before applying the utility function leads to the scalarised expected returns (SER) optimisation criterion:

$$V_u^{\pi} = u\left(\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \mid \pi, \mu_0\right]\right). \tag{1}$$

SER is the most commonly used criterion in the multi-objective (single agent) planning and reinforcement learning literature [32]. For SER, a coverage set is defined as a set of optimal solutions for all possible utility functions. If the utility function is instead applied before computing the expectation, this leads to the expected scalarised returns (ESR) optimisation criterion [31]:

$$V_u^{\pi} = \mathbb{E}\left[u\left(\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t\right) \mid \pi, \mu_0\right]. \tag{2}$$

ESR is the most commonly used criterion in the game theory literature on multi-objective games [30].

## 2.2 Monte Carlo Tree Search

One way of approaching a decision problem (in RL) is to use tree search. Perhaps the most popular of such methods is Monte Carlo Tree Search (MCTS) [8], which employs heuristic exploration to construct its search tree. MCTS builds a search tree of nodes, where each node has a number of children. Each child node corresponds to an action available to the agent. MCTS has two phases: the learning phase and the execution phase.

In the learning phase the agent implements the following four steps [6]: selection, expansion, simulation and backpropagation. **Selection:** the agent traverses the search tree until it reaches a node that not been explored, also called a leaf node. **Expansion:** at a leaf node the node must be expanded. The agent creates a random child node and then must simulate the environment for the newly created child node. **Simulation:** the agent executes a random policy through Monte Carlo simulations until a terminal state of the environment is reached. The agent then receives the returns. **Backpropagation:** the agent must backpropagate the returns received at a terminal state to each node visited during selection where a predefined algorithm statistic e.g. UCT [8, 17] is updated. Each step is repeated a specified number of times, which incrementally builds the search tree. Or, as we will discuss in the next subsection, a posterior belief on the returns, from which we can draw actions using Thompson sampling [3].

During the execution phase the agent must select a child node to traverse to next. The agent evaluates the statistic at each node and moves to the node which returns the maximum value; the selection phase is then re-executed. An episode ends when the execution phase arrives at a terminal state.

## 2.3 (Bootstrap) Thompson Sampling

As previously mentioned, during the learning phase of MCTS, we can use Thompson sampling to take exploring actions [3]. However, it is not always possible to get an exact posterior. In this case a bootstrap distribution over means can be used to approximate a posterior distribution [11, 24]. Eckles et al. [9, 10] use a bootstrap distribution to replace the posterior distribution used in Thompson Sampling. This method is known as Bootstrap Thompson Sampling (BTS) [9] and was proposed in the multi-arm bandit setting. The bootstrap distribution contains a number of bootstrap replicates, $j \in \{1, ..., J\}$, where $J$ is a hyper-parameter that can be tuned for exploration. For a small $J$, BTS can become greedy. A larger $J$ value increases exploration, but at a computational cost [9].

Each bootstrap replicate, $j$, in the bootstrap distribution contains two parameters, $\alpha_j$ and $\beta_j$, where $\frac{\alpha_j}{\beta_j}$ is an observation. At decision time, to determine the optimal action the bootstrap distribution for each arm, $i$, is sampled. The observation for the corresponding bootstrap replicate, $j$, is retrieved and the arm with the maximum observation is pulled [9].

The distribution which corresponds to the maximum arm is randomly re-weighted by simulating a coin-flip (commonly known as sampling from a Bernoulli bandit) for each bootstrap replicate, $j$, in the bootstrap distribution. If the coin-flip is heads, the observation for $j$ is re-weighted. To re-weight an observation, the return is added to the $\alpha_j$ value and 1 is added to $\beta_j$ [9].

Bootstrap methods with random re-weighting [33] are more computationally appealing as they can be conducted online rather than re-sampling data [26]. BTS addresses problems of scalability and robustness when compared to Thompson Sampling [9].

## 2.4 Expected Utility Policy Gradient

Expected Utility Policy Gradient (EUPG) is a MORL algorithm for ESR [31]. EUPG is an extension of Policy Gradient [36, 41], where Monte Carlo simulations are used to compute the returns and optimise the policy. EUPG calculates the accrued returns, $\mathbf{R}_t^-$, which is the sum of the immediate returns received as far as the current timestep, $t$. EUPG also calculates the future returns, $\mathbf{R}_t^+$, which is the sum of the immediate returns from the current timestep, $t$, to the terminal state. Using both the accrued and future returns enables EUPG to optimise over the utility of the full returns of an episode, where utility function is applied the sum of $\mathbf{R}_t^-$ and $\mathbf{R}_t^+$.

Roijers et al. [31] showed for ESR the accrued and future returns must be considered when learning. Applying this consideration to EUPG, the algorithm achieves the state-of-the-art performance under ESR. In this paper, we use the same method of adding past and future returns together before applying the utility inside of the MCTS search scheme.

# 3 DISTRIBUTIONAL MONTE CARLO TREE SEARCH

The majority of RL research focuses on learning an optimal policy based on the expected returns, known as the value. Under the expected scalarised returns (ESR), a single execution of a policy is used to derive the utility of a user. In the outlined scenario, it is crucial that the agent has sufficient information at decision time to exploit positive outcomes and avoid negative outcomes. Under ESR, taking actions based on the expected returns fails to provide the agent with this information. However, acting based on a distribution over the expected utility of returns overcomes this issue. In Section 3, we present our Distributional Monte Carlo Tree Search (DMCTS) algorithm which learns a posterior distribution over the expected returns.

DMCTS builds an expectimax search tree through the same process as MCTS. A search tree is a representation of the state-action space that is incrementally built though Monte Carlo simulations. The search tree is built using nodes, where a node represents an action available to the agent. Each node has a number of children corresponding to the number of actions available (see Section 2.2).

An expectimax search tree [39] uses both decision and chance nodes. Each decision node represents a state, action and reward of an MOMDP, and has a child chance node per action. In this paper we examine only environments with stochastic rewards. Each chance node represents the state and action of an MOMDP. At each chance node, the environment is sampled. If an unseen reward is generated when sampling the environment, a new child decision node is created. This process repeats as the agent traverses the search tree. It is important to note that each chance node and its parent decision node share the same state and action. A child decision node is only created when an unseen reward is observed from sampling the environment. DMCTS uses the phases of selection, expansion, simulation and backpropagation to build and traverse the search tree, similar to MCTS (Section 2.2).

Usually an expectation of the returns is maintained at each chance node, and the agent seeks to maximise the expectation. Making decisions based on the expected returns does not account for potential undesired outcomes. For risk-aware RL and MORL under ESR, we need to be able to make decisions with sufficient information to avoid such undesirable outcomes. Under these conditions, an alternative to making decisions based on the expected returns must be found.

Learning a posterior distribution over the utility of the returns can be used to replace the expected future returns (of vanilla MCTS) at each node. We outline our algorithm for single-objective risk-aware RL and MORL under ESR. With minor changes to our algorithm we can also apply DMCTS to multi-objective RL under SER, which we will discuss briefly at the end of this section.

To compute the distribution we first calculate the accrued returns, $\mathbf{R}_t^-$. The accrued returns is the sum of returns received during the execution phase as far as timestep, $t$, where $\mathbf{r}_t$ is the reward received at each timestep,

$$\mathbf{R}_t^- = \sum_{0}^{t-1} \mathbf{r}_t.$$

Secondly, we must calculate future returns, $\mathbf{R}_t^+$. The future returns is the sum of the rewards received when traversing the search tree during the learning phase and Monte Carlo simulations from timestep, $t$, to a terminal node, $t_n$,

$$\mathbf{R}_t^+ = \sum_{t}^{t_n} \mathbf{r}_t.$$

The cumulative returns, $\mathbf{R}_t$, is the sum of the accrued returns, $\mathbf{R}_t^-$, and the expected future returns, $\mathbf{R}_t^+$,

$$\mathbf{R}_t = \mathbf{R}_t^- + \mathbf{R}_t^+.$$

$\mathbf{R}_t$ is backpropagated to each node in the search tree, where the utility is computed, $u(\mathbf{R}_t)$. Since non-linear utility functions do not distribute across the sums for the immediate or future returns, we must calculate the cumulative returns, $\mathbf{R}_t$. Applying the utility function to the cumulative returns, $\mathbf{R}_t$, ensures we satisfy the ESR criterion. In single-objective RL where risk is not considered, the expected future returns are sufficient to base the optimal action on. By contrast, for risk-aware and ESR-MORL it is essential to use the cumulative returns $\mathbf{R}_t$ to determine the optimal action, as we have argued before. In this paper, we do not use discounting as we perform evaluations only on finite horizon tasks. We note that DMCTS can easily be adapted to discounted settings.

At each node we aim to maintain a posterior distribution over the expected utility of the returns. However, because the utility function may be non-linear, a parametric form of the posterior distribution may not exist. Since a bootstrap distribution can be used to approximate a posterior [11, 24], it is much more suitable to maintain a bootstrap distribution over the expected utility of the returns at each node.

Each bootstrap distribution contains a number of bootstrap replicates, $j \in \{1, ..., J\}$ [9] (See Section 2.3). On initialisation of a new node, for each bootstrap replicate, $j$, the parameters $\alpha_j$ and $\beta_j$ are both set to 1. Moreover, $\alpha_j$ can be set to positive values to increase initial exploration without a computational cost.

During the backpropagation phase the bootstrap distribution at each node is updated. Algorithm 1 outlines how a bootstrap distribution for a node is updated. At node $i$, for each bootstrap replicate, $j$, a coin flip is simulated (See Algorithm 1, Line 5). If the result of the coin flip is equal to 1 (heads), $\alpha_{ij}$ and $\beta_{ij}$ are updated:

$$\alpha_{ij} = \alpha_{ij} + u(\mathbf{R}_t)$$

$$\beta_{ij} = \beta_{ij} + 1$$

To select actions while learning, we use the previously computed statistics. At each timestep the agent must choose which action to execute in order to traverse the search tree (as outlined in Algorithm 2). At node $n$, we select an action by sampling the bootstrap distribution at each child node, $i$. For each sampled bootstrap replicate, $j$, the $\alpha_{ij}$ and $\beta_{ij}$ values are retrieved and $\frac{\alpha_{ij}}{\beta_{ij}}$ is computed. Since the following is true,

$$\frac{\alpha_{ij}}{\beta_{ij}} \equiv \mathbb{E}[u(\mathbf{R}_t^- + \mathbf{R}_t^+)], \tag{3}$$

by maximising over $i$ in Equation 3, we select an action corresponding to $j$ approximately proportionally to the probability of that action being optimal – as per the Bootstrap Thompson Sampling exploration strategy. The agent then executes the action, $a^*$, which corresponds to the following:

$$a^* = \arg\max_i \frac{\alpha_{ij}}{\beta_{ij}}.$$

We note that at execution time we can simply select the overall maximising action by averaging over all the acquired data (ignoring the bootstrap replicates), thereby maximising the ESR criterion:

$$ESR = \mathbb{E}[u(\mathbf{R}_t^- + \mathbf{R}_t^+)]. \tag{4}$$

Using the outlined algorithm, DMCTS is able to learn optimal policies for risk-aware settings and under ESR for multi-objective settings. In Section 4 we have evaluated DMCTS for risk-aware settings and multi-objective settings under ESR.

The majority of MORL research focuses on the SER criterion rather than the ESR criterion [32]. With a minor change to the algorithm it is also possible for DMCTS to optimise for the SER criterion. Specifically, under the SER criterion we maintain a bootstrap distribution over expected return vectors. For a node under SER, it is important to ensure that $\boldsymbol{\alpha}$ is initialised to a vector for each bootstrap replicate, $j$. The number of values in the bootstrap replicate vector, $\boldsymbol{\alpha}_j$, corresponds to the number of objectives, $o$, where each value is set to 1, $\boldsymbol{\alpha}_j = [1, ..., 1_o]$. The parameter $\beta$ is set to 1 for each bootstrap replicate, $j$.

To update the bootstrap distribution of node, $i$, we use the same process as under ESR (see Algorithm 1). At each node a coin flip is simulated for each bootstrap replicate, $j$. If the simulated coin flip returns 1 (heads), then we update the bootstrap replicate. We use the following to update $\boldsymbol{\alpha}_{ij}$ and $\beta_{ij}$:

$$\boldsymbol{\alpha}_{ij} = \boldsymbol{\alpha}_{ij} + \mathbf{R}_t,$$

$$\beta_{ij} = \beta_{ij} + 1.$$

At learning time, we sample the bootstrap distribution at each child node, $i$. For a sampled bootstrapped replicate, $j$, the parameters $\boldsymbol{\alpha}_{ij}$ and $\beta_{ij}$ are retrieved. Before we can determine the optimal action, we must compute $\frac{\boldsymbol{\alpha}_{ij}}{\beta_{ij}}$. We then apply the utility function, $u$, to $\frac{\boldsymbol{\alpha}_{ij}}{\beta_{ij}}$ to compute the utility of the expected returns. Since,

$$u(\frac{\boldsymbol{\alpha}_{ij}}{\beta_{ij}}) \equiv u(\mathbb{E}[\mathbf{R}_t^- + \mathbf{R}_t^+]), \tag{5}$$

the agent can then execute the action, $a^*$, which corresponds to the following:

$$a^* = \arg\max_i u(\frac{\boldsymbol{\alpha}_{ij}}{\beta_{ij}}).$$

**Algorithm 1:** UpdateDistribution

```
1  Input: i ⟵ Node in the tree
2  Input: u($R_t$)  ⟵ Cumulative Reward
3  J ⟵ node.bootstrapDistribution
4  for j, ..., J bootstrap replicates do
5      Sample $d_j$ from Bernoulli(1/2)
6      if $d_j$ = 1 then
7          $\alpha_{ij} = \alpha_{ij} + u(R_t)$
8          $\beta_{ij} = \beta_{ij} + 1$
9      end
10 end
```

**Algorithm 2:** ThompsonSample

```
Input: n ⟵ Node in the tree
Require: $\alpha$, $\beta$ prior parameters
$\alpha_{ij} := \alpha$, $\beta_{ij} := \beta$ {For each n child, i, and each bootstrap
  replicate, j }
for i, ..., n children do
    Sample j from uniform 1, ..., J bootstrap replicates
    Retrieve $\alpha_{ij}$, $\beta_{ij}$
end
maxChild = arg max$_i$ $\frac{\alpha_{ij}}{\beta_{ij}}$
return maxChild or maxChild.action
```

# 4 EXPERIMENTS

In order to evaluate our DMCTS algorithm, we test DMCTS in multiple settings. Firstly, we evaluate DMCTS in a risk-aware setting. Secondly, we evaluate DMCTS in multi-objective settings under both ESR and SER. In multi-objective settings we test our algorithm on variants of standard benchmark problems from the MORL literature. At each timestep for DMCTS, the learning phase is performed multiple times before an action is selected during the execution phase. To fairly evaluate all other algorithms against DMCTS, we have altered each benchmark algorithm to have the same number of policy executions of each environment at each timestep as DMCTS. So at each timestep, each algorithm gets $n_{exec}$ full policy executions worth of learning from that state and timestep onward. For the other algorithms (except DMCTS) this has the effect of increasing the learning speed. The number of policy executions $n_{exec}$ varies for each problem domain. All experiments are averaged over 10 runs.

## 4.1 Risk-Aware MDP

Before testing DMCTS on benchmark problems from the MORL literature, we evaluate DMCTS in a risk-aware problem domain under ESR. Shen et al. [34] define a risk-aware MDP where an agent must decide from a number of stocks in which to invest. The underlying MDP which has 4 actions (each action is a monetary amount, in Euros, of investment) and 7 states. At each timestep the agent must select a monetary amount to invest in the stock for a given state. We can invest €0, €1, €2 or €3 in a stock at each timestep. Each stock has a probability of making a profit and a probability of making a loss where the agent's return is the action multiplied by the stock price. In the risk-aware MDP, certain policies are risk-averse, risk-seeking or a mixture of both. For example, if an agent takes action 0 or action 1 at each timestep, the agent is said to be risk-averse. Executing action 0 at each timestep is the most risk-averse policy that an agent can learn. Investing €0 at each timestep means the agent has no gains or losses, given the returns are equal to the monetary investment multiplied by the stock price. The type of policy an agent learns depends on the utility function. For certain utility functions, the agent could be risk-seeking or risk-averse. To evaluate our DMCTS algorithm we use the following risk-averse non-linear utility function:

$$u = 1 - e^{-r_t}. \tag{6}$$

In the risk-aware setting, we compare our algorithm against Q-learning, where we aim to learn the policy that is risk-averse. The parameter $n_{exec}$ is set to 10 for each algorithm and each experiment lasts for 1,000 episodes.

As shown in Figure 1, DMCTS consistently learns the optimal policy for the above risk-averse utility function. The policy, which avoids all risk, has a cumulative utility of 0. DMCTS needs around 400 episodes to converge to the optimal policy, while Q-learning struggles to learn a stable policy for the given utility function. Maintaining a bootstrap distribution over the expected utility of the returns enables DMCTS to avoid all risk. The ability for an agent to access a distribution when learning ensures the agent can make more informed decisions to maximise its utility which, in this case, is risk-averse.

## 4.2 Fishwood

To evaluate DMCTS in a multi-objective setting under ESR, we use a number of problem domains. Firstly, we evaluate DMCTS in the Fishwood problem [31], given this is one of the very few domains for which ESR results have been published. In Fishwood the agent has two states: at the river or in the woods, two actions: move to the other state or stay, and two objectives: to catch fish (when at the river) and obtain wood (when in the woods). The Fishwood environment is parameterised by the probabilities of successfully obtaining fish and wood at these respective states. In this paper we use the following values: at the river the agent has a 0.25 chance of catching a fish and in the woods the agent has a 0.65 chance of acquiring wood. For every fish caught, two pieces of wood are required to cook the fish, which results in a utility of 1. The goal in this setting is to maximise the following non-linear utility function:

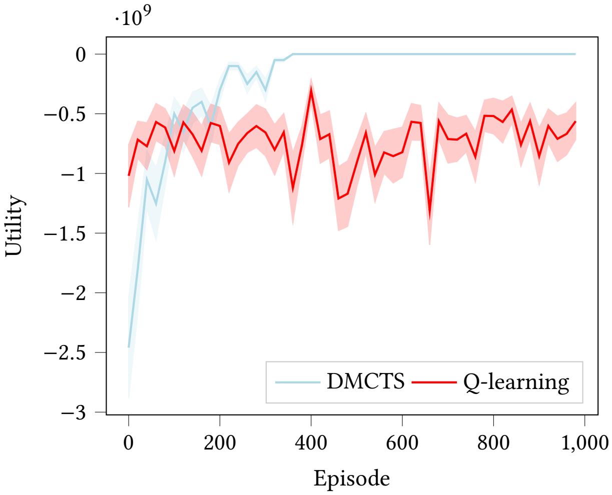

**Figure 1: Results from the risk-aware environment. Learning a bootstrap distribution over the expected utility of the returns (DMCTS) is critical to learning the optimal risk-averse policy for a risk-averse utility function.**

$$u = \min\left(\texttt{fish}, \left\lfloor \frac{\texttt{wood}}{2} \right\rfloor\right). \tag{7}$$

To maximise utility in Fishwood it is essential that both past and future returns are taken into consideration when learning. For example, if there are 5 timesteps remaining and the agent has received 2 pieces of wood, the agent should go to the river and try to catch a fish to ensure a utility of 1 [31].

To evaluate DMCTS in the Fishwood domain, we compare DMCTS against C51, Expected Utility Policy Gradient (EUPG) [31], and Q-learning. EUPG achieves state-of-the-art results in the Fishwood problem under ESR [31]. C51 [5] is a distributional deep reinforcement learning algorithm that achieved state-of-the-art results in the Atari game problem domain.

For C51 the learning parameters were set as follows: $V_{min} = 0$, $V_{max} = 2$, $\epsilon = 0.1$, $\gamma = 1$ and $\alpha = 0.1$. For Q-learning, the learning parameters were set as follows: $\epsilon = 0.1$, $\gamma = 1$ and $\alpha = 0.1$. For DMCTS we set the $\alpha_j$ parameter to 10 for each bootstrap replicate, $j$. We set $n_{exec} = 2$ and ran each experiment for 20,000 episodes where each episode has 13 timesteps. As shown in Figure 2, Q-learning and C51 fail to learn any meaningful policy. The utility for C51 fluctuates throughout experimentation and fails to learn a consistent policy, while the utility for Q-learning remains close to zero. This is because Q-learning and C51 do not take the accrued returns into consideration when learning, which has a negative

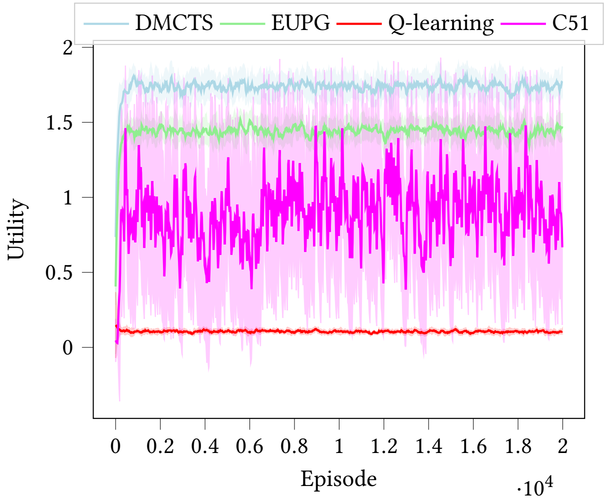

**Figure 2: Results from the Fishwood environment where DMCTS achieves state-of-the-art performance in a multi-objective setting over EUPG.**

impact on the ability of both algorithms to learn in the Fishwood domain.

By contrast, DMCTS and EUPG outperform both C51 and Q-learning. DMCTS achieves a higher utility when compared to EUPG. Both algorithms use Monte Carlo simulations of the environment and optimise over the expected utility of the returns of a full episode. Although both algorithms use Monte Carlo simulations of the environment, policy gradient algorithms are sample inefficient. DMCTS is sample efficient since DMCTS shares the learning phase steps with MCTS, which has been shown to be sample efficient [2, 7]. In the Fishwood environment, the agent is not guaranteed to obtain a fish or a piece of wood. For an action in a particular state the agent may need multiple simulations to understand the underlying distribution of the stochastic rewards. Since DMCTS builds an expectimax tree, the agent can re-sample the environment at each chance node when learning. With repeated sampling at each chance node and Monte Carlo simulations, the agent can build an accurate bootstrap distribution over the expected utility of returns. Using the learned bootstrap distribution over the expected utility of the returns enables DMCTS to outperform EUPG and achieve state-of-the-art performance under ESR.

### 4.3 Renewable Energy Dynamic Economic Emissions Dispatch

Next, we evaluate our DMCTS algorithm in a complex problem domain with a large state action space. Renewable Energy DEED (REDEED) is a variation of the traditional DEED problem [4]. In REDEED, the power demand for a city must be met over 24 hours. To supply the city with sufficient power, a number of generators are required. There are 9 fossil fuel-powered generators, including a slack generator and 1 generator powered by renewable energy which is generated by a wind turbine. The optimal power output for each generator was derived by Mannion et al. [19] and the derived values are used for the both the fossil fuel generators and the renewable energy generator. In this example, Generator 3 is controlled by an agent, Generator 1 is a slack generator and Generator 4 is powered by a wind turbine.

In this setting we imagine a period of 24 hours and for each hour we receive a weather forecast for a city. For hours $1-15$, the weather is predictable and the optimal power values derived by Mannion et al. [19] can be used to generate power. From hours $16-24$, a storm is forecast for the city. During the storm, both high and low levels of wind are expected and the weather forecast impacts how much power the wind turbine can generate. At each hour during the storm, there is a 0.15 chance the wind turbine will produce 25% less power than optimal, a 0.7 chance the wind turbine will produce optimal power and a 0.15 chance the wind turbine will produce 25% more power than optimal. In the REDEED problem we aim to learn an optimal policy that can ensure the required power is met over the entire day while reducing both the cost and emissions created by all generators.

The goal is to maximise the following linear utility function under the ESR criterion,

$$R_{+} = -\sum_{o=1}^{O} w_o f_o, \tag{8}$$

where $w_o$ is the objective weight, and $f_o$ is the objective function. The objective weights used are $w_c = 0.225$, $w_e = 0.275$ and $w_p = 0.5$ [19].

The following equation calculates the local cost for each generator $n$, at each hour $m$:

$$f_c^L(n,m) = a_n + b_n P_{nm} + c_n (P_{nm})^2 + |d_n sin\{e_n(P_n^{min} - P_{nm})\}|. \tag{9}$$

Therefore the global cost for all generators can be defined as:

$$f_c^G(m) = \sum_{n=1}^{N} f_c^L(n,m). \tag{10}$$

The local emissions for each generator, $n$, at each hour, $m$, is calculated using the following equation :

$$f_e^L(n,m) = E(a_n + b_n P_{nm} + \gamma_n (P_{nm})^2 + \eta \exp \delta P_{nm}). \tag{11}$$

Therefore the global emissions for all generators can be defined as:

$$f_e^G(m) = \sum_{n=1}^{N} f_e^L(n,m). \tag{12}$$

It is important to note the emissions for the generator controlled by the wind turbine are set to 0.

If the agent exceeds the ramp and power limits a penalty is received. A global penalty function $f_p^G$ is defined to capture the violations of these constraints,

$$f_p^G(m) = \sum_{v=1}^{V} C(|h_v + 1|\delta_v). \tag{13}$$

Along with cost and emissions, the penalty function is an additional objective that will need to be optimised. All equations and parameters absent from this paper that are required to implement this problem domain can be found in the works of Basu [4] and Mannion et al. [19].

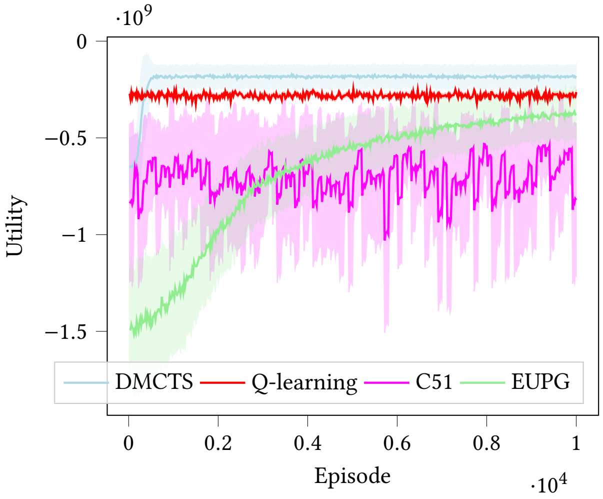

**Figure 3: Results from the REDEED environment DMCTS outperforms EUPG, C51 and Q-learning. DMCTS achieves a higher utility compared to other algorithms used throughout experimentation in the REDEED domain under ESR.**

To evaluate DMCTS in the REDEED domain, we compare DMCTS against EUPG, C51, and Q-learning [14, 19].

For C51 the learning parameters were set as follows: $V_{min} = -1.75e^9$, $V_{max} = 0$, $\epsilon = 0.1$, $\gamma = 1$ and $\alpha = 0.1$. For Q-learning, the learning parameters were set as follows: $\epsilon = 0.1$, $\gamma = 1$ and $\alpha = 0.1$. For the REDEED problem the agent learns for 10, 000 episodes and $n_{exec} = 10$ for each algorithm.

As seen in Figure 3, DMCTS outperforms EUPG, Q-learning and C51 in the REDEED domain. C51 struggles to learn a consistent policy and C51's utility fluctuates throughout experimentation. The hyper-parameters chosen for C51 provide the most optimal performance but are difficult to tune. Although the learning speed of EUPG is slow, EUPG achieves a higher utility than C51 but does not achieve a utility as high as Q-learning or DMCTS.

Although Q-learning outperforms EUPG and C51 in the REDEED environment, it does not achieve a higher utility when compared with DMCTS. C51 makes decisions based on a distribution over the expected returns and Q-learning makes decisions based on the expected future returns; due to this, both algorithms fail to learn good policies under ESR because they do not take both accrued and future returns into consideration.

The results presented in this paper evaluate C51 in an environment with a large state action space and complex returns. We hypothesise that a reason for poor performance is C51's inability to learn a distribution over the full returns and the level of discretisation of the distribution. The distribution for C51 uses 51 bins to discretise the algorithm's distribution. Bellemare et al. [5] claim this parameter for discretisation is optimal. However, the results presented in this paper show this parameter setting is not optimal in scenarios where the returns are not simple scalars over small ranges. How C51 would perform using different numbers of bins other than the 51 recommended by Bellemare et al. [5] is an open question, which we do not address here as it is outside the scope of this work. The results present in Figure 3 show that C51 struggles to scale to large problem domains with complex returns over large ranges. Instead, DMCTS is able to learn an approximate posterior distribution, i.e. a bootstrap distribution over the expected utility of the returns. DMCTS outperforms both C51 and Q-learning because a posterior over the expected utility of the returns is a sufficient statistic on which to base exploration. Moreover, learning a bootstrap distribution is an efficient yet compact approximation of a posterior distribution.

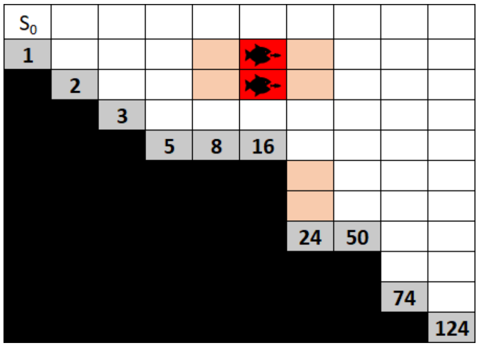

**Figure 4: In DDST, states marked in red are terminal as the submarine is destroyed by a shark. The state in light red are non-terminal states where the agent has a probability, $p_{shark}$, of been hit by a shark. A hit by a shark causes -10 damage and the submarine is destroyed.**

## 4.4 Dangerous Deep Sea Treasure

We now demonstrate that DMCTS can also learn successfully under the SER criterion as stated in Section 3, even in an environment with stochastic transitions and rewards. For this, we adapt Deep Sea Treasure (DST), which is a commonly used benchmark for MORL algorithms under the SER optimisation criterion [37]. We introduce the Dangerous Deep Sea Treasure (DDST) environment, which is a stochastic variant of the DST problem. In DDST a submarine controlled by an agent searches for treasure on the sea bed where there are three objectives: treasure, damage and time. At certain states in the environment, the submarine can be attacked by a shark resulting in a negative reward in a separate objective, with probability $p_{shark}$. If the submarine receives a hit from a shark, the submarine becomes damaged. The submarine is destroyed if it accumulates a total of −10 damage, which terminates the episode.

Equation 14 describes the non-linear utility function we use to determine if DMCTS can learn a target Pareto optimal policy. In Equation 14, $\mathbf{r}$ is a reward vector, $\mathbf{e} = \frac{r_\dagger}{|r_\dagger|}$ where $\mathbf{r}_\dagger$ is a specified target vector, and $c$ is a constant we aim to maximise. The utility of $\mathbf{r}$ is the maximum $c$ value where $\mathbf{r} - c\mathbf{e}$ is greater than 0 for all objectives. The target vector, $\mathbf{r}_\dagger$, is initialised to the desired vector we want to recover.

$$u(\mathbf{r}) = \arg\max c : \mathbf{r} - c\mathbf{e} > 0 \tag{14}$$

For this demonstration, we implemented scalarised Q-learning [38] as that it is one of the most widely used SER algorithms [37]. DMCTS uses an extra exploration strategy which was outlined by

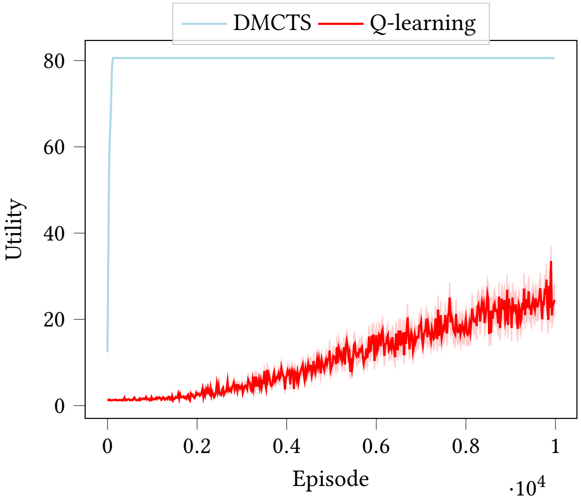

**Figure 5: Results from the Dangerous Deep Sea Treasure environment using a non-linear utility function. DMCTS learns optimal utility for the specified target vector, $\mathbf{r}_{\dagger}$.**

Osband et al. [25]. For DMCTS, artificially generated returns of the environment are randomly sampled during the learning phase to ensure sufficient exploration. Q-learning has the following learning parameters: $\alpha = 0.1$, $\gamma = 1$ and a decaying $\epsilon = 0.998^{e}$, where $e$ is equal to the episode number [20, 37]. In the utility function we set $\mathbf{r}_{\dagger}$ to $[54, 0, -14]$, where the objectives are ordered as follows: [treasure, danger, time]. We set $n_{exec} = 10$ and $p_{shark} = 0.5$. We run $10,000$ episodes per experiment.

DMCTS converges to the optimal utility after 100 episodes (Figure 5). This stands in contrast to scalarised Q-learning, which does not reach the optimal utility. Learning a bootstrap distribution over the expected returns under SER provides the agent with sufficient information to avoid states that represent negative utility (in this case, danger states), in order to maximise the known utility function.

## 5 RELATED WORK

Many risk-aware RL approaches seek to learn policies to maximise the expected return. Some research in this area focuses on learning policies which maximise the expected exponential utility [21]. Other approaches take the weighted sum of the return and risk into consideration when learning policies [12, 13]. Although most risk-aware RL approaches aim to maximise the expected utility, they often do not take into consideration the utility of the return of a full episode. It is also important to note that little research exists where decisions are made based on a learned distribution over the expected returns [22, 23] for risk-aware RL.

As previously highlighted, the majority of RL research focuses on the SER criterion. Multi-objective MCTS (MOMCTS) [40] was shown to be able to learn a coverage set under SER. However, MOMCTS can only learn a coverage set in deterministic environments. Convex Hull MCTS [27] is able to learn the convex hull of the Pareto front but focuses solely on linear utility functions. A number of other multi-objective MCTS methods exist [18, 28, 29], but no method has previously been shown to learn the Pareto front for both deterministic and stochastic environments for any unknown utility function. An interesting opportunity for future work is the possibility of building on the methods of Wang and Sebag [40] and Painter et al. [27] to extend DMCTS to learn the optimal coverage set under both SER and ESR for any unknown utility function.

A key argument in this paper is the expected utility of the future returns under ESR must be replaced with a posterior distribution over the expected utility of the returns. Bai et al. [3] extend MCTS to maintain a distribution at each node using Thompson Sampling as an exploitation strategy. However, the work presented in this paper is significantly different. In their work, Bai et al. [3] do not learn a posterior distribution over the expected utility of the return, apply their work to multi-objective settings, or incorporate the accrued returns as part of their algorithm. It is also important to note the C51 algorithm proposed by Bellemare et al. [5] achieves state-of-the-art performance in single-objective settings and learns a distribution over the future returns. Abdolmaleki et al. [1] learn a distribution over actions based on constraints set per objective. This approach ignores the utility-based approach [32] and uses constraints set by the user to learn a coverage set of policies where the value of constraints is dependent on the scale of the objectives. Abdolmaleki et al. claim setting the constraints for this algorithm is a more intuitive approach when compared to setting weights for a linear utility function. We theorise that if the user's utility function is non-linear, this approach would fail to learn a coverage set.

## 6 CONCLUSION & FUTURE WORK

In this paper we propose a novel Distributional Monte Carlo Tree Search algorithm. DMCTS is able to learn optimal policies in MORL settings under both ESR and SER for both linear and non-linear utility functions in problem domains with stochastic rewards. DMCTS replaces the expected utility of the future returns with a bootstrap distribution over the utility of the returns, and achieves state-of-the-art performance in MORL domains under ESR. We achieve this by using a bootstrap distribution as an approximate posterior over the expected utility of the returns of the episode. It is our hope that this paper will inspire further work on algorithms that replace the expected returns with a distribution over the expected utility of the returns for risk-aware and ESR settings.

In this paper, the utility function is known at the time of learning or planning. In different MORL scenarios, the utility function can be unknown at the time of learning or planning [30, 32]. In these scenarios, an algorithm must recover a coverage set of optimal policies. Multi-objective MCTS [40] can learn a coverage set for deterministic environments under SER. In future work, we aim to extend our DMCTS algorithm to be able to learn coverage sets for unknown utility functions under ESR and SER for stochastic environments. However, a coverage set of optimal policies under ESR has yet to be defined. We therefore hope to define the coverage set of optimal policies for ESR and extend DMCTS to learn it.

## ACKNOWLEDGEMENTS

Conor F. Hayes is funded by the National University of Ireland Galway Hardiman Scholarship.

# REFERENCES

[1] Abbas Abdolmaleki, Sandy H. Huang, Leonard Hasenclever, Michael Neunert, H. Song, Martina Zambelli, M. F. Martins, Nicolas Heess, Raia Hadsell, and Martin A. Riedmiller. 2020. A Distributional View on Multi-Objective Policy Optimization. *ArXiv* (2020).

[2] Bruce Abramson. 1987. *The expected-outcome model of two-player games.* Ph.D. Dissertation. Columbia University.

[3] Aijun Bai, Feng Wu, Zongzhang Zhang, and Xiaoping Chen. 2014. Thompson Sampling Based Monte-Carlo Planning in POMDPs. In *Proceedings of the Twenty-Fourth International Conference on International Conference on Automated Planning and Scheduling* (Portsmouth, New Hampshire, USA) *(ICAPS'14)*. AAAI Press, 29–37.

[4] Mousumi Basu. 2008. Dynamic economic emission dispatch using nondominated sorting genetic algorithm-II. *International Journal of Electrical Power and Energy Systems* 78 (02 2008), 140–149.

[5] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 449–458.

[6] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4:1 (03 2012), 1–43.

[7] Hyeong Soo Chang, Michael C. Fu, Jiaqiao Hu, and Steven I. Marcus. 2005. An Adaptive Sampling Algorithm for Solving Markov Decision Processes. *Oper. Res.* 53, 1 (Jan. 2005), 126–139. https://doi.org/10.1287/opre.1040.0145

[8] Rémi Coulom. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, Vol. 4630.

[9] Dean Eckles and Maurits Kaptein. 2014. Thompson sampling with the online bootstrap. *CoRR* abs/1410.4009 (2014). arXiv:1410.4009 http://arxiv.org/abs/1410.4009

[10] Dean Eckles and Maurits Kaptein. 2019. Bootstrap Thompson Sampling and Sequential Decision Problems in the Behavioral Sciences. *SAGE Open* 9, 2 (2019).

[11] Bradley Efron. 2012. Bayesian inference and the parametric bootstrap. *Ann. Appl. Stat.* 6, 4 (12 2012), 1971–1997. https://doi.org/10.1214/12-AOAS571

[12] Peter Geibel and Fritz Wysotzki. 2005. Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. *J. Artif. Int. Res.* 24, 1 (July 2005), 81–108.

[13] Abhijit Gosavi. 2009. Reinforcement Learning for Model Building and Variance-Penalized Control. In *Winter Simulation Conference* (Austin, Texas) *(WSC '09)*. Winter Simulation Conference, 373–379.

[14] Conor F. Hayes, Enda Howley, and Patrick Mannion. 2020. Dynamic Thresholded Lexicograpic Ordering. *Adaptive and Learning Agents Workshop (AAMAS 2020)*.

[15] Conor F Hayes, Mathieu Reymond, Diederik M Roijers, Enda Howley, and Patrick Mannion. 2021 In Press. Distributional Monte Carlo Tree Search for Risk-Aware and Multi-Objective Reinforcement Learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, Vol. 2021.

[16] Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. 2021. A Practical Guide to Multi-Objective Reinforcement Learning and Planning. arXiv:2103.09568 [cs.AI]

[17] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. *Machine Learning: ECML* 2006, 282–293.

[18] Jongmin Lee, Geon-hyeong Kim, Pascal Poupart, and Kee-Eung Kim. 2018. Monte-Carlo Tree Search for Constrained POMDPs. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 7923–7932.

[19] Patrick Mannion, Sam Devlin, Jim Duggan, and Enda Howley. 2018. Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning. *The Knowledge Engineering Review* 33 (2018), e23. https://doi.org/10.1017/S0269888918000292

[20] Patrick Mannion, Sam Devlin, Karl Mason, James Duggan, and Enda Howley. 2017. Policy invariance under reward transformations for multi-objective reinforcement learning. *Neurocomputing* 263 (2017), 60–73.

[21] Teodar.M Moldovan and Pieter Abbeel. 2012. Risk aversion in Markov decision processes via near-optimal Chernoff bounds. *Advances in Neural Information Processing Systems* 4 (01 2012), 3131–3139.

[22] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. 2010. Nonparametric Return Distribution Approximation for Reinforcement Learning. In *ICML*. 799–806.

[23] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. 2010. Parametric Return Density Estimation for Reinforcement Learning. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence* (Catalina Island, CA) *(UAI'10)*. AUAI Press, Arlington, Virginia, USA, 368–375.

[24] Michael Newton and Adrian Raftery. 1994. Approximate Bayesian Inference by the Weighted Likelihood Bootstrap. *Journal of the Royal Statistical Society Series B-Methodological* 56 (01 1994), 3 – 48.

[25] Ian Osband and B Van Roy. 2015. Bootstrapped Thompson Sampling and Deep Exploration. arXiv:1507.00300 [stat.ML]

[26] Nikunj C. Oza and Stuart Russell. 2005. Online bagging and boosting, Vol. 3. 2340–2345 Vol. 3.

[27] Michael Painter, Bruno Lacerda, and Nick Hawes. 2020. Convex Hull Monte-Carlo Tree-Search. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*. AAAI Press, 217–225.

[28] Diego Perez, Sanaz Mostaghim, Spyridon Samothrakis, and Simon Lucas. 2015. Multiobjective Monte Carlo Tree Search for Real-Time Games. *IEEE Transactions on Computational Intelligence and AI in Games* 7, 4 (2015), 347–360.

[29] Diego Perez, Spyridon Samothrakis, and Simon Lucas. 2013. Online and offline learning in multi-objective Monte Carlo Tree Search. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. 1–8.

[30] Roxana Rădulescu, Patrick Mannion, Diederik M. Roijers, and Ann Nowé. 2020. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems* 34, 10 (2020).

[31] Diederik M Roijers, Denis Steckelmacher, and Ann Nowé. 2018. Multi-objective Reinforcement Learning for the Expected Utility of the Return. In *Proceedings of the Adaptive and Learning Agents workshop at FAIM 2018*.

[32] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* 48 (2013), 67–113.

[33] Donald B. Rubin. 1981. The Bayesian Bootstrap. *The Annals of Statistics* 9, 1 (1981), 130–134.

[34] Yun Shen, Michael J. Tobia, Tobias Sommer, and Klaus Obermayer. 2014. Risk-Sensitive Reinforcement Learning. *Neural Computation* 26, 7 (2014), 1298–1328.

[35] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* (2016).

[36] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems* (Denver, CO) *(NIPS'99)*. MIT Press, Cambridge, MA, USA, 1057–1063.

[37] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. 2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning* (2011).

[38] K. Van Moffaert, M. M. Drugan, and A. Nowé. 2013. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 191–199.

[39] Joel Veness, Kee Siong Ng, Marcus Hutter, William Uther, and David Silver. 2011. A Monte-Carlo AIXI Approximation. *J. Artif. Int. Res.* 40, 1 (Jan. 2011), 95–142.

[40] Weijia Wang and Michèle Sebag. 2012. Multi-objective Monte-Carlo Tree Search *(Proceedings of Machine Learning Research, Vol. 25)*, Steven C. H. Hoi and Wray Buntine (Eds.). PMLR, Singapore Management University, Singapore, 507–522.

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