Multi-Objective Reinforcement Learning & Planning for the Expected Scalarised Returns

Conor F. Hayes

School of Computer Science
University of Galway

October 2022

A thesis submitted for the degree of
Doctor of Philosophy

Supervisors: *Dr. Patrick Mannion, Dr. Enda Howley & Dr. Diederik M. Roijers*

Abstract

Many problems in the real world have multiple, often conflicting, objectives. To solve such problems a multi-objective approach to decision making must be taken. In the multi-objective decision making (MODeM) literature, the utility-based approach is followed where a utility function is used to model the preferences over the objectives of a human decision maker (or user). If the utility function is known a priori a single optimal solution can be computed. However, if the utility function is unknown or uncertain, a set of optimal solutions must be computed.

When following the utility-based approach, multiple optimality criteria can arise. In scenarios where the utility function of a user is derived from multiple executions of a policy, the scalarised expected returns (SER) must be optimised. In scenarios where the utility of a user is derived from a single execution of a policy, the expected scalarised returns (ESR) criterion must be optimised. In the MODeM literature, the SER criterion has been studied extensively, while the ESR criterion has largely been ignored. In the real world, a user may only have a single opportunity to make a decision. For example, in a medical setting, a patient may only have one chance to select a treatment. Therefore, in order to effectively apply MODeM algorithms to a range of practical applications, the ESR criterion must be further investigated.

This thesis contains a number of important contributions. It is demonstrated by example that for ESR settings where the utility function is known and nonlinear, multi-objective methods that compute policies must be explicitly designed for the ESR criterion. For settings where the utility function of a user is unknown, it is shown that expected value vectors are not sufficient to determine optimality under the ESR criterion. Therefore, to determine a partial ordering over policies, new methods to compute sets of optimal policies are proposed. Finally, this thesis proposes a number of new multi-objective algorithms that can compute sets of optimal policies for the ESR criterion in various MODeM settings.

Contents

Acknowledgments

First, I would like to thank my supervisors Patrick, Enda, and Diederik. Thank you all for everything over the course of this PhD. I have thoroughly enjoyed every minute of the last few years, and I will miss our weekly meetings and chats. Thank you for your encouragement, guidance, patience, and support. But most importantly thank you for pushing me to do good work. While this is the end of my PhD, I look forward to continuing to work with you all.

I like to thank everyone in Room 307 whom I've been fortunate enough to cross paths with. I would also like to thank everyone who I have collaborated with over the last number of years, particularly Roxana Rădulescu, Mathieu Reymond, Pieter Libin, Willem Röpke, Timothy Verstraeten, and Peter Vamplew.

Thank you to Dr. Karl Mason and Prof. Maite Lopez-Sanchez for their helpful discussion, comments, and feedback on this thesis.

I would also like to thank my family. To my parents Edel and Frank, thank you for your continued unconditional support. I really appreciate all that you have done for me, especially over the last two years. Without you none of this would be possible. To my siblings Luke and Ross, thank you for always seeing the lighthearted aspect of everything!

To Theresa and Casey, thank you for your continued support.

Finally, I want to thank Megan. Megan, thank you for everything! You have been a constant throughout this PhD journey. Thank you!

Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree other than Doctor of Philosophy of the University of Galway. This thesis is the result of my own investigations, except where otherwise stated.

Mr. Mathieu Reymond (Vrije Universiteit Brussel) provided part of the source code that was used for the Renewable Energy Dynamic Economic Emissions Dispatch domain in Section 3.4.2.2.

Dr. Timothy Verstraeten (Vrije Universiteit Brussel) helped verify Theorem 3 presented in Section 4.2.

Some of the material contained in this thesis has appeared in the following published or awaiting publication papers:

1. Hayes, C. F., M. Reymond, D. M. Roijers, E. Howley, P. Mannion. Risk-Aware and Multi-Objective Decision Making with Distributional Monte Carlo Tree Search. In Proceedings of the Adaptive Learning Agents Workshop 2021 at AAMAS. 2021.

2. Hayes, C. F., M. Reymond, D. M. Roijers, E. Howley, P. Mannion. 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, pp. 1530-1532. 2021.

3. Hayes, C. F., M. Reymond, D. M. Roijers, E. Howley, P. Mannion. Monte Carlo Tree Search Algorithms for Nonlinear Utility Functions. Journal of Autonomous Agents and MultiAgent Systems.

4. Hayes, C. F., T. Verstraeten, D. M. Roijers, E. Howley, P. Mannion. Dominance Criteria and Solution Sets for the Expected Scalarised Returns. In Proceedings of the Adaptive Learning Agents Workshop 2021 at AAMAS. 2021.

5. Hayes, C. F., T. Verstraeten, D. M. Roijers, E. Howley, P. Mannion. Expected Scalarised Returns Dominance: A New Solution Concept for Multi-Objective Decision Making. Neural Computing & Applications. https://doi.org/10.1007/s00521-022-07334-x. 2022.

6. Hayes, C. F., D. M. Roijers, E. Howley, P. Mannion. Multi-Objective Distributional Value Iteration. In Proceedings of the Adaptive Learning Agents Workshop 2022 at AAMAS. 2022.

7. Hayes, C. F., D. M. Roijers, E. Howley, P. Mannion. Decision-Theoretic Planning for the Expected Scalarised Returns. In Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, pp. 1621-1623. 2022.

8. Hayes, C. F., T. Verstraeten, D. M. Roijers, E. Howley, P. Mannion. Multi-Objective Coordination Graphs for the Expected Scalarised Returns with Generative Flow Models. In Proceeding of the European Workshop on Reinforcement Learning. 2022.

9. Hayes, C. F., D. M. Roijers, E. Howley, P. Mannion. A Distributional Perspective on Multi-Objective Decision Making. Under Review: Journal of Artificial Intelligence Research.

1 | Introduction

Machine learning [Russell and Norvig, 2010] can be defined as a computer system that learns from experience to solve a predefined task. It is becoming increasingly common to use machine learning approaches to solve real-world decision making problems. For example, algorithms like AlphaFold [Jumper et al., 2021] and AlphaTensor [Fawzi et al., 2022] have shown that machine learning can be used to make major scientific advancements across multiple disciplines.

Reinforcement learning (RL) and planning are sub-fields of machine learning whereby an autonomous agent solves a task by interacting with its environment. An RL or planning agent perceives the state of the environment and the agent interacts with the environment by executing actions. For each action, the agent receives feedback and this known as a reward. Over time, the agent solves a predefined task by acting in a way that maximises its long term reward, RL and planning are popular methods used to solve real-world decision making problems. For example, RL and planning have been applied to autonomous driving [Kiran et al., 2021], healthcare [Yu et al., 2021], finance [Rao and Jelvis, 2022], and many other areas [Li, 2017; Mason and Grijalva, 2019].

However, many real-world problems have multiple, often conflicting, objectives [Roijers et al., 2013; Dulac-Arnold et al., 2021; White, 1982]. For example, consider a user planning their commute to work. The user aims to minimise the following objectives: time taken to commute to work and the cost of the commute. In this case, driving will get the user to work much faster than public transport. However, the cost of fuel will be significantly higher when compared to the cost of public transport [Hayes et al., 2022c]. When faced with a multi-objective problem, RL and planning practitioners generally either select a single objective to optimise or combine the objectives using a hand-tuned weighted sum. These approaches reduce a multi-objective problem to a single-objective problem, where a scalar reward can be optimised. However, both of the aforementioned approaches can produce sub-optimal results [Vamplew et al., 2008]. Furthermore, it can be argued that maximising for a scalar reward is not sufficient when optimising for multiple objectives [Vamplew et al., 2022]. Therefore, in order to solve multi-objective problems, all objectives must be taken into consideration and an optimal solution or set of optimal solutions can be computed. To do so, an explicitly multi-objective approach to RL and planning must be taken [Bryce et al., 2007].

In the multi-objective decision making (MODeM) literature, multi-objective RL and multi-objective planning are used to compute optimal policies for multi-objective problems. In MODeM, the agent has a state and can execute actions. In this case, the agent receives a reward vector, where a value in the vector exists per objective. In some settings, a utility function (also known as a scalarisation function) is used to reduce the reward vector to a scalar and a total ordering over policies can be determined by comparing the scalarised values [Roijers et al., 2013]. However, in many settings, applying a utility function is impossible, infeasible, or undesirable [Rădulescu et al., 2020]. Early work in the MODeM literature adopted an axiomatic based approach to determine a partial ordering over policies, where the Pareto front is always assumed to be the optimal set of solutions [Wang and Sebag, 2012]. By following the axiomatic based approach, it is difficult to encode domain knowledge into the decision making process. As such, time and resources can be wasted computing unnecessary solutions that may have low utility for the user [Hayes et al., 2022c].

A widely used approach to solving decision problems, is maximising user utility [Roijers, 2016]. MODeM explicitly models the human decision maker in the decision making process by adopting the utility-based approach [Roijers et al., 2013; Hayes et al., 2022c], where a utility function is used to represent a human decision maker's preferences over objectives. In some settings, a user may know their preferences over the objectives a priori. By using the user's preferences, a utility function can be modelled, and a single optimal policy can be computed. In many scenarios, explicitly modelling a user's utility function can be difficult. When a utility function is unavailable a set of optimal policies must be computed. A user can then simply select a policy from the computed set that best reflects their preferences. MODeM has been applied to many real-world problems, like water resource management [Castelletti et al., 2013], energy management [Mannion et al., 2016], and public health [Reymond et al., 2022b].

In contrast to single-objective RL and planning, different optimality criteria exist in MODeM. In scenarios where the utility of a user is derived from multiple executions of a policy, the scalarised expected returns (SER) criterion must be optimised. Consider an electrical generation facility powered by fossil fuels. Government regulations limit the amount of CO_2 that the facility can emit annually. In this example, a policy is executed every day. Subject to meeting the annual CO_2 emissions regulation, CO_2 emissions can be high for some days because days where CO_2 emissions are low compensate. As such, computing policies for this scenario under the SER criterion is optimal as average CO_2 emissions are important, rather than daily limits. However, many scenarios exist where the utility of a user is derived from the single execution of a policy. For example, strict government regulations might require the daily CO_2 emissions of an electrical generation facility to fall below a certain limit. In this example, optimising under the ESR criterion is optimal because every policy execution must ensure the CO_2 emissions fall below the regulated limit for a given day.

The SER criterion has been studied extensively in the existing MODeM literature. Many methods exist that can compute a single policy [Pan et al., 2020] or set of polices [Van Moffaert and Nowé, 2014a] under the SER criterion. For example, under the SER criterion, the Pareto front can be computed as a set of optimal polices [Wang and Sebag, 2012]. In contrast, the ESR criterion has largely been ignored by the MODeM community. Only a small number of single policy methods have been proposed that can compute policies for the ESR criterion [Roijers et al., 2018b; Reymond et al., 2021; Malerba and Mannion, 2021]. Furthermore, a method to determine a partial ordering over policies for the ESR criterion has yet to be defined. For many real-world problems, a user may only have one opportunity to make a decision. For example, in a medical setting a patient may only have one opportunity to select a treatment. Therefore, the ESR criterion requires further study in order to effectively utilise MODeM in the real world.

The work presented in this thesis explores the ESR criterion. First, I evaluate the ramifications of nonlinear utility functions in single-agent settings and their impact on the policies computed under the SER criterion and ESR criterion. Second, I investigate if the current state-of-the-art methods that are used to determine a partial ordering over policies for the SER criterion can be used under the ESR criterion. Finally, I present a number of novel algorithms that can compute a set of optimal policies for the ESR criterion. The aim of this thesis is to investigate the ESR criterion because I believe exploring the ESR criterion is essential for extending MODeM to a broad range of application domains.

1.1 Research Questions

This thesis aims to answer the following research questions:

1. Is it necessary to design algorithms specifically for the ESR criterion in single-agent settings where the utility function is known? (RQ1)
2. When the utility function is unknown, what methodologies can be used to derive a partial ordering over policies for the ESR criterion? (RQ2)
3. How can multi-policy algorithms be designed for the ESR criterion for different multi-objective settings (e.g., multi-armed bandits, Markov decision processes and coordination graphs)? (RQ3)

1.2 Hypotheses

From the research questions outlined above, in this thesis I expect to demonstrate that:

1. It can be shown that the policies computed under the SER criterion and the ESR criterion can be different for nonlinear utility functions in single-agent settings. As a result, dedicated methods that can optimise for the ESR criterion must be developed.
2. The state-of-the-art methods, like Pareto dominance, that use expected value vectors cannot be used to determine a partial ordering over policies under the ESR criterion. An alternative to the expected value approach is to, instead, maintain distributions over the range of possible rewards. Therefore, methods like stochastic dominance (SD) can be used to determine a partial ordering over policies.
3. By taking a distributional approach to MODeM, it is possible to define new multi-policy algorithms for many multi-objective settings that can compute a set of optimal policies for the ESR criterion.

1.3 Thesis Overview

This thesis is structured as follows:

- **Chapter 2** - introduces the concepts for reinforcement learning and planning, stochastic dominance, and multi-objective reinforcement learning and planning that are relevant to understanding the contributions of this thesis.

- **Chapter 3** - considers the implications of nonlinear utility functions on the policies computed under the SER criterion and the ESR criterion. This investigation shows that the policies can be different in single-agent settings. Therefore, methods that explicitly compute policies for the ESR criterion must be developed. This chapter also proposes two model-based multi-objective Monte Carlo tree search (MCTS) algorithms that can compute policies for nonlinear utility functions under the ESR criterion.

- **Chapter 4** - investigates whether methods that utilise expected value vectors to determine a partial order over policies under the SER criterion can be used under the ESR criterion. By example, it is demonstrated that using expected value vectors to determine optimality is fundamentally incompatible with the ESR criterion. As such, it is shown that a distributional approach to MODeM must be taken under the ESR criterion. Furthermore, several novel dominance relations and solution sets for the ESR criterion are defined.

- **Chapter 5** - defines three novel distributional multi-objective algorithms that can compute sets of optimal policies for the ESR criterion. These algorithms show that optimal policies for the ESR criterion can be computed for multi-objective multi-armed bandit settings, multi-objective Markov decision processes, and multi-objective coordination graphs.

- **Chapter 6** - concludes with a summary of the main contributions of this thesis, a discussion of the limitations of this work, and an outline of some promising directions for future research.

2 | Background

This chapter presents the relevant background material required to understand the contributions presented in this thesis. The topics covered address reinforcement learning and planning, deep reinforcement learning, distributional reinforcement learning, stochastic dominance, multi-objective reinforcement learning and planning, multi-objective optimality criteria, and the expected scalarised returns optimality criterion.

2.1 Reinforcement Learning & Planning

Reinforcement learning (RL) and planning [Sutton and Barto, 2018] are sub-fields of machine learning where an autonomous agent solves a predefined task through interactions with an environment. Figure 2.1 outlines how an RL and planning agent interacts with an environment. At each timestep, t, the agent perceives the current state, s_t, of the environment. The agent receives feedback from the environment in the form of a *scalar reward*, r_{t+1}, by performing an action, a_t, and transitioning to a new state, s_{t+1}.

The reward the agent receives can be positive or negative, and the agent aims to maximise the reward. Through multiple interactions with the environment, the agent computes a solution that maximises its reward.

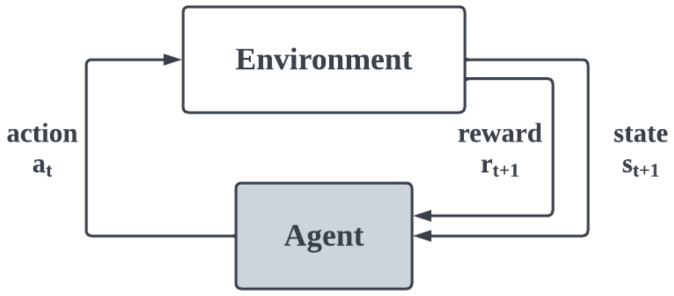


Figure 2.1: Interaction between an agent and its environment.

2.1.1 Markov Decision Processes

Generally, in RL and planning, a Markov decision process (MDP) is used to model single-agent sequential decision-making problems. A MDP can be defined as follows:

> **Definition 1**
>
> A Markov decision process (MDP) [Puterman, 1990] is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mu, \mathcal{R} \rangle$, where:
>
> - $\mathcal{S}$ is the state space
> - $\mathcal{A}$ is the action space
> - $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a probabilistic transition function
> - γ is a discount factor
> - $\mu : S \leftarrow [0, 1]$ is a probability distribution over states
> - $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the immediate reward function

MDPs are commonly used in RL and planning to model the environment. A MDP is a tuple consisting of a state space, $\mathcal{S}$, an action space, $\mathcal{A}$, a probabilistic transition function, $\mathcal{T}$, a discount factor, γ, and a reward function, $\mathcal{R}$. The state space $\mathcal{S}$ is a set of all possible states of the environment where the current state is denoted s. The action space $\mathcal{A}$ is the set of all possible actions an agent can take in an environment. The transition function $\mathcal{T}$ encodes the dynamics of the environment and determines which next state, s', an agent will transition to, having executed action a in state s. The discount factor γ can be used to determine the relative importance of future rewards. Finally, the reward function $\mathcal{R}$ determines the *scalar reward* the agent receives for taking action a in state s and transitioning to state s'.

Typically, in a MDP, a horizon $\mathcal{H}$ is defined to determine the number of timesteps in a RL or planning problem. Some problems have an infinite horizon, $\mathcal{H} = \infty$. In contrast, finite horizon problems have a finite number of timesteps. Finite horizon problems are episodic and terminate after a fixed number of timesteps.

While both RL and planning use MDPs to model sequential decision making problems, the dynamics of the environment are not known to the agent in RL. However, in planning, the dynamics of the environment (also known as a model) are known to the agent.

The agent's goal is to compute a policy, π, that maximises the expected discounted sum of rewards. A policy π can be defined as a mapping from states to probabilities of selecting each possible action, $\pi(a|s)$ [Sutton and Barto, 2018]. The expected discounted sum of rewards can be defined as follows:

$$V^{\pi} = \mathbb{E}\Bigg[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi,\ \mu_0\Bigg], \tag{2.1}$$

where r_t is the reward $\mathcal{R}(s_t, a_t, s_{t+1})$ for taking action a_t in state s_t and transitioning to state s_{t+1} at timestep t, and μ_0 is a probability distribution over initial states.

To compute optimal policies, the agent needs to reason about how good being in a given state is and how good selecting a certain action in a given state is [Sutton and Barto, 2018]. To evaluate a state s, the state-value function can be defined. The state-value function of a state s under a policy π can be used to determine the expected future reward the agent can expect when in state s and following policy π thereafter:

$$V^{\pi}(s) = \mathbb{E}\Bigg[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid \pi,\ s_t = s\Bigg], \tag{2.2}$$

where r_{t+k} is the reward $\mathcal{R}(s_{t+k}, a_{t+k}, s_{t+k+1})$ for taking action a_t in state s_t and transitioning to state s_{t+1} at timestep t. To evaluate the possible expected future reward for selecting a given action a in a given state s, the action-value function can be defined as follows:

$$Q^{\pi}(s,a) = \mathbb{E}\Bigg[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid \pi,\ s_t = s,\ a_t = a\Bigg]. \tag{2.3}$$

The action-value function defines the value of taking action a in state s under a policy π, as the expected return, starting from state s, selecting action a and following policy π thereafter.

A fundamental property of value functions used in MDPs and dynamic programming is that value functions satisfy recursive relationships. For example, the Bellman equation [Bellman, 1957a] expresses a relationship between the value of a state and the values of its successor states. The Bellman equation can be defined as follows:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{T}(s,a,s')(\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')) \tag{2.4}$$

The Bellman equation states that the value of the start state must equal the discounted value of the expected next state and the expected future reward. The Bellman equation averages over all possibilities, weighting each by its probability of occurring. The Bellman equation is the foundation of numerous MDP algorithms, such as Q-learning [Watkins and Dayan, 1992] and value iteration [Bellman, 1957b]. Q-learning has been used extensively in the RL literature [Dearden et al., 1998; Greenwald et al., 2003], whereas value iteration algorithms have been used extensively in the planning literature [Tamar et al., 2016; Pineau et al., 2003].

The agent aims to compute an optimal policy, π^*, that maximises the expected sum of future reward. There may be multiple optimal policies, however, all optimal policies share the same state-value function called the optimal state-value function, V^{π^*}, where $V^{\pi^*} = \max_\pi V^\pi(s)$ for all $s \in \mathcal{S}$. Each optimal policy also shares the same optimal action-value function $Q^{\pi^*}(s,a)$, where $Q^{\pi^*}(s,a) = \max_\pi Q^\pi(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

2.1.1.1 Q-Learning

One of the earliest MDP algorithms is Q-learning [Watkins and Dayan, 1992], which is a temporal-difference control algorithm. For Q-learning, the learned action-value function directly approximates the optimal action-value function independently of the policy being followed. Over multiple iterations, the Q-learning algorithm updates the action-value function by bootstrapping and using the estimated action-value of the next state. Q-learning updates the action-value function for state s and the selected action a by executing the following equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \Big], \tag{2.5}$$

where α is the learning rate and γ is the discount factor. Q-learning is a tabular method and stores the action-values for each state in memory using a Q-table. The use of tabular Q-learning is limited to settings with discrete state-action spaces, given a value for all states and actions must be stored in memory.

2.1.1.2 Policy Gradient

Policy gradient methods are commonly used in RL to solve MDPs. Policy gradient methods learn a policy without learning a value function. To learn a policy, π, the policy π is paramaterised using $\boldsymbol{\theta}$. Using this approach, it is possible to follow the policy gradient, $\nabla_\theta J(\pi_\theta)$, to find the $\boldsymbol{\theta}$ that maximises the expected future returns. To do so, policy gradient methods calculate the gradients of the objective, $J(\boldsymbol{\theta})$, with respect to $\boldsymbol{\theta}$, using the agent's interactions with the environment. The parameters of $\boldsymbol{\theta}$ can then be updated by taking a step in the direction of the gradient,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta}_t). \tag{2.6}$$

The policy gradient theorem [Sutton and Barto, 2018] shows the policy gradient can be computed as follows:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) Q^\pi(s,a), \tag{2.7}$$

where μ is the on-policy distribution under π [Sutton and Barto, 2018]. Algorithms like REINFORCE [Williams, 1992] estimate the objective by using a single Monte Carlo rollout of a policy. Such methods suffer from slow learning rates and high variance. Various other policy gradient methods, like actor-critic methods [Haarnoja et al., 2018; Konda and Tsitsiklis, 1999; Mnih et al., 2016], also exist.

2.1.1.3 Monte Carlo Tree Search

Another common way of solving MDPs is to use tree search [Bonet and Geffner, 2006, 2012; Bai et al., 2016]. Perhaps the most popular of such methods is Monte Carlo tree search (MCTS) [Coulom, 2006], which employs heuristic exploration to construct its search tree through planning. MCTS is a model-based algorithm and, therefore, the reward function and transition function are known to the agent a priori [Moerland et al., 2020]. 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 planning phase and the execution phase.

In the planning phase, the agent implements the following four steps [Browne et al., 2012]:

- **Selection:** the agent traverses the search tree until it reaches a node for which not all of its possible child nodes have been explored.
- **Expansion:** at a node whose children have not all been expanded, 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 computes the returns.
- **Backpropagation:** the agent must backpropagate the returns received at a terminal state to each node visited during the selection phase, where a predefined algorithm statistic, e.g., upper confidence bound (UCB) [Coulom, 2006; Kocsis and Szepesvári, 2006], is updated.

Each step is repeated a specified number of times, which incrementally builds the search tree.

During the execution phase, the agent must select a child node corresponding to an action and associated state transition to which it will traverse. The agent evaluates the statistic at each node that is reachable from the root node and moves to the node which returns the maximum value. Once the execution phase has completed, the agent repeats the planning phase.

As already highlighted, MCTS makes decisions and explores based on a predefined algorithm statistic. One such version of MCTS is upper confidence trees (UCT) [Kocsis and Szepesvári, 2006], which uses the following formula to derive the optimal action at decision time while also incorporating exploration during learning:

$$v_i + C \times \sqrt{\frac{ln(N)}{n_i}}, \tag{2.8}$$

where v_i is the approximated value of the node i, n_i is the number of the times the node i has been visited, and N is the total number of times that the parent of node i has been visited. C is a hyperparameter that can be tuned for exploration, however C is often set to $\sqrt{2}$.

2.1.1.4 Exploration-Exploitation Dilemma

In RL, an agent must trade off between exploring new areas of the environment and exploiting the knowledge already accumulated. This is known as the exploration-exploitation dilemma. By exploring, the agent can potentially find new states that may lead to a higher reward. However, if the agent focuses too much on exploration, actions known to return a high reward may not be effectively exploited. A common approach to balancing exploration and exploitation is to follow an ϵ-greedy strategy. In this scenario, the agent executes a random action (explores) with probability ϵ and executes the known best action (exploitation) with probability $1 - \epsilon$. It is important to note that other methods, like Thompson sampling (TS) [Russo et al., 2018], can also be used to address the exploration-exploitation dilemma.

2.1.2 Multi-Armed Bandits

Beyond MDPs, RL and planning have been applied to many other problem settings. Multi-armed bandit (MAB) settings have been studied extensively using RL. The MAB setting is stateless and presents the agent with a choice among $\mathcal{A}$ different options, or arms [Slivkins et al., 2019]. The agent selects a given arm, a, and receives a scalar reward, $\mathcal{R}(a)$, where the rewards are generated from a stationary probability distribution. The agent aims to maximise its reward by selecting the arm with the highest expected reward. The rewards for each arm are not known a priori and are stochastic [Auer et al., 2002]. Therefore, MABs can be defined as follows:

> **Definition 2**
>
> The multi-armed bandit (MAB) has a finite set of arms $\mathcal{A}$, where each arm $a \in \mathcal{A}$ returns some reward $\mathcal{R}(a)$ when pulled. Each arm a has an associated stationary reward distribution that is unknown to the agent, where $\mu(a) = \mathbb{E}[\mathcal{R}(a)]$

Generally, in MAB settings, the objective is to minimise the expected cumulative regret, which can be defined as follows:

$$\nabla(a_t) = \mu(a_*) - \mu(a_t), \tag{2.9}$$

where a_* is an optimal value, which is known a priori. To minimise the expected cumulative regret, the agent learns a policy, π, that balances both exploration and exploitation. MABs have been used to model real-world decision making problems, like online advertising [Chapelle and Li, 2011] and wind-farm control [Bargiacchi et al., 2018]

2.1.2.1 Upper Confidence Bounds

One of the most commonly used methods to learn optimal policies in MAB settings is upper confidence bound (UCB) [Auer, 2002]. UCB is a frequentist approach that constructs confidence bounds around the mean of each arm. UCB optimistically selects arms with a high confidence bound. UCB is defined as follows:

$$arg\, max_a \; \mu_{t-1}(a) + C\sqrt{\frac{ln(t)}{n_{t-1}(a)}}, \tag{2.10}$$

where C is a constant that can be tuned for exploration, $\mu_{t-1}(a)$ is the expected returns for arm a at timestep $t-1$ and $n_{t-1}(a)$ is the total number of times arm a has been pulled at timestep $t-1$.

2.1.2.2 Thompson Sampling

Another approach commonly used to solve MABs is Thompson sampling (TS) [Thompson, 1933]. TS, also known as *probability matching* or *posterior sampling*, takes a Bayesian approach to decision making and selects arms based on the probability of that arm being optimal. At each timestep, t, the agent draws a sample, $\mu_{t-1}(a)$, from the posterior distribution of a given arm, a. The posterior distribution is the prior distribution conditioned on the history, $\mathcal{H}_{t-1}$ [Chapelle and Li, 2011]. The history at timestep t, $\mathcal{H}_{t-1}$, consists of the previously pulled arms and the observed rewards. Once an arm a_t is pulled, a reward $r(a_t)$ is received and the history is updated [Russo et al., 2018]. The arm, a, that returns the maximum sample is selected according to the following:

$$arg\,max_a\ \mu_t(a) \tag{2.11}$$

where $\mu_t(a)$ is the sample mean at timestep t for arm a. TS has been used in real-world decision making settings like wind-farm control [Verstraeten et al., 2020], online advertising [Chapelle and Li, 2011], and news article recommendation [Li et al., 2010].

2.1.2.3 Bootstrap Thompson Sampling

When using TS, 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 [Efron, 2012; Newton and Raftery, 1994]. Eckles and Kaptein [2014, 2019] use a bootstrap distribution to replace the posterior distribution used in TS. This method is known as bootstrap Thompson sampling (BTS) [Eckles and Kaptein, 2014] and was proposed in the MAB setting. The bootstrap distribution contains a number of bootstrap replicates, $j \in \{1, ..., J\}$, where J is a hyperparameter 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 [Eckles and Kaptein, 2014].

Each bootstrap replicate, j, in the bootstrap distribution has two parameters, α_j and β_j, where $\frac{\alpha_j}{\beta_j}$ is an estimate of replicate j's expected utility. 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 expected utility is pulled [Eckles and Kaptein, 2014].

The distribution that 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 (see Algorithm 1). If

Algorithm 1: Bootstrap Thompson Sampling Update

```
1 for $j \in J$ do
2     sample $d_j$ from Bernoulli(1/2)
3     if $d_j = 1$ then
4         $\alpha_j = \alpha_j + u(\mathbf{r})$
5         $\beta_j = \beta_j + 1$
```

the coin flip is heads, the α and β parameters for j are re-weighted[1]. To do so, the return is added to the α_j value and 1 is added to β_j [Eckles and Kaptein, 2014].

Bootstrap methods with random re-weighting [Rubin, 1981] are more computationally appealing as they can be conducted online rather than having to re-sample data [Oza and Russell, 2001]. BTS addresses problems of scalability and robustness when compared to TS [Eckles and Kaptein, 2014]. Furthermore, bootstrap distributions can approximate posteriors that are difficult to represent exactly.

2.1.3 Coordination Graphs

Another common method to model stateless decision problems is using coordination graphs (CoGs) [Guestrin et al., 2001; Kok and Vlassis, 2004]. CoGs consider multiple agents where some dependency structure exists between the agents that can be exploited. The goal of a CoG is to learn a single joint action, across all agents, that is optimal. A CoG can be defined as follows [Roijers et al., 2015; Verstraeten et al., 2020]:

[1]Updating the distribution in this way is known as "double-or-nothing" or online half sampling [Eckles and Kaptein, 2014]. It is important to note that the absolute scale of the weights does not matter for most estimators [Eckles and Kaptein, 2014]. In the literature various other weight distributions have been used. For example, Rubin [1981] uses a Bayesian bootstrap which uses exponential weights. While this overcomes some numerical problems, it requires updating all replicates and therefore can be more computationally expensive. For an extensive study on weight distributions for bootstrapping the sample mean see Owen and Eckles [2012].

Definition 3

A coordination graph (CoG) [Roijers, 2016] is a tuple $\langle \mathcal{D}, \mathcal{A}, \mathcal{P} \rangle$ where:

- $\mathcal{D} = \{1, ..., n\}$ is the set of n agents. $\mathcal{D}$ is factorised into p, possibly overlapping, groups of agents $\mathcal{D}^e$.
- $\mathcal{A} = \mathcal{A}_i, ..., \mathcal{A}_n$ is the set of joint actions, which is the Cartesian product of the finite action spaces of all agents. A joint action is a tuple containing an action for each agent $\mathbf{a} = \langle a_1, ..., a_n \rangle$. $\mathcal{A}^e$ denotes the set of local joint actions for the group $\mathcal{D}^e$.
- $\mathcal{P} = p^1, ..., p^l$ is a set of l, scalar local payoff functions. The joint payoff for all agents is the sum of local payoff functions: $p(\mathbf{a}) = \sum_{e=1}^{l} p^e(\mathbf{a}^e)$.

The dependencies between the local reward functions and agents can be represented by a bipartite graph with a set of nodes, $\mathcal{D}$, and a set of edges, $\mathcal{E}$. In this setting the nodes, $\mathcal{D}$, are agents and components of a factored payoff function, and an edge $(i, p^e) \in \mathcal{E}$ exists if and only if agent i influences component p^e. The set of all possible joint action value vectors is denoted by the set $\mathcal{V}$ [Verstraeten et al., 2020].

2.1.3.1 Variable Elimination

Variable elimination is a method that can be used to solve CoGs [Zhang and Poole, 1996]. Variable elimination can exploit the loose couplings between the local payoff functions to compute the optimal joint global action. Variable elimination utilises two passes: the forward pass and the backward pass. First, variable elimination executes a forward pass, where variable elimination eliminates each agent in some predefined or random order by calculating the value of that agent's best response to every joint action of its neighbors [Roijers et al., 2015]. A new local payoff function is then created conditioned on the best response. The agent and the payoff function previously used to determine optimality are then removed. Once all agents have been eliminated, a backward pass computes the optimal joint action. Typically, both passes are utilised. However, in multi-objective settings, only a forward pass is used and the backward pass is replaced with a tagging scheme [Roijers et al., 2015]. Algorithm 2, presented by Roijers et al. [2015], determines the elimination procedure for variable eliminationusing a tagging scheme. Variable elimination has been used in real-world settings, like wind-farm control [Verstraeten et al., 2020, 2021], to determine optimality.

Algorithm 2: eliminateVE($\mathcal{P}$, i)

```
1  **Input**: A CoG $\mathcal{P}$, and an agent $i$
2  $\mathcal{P}_i \leftarrow$ a set of local payoff functions involving $i$
3  $n_i \leftarrow$ a set of neighboring agents of $i$
4  $u^{new} \leftarrow$ a new factor taking joint actions of $n_i, \mathbf{a}^{n_i}$ ,as input
5  for $\boldsymbol{a}^{n_i} \in \mathcal{A}^{n_i}$ do
6      $\mathcal{S} \leftarrow \emptyset$
7      for $\boldsymbol{a}^{n_i} \in \mathcal{A}^{n_i}$ do
8          $v \leftarrow \sum_{p_j \in \mathcal{P}_i} p_j(a_{n_i}, a_i)$
9          tag $v$ with $a_i$
10         $\mathcal{S} \leftarrow \mathcal{S} \cup \{v\}$
11     end
12     $p^{new}(a_{n_i}) \leftarrow \max(\mathcal{S})$
13 end
14 Return $(\mathcal{P} \setminus \mathcal{P}_i) \cup \{p^{new}\}$
```

2.1.4 Other Settings

While this thesis extensively covers MDPs, MABs and CoGs, other RL and planning settings exist which have not been mentioned. For example, a partially observable Markov decision process (POMDP) is another common way of formulating sequential decision making in RL and planning. In MDPs, the state of the agent is fully observable, whereas, in POMDPs, the agent can not directly observe the state of the environment. Therefore, the agent must make decisions based on imperfect information. POMDPs have been extensively covered in the existing literature [Monahan, 1982; Spaan, 2012] and algorithms like MCTS [Silver and Veness, 2010], point-based algorithms [Pineau et al., 2003], and other methods [Sunberg and Kochenderfer, 2018; Ross et al., 2008; Somani et al., 2013] can be used to solve POMDPs.

2.1.5 Deep Reinforcement Learning

Tabular methods, like Q-learning, use a table to store the action-value functions for each state-action pair in memory. While tabular methods work well for settings with a small discrete state-action space, they cannot scale to settings with continuous states or actions. With the advent of deep learning [LeCun et al., 2015; Goodfellow et al., 2016], a large number of deep RL methods have been presented in recent years. Deep RL methods can scale to settings with continuous state and action spaces through approximation and have achieved astounding results.

Deep Q-networks (DQN) [Mnih et al., 2013] extend Q-learning to settings with high dimensional state spaces. DQN approximates a state-value function used in the Q-learning framework using deep neural networks. DQN takes a state as input and outputs state-values per action. Generally, DQN maintains two networks: the Q-network and the target network. The Q-network, parameterised by $\boldsymbol{\theta}$, is trained over a number of iterations, i, for sequences of loss functions, $\mathcal{L}_i\boldsymbol{\theta}_i$, as follows:

$$\mathcal{L}_i\boldsymbol{\theta}_i = \mathbb{E}\Bigg[(y_i - Q(s,a;\boldsymbol{\theta}_i))^2\Bigg], \tag{2.12}$$

where $y_i = \mathbb{E}[r+\gamma \max_{a'} Q(s',a';\boldsymbol{\theta_{i-1}})]$, which is known as the target. The target is generated by the target network. The target network is the same as the Q-network except that its parameters are copied every τ steps from the Q-network. DQN learns by storing experiences ((s,a,s',r) transitions) in a replay buffer [Lin, 1992]. DQN then samples experiences from the replay buffer in batches when learning. In the Atari gaming platform [Bellemare et al., 2013], DQN achieves superhuman levels of play [Mnih et al., 2013]. Multiple variants of DQN also exist, some of which are listed here: [Osband et al., 2016; Van Hasselt et al., 2016; Wang et al., 2016].

Deep RL has enabled significant advances in artificial intelligence. Some examples include AlphaGo, which learned to play the game of Go through a combination of tree search, supervised learning, and RL [Silver et al., 2016]. Alpha Zero mastered the game of Go, chess, and Shogi [Silver et al., 2017]. Alpha Zero learns without any human input by repeatedly playing itself in matches of the aforementioned games, using a process called "self-play" [Silver et al., 2017]. Deep RL has also been applied to competitive computer games [Wurman et al., 2022], silicon chip design [Mirhoseini et al., 2020], and many other applications [Fawzi et al., 2022; Bellemare et al., 2020; Kiran et al., 2021].

2.1.6 Distributional Reinforcement Learning

Recently, distributional RL [Bellemare et al., 2023] has become an active area of research. An example of a distributional RL method is categorical deep Q-networks (C51) [Bellemare et al., 2017]. To make distributional updates, Bellemare et al. [2017] define the distributional Bellman operator as follows:

$$\mathcal{T}_D^{\pi}\ z(s,a) \overset{D}{=} r_{s,a} + \gamma\ z(s',a'), \tag{2.13}$$

where z is the distribution over the returns. The distributional Bellman operator is used to replace the Bellman operator[2]. To represent a return distribution,

[2]The Bellman operator can be defined as follows: $\mathcal{T}^{\pi}Q(s,a) = r_{s,a} + \gamma\mathbb{E}\ Q(s',a')$.

Bellemare et al. [2017] utilise a paramaterised categorical distribution. The distribution is parameterised by a number of categories $n \in \mathcal{N}$ and is bounded by the minimum returns, R_{min}, and maximum returns, R_{max}, whose support is the set of categories: $\{z_i = R_{min} + i \bigtriangleup_z \colon 0 \leq i < n\}$. The category probabilities are given by a parametric model, θ.

Bellemare et al. [2017] project the Bellman update of $\mathcal{T} z_\theta$ onto the support of z_θ, which reduces the Bellman update to multi-class classification. Therefore, for a sampled transition from the replay buffer the Bellman update $\mathcal{T} z_j := r + \gamma z_j$ for each category, z_j, is performed. The probability $p_j(s', \pi(s'))$ is distributed to the immediate neighbours of $\mathcal{T} z_j$. As such, the i^{th} component of the projected update $\phi \mathcal{T} z_\theta(s, a)$ is:

$$(\phi \mathcal{T} z_\theta(s,a))_i = \sum_{j=0}^{n-1} \left[1 - \frac{|[\mathcal{T} z_j] - z_i|}{\bigtriangleup_z}\right]_0^1 p_j(s', \pi(s')). \tag{2.14}$$

Bellemare et al. [2017] use the cross-entropy term of the KL divergence [Thomas and Joy, 2006] as the loss function $\mathcal{L}(\theta)$,

$$\mathcal{L}(\theta) = D_{KL}(\phi \mathcal{T} z_{\theta'}(s,a) || z_\theta(s,a)), \tag{2.15}$$

which can be minimised by gradient descent. Bellemare et al. [2017] use the same architecture as DQN, except C51 outputs category probabilities instead of action-values. C51 is evaluated using the Atari learning environment and achieves state-of-the-art performance.

Many other distributional RL algorithms exist in the literature. For example, Dabney et al. [2018b] define a distributional RL algorithm using quantile regression. Martin et al. [2020] define a distributional RL algorithm and utilise stochastic dominance (SD) for action selection to compute a policy that is optimal for all risk preferences. Some other example of distributional RL algorithms are: [Petersen et al., 2020; Dabney et al., 2018a; Eriksson et al., 2022; Mavrin et al., 2019; Lyle et al., 2019].

2.2 Stochastic Dominance

When taking a distributional approach to decision making, stochastic dominance (SD) [Hadar and Russell, 1969; Bawa, 1975] can be used to give a partial ordering over distributions (see Figure 2.2). In sequential decision making settings, SD is particularly useful when a decision maker needs to take the full distribution over the returns into consideration, and not just the expected returns. When making

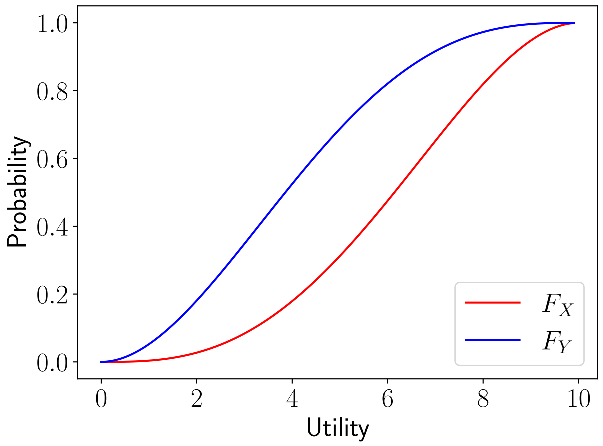


Figure 2.2: For random variables X and Y, $X \succeq_{FSD} Y$, where F_X and F_Y are the CDFs of X and Y respectively. In this case, X is preferable to Y because higher utilities occur with greater frequency in F_X.

decisions under uncertainty, SD can be used to determine the most risk averse decision.

In the literature, varying orders of SD exist. To illustrate the orders of SD, two random variables X and Y are considered. First-order stochastic dominance (FSD) can be defined, in terms of X and Y, as follows:

Definition 4

For random variables X and Y, X first-order stochastically dominates ($\succeq_{FSD}$) Y if the following is true:

$$X \succeq_{FSD} Y \Rightarrow P(X > z) \geq P(Y > z), \forall\, z$$

It is also possible to define FSD in terms of the cumulative distribution function (CDF) of a random variable. Consider the CDF of X, denoted by F_X, and the CDF of Y, denoted by F_Y, therefore X $\succeq_{FSD}$ Y if:

$$F_X(z) \leq F_Y(z), \forall\, z.$$

Definition 4 presents the necessary conditions for FSD, and Theorem 1 proves that, if a random variable is FSD dominant, it has at least as high of an expected value as another random variable [Wolfstetter, 1999]. It is important to note that the work of Wolfstetter [1999] is used to prove Theorem 1.

Theorem 1

If X $\succeq_{FSD}$ Y, then X has a greater than or equal expected value as Y.

$$X \succeq_{FSD} Y \implies \mathbb{E}(X) \geq \mathbb{E}(Y).$$

Proof. By a known property of expected values the following is true for any random variable:

$$\mathbb{E}(X) = \int_0^{+\infty} (1 - F_X(x))\, dx$$
$$\mathbb{E}(Y) = \int_0^{+\infty} (1 - F_Y(x))\, dx$$

Therefore, if X $\succeq_{FSD}$ Y then:

$$\int_0^{+\infty} (1 - F_X(x))\, dx \geq \int_0^{+\infty} (1 - F_Y(x))\, dx$$

Which gives,

$$\mathbb{E}(X) \geq \mathbb{E}(Y).$$

□

Second-order stochastic dominance (SSD) can be defined as follows:

Definition 5

A random variable X second-order stochastically dominates ($\succeq_{SSD}$) a random variable if the following is true:

$$X \succeq_{SSD} Y \Rightarrow \int_n^k F_X dx \leq \int_n^k F_Y dy \;\forall\; k \tag{2.16}$$

Furthermore, if $X \succeq_{SSD} Y$, then X has a greater than or equal expected utility as Y.

Proposition 1

If X $\succeq_{SSD}$ Y, then the expected utility of X is greater than or equal to the expected utility of Y .

$$X \succeq_{SSD} Y \implies \mathbb{E}(X) \geq \mathbb{E}(Y).$$

Moreover, if $X \succeq_{FSD} Y$, it can be assumed $X \succeq_{SSD} Y$. However, if $X \succeq_{SSD} Y$, it cannot be assumed that $X \succeq_{FSD} Y$ [Wolfstetter, 1999]. SD has been shown to be transitive [Hadar and Russell, 1969]. The transitive properties of SD are preserved when the original random variables are multiplied by a constant or when another random variable is added [Wolfstetter, 1999], see Proposition 2.

Proposition 2

Consider three independent non-negative random variables, X, Y, and W, and the linear combination $aX + bW$ and $aY + bW$ with $a > 0$, $b > 0$. Then:

1. $X \succeq_{FSD} Y \Rightarrow (aX + bW) \succeq_{FSD} (aY + bW)$
2. $X \succeq_{SSD} Y \Rightarrow (aX + bW) \succeq_{SSD} (aY + bW)$

SD has been used extensively in economics [Choi and Johnson, 1988], finance [Ali, 1975; Bawa, 1978], game theory [Fishburn, 1978], and various other real-world scenarios [Bawa, 1982; Cook and Jarrett, 2018]. For example, Levy and Robinson [2006] use SD in financial settings to build sets of optimal portfolios, where each portfolio in the set is optimal for different preferences of risk.

2.3 A Note on the Limitations of Reinforcement Learning & Planning

Many problems in the real world have multiple, often conflicting, objectives [Vamplew et al., 2022]. Generally, AI engineers deploy RL and planning algorithms to compute policies in real-world settings with multiple objectives by: (a) selecting a single objective to compute a policy for and ignoring the other objectives (b) combining the objectives using a weighted sum (linear scalarisation) [Hayes et al., 2022c]. Utilising a weighted sum to combine the objectives is the most common method used to tackle multi-objective problems with standard RL and planning methods [Kompella et al., 2020; Wurman et al., 2022]. Both of the outlined approaches reduce a multi-objective problem to a single-objective problem, where traditional RL and planning methods can be used to optimise for a single scalar reward. However, ignoring objectives, or, combining objectives via a weighted sum, has been shown to produce sub-optimal results [Vamplew et al., 2008]. For example, combining objectives via a weighted sum assumes that the utility function of a user is linear. In many cases the user's preferences may be nonlinear. For such cases, using a linear utility function will be inadequate to represent the user's true utility [Hayes et al., 2022c]. This will be discussed in more detail in the next paragraph.

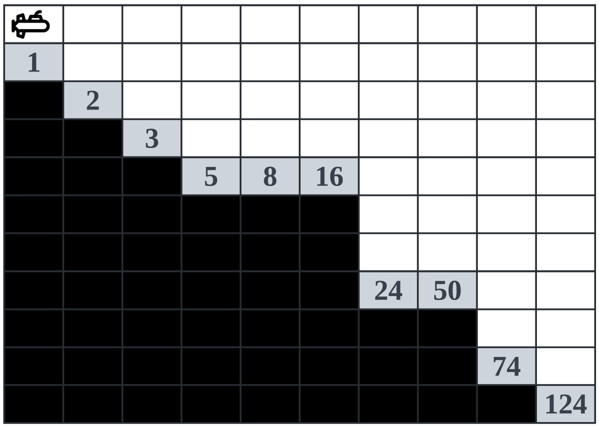


Figure 2.3: The Deep Sea Treasure problem where the agent aims to find and collect treasures on the sea floor while minimising fuel.

To illustrate why the aforementioned approaches to solving multi-objective problems can have limitations, consider the following problem presented in Figure 2.3, known as Deep Sea Treasure [Vamplew et al., 2008]. In Deep Sea Treasure, the agent controls a submarine and aims to collect treasure on the sea floor. At each timestep the submarine uses a single unit of fuel, which incurs a cost. Therefore, depending on the cost of the fuel, the agent may only be able to collect certain treasures. Deep Sea Treasure is a multi-objective problem where the agent aims to minimise fuel and maximise treasure. Solving the Deep Sea Treasure problem is a challenging task. Given the nature of the problem, it is not possible to simply ignore one of the objectives. Therefore, to apply RL and planning algorithms to Deep Sea Treasure, the objectives must be combined by using linear weights. When using linear weights, a number of problems arise. Tuning the weights for each objective must be completed by an AI engineer using a non-intuitive, semi-blind, and iterative process. To find the weights for each objective, the AI engineer will execute the following steps: (a) choose the weights manually for each objective, where the weights must be positive and sum to 1 (b) execute the learning or planning algorithm to evaluate if, for the chosen weights, the desired behaviour can be achieved (c) repeat steps (a) and (b) until weights that can be used to achieve the desired behaviour are identified. This process can be long, frustrating, and, in the end, may not provide the exact behaviour desired (just something close to it). Moreover, tuning the weights is non-intuitive because a small change in the

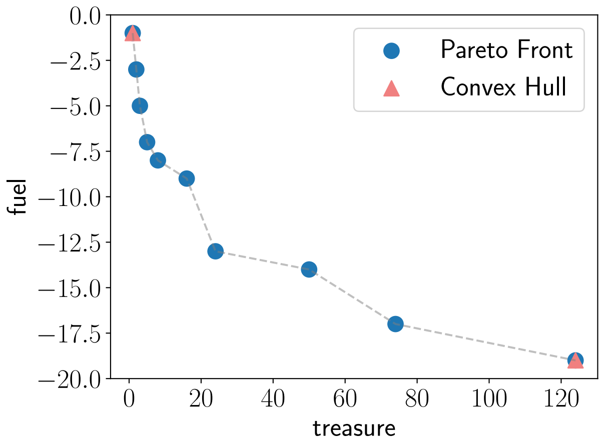


Figure 2.4: The policies on Pareto front and the convex hull for the multi-objective problem known as Deep Sea Treasure.

inputs can lead to a large change in the output, especially if there is a nonlinear relationship between the objectives [Hayes et al., 2020, 2022c].

Additionally, an AI engineer, not a problem domain expert, is tasked with determining the weights. In this scenario, the weights the AI engineer selects may not be accurately aligned with the preferences of a problem domain expert. As such, the policy computed using the weights specified by an AI engineer may cause unintended and potentially serious negative side effects [Vamplew et al., 2018]. In contrast, a problem domain expert has extensive knowledge about the underlying problem and can leverage this knowledge to avoid such undesirable outcomes. Therefore, leaving such important decision factors to an AI engineer has both moral and ethical implications [Vamplew et al., 2018]. A potential solution to overcome the aforementioned implications would be to elicit the preferences from a problem domain expert in the form of a utility function and compute a single optimal policy [Roijers et al., 2013]. Another potential solution is to compute a set of optimal policies. A problem domain expert can then select a policy from the computed set of policies that best reflects their preferences [Hayes et al., 2022c]. Both of the highlighted methods effectively remove the AI engineer from the decision making process. However, the aforementioned solutions require an explicitly multi-objective approach, which will be discussed in detail later.

Furthermore, when using linear weights, the range of policies that can be recovered during learning or planning is limited. By using linear weights to combine objectives, only policies that lie on the convex hull can be computed. This observation has been studied extensively in the multi-objective optimisation [Coello, 2000] literature and the multi-objective decision making (MODeM) literature [Vamplew et al., 2008]. For example, in Figure 2.4, both the convex hull and Pareto front are presented for the Deep Sea Treasure problem. Using linear weights, only policies on the convex hull (highlighted in red) can be computed. Therefore, several other policies (highlighted in blue) that may be optimal cannot be computed. In some settings computing the convex hull may be optimal. However, for many real-world settings, like Covid-19 modelling [Reymond et al., 2022b], calculating the Pareto front would be more beneficial, given the Pareto front generally contains a more diverse set of solutions. A solution to overcome the challenges and limitations highlighted above is to take a multi-objective approach to decision making.

2.4 Multi-Objective Reinforcement Learning & Planning

Multi-objective RL and planning are a natural extension of RL and planning where multiple objectives are explicitly modelled. Both approaches are described extensively in the multi-objective decision making (MODeM) literature, therefore MODeM is used to refer to multi-objective RL and planning. For MODeM, at timestep t an agent has a state s_t. At state s_t, the agent selects an action a_t and transitions to the next state s_{t+1}. To model problems with multiple objectives, the reward received by the agent is a *reward vector*, where there exists a value for each objective. Therefore, the agent receives a reward, $\mathbf{r}_{t+1}$, for each action a and next state s_{t+1} transition.

2.4.1 Problem Setting

A common way to model sequential MODeM problems is to utilise a multi-objective Markov decision process (MOMDP). A MOMDP is a natural extension to MDPs where multiple objectives are explicitly modelled.

Definition 6

A multi-objective Markov decision process (MOMDP) is a tuple, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mu, \mathbf{R} \rangle$, where:

- $\mathcal{S}$ is the state space
- $\mathcal{A}$ is the set of actions
- $\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the probabilistic transition function
- γ is the discount factor
- $\mu : S \leftarrow [0, 1]$ is a probability distribution over states
- $\mathbf{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^n$ is the probabilistic vectorial reward function. The reward, $\mathbf{r}$, from this function at each timestep is a vector, where each component of the vector is a reward for each of the n objectives.

An agent acts according to a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Given a state, actions are selected according to a certain probability distribution. The value function of a policy π in a MOMDP can be defined as follows:

$$\mathbf{V}^{\pi} = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{k+1} \mid \pi, \mu\right], \tag{2.17}$$

where $\mathbf{r}_{k+1} = \mathbf{R}(s_k, a_k, s_{k+1})$ is the reward received at timestep $k+1$, and $\mathbf{V}^{\pi}$ is a vector. The state-value function of a MOMDP can be defined as follows:

$$\mathbf{V}^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{t+k+1} \mid \pi, s_t = s\right], \tag{2.18}$$

where $\mathbf{V}^{\pi}(s)$ is a vector.

For MODeM a utility function can be used to represent a user's (human decision maker's) preferences over the objectives. The utility function maps the multi-objective value of a policy to a scalar value,

$$V_u^{\pi} = u(\mathbf{V}^{\pi}). \tag{2.19}$$

Applying the utility function, u, to the multi-objective value function effectively converts a MOMDP to a MDP, where a total ordering over policies can be determined [Hayes et al., 2022c] by comparing the scalarised values of each policy.

In many scenarios applying the utility function to the value function may be impossible, infeasible, or undesirable. In this case the value functions must be utilised. In MOMDPs, value functions only allow for a partial ordering over policies to be obtained. For value functions, it may be that values on one policy π may be superior to a policy π' on one objective, i, but policy π' may be superior to policy π on another objective, j, i.e., $V_i^\pi > V_i^{\pi'}$ but $V_j^\pi < V_j^{\pi'}$ [Roijers et al., 2013]. In this case, to determine which value functions are optimal, further information on how to prioritise the objectives is required [Hayes et al., 2022c]. Therefore, a utility function, u, is always assumed to be monotonically increasing in all objectives. Utilising monotonically increasing utility functions is the minimal assumption for MODeM, given a user will always want more value in each of the objectives [Hayes et al., 2022c; Roijers et al., 2013].

Definition 7

A monotonically increasing utility function, u, adheres to the constraint that if a policy increases for one or more of its objectives without decreasing any of the objectives, the scalarised value also increases:

$$(\forall i : \mathbf{V}_i^\pi \geq \mathbf{V}_i^{\pi'}) \wedge (\exists i : \mathbf{V}_i^\pi > \mathbf{V}_i^{\pi'}) \implies u(\mathbf{V}^\pi) \geq u(\mathbf{V}^{\pi'})$$

Monotonically increasing utility functions are general and can be used to represent both linear and nonlinear user preferences. Generally, a total ordering over policies cannot be determined in MODeM. By assuming utility functions are monotonically increasing, a partial ordering over polices can be determined to compute sets of possibly optimal policies.

2.4.1.1 Solution Sets

Depending on the setting, a user may be uncertain about their preferences over the objectives. As a result, a utility function may not be available and an a priori scalarisation may not be possible. Furthermore, an a priori scalarisation may be undesirable even if the utility function is known. In this case, a partial ordering over policies can be determined using dominance relations, like Pareto dominance [Pareto, 1896], and sets of policies that are optimal for all monotonically increasing utility functions can be computed.

Definition 8

The undominated set, $U(\Pi)$, is the subset of all possible policies Π and associated value vectors for which there exists a possible utility function u with a maximal scalarised value:

$$U(\Pi) = \left\{ \pi \in \Pi \;\middle|\; \exists u, \forall \pi' \in \Pi : u(\mathbf{V}^\pi) \geq u(\mathbf{V}^{\pi'}) \right\}. \tag{2.20}$$

Furthermore, if a user's utility function is monotonically increasing, then the undominated set can be considered to be the Pareto front:

Definition 9

If the utility function u is any monotonically increasing function, then the *Pareto Front (PF)* is the undominated set [Roijers et al., 2013]:

$$PF(\Pi) = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \mathbf{V}^{\pi'} \succ_P \mathbf{V}^{\pi}\}, \tag{2.21}$$

where $\succ_P$ is the Pareto dominance relation,

$$\mathbf{V}^{\pi} \succ_P \mathbf{V}^{\pi'} \iff (\forall i : \mathbf{V}_i^{\pi} \geq \mathbf{V}_i^{\pi'}) \wedge (\exists i : \mathbf{V}_i^{\pi} > \mathbf{V}_i^{\pi'}). \tag{2.22}$$

However, the definition of the undominated set allows for excess policies to be included, whereby excess policies have the same value vector. As a result, not all policies must be retained to ensure optimal utility. Therefore a coverage set can be defined where the goal is to make a coverage set as small as possible.

Definition 10

A set $CS(\Pi)$ is a *coverage set* if it is a subset of $U(\Pi)$ and if, for every u, it contains a policy with maximal scalarised value, i.e.,

$$CS(\Pi) \subseteq U(\Pi) \wedge \left(\forall u, \exists \pi \in CS(\Pi), \forall \pi' \in \Pi : u(\mathbf{V}^{\pi}) \geq u(\mathbf{V}^{\pi'})\right). \tag{2.23}$$

In this case, excess policies on the Pareto front can be removed, given only one of the policies that has the same value vector needs to be maintained to ensure optimality. Therefore, a set of policies whose value function corresponds to the Pareto front is called a Pareto coverage set [Hayes et al., 2022c].

As previously highlighted, a user's preferences over the objectives may be linear (a positively weighted linear sum). For linear utility functions, the policies in the undominated set will be the convex hull.

Definition 11

A linear utility function computes the inner product of a weight vector $\mathbf{w}$ and a value vector $\mathbf{V}^{\pi}$

$$u(\mathbf{V}^{\pi}) = \mathbf{w}^{\top}\mathbf{V}^{\pi}. \tag{2.24}$$

Each element of $\mathbf{w}$ specifies how much one unit of value for the corresponding objective contributes to the scalarised value. The elements of the weight vector $\mathbf{w}$ are all positive real numbers and constrained to sum to 1.

Definition 12

The *convex hull (CH)* is the subset of Π for which there exists a $\mathbf{w}$ (for a linear u), for which the linearly scalarised value is maximal, i.e., it is the undominated set for linear utility functions:

$$CH(\Pi) = \{\pi \in \Pi \mid \exists \mathbf{w}, \forall \pi' \in \Pi : \mathbf{w}^\top \mathbf{V}^\pi \geq \mathbf{w}^\top \mathbf{V}^{\pi'}\}. \tag{2.25}$$

Finally, to reduce the size of the convex hull, a convex coverage set (CCS) can be computed, which can often be significantly smaller than the convex hull and Pareto front.

Definition 13

A set $CCS(\Pi)$ is a *convex coverage set* if it is a subset of $CH(\Pi)$ and if for every $\mathbf{w}$ it contains a policy whose linearly scalarised value is maximal, i.e., if:

$$CCS(\Pi) \subseteq CH(\Pi) \wedge \left(\forall \mathbf{w}, \exists \pi \in CCS(\Pi), \forall \pi' \in \Pi : \mathbf{w}^\top \mathbf{V}^\pi \geq \mathbf{w}^\top \mathbf{V}^{\pi'}\right). \tag{2.26}$$

Depending on the availability of a user's utility function and the setting, MODeM algorithms can be deployed to compute a single optimal policy or a set of policies that are optimal for all monotonically increasing utility functions. Such an approach highlights the flexibility of MODeM which can be utilised in real-world problems.

2.4.2 Other Problem Settings

A multi-objective approach can also be taken in other decision making settings, like MABs and CoGs. A multi-objective approach to these settings is outlined below.

2.4.2.1 Multi-Objective Multi-Armed Bandits

Multi-objective multi-armed bandits (MOMABs) [Drugan and Nowe, 2013] are a multi-objective extension to MABs. MOMABs lead to important differences compared to MABs, where there could be several optimal arms for a given problem. MOMABs can be defined as follows:

Definition 14

A multi-objective multi-armed bandit (MOMAB) has a finite set of arms, $\mathcal{A}$, where each arm $a \in \mathcal{A}$ returns a reward vector, $\mathbf{R}(a)$, when it is pulled. The reward vector has a value per objective. Each arm a has an associated stationary reward distribution that is unknown to the agent, where $\boldsymbol{\mu}(a) = \mathbb{E}[\mathbf{R}(a)]$

Generally, in MOMABs the utility function is unknown. Therefore, the goal of MOMABs is to learn a set of optimal policies (e.g. the Pareto front or convex hull). Various regret metrics have been formulated for the multi-objective case. For example, Pareto regret and a scalarised regret metric have been defined by Drugan and Nowe [2013] and have been used extensively to evaluate performance of MOMAB algorithms [Yahyaa et al., 2014; Yahyaa and Manderick, 2015].

2.4.2.2 Multi-Objective Coordination Graphs

Multi-objective coordination graphs (MO-CoGs) are a natural extension to CoGs. Generally, for MO-CoGs the utility function of a user is unknown a priori, and a set of optimal policies must be computed. For MO-CoGs, a policy is represented by a global joint action and the expected value vector for executing the associated global joint action [Rollón and Larrosa, 2006; Roijers et al., 2013]. A MO-CoG can be defined as follows:

Definition 15

A multi-objective coordination graph (MO-CoG), a multi-objective extension of CoGs, is a tuple $(\mathcal{D}, \mathcal{A}, \mathcal{P})$.

- $\mathcal{D} = \{1, ..., n\}$ is a set of n agents. $\mathcal{D}$ is factorised into p, possibly overlapping, groups of agents $\mathcal{D}^e$, where e is used to denote a group.
- $\mathcal{A} = \mathcal{A}_i, ..., \mathcal{A}_n$ is the set of joint actions. $\mathcal{A}^e$ denotes the set of local joint actions for the group $\mathcal{D}^e$.
- $\mathcal{P} = \mathbf{p}^1, ..., \mathbf{p}^l$ is a set of l, d-dimensional local payoff functions.
- The joint payoff for all agents is the sum of local payoff functions: $\mathbf{p}(\mathbf{a}) = \sum_{e=1}^{l} \mathbf{p}^e(\mathbf{a}^e)$.

2.4.3 A Note on Scalar and Vector Rewards

Some researchers would argue that modelling problems as multi-objective is unnecessary and that all rewards can be represented as a single scalar signal. For example, Sutton's reward hypotheses states, "All of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)" [Sutton and Barto, 2018]. Furthermore, Silver et al. [2021] define the reward-is-enough hypotheses as follows: "Intelligence, and its associated abilities, can be understood as subserving the maximisation of reward by an agent acting in its environment," where Silver et al. [2021] define a reward as a special scalar observation emitted at each timestep by a reward signal in the environment. Scalar rewards are suitable for many problems, however, many other researchers would argue that vector rewards are more appropriate for general use in decision making problems. Vamplew et al. [2022] argue against the general use of scalar rewards and the reward-is-enough hypothesis. Instead, Vamplew et al. [2022] promote following a multi-objective approach using vector rewards[3]. A brief overview of some of the arguments for vector reward outlined by Vamplew et al. [2022] is presented below.

Silver et al. [2021] imply that the maximisation of the cumulative scalar reward is the general case for decision making problems. However, Vamplew et al. [2022] provide strong arguments that the maximisation of the cumulative scalar reward is the special case for decision making problems. Vamplew et al. [2022] argue that scalar rewards (where the number of rewards $n = 1$) are a subset of vector rewards (where the number of rewards $n > 1$). Therefore, an agent designed to optimise for a reward vector can also be used in scenarios with a single scalar reward, as the reward can simply be treated as a one-dimensional vector [Vamplew et al., 2022]. However, the inverse is not true. Silver et al. [2021] state that a solution to a general problem also provides a solution to any special cases. However, based on the argument outlined by Vamplew et al. [2022], vector rewards are more general.

Furthermore, Silver et al. [2021] acknowledge that multiple objectives can exist, but outline that a scalar reward signal can be represented using a linearly weighted combination of the objectives. As previously discussed in Section 2.3, such an approach has several limitations and a scalar representation may not be adequate to represent a user's true utility [Vamplew et al., 2022]. Additionally, a scalar reward representing a weighted combination of objectives encodes a fixed weighting of the objectives. In this case, the agent can only compute policies with respect to that weighting. In contrast, an agent which uses an explicitly multi-objective approach

[3]Silver et al. [2021] and Vamplew et al. [2022] make respective arguments about scalar rewards and vector rewards in the context of artificial general intelligence. Discussion on artificial general intelligence is considered out of scope of this work and is therefore not included.

that maintains vector rewards can compute a set of policies which are optimal for all utility functions. For scenarios where the user's preferences change over time, a standard single-objective RL algorithm must start learning from scratch. However, this is not the case for dynamic utility and multi-policy MODeM methods. For example, if using a multi-policy method, another policy which best reflects the new preferences can be selected without the need for further learning. This ensures rapid, or even immediate, adaption if the utility function changes [Vamplew et al., 2022]. Such an approach cannot be taken with scalar rewards.

Further arguments against the general use of scalar rewards have been made by Roijers et al. [2013]. To utilise scalar rewards for problems with multiple objectives, a MOMDP must be converted to a MDP by using an a priori scalarisation. However, Roijers et al. [2013] have argued that when following the utility-based perspective (which will be discussed in the next section), an a priori scalarisation can be impossible, infeasible, or undesirable [Roijers et al., 2013].

In line with the discussion above, this work advocates for a multi-objective approach using vector rewards.

2.4.4 The Utility-Based Perspective

An approach often adapted in the MODeM literature is the axiomatic approach, where the Pareto front is always assumed to be the optimal solution. In certain settings, taking an axiomatic approach can be useful because the Pareto front is a set of optimal policies for all monotonically increasing utility functions. However, the axiomatic approach has several limitations. In many practical settings, domain knowledge may be readily available, which can be used to model a utility function. However, by taking an axiomatic based approach it is difficult, if not impossible, to encode domain knowledge into the process of computing optimal policies. By exploiting domain knowledge, it may be possible to compute a single optimal policy. However, when taking an axiomatic approach the Pareto front must be computed. In this case, any available domain knowledge is not utilised and both time and computation are wasted calculating solutions which may be sub-optimal with respect to the user's utility function. Another limitation is that computing the Pareto font may be infeasible in certain settings. If the state-action space for a given problem domain is large enough, it may be prohibitively expensive, and perhaps infeasible, to compute the Pareto front. As a result, Roijers et al. [2013] and Hayes et al. [2022c] recommend following the utility-based approach, which is becoming widely adapted in the MODeM literature [Roijers and Whiteson, 2017; Reymond et al., 2021].

The utility-based approach considers user utility first, and aims to derive the optimal solutions from the information available about a user's utility function.

By exploiting this knowledge, it is possible to put constraints on the solution set, which can improve efficiency and make it easier for users to select their preferred policy. In contrast to the axiomatic approach, it is also possible to encode system domain knowledge and represent this information as a utility function. The utility-based approach has the following steps [Hayes et al., 2022c]:

1. Collect all a priori available information regarding a user's utility.
2. Decide which type of policies (e.g., stochastic or only deterministic) are allowed.
3. Derive the optimal solution concept from the resulting information of the first two points.
4. Select or design a multi-objective reinforcement learning or planning algorithm that fits the solution concept.
5. When multiple policies are required for the solution, design a method for the user to select the desired policy from the set of optimal policies.

In Step 1, all available information about a user's utility function is collected. Using this information it is possible to determine which class of utility functions should be used. For example, a user's utility function may be linear. A user may also not know their preferences and, therefore, a utility function cannot be derived before planning or learning.

In Step 2, the policy types that are allowed must be decided. For example, in MODeM settings stochastic policies have been shown to dominate deterministic policies [Vamplew et al., 2021b; Wakuta and Togawa, 1998].

Using the information gathered in Step 1 and Step 2, the appropriate solution concept must be selected. For example, if the utility function is unknown, then a set of optimal solutions must be computed. However, if the utility function is known to be linear, then the convex hull can be computed. Moreover, if the utility function is known, then a single optimal policy can be computed for the known utility function.

In Step 3, the appropriate solution concept must derived. The selection of the solution concept depends on Step 1 and Step 2. For example, if the utility function of a user is known and linear, then any type of policy is allowed. In this case, the known utility function scenario can be selected as the solution concept, and a single optimal policy can be computed. In Section 2.4.5, each solution concept is discussed in detail.

In Step 4 an algorithm to compute the desired solution must be selected or designed. Therefore, an algorithm from the literature can be selected or a novel algorithm must be designed to solve the multi-objective problem. Depending on the availability of the utility function, different types of algorithms must be utilised. If the utility function is known, then a single-policy algorithm can be used. Single-policy algorithms compute a single optimal policy for a given utility function. However, if the utility function is unknown, then a multi-policy algorithm must be used to compute a set of optimal policies for all monotonically increasing utility functions.

Finally, the aim of Step 5 is to aid the user in selecting a policy which best reflects their preferences. Step 5 is only relevant when a set of optimal policies has been computed in Step 4. Therefore, a user must select a single policy from the set of returned policies that best reflects their preferences. In settings where a small number of policies is returned to a user, this process may be straightforward. However, for continuous settings the computed solution set may be large and, therefore, a user may have difficulty in selecting a policy. While some work has been presented in this area [Zintgraf et al., 2018], how to appropriately visualise and present a set of polices to a user remains an open question.

Once the desired solution is selected in Step 5, then the selected policy can be executed during the execution phase. Together, these steps form a complete pipeline to set up a multi-objective reinforcement learning or planning system [Hayes et al., 2022c].

2.4.5 Multi-Objective Solution Concepts

Many scenarios require an explicitly multi-objective approach. As a result, Roijers et al. [2013] present three scenarios where a multi-objective approach is required as illustrated in (a), (b) and (c) in Figure 2.5. Recently, Hayes et al. [2022c] proposed three new scenarios hat require a multi-objective approach: the interactive decision support scenario (d), the dynamic utility function scenario (e), and the review and adjust scenario (f). Figure 2.5 presents each of the outlined scenarios, and shows that each scenario consists of a planning or learning phase, an execution phase, and for some scenarios a selection phase. Each of the proposed scenarios is described below.

In the **unknown utility function scenario** (a) [Rădulescu et al., 2020], the utility functions of a user is unknown at the time of learning or planning, therefore a priori scalarisation is undesirable. In this scenario, a single optimal policy cannot be computed, given there is little information available about the user's utility function. Therefore, a set of policies, that are optimal for all monotonically increasing utility functions, must be computed (e.g. a coverage set). A policy can then be selected from the set of optimal policies when more information about the user's utility functions becomes available. During the selection phase, it is assumed

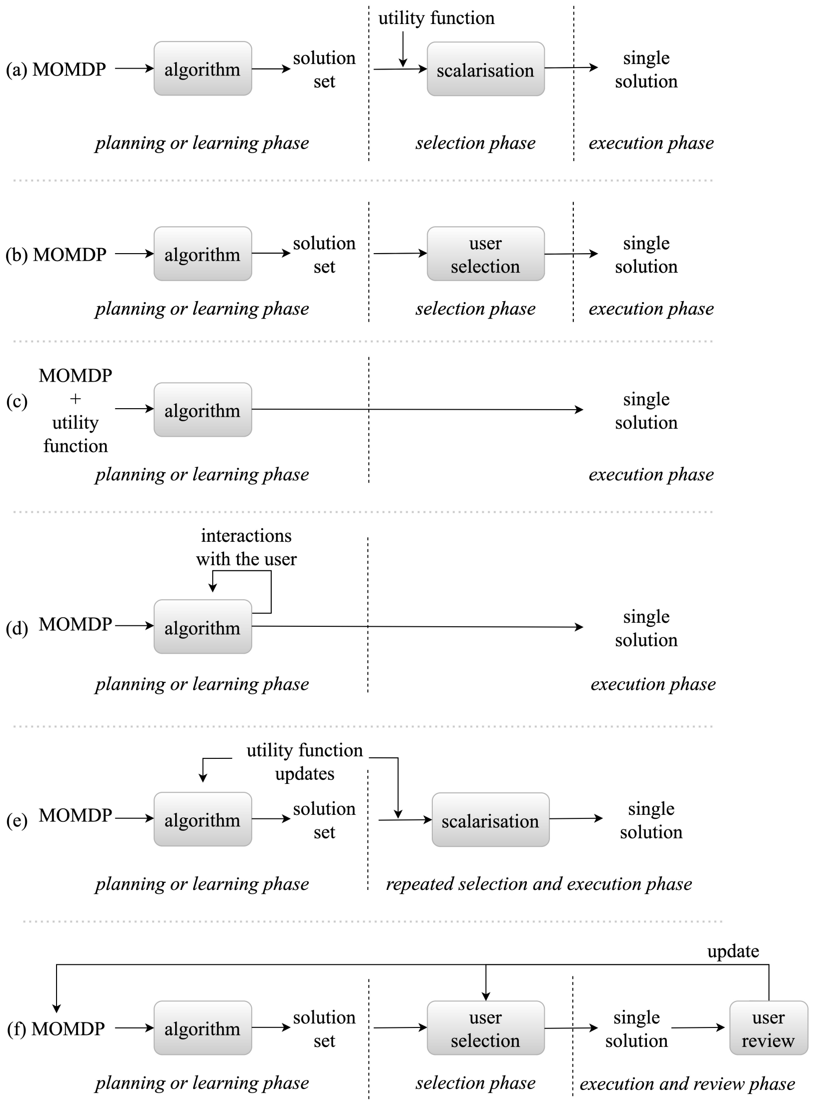


Figure 2.5: The six motivating scenarios for MOMDPs: (a) the unknown utility function scenario, (b) the decision support scenario, (c) the known utility function scenario, (d) the interactive decision support scenario, (e) the dynamic utility function scenario, and (f) the review and adjust scenario [Hayes et al., 2022c].

that the utility function of a user becomes known, therefore, a utility revelation step is included where the utility function becomes explicit.

In the **decision support scenario** (b), the user's preferences over the objectives are unknown or difficult to specify. Similarly to the unknown utility function scenario, an a priori scalarisation is infeasible given the utility function is unknown. Therefore a set of optimal policies must be computed. Defining a utility function can be difficult, if not infeasible. Therefore, during the selection phase the decision relies on the user, and the utility function remains implicit in the decision taken.

In the **known utility function scenario** (c), the user's preferences are known. Therefore, the user's utility function is known at the time of learning or planning. As a result, scalarisation is possible and a single optimal policy can be computed. It is important to note, in some cases a priori scalarisation can lead to an intractable problem [Rădulescu et al., 2020; Roijers et al., 2013; Hayes et al., 2022c].

In the **interactive decision support scenario** (d), the agent has to learn about both the preferences of the user and the environment [Roijers et al., 2018a]. Given the uncertainty about the user's preferences, applying a priori scalarisation in this scenario can be both undesirable and infeasible. Therefore, preference elicitation is utilised during learning to remove uncertainty about the user's utility function. A method to elicit information about a user's preferences is to present the user with different solutions during the learning phase. The user can then rank the solutions in order of preference. Using this information it would be possible to get a more accurate representation of the user's preferences and compute an optimal solution.

In the **dynamic utility function scenario** (e), the user's preferences over objectives changes over time [Natarajan and Tadepalli, 2005]. Therefore, applying a priori scalarisation would be undesirable. In this case, it would be optimal to compute a finite number of policies over time. Then a non-dominated policy for any utility function can be chosen and improved upon by further learning for that utility. As shown by Abels et al. [2019]; Natarajan and Tadepalli [2005] efficiency can be improved by reusing information from previously encountered utilities.

In the **review and adjust scenario** (f), a user may be uncertain about their preferences over objectives. A further complicating factor in this scenario is a user's preferences over objectives may also change over time. Therefore, applying a priori scalarisation is infeasible because there is too much uncertainty around the utility function of the user. As a result a set of optimal policies must be computed during the learning or planning phase. During the selection phase the user can select a policy which accurately represents their preferences. During the execution phase a review step is introduced, where the user can review their chosen solution before the solution is executed. If the user's preferences have changed, the user can simply adjust their selected solution to accurately reflect their updated preferences.

The review process can also update the MOMDP, which can alter the computed set of solutions. This may, for example, occur when a new objective is identified that was previously missed.

2.4.6 Multi-Objective Reinforcement Learning & Planning Algorithms

To compute policies for each of the scenarios outlined above, explicitly multi-objective algorithms must be deployed. In this section an overview of various algorithms for a number of settings is provided.

2.4.6.1 Stateless & Bandit Algorithms

Stateless algorithms can be utilised to solve MO-CoGs. For example, multi-objective variable elimination (MOVE) methods extend traditional variable elimination (VE) [Koller and Friedman, 2009] to problems with multiple-objectives. MOVE methods have been used to compute sets of optimal policies for MO-CoGs. Pareto multi-objective variable elimination (PMOVE) [Rollón and Larrosa, 2006; Rollón, 2008] computes the Pareto front for MO-CoGs. Convex hull variable elimination (CMOVE) [Roijers et al., 2015] computes the convex hull for MO-CoGs. CMOVE shows that computing the convex hull can be more computationally efficient compared to computing the Pareto front [Roijers et al., 2015]. Other stateless algorithms use AND/OR branch and bound methods [Marinescu, 2009] or influence diagrams [Marinescu et al., 2017, 2012] to solve multi-objective problems.

Many algorithms have been proposed to solve MOMABs. For example, the UCB algorithm has been used as a starting point for several multi-objective algorithms. Drugan and Nowe [2013] extend UCB using both linear and Chebyshev scalarisations; they also extend UCB with Pareto dominance. Furthermore, Gaussian process Thompson sampling [Roijers et al., 2020] utilises both TS with Gaussian processes to learn for the interactive decision support scenario. Turgay et al. [2018] extended contextual MABs to model multiple-objectives and propose a Pareto contextual zooming algorithm to minimise Pareto regret.

2.4.6.2 Single-Policy Algorithms

In the known utility function scenario, a single optimal policy must be computed and executed. For this setting, single-policy algorithms are deployed when learning or planning. The simplest and most-widely adopted approach is to extend existing RL or planning methods to handle multiple-objectives. For example, when the utility function is linear, applying a linear scalarisation is the equivalent of transforming a MOMDP to a MDP. In this case, a single-objective RL or planning algorithm can be used to compute a single optimal policy using weighted or unweighted linear utility functions [Aissani et al., 2008; Guo et al., 2009; Perez et al., 2009; Shabani, 2009]. A limitation of linear utility functions is that they are not always an adequate representation of a user's preferences over objectives [Hayes et al., 2022c]. To overcome this limitation, nonlinear utility functions must be used [Hayes et al., 2022c; Roijers et al., 2013]. However, nonlinear utility functions do not distribute across the sum of the immediate and future returns [Roijers et al., 2018b], which violates the assumption of additive returns in the Bellman equation [Hayes et al., 2022c; Roijers et al., 2018b]. Therefore, explicitly multi-objective algorithms must be used to compute policies for nonlinear utility functions. Reymond et al. [2021] and Roijers et al. [2018b] propose single policy multi-objective algorithms that can learn policies for nonlinear utility functions. These approaches use Monte Carlo rollouts to compute the future returns, and apply the utility function to the cumulative return vector.

Policy gradient methods have also been extended to multi-objective settings. For example, Siddique et al. [2020] extend PPO [Schulman et al., 2017] and A2C [Mnih et al., 2016] to compute a single policy that is fair in all objectives. Siddique et al. [2020] define a fair solution as one that is Pareto optimal, satisfies the equal treatment of equals principal, and satisfies the Pigou-Dalton principle. To implement these concepts Siddique et al. [2020] use the generalised Gini social welfare function. Furthermore, Pan et al. [2020] propose a policy gradient and planning method to compute a single policy in multi-objective settings.

2.4.6.3 Multi-Policy Algorithms

Many multi-objective scenarios exist where the utility function of a user is unknown or uncertain. In this setting, it is not possible to apply an a priori scalarisation. Therefore, multi-policy algorithms must be deployed to compute a set of optimal policies. Multi-policy methods are divided into two categories: outer loop methods and inner loop methods.

Outer loop methods solve a series of single objective problems to compute a coverage set. For example Mossalam et al. [2016] create an outer loop deep RL multi-policy algorithm by applying the optimistic linear support algorithm [Roijers et al., 2015] to a deep scalarised Q-learning algorithm. The proposed algorithm learns an optimal policy for each optimal linear weight, where the policy learned is represented by a DQN instance. The set of policies returned to the user is the convex hull.

Inner loop methods are explicitly designed to learn multiple policies in a single pass. Some inner loop methods follow the utility-based approach [Abels et al., 2019; Castelletti et al., 2012], however, many follow the axiomatic based approach

[Parisi et al., 2017; Ruiz-Montiel et al., 2017; Van Moffaert and Nowé, 2014b; Wiering et al., 2014]. An example of an inner loop method is Pareto Q-learning [Van Moffaert and Nowé, 2014b]. Pareto Q-learning learns sets of Pareto optimal policies in a single run in episodic environments with deterministic and stochastic reward functions. Pareto Q-learning learns by bootstrapping sets of Q-vectors [Van Moffaert and Nowé, 2014b], where Van Moffaert and Nowé [2014b] propose a mechanism that separates the expected immediate reward vector from the set of expected future discounted reward vectors. By taking this approach, it is possible to update the sets of policies and to exploit the learned policies consistently throughout the state space [Van Moffaert and Nowé, 2014b]. Pareto Q-learning returns the Pareto front to the user during the selection phase. However, when the reward function is stochastic, Pareto Q-learning suffers from the policy following problem [Roijers et al., 2021] making policy execution difficult in settings where the reward function is stochastic.

Many other multi-policy methods exist in MODeM literature. Reymond et al. [2022a] extend upside down RL [Kumar et al., 2019; Schmidhuber, 2019] to multi-objective settings and learn to approximate the Pareto front for settings with continuous state and action spaces [Reymond et al., 2022b]. Furthermore, Yang et al. [2019], Abels et al. [2019], Reymond and Nowé [2019], and Alegre et al. [2022] propose interesting multi-policy MODeM methods.

For planning settings, methods like convex hull Monte Carlo tree search (CHMCTS) extend MCTS to compute the convex hull. Multi-objective Monte Carlo tree search (MOMCTS) [Wang and Sebag, 2012] extends MCTS to multi-objective settings and can learn the Pareto front in deterministic environments. Bryce et al. [2007] propose a multi-objective LAO^* algorithm to compute the Pareto front in multi-objective planning settings. White [1982] adapted dynamic programming to find Pareto optimal policies for infinite horizon MOMDPs.

2.4.7 Multi-Objective Optimality Criteria

When following the utility-based perspective, depending on how the utility function is utilised, different optimality criteria can arise. The MODeM literature distinguishes between two optimality criteria: the scalarised expected returns (SER) and the expected scalarised returns (ESR). The selection of which optimality criterion to apply depends on how the utility of a user is derived. In scenarios where the utility of a user is derived from the expected outcome over multiple executions of a policy, the SER criterion should be optimised [Hayes et al., 2022c]:

$$V_u^\pi = u\left(\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \mid \pi, \mu_0\right]\right). \tag{2.27}$$

Under the SER criterion the expected value vector is computed, then the utility function is applied. Therefore, under the SER criterion the utility of the expectation is computed. In this case, a user will execute the selected policy multiple times during the execution phase. The SER criterion is the most commonly used criterion in the multi-objective (single-agent) MODeM literature [Alegre et al., 2022; Wang and Sebag, 2012; White, 1982; Xu et al., 2020]. For SER, a set of non-dominated policies that are optimal for all possible utility functions is known as a coverage set.

Applying the utility function to the returns and then calculating the expected utility leads to the ESR criterion:

$$V_u^\pi = \mathbb{E}\left[u\left(\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \right) \mid \pi, \mu_0 \right]. \tag{2.28}$$

In scenarios where the utility function of a user is derived from the single execution of a policy, the ESR criterion should be optimised [Hayes et al., 2022c]. Under the ESR criterion, the utility function is applied to the returns first, and then the expectation is computed. Therefore, the expected utility is computed. In this case, a user may execute their selected policy once during the execution phase. The ESR criterion is the most commonly used criterion in the game theory literature on multi-objective games [Rădulescu et al., 2020; Röpke et al., 2021]. However, the ESR criterion has largely been ignored in the MODeM literature [Hayes et al., 2022c].

2.5 The Expected Scalarised Returns Optimality Criterion

The expected scalarised returns (ESR) criterion was introduced by Roijers et al. [2013], where the ESR criterion was identified as an open research question in the MODeM literature. Recently, Rădulescu et al. [2020] identified that the distinction between the ESR criterion and the SER criterion does not exist when the utility function of a user is linear. However, Rădulescu et al. [2020] also identified that the policies computed under the ESR criterion and the SER criterion can be different when the utility function of a user is nonlinear. The findings presented by Rădulescu et al. [2020] were demonstrated only for multi-agent settings; no results for single-agent settings were presented. However, in the MODeM literature it has been assumed that the policies for the ESR criterion and the SER criterion can be different for nonlinear utility functions in single-agent settings [Roijers et al., 2018b; Hayes et al., 2022c]. As a result, some explicitly multi-objective methods have been developed that compute policies for the ESR criterion [Roijers et al., 2018b; Reymond et al., 2021; Malerba and Mannion, 2021; Vamplew et al., 2021a]. Furthermore, no multi-policy methods for the ESR criterion have been developed. It has not been explored in the literature if expected value vectors based methods, which are used to compute policies under the SER criterion, can be utilised to determine a partial ordering over policies under the ESR criterion. Moreover, a set of optimal policies for the ESR criterion has yet to be defined.

As previously mentioned, in the MODeM literature, when computing policies under the ESR criterion, the utility function is assumed to be nonlinear. Computing policies for nonlinear utility function can be challenging because nonlinear utility functions do not distribute across the sum of the immediate and future returns, which violates the assumption of additive returns in the Bellman equation [Hayes et al., 2022c],

$$\begin{aligned} &\max_{\pi} \mathbb{E}\left[u\left(\mathbf{R}_t^- + \sum_{i=t}^{\infty} \gamma^i \mathbf{r}_i \right) \,\middle|\, \pi, s_t \right] \neq \\ &u(\mathbf{R}_t^-) + \max_{\pi} \mathbb{E}\left[u\left(\sum_{i=t}^{\infty} \gamma^i \mathbf{r}_i \right) \,\middle|\, \pi, s_t \right], \end{aligned} \tag{2.29}$$

where u is a nonlinear utility function and $\mathbf{R_t^-} = \sum_{i=0}^{t-1} \gamma^i \mathbf{r}_i$. To compute policies for nonlinear utility functions, methods which utilise the Bellman equation must augment the state by conditioning the state on the accrued returns. This approach ensures the utility for nonlinear utility functions can be correctly calculated. Furthermore, if the accrued returns are utilised to calculate the utility, but the state is not augmented with the accrued returns, the Markov property is broken [Reymond et al., 2021]. Therefore, to ensure the Markov property is intact, the state must be augmented using the accrued returns [Reymond et al., 2021]. The accrued returns, $\mathbf{R_t^-}$, is the sum of the rewards received from timestep 0 to timestep $t-1$. However, by taking this approach the algorithm may fail to converge to the optimal policy [Hayes et al., 2022c]. Furthermore, explicitly multi-objective methods must be used to compute policies for nonlinear utility functions.

A method that can learn policies for nonlinear utility functions under ESR criterion is expected utility policy gradient (EUPG) [Roijers et al., 2018b]. EUPG is an extension of policy gradient [Sutton and Barto, 2018; Williams, 1992], 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, $t-1$. EUPG also calculates the future returns, $\mathbf{R}_t^+$, and uses Monte Carlo rollouts to calculate the future returns, $\mathbf{R}_t^+$. Using both the accrued and future returns enables EUPG to optimise over the utility of the full returns of an episode, where the utility function is applied to the sum of $\mathbf{R}_t^-$ and $\mathbf{R}_t^+$.

Policy gradient methods adapt the policy towards the attained utility by gradient descent. For EUPG the utility of the sum of the accrued and future returns is calculated inside the loss function, which results in the following:

$$\mathcal{L}(\pi) = -\sum_{t=0}^{T} u(\mathbf{R}_t^- + \mathbf{R}_t^+) \log(\pi_\theta(a|s, \mathbf{R}_t^-, t)). \tag{2.30}$$

Roijers et al. [2018b] demonstrated that for the ESR criterion the accrued and future returns must be considered when learning in order to learn a good policy. Applying this consideration to EUPG, the algorithm achieves the state-of-the-art performance under the ESR criterion.

Although the ESR criterion has received some attention in recent years [Roijers et al., 2018b; Rădulescu et al., 2020; Malerba and Mannion, 2021], the ESR criterion still largely remains under-explored. It has yet to be determined for single-agent settings if the policies computed for the ESR criterion and the SER criterion are different for nonlinear utility functions. If the policies can be different for nonlinear utility functions, algorithms that can compute policies explicitly for the ESR criterion must be developed. Furthermore, it is unknown whether current multi-policy methods that utilise expected value vectors to determine a partial ordering over policies under the SER criterion can be used for the ESR criterion. Additionally a set of optimal policies has yet to be determined for the ESR criterion.

In scenarios where a user may only have a single opportunity to execute the policy, the ESR criterion should be optimised. Therefore, the ESR criterion aligns with many real-world decision making scenarios. To effectively extend MODeM to a broad range of real-world problems, the ESR criterion must be investigated further.

3 Algorithms for Known Utility Functions[1]

Chapter 3 introduces the first contributions of this thesis. As previously highlighted in Chapter 2, designing algorithms that compute policies for nonlinear utility functions poses a significant challenge in multi-objective decision making (MODeM). The policies computed for nonlinear utility functions under the scalarised expected returns (SER) criterion and the expected scalarised returns (ESR) criterion have been shown to be different in multi-agent settings. However, whether this also holds in single-agent settings has not been comprehensively determined.

This chapter aims to investigate the impact of nonlinear utility functions on different MODeM optimality criteria in single-agent settings, and also proposes algorithms that can compute policies for the ESR criterion. The contributions of Chapter 3 are as follows:

1. Section 3.1 investigates if, for single-agent settings, the policies computed under the SER criterion and ESR criterion can be different when the utility function is nonlinear. It is demonstrated by example that policies computed for nonlinear utility functions in single-agent settings can be different when optimising for the SER criterion and the ESR criterion.

[1]The contributions presented in Chapter 3 are published in the following papers: [Hayes et al., 2021a,b, 2022d]

2. Section 3.2 outlines Monte Carlo tree search for nonlinear utility functions (NLU-MCTS). The outlined algorithm can also compute policies for the ESR criterion. The proposed algorithm overcomes the challenges previously outlined when optimising for nonlinear utility functions.

3. Section 3.3 proposes a novel distributional Monte Carlo tree search (DMCTS) algorithm that computes an approximate posterior distribution and utilises Thompson sampling (TS) for exploration during planning. DMCTS computes policies for nonlinear utility functions and can also optimise for the ESR criterion. DMCTS also overcomes the previously highlighted challenges that arise when optimising for nonlinear utility functions.

4. Finally, Section 3.4 performs an empirical evaluation where the algorithms proposed in Section 3.2 and Section 3.3 are evaluated against state-of-the-art algorithms from the literature in several sequential MODeM problems. Section 3.4 outlines how both proposed algorithms overcome the challenges associated with nonlinear utility functions to achieve good performance in each evaluation domain.

3.1 A Note on Nonlinear Utility Functions

As previously mentioned, different optimality criteria exist for MODeM. In scenarios where the utility of a user is derived from multiple executions of a policy, the agent should optimise over the SER criterion. In scenarios where the utility of a user is derived from a single execution of a policy, the agent should optimise for the ESR criterion.

Consider the following example: a power plant that generates electricity for a city and emits harmful CO_2 and greenhouse gases. City regulations have been imposed which limit the amount of pollution that the power plant can generate. If the regulations require that the emissions from the power plant do not exceed a certain amount over an entire year, the SER criterion should be optimised. In this scenario, the regulations allow for the pollution to vary day to day, as long as the emissions do not exceed the regulated level for a given year. However, if the regulations are much stricter and the power plant is fined every day it exceeds a certain level of pollution, it is beneficial to optimise under the ESR criterion.

The majority of MODeM research focuses on linear utility functions. However, in the real world, a user's utility function may be nonlinear. For example, a utility function is nonlinear in situations where a minimum value must be achieved on each objective [O'Callaghan and Mannion, 2021]. Focusing on linear utility functions limits the applicability of MODeM in real-world decision making problems. For

L_1

P(L_1= $\mathbf{R}$)	$\mathbf{R}$
0.5	(4, 3)
0.5	(2, 3)

L_2

P(L_2=$\mathbf{R}$)	$\mathbf{R}$
0.9	(1, 3)
0.1	(10, 2)

Table 3.1: A lottery, L_1, has two possible returns, (4, 3) and (2, 3), each with a probability of 0.5. A lottery, L_2, has two possible returns, (1, 3) with a probability of 0.9 and (10, 2) with a probability of 0.1.

example, linear utility functions cannot be used to learn policies in concave regions of the Pareto front [Vamplew et al., 2008][2]. Furthermore, if a user's preferences are nonlinear, these are fundamentally incompatible with linear utility functions. In this case, strictly multi-objective methods must be used to learn optimal policies for nonlinear utility functions. In MODeM, for nonlinear utility functions, different policies are preferred when optimising under the ESR criterion versus the SER criterion [Rădulescu et al., 2020]. While this has been shown in multi-agent settings, the difference in policies for nonlinear utility functions for different optimality criteria has not been shown in single-agent settings. Therefore, the example below is used to investigate if different policies are preferred for nonlinear utility function under different MODeM optimality criteria[3].

For example, a decision maker has to choose between the following lotteries, L_1 and L_2, which are highlighted in Table 3.1.

The decision maker has the following nonlinear utility function:

$$u(\mathbf{x}) = x_1^2 + x_2^2, \tag{3.1}$$

where $\mathbf{x}$ is a vector returned from $\mathbf{R}$ in Table 3.1, and x_1 and x_2 are the values of two objectives. Note that this utility function is monotonically increasing for $x_1 \geq 0$ and $x_2 \geq 0$. Under the SER criterion, the decision maker will compute the expected value of each lottery, apply the utility function, and select the lottery that maximises their utility function. Let us consider which lottery the decision maker will play under the SER criterion:

$$L_1 : \mathbb{E}(L_1) = 0.5(4,3) + 0.5(2,3) = (2,1.5) + (1,1.5) = (3,3)$$
$$L_1 : u(\mathbb{E}(L_1)) = (3^2 + 3^2) = 9 + 9 = 18$$

[2]When using linear utility functions only policies that lie on the convex hull can be recovered, reducing the number of optimal policies that can be computed. See to Section 2.3.

[3]It is important to note that, for linear utility functions, the distinction between ESR and SER does not exist [Rădulescu et al., 2020].

$$L_2 : \mathbb{E}(L_2) = 0.9(1,3) + 0.1(10,2) = (0.9, 2.7) + (1, 0.2) = (1.9, 2.9)$$
$$L_2 : u(\mathbb{E}(L_2)) = (1.9^2 + 2.9^2) = 3.61 + 8.41 = 12.02$$

Therefore, a decision maker with the utility function in Equation 3.1 will prefer to play lottery L_1 under the SER criterion.

Under the ESR criterion, the decision maker will first apply the utility function to the return vectors, compute the expectation, and select the lottery to maximise their utility function. Let us consider how a decision maker will choose which lottery to play under the ESR criterion:

$$L_1 : \mathbb{E}(u(L_1)) = 0.5(u(4,3)) + 0.5(u(2,3)) = 0.5(4^2 + 3^2) + 0.5(2^2 + 3^2)$$
$$= 0.5(25) + 0.5(13) = 12.5 + 6.5 = 19$$
$$L_2 : \mathbb{E}(u(L_2)) = 0.9(u(1,3)) + 0.1(u(10,2)) = 0.9(1^2 + 3^2) + 0.1(10^2 + 2^2)$$
$$= 0.9(10) + 0.1(104) = 9 + 10.4 = 19.4$$

Therefore, a decision maker with the utility function in Equation 3.1 will prefer to play lottery L_2 under the ESR criterion. From the example, it is clear that users with the same nonlinear utility function can prefer different policies, depending on which multi-objective optimisation criterion is selected. Therefore, it is critical that the distinction between ESR and SER is taken into consideration when selecting a MODeM algorithm to compute optimal policies[4].

The majority of MODeM research focuses on the SER criterion [Rădulescu et al., 2020]. By comparison, the ESR criterion has received very little attention from the MODeM community [Roijers et al., 2013; Hayes et al., 2022c; Roijers et al., 2018b; Rădulescu et al., 2020]. Many of the traditional MODeM methods cannot be used when optimising under the ESR criterion, given nonlinear utility functions in MOMDPs do not distribute across the sum of immediate and future returns, which invalidates the Bellman equation [Roijers et al., 2018b]. As a result single objective methods cannot be used to compute policies for nonlinear utility functions. This poses a significant challenge for the RL and planning community given the majority of methods cannot be utilised to compute policies for nonlinear utility functions. However, it is possible to utilise multi-objective methods [Roijers et al., 2018b]. Furthermore, Section 3.2 and Section 3.3 propose two novel multi-objective algorithms that can compute policies for nonlinear utility functions under the ESR criterion.

[4]It is important to note, Roijers et al. [2018b] briefly discuss the differences between the SER criterion and the ESR criterion in single-agent settings for nonlinear utility functions. However, the work of Roijers et al. [2018b] does not formally address or investigate the differences in the values of policies under the different optimality criteria for nonlinear utility functions.

3.2 Monte Carlo Tree Search for Nonlinear Utility Functions

In Section 3.1 it was shown that the policies computed under the SER criterion and the ESR criterion can be different when the utility function is nonlinear. Therefore dedicated methods that can optimise for the ESR criterion must be developed.

To compute policies for the ESR criterion when the utility function is nonlinear and known a priori [Hayes et al., 2022c], a Monte Carlo tree search for nonlinear utility functions (NLU-MCTS) algorithm is presented. As shown by Roijers et al. [2018b], to compute optimal policies for nonlinear utility functions under the ESR criterion, both the accrued and future returns must be taken into consideration before applying the utility function. Therefore, an algorithm must either maintain a distribution over the returns or have some method which allows the agent to sample from the underlying return distribution of the environment. NLU-MCTS utilises the latter, by performing Monte Carlo simulations to compute the future returns.

Usually, in single-objective MCTS an expectation of the returns is maintained at each chance node and the agent seeks to maximise the expectation. When the utility function is nonlinear, making decisions based on the expected returns does not account for the potential undesired outcomes a decision might have. For MODeM under the ESR criterion, decisions must be made with sufficient information to avoid undesirable outcomes and exploit positive outcomes. Therefore, computing the utility of the cumulative returns (the returns received from executing a policy), can be used to replace the expected future returns (of vanilla MCTS) at each node.

Before a method to compute the accrued and future returns is presented, the structure of the search tree utilised by NLU-MCTS is outlined. Under the ESR criterion, the environment can be stochastic, where the state transitions or reward function are stochastic. To handle this uncertainty, NLU-MCTS builds an expectimax search tree using the same planning phase as MCTS (see Section 2.1.1.3). A search tree is a representation of the state-action space that is incrementally built via the steps of the underlying MCTS algorithm. An expectimax search tree [Veness et al., 2011] uses both decision and chance nodes.

Figure 3.1 describes a search tree constructed by NLU-MCTS, which contains both decision and chance nodes. Each decision node represents a state, action, and reward of a MOMDP, where each decision node has a child chance node per action. In Section 3.4, environments with stochastic rewards are examined. Each chance node represents the state and action of a MOMDP. At each chance node, the environment is sampled. For NLU-MCTS, if a new observation-reward combination

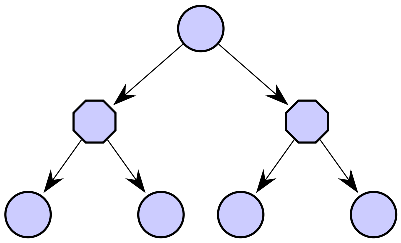


Figure 3.1: A representation of a search tree constructed using NLU-MCTS for a problem with stochastic rewards and two actions. The search tree contains both decision nodes, represented by circular nodes, and chance nodes, represented by octagons.

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 a new observation-reward combination is received when sampling the environment. To build and traverse a search tree similar to MCTS, NLU-MCTS uses the following phases: selection, expansion, simulation, and backpropagation.

Now that the structure of the underlying search tree has been outlined, it is possible to describe how the cumulative returns and future returns are calculated. The accrued returns is the sum of returns the NLU-MCTS algorithm receives during the execution phase from timestep 0, t_0, to timestep $t-1$, where $\mathbf{r}_t$ is the reward vector received at each timestep,

$$\mathbf{R}_t^- = \sum_{t_0}^{t-1} \mathbf{r}_t. \tag{3.2}$$

Given the underlying planning phases of MCTS are utilised (see Section 2.1.1.3), it is possible to use the simulation phase to compute the future returns. As already mentioned, during the simulation phase the agent performs a random rollout (also known as a Monte Carlo simulation) until a terminal state is reached. Therefore, the future returns can be computed from Monte Carlo simulations performed at each node during planning. Taking this into consideration, the future returns, $\mathbf{R}_t^+$, is the sum of the rewards received when traversing the search tree during the planning phase and Monte Carlo simulations from timestep, t, to a terminal node, t_n,

$$\mathbf{R}_t^+ = \sum_t^{t_n} \mathbf{r}_t. \tag{3.3}$$

Finally, before the utility function is applied, the cumulative returns must be calculated. The cumulative returns, $\mathbf{R}_t$, is the sum of the accrued returns, $\mathbf{R}_t^-$, and the future returns, $\mathbf{R}_t^+$,

$$\mathbf{R}_t = \mathbf{R}_t^- + \mathbf{R}_t^+. \tag{3.4}$$

In other words, the cumulative returns is the returns received from a full policy execution. Once the cumulative returns, $\mathbf{R}_t$, have been calculated, it is possible to compute the utility of the returns, $u(\mathbf{R}_t)$, to optimise for the ESR criterion.

As already highlighted, NLU-MCTS builds an expectimax search tree and utilises both decision and chance nodes. Over multiple iterations of the planning phase, NLU-MCTS constructs a search tree using the selection, expansion, simulation, and backpropagation phases used by traditional MCTS [Silver et al., 2016]. The NLU-MCTS algorithm is outlined in Algorithm 3.

Firstly, NLU-MCTS utilises the selection phase (Algorithm 4, see Figure 3.2), where the agent traverses the search tree starting at the current root decision node [Shen et al., 2019]. During the selection phase, outcome selection is utilised for chance nodes and action selection is utilised for decision nodes. When the agent arrives at a chance node, outcome selection is performed where the agent simulates the environment model (Algorithm 6). The agent then moves to the child decision node corresponding to the observation-reward combination received from the simulation [Shen et al., 2019]. When the agent arrives at a decision node, n_d, the agent must decide which of its child chance nodes, C_{n_d} to select. To do so, NLU-MCTS selects the chance node n_c, which maximises the UCB term:

$$\text{bestChild} = arg\ max_{n_c \in C_{n_d}} \text{UCB}(n_d, n_c) \tag{3.5}$$

where the UCB term is defined as follows:

$$\text{UCB}(n_d, n_c) = \frac{v_{n_c}}{N_{n_c}} + C \times \sqrt{\frac{ln(N_{n_d})}{N_{n_c}}}, \tag{3.6}$$

where v_{n_c} is the total utility of the child node n_c, $\frac{v_{n_c}}{N_{n_c}}$ is the expected utility of the child node n_c, C is an exploration value, and N_{n_d} and N_{n_c} are the number of times n_d and n_c have been visited respectively. Equation 3.6 ensures that the agent explores areas of the tree that have not been visited often while also ensuring that

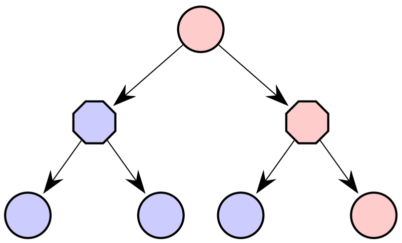


Figure 3.2: During the selection phase, NLU-MCTS starts at the root node and traverses down the search tree (nodes highlighted in red). The agent traverses the search tree until a leaf decision node is found.

the agent exploits nodes that have good returns. The agent then traverses to the chance node corresponding to the best action. The agent continues to traverse the search tree until a decision node is encountered that has not had all of its children expanded. The agent then progresses to the expansion phase (Algorithm 5) where the selected decision node is utilised. It is important to note that, as the agent traverses the search tree, the future returns, $\mathbf{R}_t^+$, is being computed incrementally.

During the expansion phase (Algorithm 5, see Figure 3.3), the agent considers a decision node selected during the previous phase that has not had all of its children expanded. There are three steps to the expansion phase. First, for the decision node, a child chance node corresponding to a previous remaining action is created for a randomly selected action. Second, the agent simulates the environment model for the newly created chance node. Finally, for the previously created chance node, the agent creates a child decision node corresponding to the observation-reward combination received. It is important to note that both a chance node and a decision node are generated during the expansion phase. The newly created decision node is then utilised in the next phase, known as the simulation phase.

After expansion, the created decision node must be simulated. Figure 3.4 highlights the simulation phase (Algorithm 7) for NLU-MCTS. When a decision node is simulated, a random rollout is executed. During the rollout, a random policy is followed until it reaches a terminal state. Once the simulation is completed, the cumulative returns, $\mathbf{R}_t$, can be computed. The future returns, $\mathbf{R}_t^+$, is equal to the sum of the rewards received when traversing the search tree and the returns from the random rollout in the simulation phase. The cumulative returns, $\mathbf{R}_t$, is then computed by adding both the accrued returns, $\mathbf{R}_t^-$, and the future returns, $\mathbf{R}_t^+$. It is important to note that $\mathbf{R}_t$ is the same for every node visited during backpropagation.

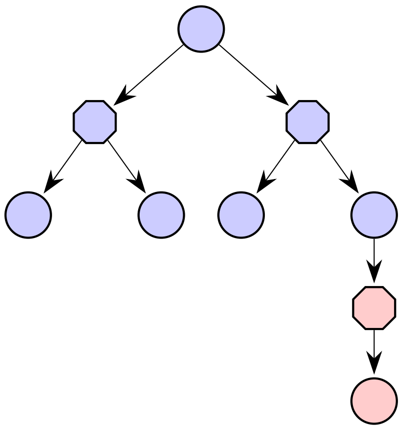


Figure 3.3: During the expansion phase of NLU-MCTS (nodes highlighted in red), a child chance node is created. The newly generated chance node simulates the environment and creates a child decision for the corresponding reward received.

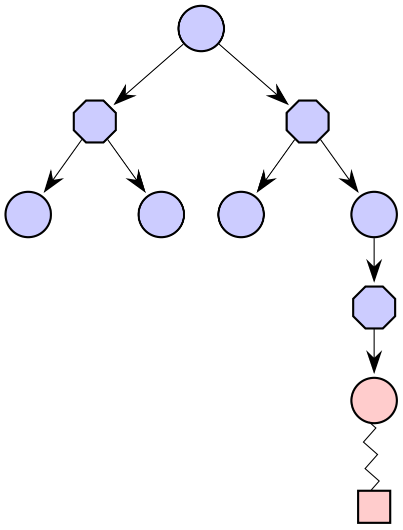


Figure 3.4: During the simulation phase of NLU-MCTS (nodes highlighted in red), the decision node generated in the expansion phase executes a random policy until a terminal state. Finally, the cumulative returns, $\mathbf{R}_t$, is computed.

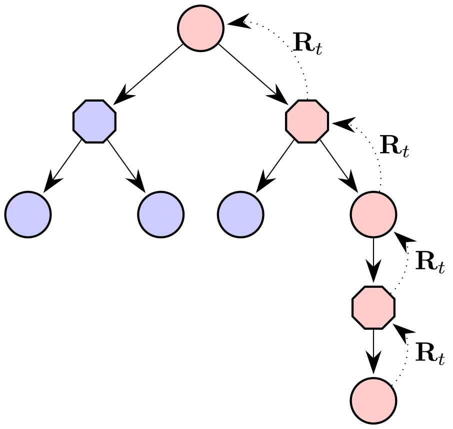


Figure 3.5: During the backpropagation phase, the cumulative returns, $\mathbf{R}_t$, is backpropagated to each node visited during the planning phase.

Figure 3.5 and Algorithm 8 outline the backpropagation phase of NLU-MCTS. Once the simulation phase is completed, the cumulative returns, $\mathbf{R}_t$, is backpropagated to each node visited during the previous phases of the search tree. As the agent backpropagates the cumulative returns, the agent updates the required statistic for each node.

Under the ESR criterion, the utility of the cumulative returns, $u(\mathbf{R}_t)$, is computed during the backpropagation phase[5] by applying the known utility function, u, to the cumulative returns, $\mathbf{R}_t$. Therefore, during backpropagation, the statistics at a chance node are updated by updating the total utility, v, of the node:

$$v_{n_c} \leftarrow v_{n_c} + u(\mathbf{R}_t). \tag{3.7}$$

The visit count for both chance nodes and decision nodes is also updated as follows:

$$N_{n_c} \leftarrow N_{n_c} + 1, \tag{3.8}$$

$$N_{n_d} \leftarrow N_{n_d} + 1. \tag{3.9}$$

The NLU-MCTS algorithm runs each step of the planning phase (selection, expansion, simulation, and backpropagation) a specified number of times. The

[5]To compute policies under the ESR criterion its is also possible to backpropagate the utility of the cumulative returns, $u(\mathbf{R}_t)$. The relevant statistics can then be updated using the utility of the cumulative returns. See Appendix A.1

number of times the planning phase is run is denoted by n_{exec}. Once the NLU-MCTS algorithm has run the planning phase an n_{exec} number of times, the algorithm returns the best action to take from the current root node, n_r. Under the ESR criterion, the best action, a^*, can be calculated by evaluating the expected utility, v, of each of the current root nodes, n_r, children, C_{n_r} and taking the action that returns the maximum expected utility:

$$a^* = \arg\max_{n \in C_{n_r}} \frac{v_n}{N_n}. \tag{3.10}$$

Using the outlined algorithm, NLU-MCTS is able to learn policies for nonlinear utility functions under the ESR criterion for multi-objective settings.

Algorithm 3: Monte Carlo Tree Search for nonlinear Utility Functions

```
1  Input : $N_{root}$ ← Root node; $\mathbf{R}_t^-$ ← Accrued returns
2  Output: Action $a$
3  while Not out of computation do
4      N ← $N_{root}$
5      $\mathbf{R}_t^+$ ← Future returns with 0 value entry per objective
6      N, $\mathbf{R}_t^+$ ← Selection(N, $\mathbf{R}_t^+$)
7      $\mathbf{R}_t^+$ ← Simulation(N, $\mathbf{R}_t^+$)
8      $\mathbf{R}_t$ ← $\mathbf{R}_t^+ + \mathbf{R}_t^-$
9      Backpropagate(N, $\mathbf{R}_t$)
10 end
11 bestAction ← calculateBestAction($N_{root}$)(Equation 3.10)
12 return bestAction
```

Algorithm 4: Selection

```
1  Input: N ← Node in the tree; $\mathbf{R}_t^+$ ← Future returns
2  if N is terminal then
3      return N, $\mathbf{R}_t^+$
4  end
5  if N is a chance node then
6      $N, \mathbf{R}_t^+$ ← Sample(N, $\mathbf{R}_t^+$)
7  end
8  if N has children to expand then
9      $N^*, \mathbf{R}_t^+$ ← Expansion($N^*, \mathbf{R}_t^+$)
10     return $N^*, \mathbf{R}_t^+$
11 end
12 $N, \mathbf{R}_t^+$ ← BestChild
13 Selection(N, $\mathbf{R}_t^+$)
```

Algorithm 5: Expansion

```
1  Input : N ← Node in the tree; $\mathbf{R}_t^+$ ← Future returns
2  $N^*$ ← a new child chance node for a remaining action
3  add $N^*$ to N's children
4  $N^*, \mathbf{R}_t^+$ ← Sample($N^*, \mathbf{R}_t^+$)
5  return $N^*$, $\mathbf{R}_t^+$
```

Algorithm 6: Sample

```
1  Input: N ← Chance Node; $\mathbf{R}_t^+$ ← Future returns
2  observation, reward ← simulate agent environment for N
3  $\mathbf{R}_t^+ \leftarrow \mathbf{R}_t^+ +$ reward
4  for DN in N.Children do
5      if (DN.observation, DN.reward) = (observation, reward) then
6          return DN, $\mathbf{R}_t^+$
7      end
8  end
9  DN ← N create child decision node (observation, reward)
10 return DN, $\mathbf{R}_t^+$
```

Algorithm 7: Simulation

```
1 Input : N ← Node in the tree; s ← Node.state; $\mathbf{R}_t^+$ ← future returns
2 while s is not terminal do
3     a ← random action
4     s, $\mathbf{r}_t$ ← env(s, a)
5     $\mathbf{R}_t^+ \leftarrow \mathbf{R}_t^+ + \mathbf{r}_t$
6 end
7 return $\mathbf{R}_t^+$
```

Algorithm 8: Backpropagate

```
1 Input : N ← Node in the tree; $\mathbf{R}_t$ ← Cumulative returns
2 while N is not null do
3     UpdateStatistics(N, $\mathbf{R}_t$)
4     N ← N.parent
5 end
```

3.3 Distributional Monte Carlo Tree Search

NLU-MCTS utilises the UCB statistic to explore during planning. However, Thompson sampling (TS) methods have been shown to outperform UCB methods in bandit settings [Russo and Van Roy, 2014; Chapelle and Li, 2011]. Therefore, to exploit the potential performance increases associated with TS methods, a novel algorithm is presented, called *distributional Monte Carlo tree search (DMCTS)*, which learns an approximate posterior distribution over the expected utility of the returns.

Before the DMCTS algorithm is outlined, it is important to discuss the underlying methods DMCTS utilises to construct a search tree. DMCTS builds an expectimax search tree using the same planning phase as NLU-MCTS (see Section 3.2). However, DMCTS takes a distributional approach to decision making.

DMCTS aims to maintain a posterior distribution over the expected utility of the returns at each chance node. However, because the utility function may be nonlinear, a parametric form of the posterior distribution may not exist. Since a bootstrap distribution can be used to approximate a posterior [Efron, 2012; Newton and Raftery, 1994], it is much more suitable to maintain a bootstrap distribution over the expected utility of the returns at each chance node.

Algorithm 9: Distributional Monte Carlo Tree Search

```
1  **Input** : $\mathrm{N}_{root} \leftarrow$ Root node; $\mathbf{R}_t^- \leftarrow$ Accrued returns
2  **Output**: Action $a$
3  while Not out of computation do
4      $\mathrm{N} \leftarrow \mathrm{N}_{root}$
5      $\mathbf{R}_t^+ \leftarrow$ Future returns with 0 value entry per objective
6      N, $\mathbf{R}_t^+ \leftarrow$ Selection$(\mathrm{N}, \mathbf{R}_t^+)$
7      $\mathbf{R}_t^+ \leftarrow$ Simulation$(\mathrm{N}, \mathbf{R}_t^+)$
8      $\mathbf{R}_t \leftarrow \mathbf{R}_t^+ + \mathbf{R}_t^-$
9      Backpropagate$(\mathrm{N}, \mathbf{R}_t)$
10 end
11 bestAction $\leftarrow$ calculateBestAction$(\mathrm{N}_{root})$
12 return bestAction
```

Each bootstrap distribution contains a number of bootstrap replicates, $j \in \{1, ..., J\}$ [Eckles and Kaptein, 2014] (see Section 2.1.2.3). It is important to note the number of bootstrap replicates, J, is a hyperparameter that can be tuned for exploration [Eckles and Kaptein, 2014]. Each bootstrap replicate, j, in the bootstrap distribution has two parameters, α_j[6] and β_j, where $\frac{\alpha_j}{\beta_j}$ is the expected utility for replicate j. On initialisation of a new node, for each bootstrap replicate, j, the parameters α_j and β_j are both set to 1. Moreover, α_j can be set to positive or negative values to increase initial exploration without a computational cost. Figure 3.6 outlines a bootstrap distribution learned by the DMCTS algorithm. For ESR settings, the expected utility of each bootstrap replicate, j, can be computed as follows:

$$\mathbb{E}(u(j)) = \frac{\alpha_j}{\beta_j}. \tag{3.11}$$

It is important to note that, similarly to NLU-MCTS, DMCTS requires the utility function of the user to be known a priori. The bootstrap distribution is updated during the backpropagation phase of the DMCTS algorithm.

During the backpropagation phase, the bootstrap distribution at each chance node is updated. Algorithm 12 outlines how a bootstrap distribution for a node is updated for the ESR criterion. At chance node, i, for each bootstrap replicate, j, a coin flip is simulated (See Algorithm 12, Line 4). If the result of the coin flip is

[6] In this work our use of α differs slightly from that of Eckles and Kaptein [2014]. The parameter α is utilised to track the sum of the utility, which can then be utilised to compute the expectation. Whereas, Eckles and Kaptein [2014] utilise α as a count for the returns of a Bernoulli bandit.

Algorithm 10: Selection

```
1  Input: N ← Node in the tree; $\mathbf{R}_t^+$ ← Future returns
2  if N is terminal then
3      return N, $\mathbf{R}_t^+$
4  end
5  if N is a chance node then
6      N, $\mathbf{R}_t^+$ ← Sample(N, $\mathbf{R}_t^+$)
7  end
8  if N has children to expand then
9      N*, $\mathbf{R}_t^+$ ← Expansion(N*, $\mathbf{R}_t^+$)
10     return N*, $\mathbf{R}_t^+$
11 end
12 N, $\mathbf{R}_t^+$ ← ThompsonSampling
13 Selection(N, $\mathbf{R}_t^+$)
```

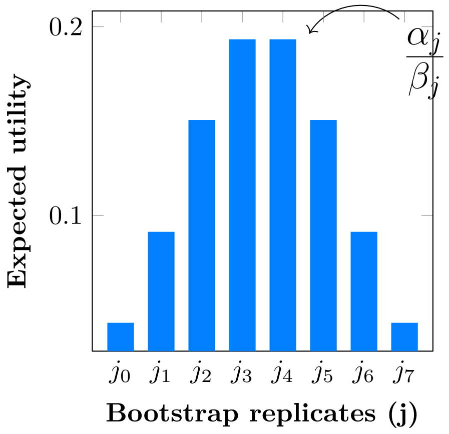


Figure 3.6: A bootstrap distribution learned by DMCTS with the number of bootstrap replicates, J, set to 8. The expected utility for each bootstrap replicate, j, can be calculated by $\frac{\alpha_j}{\beta_j}$. For example, the expected utility for bootstrap replicate j_4 can be calculated as follows: $\mathbb{E}(u(j_4)) = \frac{\alpha_{j_4}}{\beta_{j_4}}$.

Algorithm 11: Backpropagate

```
1 Input : N ← Node in the tree
2 Input : $\mathbf{R}_t$ ← Cumulative returns
3 while N is not null do
4     UpdateDistribution(N, $\mathbf{R}_t$)
5     N ← N.parent
6 end
```

equal to 1 (heads), α_{ij} and β_{ij} are updated:

$$\alpha_{ij} \leftarrow \alpha_{ij} + u(\mathbf{R}_t) \tag{3.12}$$

$$\beta_{ij} \leftarrow \beta_{ij} + 1 \tag{3.13}$$

To select actions while planning, the previously computed statistics are utilised. At each timestep the agent must choose which action to execute in order to traverse the search tree (as outlined in Algorithm 13). At decision node n, an action is selected by sampling the bootstrap distribution at each child chance node, i. For each sampled bootstrap replicate, j, the α_{ij} and β_{ij} values are retrieved and $\frac{\alpha_{ij}}{\beta_{ij}}$ is computed. Since the following approximation is true,

$$\frac{\alpha_{ij}}{\beta_{ij}} \equiv \mathbb{E}[u(\mathbf{R}_t^- + \mathbf{R}_t^+)], \tag{3.14}$$

by maximising over i in Equation 3.14, an action is selected corresponding to j approximately proportional to the probability of that action being optimal (as per the BTS exploration strategy). The agent then executes the action, a^*, which corresponds to the following:

$$a^* = \arg\max_i \frac{\alpha_{ij}}{\beta_{ij}}. \tag{3.15}$$

At execution time, we can calculate the best action (Algorithm 9 Line 11) by simply selecting the overall maximising action by averaging over all the acquired data, thereby maximising the ESR criterion:

$$ESR = \mathbb{E}[u(\mathbf{R}_t^- + \mathbf{R}_t^+)]. \tag{3.16}$$

Using the outlined algorithm, DMCTS is able to learn policies for nonlinear utility functions under the ESR criterion for multi-objective settings.

Algorithm 12: UpdateDistribution

```
1 Input: i ← Node in the tree; $\mathbf{R}_t$ ← Cumulative returns
2 J ← node.bootstrapDistribution
3 for j, ..., J bootstrap replicates do
4     Sample $d_j$ from Bernoulli($\frac{1}{2}$)
5     if $d_j$ = 1 then
6         $\alpha_{ij} = \alpha_{ij} + u(\mathbf{R}_t)$
7         $\beta_{ij} = \beta_{ij} + 1$
8     end
9 end
```

Algorithm 13: ThompsonSample

```
1 Input: n ← Node in the tree
2 Require: $\alpha$, $\beta$ prior parameters
3 $\alpha_{ij} := \alpha$, $\beta_{ij} := \beta$ {For each n child, $i$, and each bootstrap replicate, $j$ }
4 for i, ..., n children do
5     Sample $j$ from uniform 1, ..., $J$ bootstrap replicates
6     Retrieve $\alpha_{ij}$, $\beta_{ij}$
7 end
8 maxChild = $\arg\max_i \frac{\alpha_{ij}}{\beta_{ij}}$
9 return maxChild or maxChild.action
```

3.4 Empirical Evaluation

In order to evaluate NLU-MCTS and DMCTS, both algorithms are tested in multiple multi-objective settings. First, an ablative study is performed to outline the effect on computation and performance the J parameter has when computing the BTS distribution for DMCTS. Next, NLU-MCTS and DMCTS are evaluated in multi-objective settings under the ESR criterion for nonlinear utility functions. Both NLU-MCTS and DMCTS are evaluated against state-of-the-art algorithms using variants of standard benchmark problems from the MODeM literature.

NLU-MCTS and DMCTS are evaluated against two other state-of-the-art RL algorithms: expected utility policy gradient (EUPG) [Roijers et al., 2018b] and categorical deep Q-networks (C51) [Bellemare et al., 2017]. EUPG is the only MODeM algorithm that can compute policies under the ESR criterion and therefore has achieved state-of-the-art performance in this setting [Roijers et al., 2018b]. The C51 algorithm is used as a baseline algorithm during experimentation because C51 is a distributional RL algorithm and has achieved state-of-the-art performance in single-objective settings [Bellemare et al., 2017].

At each timestep for NLU-MCTS and DMCTS, the planning phase is performed multiple times before an action is selected during the execution phase. It is important to note that NLU-MCTS and DMCTS are model-based algorithms, while C51 and EUPG are model-free algorithms. To fairly evaluate all other algorithms against NLU-MCTS and DMCTS, each benchmark algorithm has been altered to have the same number of policy executions of each environment at each timestep as NLU-MCTS and DMCTS. At each timestep, each algorithm gets n_{exec} full policy executions worth of learning from that state and timestep onward. Therefore, if $n_{exec} = 10$, NLU-MCTS and DMCTS perform the planning phase ten times before selecting an action. To ensure C51 and EUPG get the same opportunity to learn, both algorithms are altered to execute a policy n_{exec} number of times from the current state. For the other algorithms (except NLU-MCTS and DMCTS), this has the effect of increasing the learning speed. The number of policy executions n_{exec} varies for each problem domain. Finally, all experiments are averaged over 10 runs.

3.4.1 Ablation Study

Before both NLU-MCTS and DMCTS are evaluated using multi-objective sequential decision making problems, the BTS parameter settings are empirically evaluated to determine their effect on performance and run time. An example is also provided which visualises how a BTS distribution is updated over time to estimate the underlying posterior distribution over the expected utility. Finally, the performance of DMCTS is evaluated under different J values in a MOMDP to highlight how the selection of the J value can effect performance in sequential decision making settings.

3.4.1.1 Bootstrap Thompson Sampling J Value & Runtime

To illustrate how a BTS distribution evolves over time, a single BTS distribution is updated based on the returns of a simple multi-objective bandit. In this setting the bandit has one arm, where there is a 0.5 chance of receiving the following return: $\mathbf{r} = [1, 1]$, and a 0.5 chance of receiving the following return: $\mathbf{r} = [0, 0]$. The returns are then scalarised using the following nonlinear utility function:

$$u = r_1 r_2, \tag{3.17}$$

where r_1 and r_2 are the returns for objective 1 and objective 2 respectively. In this example, the expected utility is 0.5.

Using this bandit, a single BTS distribution is updated over time by utilising multiple update steps at each timestep. Figure 3.7 outlines how a BTS distribution with 100 bootstrap replicates evolves after $1, 8, 32, 128, 250$ & 500 updates. After 500, updates the BTS distribution moves close to the optimal expected utility of 0.5. As a result BTS distributions can learn the optimal expected utility in the outlined setting. Figure 3.7 outlines similar results to those reported by Eckles and Kaptein [2019].

Next, the computational run time for a BTS distribution with a varying number of replicates, J, is investigated. To evaluate the run time for each chosen J value, the time in seconds taken to perform $1,000$ updates of a BTS distribution is computed. This experiment was performed 10 times for each J value and the average run time was computed. To evaluate the run time, the following J values were used: $10, 100, 200, 300, 400, 500, 600, 700, 800, 900$ & $1,000$.

Figure 3.8 shows that the run time in seconds increases linearly with the number of replicates J. Therefore, the hyperparameter J can have an impact on the run time of the algorithm and therefore should be taken into consideration in order to optimise performance. Next, the performance of a BTS distribution for multiple J values in a MOMDP will be evaluated. By comparing run time and performance, it should be possible to determine which J values can be selected for good performance and efficiency.

3.4.1.2 Random Multi-Objective Markov Decision Process

To evaluate the impact the selection of the J parameter for the BTS distribution has on the performance of DMCTS, various different J values are used in a MOMDP.

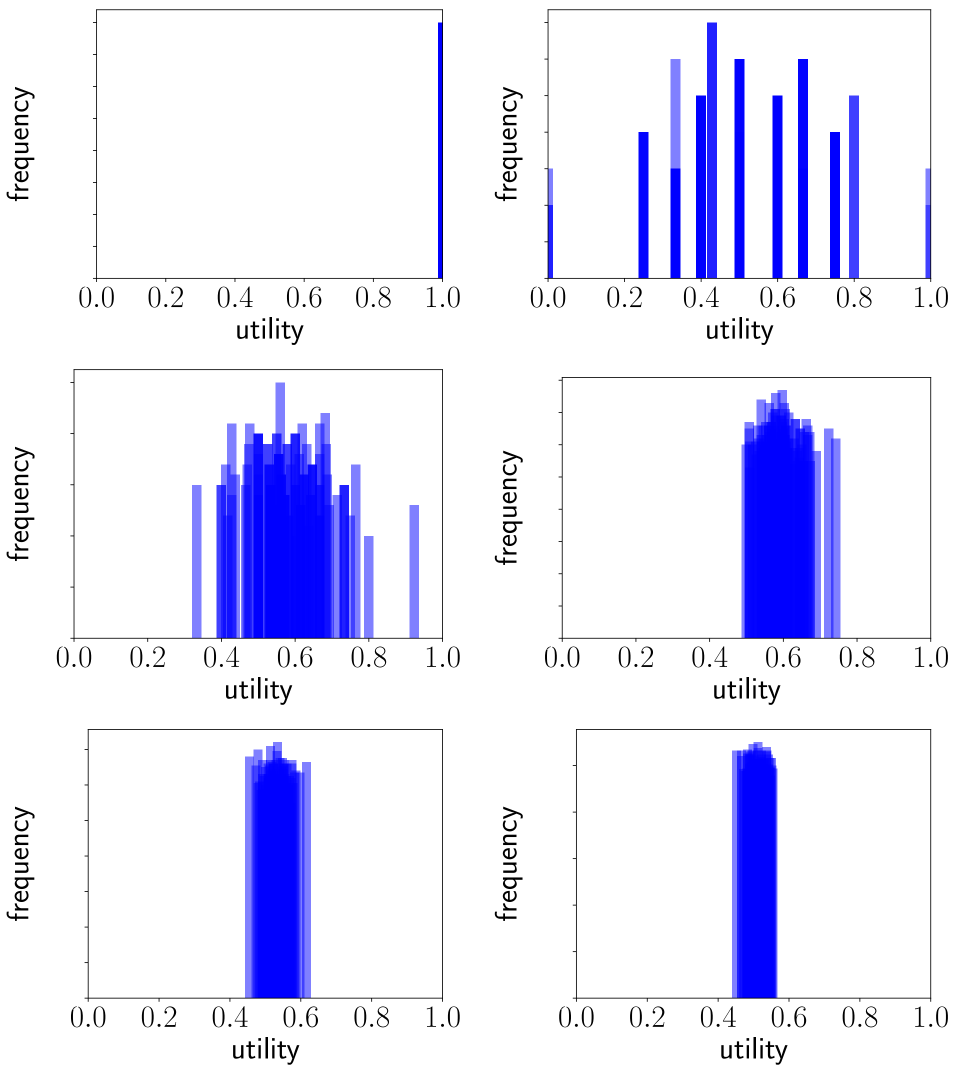


Figure 3.7: A BTS distribution after 1, 8, 32, 128, 250 & 500 updates. After 500 updates the distribution converges to the correct expected utility, where expected utility is on the x-axis.

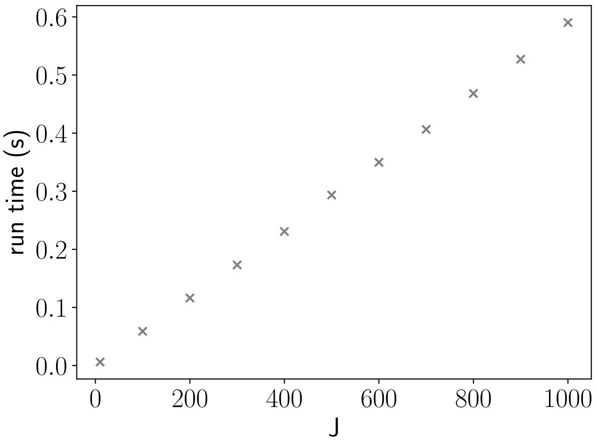


Figure 3.8: The run time is seconds required to complete 1,000 updates of a BTS distribution for different J values. The run time required increases linearly with the increase in the J value.

To do so, a random MOMDP from the literature [Roijers et al., 2018b] is utilised. The random MOMDP is configurable based on the requirements of the experiments, where the number of states, actions, objectives, timesteps, and possible successor states can be determined a priori. The random MOMDP can then be initialised for each experiment by selecting a consistent seed. For experimentation a random MOMDP with 20 states, 2 actions, and 2 objectives is generated. The transition function $T(s, a, s')$ is generated using $N = 8$ possible successor states per action, with random probabilities drawn from a uniform distribution [Roijers et al., 2018b]. The following nonlinear utility function is used for evaluation:

$$u = r_1^2 + r_2^2. \tag{3.18}$$

DMCTS is evaluated using the following J values of $1, 2, 10, 100, 500$ and $1,000$ for the BTS distributions. Figure 3.9 outlines the results from the random MOMDP. Utilising a J value of 1 has an impact on performance, given DMCTS with J set to 1 achieves a lower utility compared to the other parameter settings. As the J value increases to 2, it is clear that performance begins to improve. However, for a very low J value ($J = 1$ or $J = 2$) DMCTS will select actions greedily and will not explore the environment enough to obtain a good utility. As the J value increases even further, the performance also increases. Once the J value is set to 10, DMCTS has a large increase in performance. Similarly, once the J value increases to 100, a

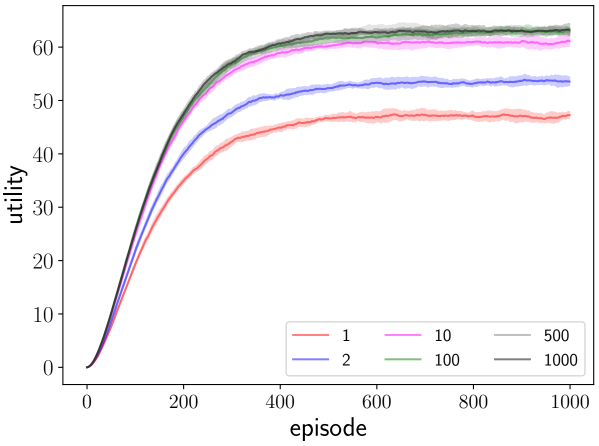


Figure 3.9: Evaluation of different J values in a random MOMDP with 20 states, 2 actions, 2 objectives and 8 successor states reachable from each state.

further improvement can be seen. The performance increases plateau for DMCTS after a certain J value for the random MOMDP. When the J value is set to 500 or $1,000$, the performance does not increase relative to $J = 100$. However, the computational cost of updating a BTS distribution scales linearly with the number of J values. Therefore, there is a trade-off between performance and computational cost. Moreover, it is important to ensure that the J value is set sufficiently high for exploration, while also avoiding J values with a high computational cost. As a result, it may be important to tune the J value depending on the evaluation setting.

3.4.2 Evaluation using Multi-objective Markov Decision Processes

To evaluate NLU-MCTS and DMCTS in multi-objective settings under the ESR criterion, two problem domains are considered. First, NLU-MCTS and DMCTS are evaluated in the Fishwood problem [Roijers et al., 2018b], given this is one of the very few domains for which ESR results have been published. Second, NLU-MCTS and DMCTS are evaluated using the newly proposed Renewable Energy Dynamic

Economic Emissions Dispatch (REDEED) problem domain[7]. Finally, DMCTS is evaluated using a number of nonlinear utility functions to highlight the flexibility of the DMCTS algorithm.

3.4.2.1 Fishwood

Fishwood is a multi-objective benchmark problem proposed by Roijers et al. [2018b]. In Fishwood, the agent has two states: in the woods or at the river. The goal of the agent is to catch fish and collect wood. The Fishwood environment is parameterised by the probabilities of successfully obtaining fish and wood at these respective states. The following parameters are used: 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 nonlinear utility function:

$$u = \min\left(\text{fish}, \left\lfloor \frac{\text{wood}}{2} \right\rfloor\right). \tag{3.19}$$

As demonstrated by Roijers et al. [2018b], to maximise utility in the Fishwood problem, 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 [Roijers et al., 2018b].

Both NLU-MCTS and DMCTS are evaluated in the Fishwood domain against EUPG [Roijers et al., 2018b] and C51 [Bellemare et al., 2017]. EUPG achieves state-of-the-art results in the Fishwood problem under the ESR criterion [Roijers et al., 2018b]. C51 [Bellemare et al., 2017] is a distributional RL algorithm that achieved state-of-the-art results in the Atari game problem domain.

For C51, the learning hyperparameters are set as follows: $V_{min} = 0$, $V_{max} = 2$, $\epsilon = 0.01$, $\gamma = 1$ and $\alpha = 0.0001$. For DMCTS the number of bootstrap replicates, J, in the bootstrap distribution is set as follows: $J = 100$. NLU-MCTS has the following hyperparameters: $C = \sqrt{2}$. EUPG is conditioned on the accrued returns and the current timestep, t. Each experiment runs for $10,000$ episodes, where each episode has 13 timesteps, and $n_{exec} = 2$.

As shown in Figure 3.10, the utility for C51 fluctuates throughout experimentation and it fails to learn a consistent policy. Given C51 does not take the accrued

[7]It is important note, in Section 3.4 NLU-MCTS and DMCTS (both model-based algorithms) are evaluated against a number of model-free algorithms. Therefore, to fairly evaluate model-based and model-free approaches, both NLU-MCTS and DMCTS maintain the search tree across episodes. This has the effect that both algorithms require less simulations during experimentation.

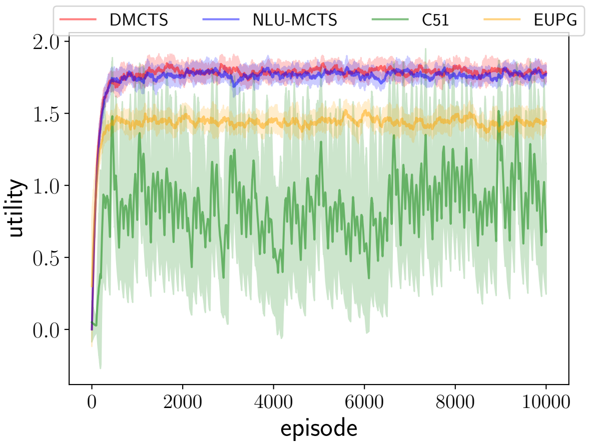


Figure 3.10: Results from the Fishwood environment where DMCTS achieves state-of-the-art performance in a multi-objective setting over EUPG.

returns into consideration during learning, the utility function is applied directly to the reward received by the agent. The reward received by an agent in the Fishwood domain can be $[0, 1]$ or $[1, 0]$. By applying the utility function (presented in Equation 3.19) to the reward, the C51 agent can only receive a utility of 0. DMCTS, NLU-MCTS, and EUPG all take the accrued and future returns into consideration and can compute better policies for nonlinear utility functions when compared to C51. It is important to note that C51 is not explicitly designed for nonlinear utility functions. C51 relies on the distributional Bellman operator, which assumes additive returns. However, it is expected if the utility function were linear, that the performance of C51 would improve.

DMCTS and NLU-MCTS achieve a higher utility when compared to EUPG. All algorithms, except C51, use Monte Carlo simulations of the environment and optimise over the expected utility of the returns of a full episode. Although EUPG uses Monte Carlo simulations of the environment, policy gradient algorithms are sample inefficient. DMCTS and NLU-MCTS are sample efficient, given both algorithms utilise the planning phase of MCTS, which has been shown to be sample efficient [Abramson, 1987; Chang et al., 2005].

Both DMCTS and NLU-MCTS learn good policies with respect to the specified utility function. Given the utility function aims to collect sufficient wood to cook fish and feed the agent, it can be said that both algorithms learn a behaviour which reflects the predefined preferences over the objectives which are encoded in the utility function. Therefore, compared to EUPG and C51, both DMCTS and NLU-MCTS compute policies that are more effective at switching states at an appropriate time to collect the right balance of fish and wood to ensure their utility is maximised.

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. Both DMCTS and NLU-MCTS build a search tree, which enables the agent to re-sample the environment at each chance node during planning. However, DMCTS achieves a higher overall utility when compared with NLU-MCTS despite both algorithms utilising repeated sampling at each chance node and Monte Carlo simulations.

3.4.2.2 Renewable Energy Dynamic Economic Emissions Dispatch

Next, NLU-MCTS and DMCTS are evaluated in a complex problem domain with a large state action space. Renewable Energy Dynamic Economic Emissions Dispatch (REDEED) is a variation of the traditional Dynamic Economic Emissions Dispatch (DEED) problem [Basu, 2008]. 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 [2017] and the derived values are used for the both the fossil fuel generators and the renewable energy generator (see Table A.1). In this example, Generator 3 is controlled by an agent, Generator 1 is a slack generator and Generator 4 is replaced by a wind turbine.

This setting covers a period of 24 hours and for each hour a weather forecast is received for a city. For hours $1 - 15$, the weather is predictable and the optimal power values derived by Mannion [2017] 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 the aim is to compute a policy that can ensure the required power is met over the entire day while reducing both the cost, emissions and penalty violations created by all generators.

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

$$R_+ = \prod_{o=1}^{O} f_o, \tag{3.20}$$

where f_o is the objective function for each objective, $o \in O$ [Mannion et al., 2018; Hayes et al., 2020].

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

$$f_c^L(n,m) = a_n + b_n P_{nm} + c_n(P_{nm})^2 + |d_n sin\{e_n(P_n^{min} - P_{nm})\}|. \tag{3.21}$$

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

$$f_c^G(m) = \sum_{n=1}^{N} f_c^L(n,m). \tag{3.22}$$

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

$$f_e^L(n,m) = E(a_n + b_n P_{nm} + \gamma_n(P_{nm})^2 + \eta \exp \delta P_{nm}). \tag{3.23}$$

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

$$f_e^G(m) = \sum_{n=1}^{N} f_e^L(n,m). \tag{3.24}$$

It is important to note the emissions for the wind turbine are 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,

$$f_p^G(m) = \sum_{v=1}^{V} C(|h_v + 1|\delta_v). \tag{3.25}$$

Along with cost and emissions, the penalty function is an additional objective that will need to be optimised. Some parameters for this problem domain have not been included here; all equations and parameters absent from this description that are required to implement this problem domain can be found in the Appendix A.2.

To evaluate NLU-MCTS and DMCTS in the REDEED domain, EUPG and C51 are again used for comparison. For DMCTS the number of bootstrap replicates, J,

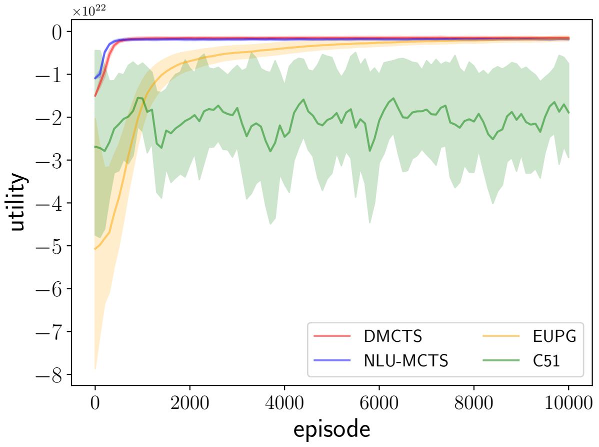


Figure 3.11: Results from the REDEED environment DMCTS outperforms EUPG, C51 and NLU-MCTS. DMCTS achieves a higher utility compared to other algorithms used throughout experimentation in the REDEED domain under the ESR criterion.

for the bootstrap distribution is set as follows: $J = 100$. For NLU-MCTS $C = \sqrt{2}$. For C51 the learning hyperparameters are set as follows: $V_{min} = -8e^{22}$, $V_{max} = 0$, $\epsilon = 0.01$, $\gamma = 1$ and $\alpha = 0.0001$. For the REDEED problem the agent learns for $10,000$ episodes and $n_{exec} = 2$ for each algorithm.

As seen in Figure 3.11, DMCTS outperforms EUPG, NLU-MCTS, and C51 in the REDEED domain. C51 struggles to learn a consistent policy and C51's utility fluctuates throughout experimentation. The hyperparameters chosen for C51 provide good performance but are difficult to tune. Although the learning speed of EUPG is slow, EUPG achieves a higher utility than C51.

Both DMCTS and NLU-MCTS learn good policies faster than EUPG. MCTS algorithms are much more sample efficient when compared to policy gradient algorithms like EUPG. Figure 3.11 highlights the difference in sample efficiency of DMCTS, NLU-MCTS and EUPG given the differences in the number episodes required for each of the aforementioned algorithms to compute stable policies for the defined nonlinear utility function.

DMCTS, NLU-MCTS and EUPG all compute stable policies. However, DMCTS achieves a higher utility when compared to NLU-MCTS and EUPG. DMCTS converges to a policy with an average utility of -1.54×10^{21}. In comparison, NLU-MCTS converges to a policy with an average utility of -1.80×10^{21}, while EUPG

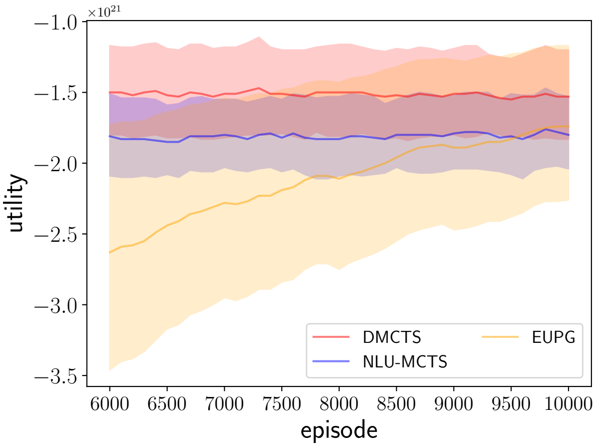


Figure 3.12: Results from the final 4,000 episodes of the REDEED environment to highlight how DMCTS outperforms NLU-MCTS, EUPG, and C51.

converges to a stable policy with an average utility of -1.75×10^{21}. Given the scale of the utility computed throughout REDEED experimentation, it is difficult to see the final difference in utility in Figure 3.11. Therefore, to highlight the difference in utility between DMCTS, NLU-MCTS, and EUPG the final 4,000 episodes have been plotted in Figure 3.12. It is important to note, given C51 has performed poorly in the REDEED domain, C51 has not been included in Figure 3.12. For the highlighted episodes in Figure 3.12, it is clear that DMCTS achieves a higher utility when compared to both NLU-MCTS and EUPG.

The REDEED environment has a large state-action space with stochastic returns. Although C51 has achieved state-of-the-art results in the Atari environment Bellemare et al. [2017], C51 fails to learn any meaningful policy for REDEED. A potential reason for such 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 categorical distribution. In the work presented by Bellemare et al. [2017] the number of bins is set to 51. While this provides good performance in certain problem domains, Bellemare et al. [2017] highlight that increasing the number of bins may lead to increased performance. However, to remain consistant with the literature the number of bins is fixed to 51, given the potential added performance when increasing the number of bins has not been thoroughly explored. The results presented for C51 show this parameter setting is sub-optimal in scenarios where the returns are not simple scalars over small ranges. The results present in Figure 3.11 show that C51 struggles to scale to large problem domains with complex returns over large ranges.

3.4.2.3 Nonlinear Utility Functions

During experimentation, DMCTS has been evaluated using utility functions for each experimental benchmark that were previously defined in the literature. To show that DMCTS can learn good policies for a wide range of nonlinear utility functions, DMCTS is evaluated in the Fishwood problem domain using the following four nonlinear utility functions under the ESR criterion:

$$u_1 = max(\frac{r_1}{2}, \frac{r_2}{2}),$$

$$u_2 = \frac{r_1}{2} + r_2^4,$$

$$u_3 = min(\frac{r_1}{2}, \frac{r_2}{4}),$$

$$u_4 = r_1^2 + r_2^2,$$

where r_1 is the returns received for the fish objective and r_2 is the returns received for the wood objective.

For this demonstration, $n_{exec} = 2$ and each experiment lasts $10,000$ episodes. For DMCTS the number of bootstrap replicates, J, for the bootstrap distribution is set as follows: $J = 100$.

In Figure 3.13, for each utility function the utility has been scaled between 0 and 1. For the scaled utility, 1 represents the maximum utility and 0 represents the minimum utility obtained by DMCTS. The utility has been scaled to show the performance of DMCTS for each utility function on a single plot.

Figure 3.13 outlines the performance of DMCTS when optimising for each nonlinear utility function. It is clear from Figure 3.13 that DMCTS converges to a good policy for each utility function. Therefore, DMCTS can learn a good policy for many forms of nonlinear utility functions and is not limited to the utility functions associated with predefined benchmark problems. The ability of DMCTS to learn a good policy for a range of nonlinear utility showcases how DMCTS could potentially be used in real-world scenarios, where different decision makers may have very different nonlinear utility functions for the same problem [Hayes et al., 2022c].

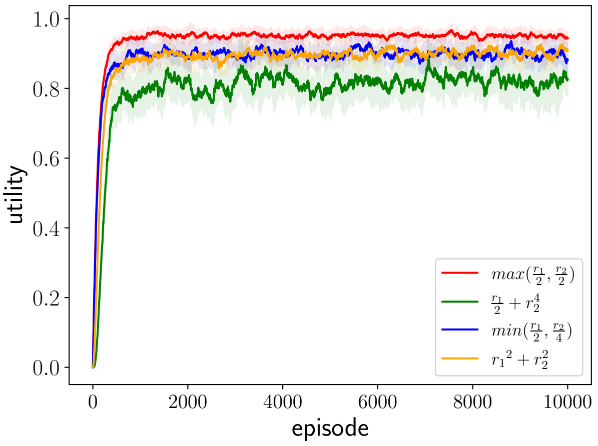


Figure 3.13: Results from the Fishwood environment where DMCTS is evaluated against multiple nonlinear utility functions.

3.4.2.4 Discussion

In multi-objective settings both NLU-MCTS and DMCTS compute good policies for the specified nonlinear utility functions. Applying the utility function to the cumulative returns (rather than just the expected future return) ensures that both NLU-MCTS and DMCTS can compute good policies. It is clear from the performance of C51 that applying the utility function to the cumulative returns is crucial for good performance in multi-objective settings when the utility function is nonlinear.

As previously highlighted, the difference between NLU-MCTS and DMCTS is the method used to explore during planning. NLU-MCTS uses UCB to determine which action to take during planning. UCB selects actions deterministically based on the expected utility and an exploration bonus [Russo and Van Roy, 2014; Auer, 2002]. In contrast, DMCTS selects actions using TS, which stochastically samples from the underlying approximate posterior distribution (BTS distribution) and selects the action proportional to the probability of the action being optimal [Chapelle and Li, 2011; Eckles and Kaptein, 2014]. In the bandit literature, TS has been shown to empirically outperform UCB [Chapelle and Li, 2011; Russo et al., 2018; Russo and Van Roy, 2014]. MCTS methods utilise independent nodes, therefore it is possible to consider each node itself to be a bandit. In this case, it is expected that TS methods will outperform UCB in sequential settings. The results presented for sequential multi-objective settings are consistent with prior bandit literature that suggests that TS can outperform UCB [Chapelle and Li, 2011].

Additionally, UCB makes highly pessimistic assumptions regarding the underlying reward/return distributions in order to guarantee a bound on the regret of its action selection procedure [Russo and Van Roy, 2014]. For known parametric distributions, tighter bounds have been proven using tighter upper confidence bounds (e.g. for Gaussian reward distributions [Russo and Van Roy, 2014]). However, doing something similar in the setting explored during experimentation isn't opportune because, even if the return distributions are nicely parametric, the nonlinear transformation resulting from the application of the utility function would no longer allow for a closed-form distribution [Russo and Van Roy, 2014]. As such, this will limit the assumptions to be highly pessimistic and, therefore, will result in sub-optimal performance. Bootstrap distributions and the resulting BTS algorithm for action selection is able to approximate and effectively exploit knowledge about the utility distributions, regardless of the shape of this underlying distribution.

3.5 Related Work

Many MCTS methods have been developed for situations involving reward uncertainty. For example, Tesauro et al. [2010] take a Bayesian approach to UCT with Gaussian approximation. Their method backpropagates probability distributions over rewards. To select actions, Tesauro et al. [2010] use UCB without taking the distributions into consideration. Cazenave and Saffidine [2010] define a MCTS algorithm that takes into account the bounds on the possible values of a node to select nodes for exploration. They apply their algorithm to problems that have more than two outcomes and show that taking the bounds into consideration can increase performance. Kaufmann and Koolen [2017] and Huang et al. [2017] also have developed MCTS algorithms that can compute policies for settings with reward uncertainty. Additionally, Bai et al. [2014] extend MCTS to maintain a distribution at each node using TS as an exploration strategy. In their work, Bai et al. [2014] do not compute a posterior distribution over the expected utility of the returns or apply their work to multi-objective settings, nor do they incorporate the accrued returns as part of their algorithm.

Furthermore, Zhang et al. [2021] compute a multi-variate distribution over the returns for RL settings. While this work considers reward vectors, it is likely that this algorithm will suffer from similar limitations to those of traditional RL algorithms when applied to nonlinear utility functions. The method proposed by Zhang et al. [2021] does not take the accrued returns into consideration. Similarly to C51, their approach may fail to achieve a high utility [Roijers et al., 2018b].

However, this method could be an interesting starting point for developing model-free multi-objective distributional RL algorithms.

It is also important to note the C51 algorithm proposed by Bellemare et al. [2017] achieves state-of-the-art performance in single-objective settings and learns a distribution over the future returns. Abdolmaleki et al. [2020] learn a distribution over actions based on constraints set per objective. This approach ignores the utility-based approach [Roijers et al., 2013] 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. [2020] claim setting the constraints for this algorithm is a more intuitive approach when compared to setting weights for a linear utility function. It may be the case that, if the user's utility function is nonlinear, this approach would fail to learn a coverage set.

3.6 Summary

This chapter outlines the first contributions of this thesis. First, an example is used to demonstrate the implications of using nonlinear utility functions to compute policies for the SER criterion and the ESR criterion in single-agent settings. This investigation shows that in single-agent settings for nonlinear utility functions, the policies computed under the SER criterion and ESR criterion can be different. Therefore, algorithms that can compute policies explicitly for the ESR criterion must be developed. These findings are similar to the results shown by Rădulescu et al. [2020] for multi-agent settings.

Second, two novel multi-objective MCTS algorithms are proposed that can compute policies for nonlinear utility functions by taking the accrued and future returns into consideration. Both algorithms aim to maximise expected utility and therefore optimise for the ESR criterion.

Finally, both algorithms are empirically evaluated. An ablative study is used to investigate the optimal hyperparameter settings for the DMCTS algorithm. Both NLU-MCTS and DMCTS are then evaluated in benchmark MOMDPs alongside state-of-the-art algorithms. Both NLU-MCTS and DMCTS compute good policies for multiple nonlinear utility functions for the ESR criterion. DMCTS achieves state-of-the-art performance for the ESR criterion.

4 Theory for Unknown Utility Functions[1]

Chapter 3 demonstrates that the policies computed for known nonlinear utility functions can be different under the scalarised expected returns (SER) criterion and the expected scalarised returns (ESR) criterion. However, in the real world many scenarios exist where a user's utility function may be unknown a priori. In this scenario, a set of optimal policies must be computed and returned to the user [Roijers et al., 2013; Hayes et al., 2022c]. The majority of multi-policy methods compute sets of optimal policies by using expected value vectors to determine optimality. Furthermore, the existing multi-policy methods only compute sets of optimal policies for the SER criterion. How to determine a partial ordering over policies for the ESR criterion has yet to be determined and, as a result, a set of optimal policies for the ESR criterion has yet to be defined. Therefore, multi-policy methods have not been explored in the multi-objective decision making (MODeM) literature for the ESR criterion.

Chapter 4 investigates if expected value vector methods, which are used under the SER criterion, can be used to determine a partial ordering over policies for the ESR criterion. Based on the findings of this investigation, Chapter 4 also aims to define a dominance relation to compute sets of optimal policies for the ESR criterion. The contributions of Chapter 4 are as follows:

[1]The contributions presented in Chapter 4 are published in the following papers: [Hayes et al., 2021c, 2022b,d]

1. Section 4.1 investigates if expected value vector methods (SER) can be used to determine a partial ordering over policies under the ESR criterion when the utility function is unknown a priori. By using a simple example, Section 4.1 shows that expected value vector methods are fundamentally incompatible with the ESR criterion. Additionally, in Section 4.1 it is shown that a distributional approach to MODeM must be taken to determine a partial ordering over policies under the ESR criterion when the utility function is unknown.

2. In Section 4.2, stochastic dominance (SD) is proposed as a solution to determine a partial ordering over policies when taking a distributional approach. SD utilises a distribution over the returns to determine a partial ordering over policies and can be used under the ESR criterion when the utility function is unknown.

3. Section 4.3 extends SD to define a novel dominance relation known as ESR dominance. Finally, both SD and ESR dominance are used to define sets of optimal policies under the ESR criterion for scenarios when the utility function of a user is unknown.

4.1 Motivating a Distributional Approach

As shown in Chapter 3, when the utility function of a user is known and nonlinear, the policies computed under the SER criterion and ESR criterion can be different. However, many scenarios exist where the utility function of a user is unknown. In the taxonomy of MODeM, this is known as the unknown utility function scenario. In this case, a multi-policy algorithm is utilised to compute a set of optimal policies which are returned to the user. To date, only multi-policy methods that compute sets of optimal polices for the SER criterion have been proposed. Given computing policies under the ESR criterion is optimal for many real-world scenarios, multi-policy methods must be explored for the ESR criterion. A further complicating factor is that a set of optimal solutions has yet to be defined for the ESR criterion.

Generally, multi-policy methods use expected value vectors to represent a policy. By taking this approach, dominance relations like Pareto dominance can be used to determine a partial ordering over policies. In this case, the Pareto dominated solutions are removed from consideration and the resulting set of optimal policies is returned to the user. When Pareto dominance is used as a dominance relation, the set of optimal policies is known as the Pareto front. Multi-policy methods that utilise expected value vectors have only been explored for the SER criterion. However, the SER criterion and the ESR criterion utilise the utility function

L_1

P(L_1= **R**)	**R**
0.6	(8, 2)
0.4	(6, 1)

L_2

P(L_2=**R**)	**R**
0.9	(5, 1)
0.1	(8, 0)

Table 4.1: Lottery L_1 has two possible returns, (8, 2) with probability 0.6 and (6, 1) with probability 0.4. Lottery L_2 has two possible returns (5, 1) with probability 0.9 and (8, 0) with probability 0.1.

differently. Therefore to determine if expected value vector methods are sufficient to determine a partial ordering over policies under the ESR criterion, consider the example below.

Consider the lotteries, L_1 and L_2 in Table 4.1. In this example the utility function, u, is unknown. To determine which lottery to play in Table 4.1 when optimising for the SER criterion, the expected value vector for L_1 and L_2 must be computed first (see Equation 2.27):

$$\mathbb{E}(L_1) = 0.6((8,2)) + 0.4((6,1)) = (4.8, 1.2) + (2.4, 0.4) = (7.2, 1.6)$$

$$u(\mathbb{E}(L_1)) = u((7.2, 1.6))$$

$$\mathbb{E}(L_2) = 0.9((5,1)) + 0.1((8,0)) = (4.5, 0.9) + (0.8, 0) = (5.3, 0.9)$$

$$u(\mathbb{E}(L_2)) = u((5.2, 0.9))$$

Given that the utility function is unknown, Pareto dominance [Pareto, 1896] can be used to define a partial ordering over expected value vectors for all monotonically increasing utility functions. For example, methods like [White, 1982; Wang and Sebag, 2012; Wiering and de Jong, 2007; Wray et al., 2015] compute the Pareto front. In this example, a user with a monotonically increasing utility function will always prefer lottery L_1 over L_2, given the expected value vector for L_1 Pareto dominates the expected value vector of L_2.

To determine which lottery to play when optimising for the ESR criterion, the utility function must first be applied, then the expected utility can be computed (see Equation 2.28):

$$u(L_1) = u((8,2)) + u((6,1))$$

$$\mathbb{E}(u(L_1)) = 0.6(u((8,2))) + 0.4(u((6,1)))$$

$$u(L_2) = u((5,1)) + u((8,0))$$

$$\mathbb{E}(u(L_2)) = 0.9(u((5,1))) + 0.1(u((8,0)))$$

As the utility function is unknown, it is impossible to compute the expected utility. By using expected value vectors, the utility of the expectation is computed. However, in order to optimise for the ESR criterion, the expected utility must be computed. Therefore, expected value vectors cannot be used to determine a partial ordering over policies for the ESR criterion. Moreover, multi-policy algorithms that utilise expected value vectors cannot be used for the ESR criterion. Therefore, new methods must be developed that can determine a partial ordering over policies for the ESR criterion and also ensure the expected utility can be computed.

As previously highlighted, to compute the expected utility, the utility function must first be applied to the returns, and then the expected utility can be calculated. In the unknown utility function scenario, the utility function of a user becomes known during the selection phase. Therefore, under the ESR criterion, a distributional approach to MODeM must be taken. By taking this approach, a distribution over the returns for a policy is maintained. Once the utility function becomes known during the selection phase, the utility function can be applied to each of the returns in the distribution and the expected utility can be computed.

To adopt a distributional approach to MODeM, a multi-objective version of the return distribution proposed by Bellemare et al. [2017][2] must be introduced. A return distribution, $\mathbf{z}^{\pi}$, is equivalent to a multivariate distribution where a dimension exists per objective. The return distribution, $\mathbf{z}^{\pi}$, gives the distribution over returns of a random vector [Sutton and Barto, 2018] when a policy π is executed, such that,

$$\mathbb{E}\,\mathbf{z}^{\pi} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t}\mathbf{r}_{t} \,\middle|\, \pi, \mu_{0}\right]. \tag{4.1}$$

Moreover, a return distribution can be used to represent policies. Under the ESR criterion, the utility-of-the-return-distribution, z_{u}^{π}, is defined as a distribution over the scalar utilities received from applying the utility function to each vector in the return distribution, $\mathbf{z}^{\pi}$. Therefore, z_{u}^{π} is a distribution over the scalar utility of vector returns of a random vector received from executing a policy π, such that,

$$\mathbb{E}\,z_{u}^{\pi} = \mathbb{E}\left[u\left(\sum_{t=0}^{\infty} \gamma^{t}\mathbf{r}_{t}\right) \,\middle|\, \pi, \mu_{0}\right]. \tag{4.2}$$

The utility-of-the-return-distribution can only be calculated when the utility function is known (or becomes known).

[2]Bellemare et al. [2017] introduce a value distribution. However given the distribution is a distribution over the returns, not values, the term return distribution is preferred.

As previously highlighted, when the utility function of a user is unknown, a set of policies that are optimal for all monotonically increasing utility functions must be computed. However, for the ESR criterion, a method to determine a partial ordering over policies and a corresponding set of optimal solutions has yet to be defined. Therefore, in Section 4.2 methods that determine a partial ordering over return distributions (policies) under the ESR criterion are explored.

4.2 Stochastic Dominance for the Expected Scalarised Returns

First-order stochastic dominance (FSD) is a method that can be used to determine a partial ordering over random variables [Wolfstetter, 1999; Levy, 1992]. FSD compares the cumulative distribution functions (CDFs) of the underlying probability distributions of random variables to determine optimality. When computing policies under the ESR criterion, it is essential that the expected utility is maximised. To use FSD for the ESR criterion, the FSD conditions presented in Chapter 2 Section 2.2 must be shown to also hold when optimising the expected utility for unknown monotonically increasing utility functions.

For the single-objective case, Theorem 2 proves for random variables X and Y, if X first-order stochastically dominates ($\succeq_{FSD}$) Y, the expected utility of X is greater than or equal to the expected utility of Y for monotonically increasing utility functions. In Theorem 2, random variables X and Y are considered, and their corresponding CDFs F_X, F_Y. The work of Mas-Colell et al. [1995] is used as a foundation for Theorem 2.

Theorem 2

A random variable, X, is preferred to a random variable, Y, for all decision makers with a monotonically increasing utility function if, $X \succeq_{FSD} Y$.

$$X \succeq_{FSD} Y \implies \mathbb{E}(u(X)) \geq \mathbb{E}(u(Y))$$

Proof. If X $\succeq_{FSD}$ Y, then[3],

$$F_X(z) \leq F_Y(z), \forall\, z$$

Since,

$$\mathbb{E}(u(X)) = \int_{-\infty}^{\infty} u(z) dF_X(z)$$

[3]CDFs with lower probability values for a given z are preferable. Figure 2.2 outlines why this is the case.

$$\mathbb{E}(u(Y)) = \int_{-\infty}^{\infty} u(z) dF_Y(z)$$

When integrating both $\mathbb{E}(u(X))$ and $\mathbb{E}(u(Y))$ by parts, the following results are generated:

$$\mathbb{E}(u(X)) = [u(z)F_X(z)]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} u'(z)F_X(z)\, dz$$

$$\mathbb{E}(u(Y)) = [u(z)F_Y(z)]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} u'(z)F_Y(z)\, dz$$

Given $F_X(-\infty) = F_Y(-\infty) = 0$ and $F_X(\infty) = F_Y(\infty) = 1$, the first terms in $\mathbb{E}(u(X))$ and $\mathbb{E}(u(Y))$ are equal, and thus

$$\mathbb{E}(u(X)) - \mathbb{E}(u(Y)) = \int_{-\infty}^{\infty} u'(z)F_Y(z)\, dz - \int_{-\infty}^{\infty} u'(z)F_X(z)\, dz$$

Since $F_X(z) \leq F_Y(z)$ and $u'(z) \geq 0$ for all monotonically increasing utility functions, then

$$\mathbb{E}(u(X)) - \mathbb{E}(u(Y)) \geq 0$$

and thus,

$$\mathbb{E}(u(X)) \geq \mathbb{E}(u(Y))$$

□

A utility function maps an input (scalar or vector return) to an output (scalar utility). Since the probability of receiving some utility is equal to the probability of receiving some return for a random variable, X, the following can be written:

$$P(X > c) = P(u(X) > u(c)), \tag{4.3}$$

where c is a constant. Using the results shown in Theorem 2 and Equation 4.3, the FSD conditions highlighted in Section 2.2 can be rewritten to include monotonically increasing utility functions:

$$P(u(X) > u(z)) \geq P(u(Y) > u(z)) \tag{4.4}$$

Definition 16

Let X and Y be random variables. X is preferred over Y for all decision makers with a monotonically increasing utility function if the following is true:

$$X \succeq_{FSD} Y \Leftrightarrow$$
$$\forall u : \forall v : P(u(X) > u(v)) \geq P(u(Y) > u(v)).$$

In MODeM, the return from the reward function is a vector where each element in the return vector represents an objective. To apply FSD to MODeM under the ESR criterion, random vectors must be considered. In this case, a random vector (or multi-variate random variable) is a vector whose components are scalar-valued random variables on the same probability space. For simplicity, the case in which a random vector has two random variables, known as the bi-variate case, is examined. FSD conditions have been proven to hold for random vectors with n random variables in the works of Sriboonchitta et al. [2009], Levhari et al. [1975], Nakayama et al. [1981] and Scarsini [1988].

In Theorem 3, the work of Atkinson and Bourguignon [1982] is distilled into a suitable theorem for MODeM. Theorem 3 highlights how the conditions for FSD hold for random vectors when optimising under the ESR criterion for a monotonically increasing utility function, u, where $\frac{\partial^2 u(x_1,x_2)}{\partial x_1 \partial x_2} \leq 0$ [Richard, 1975]. It is important to note Atkinson and Bourguignon [1982] have shown the conditions for FSD hold for random vectors for utility functions where $\frac{\partial^2 u(x_1,x_2)}{\partial x_1 \partial x_2} \geq 0$. In Theorem 3, $\mathbf{X}$ and $\mathbf{Y}$ are random vectors where each random vector consists of two random variables, $\mathbf{X} = [X_1, X_2]$ and $\mathbf{Y} = [Y_1, Y_2]$. $F_{X_1X_2}$ and $F_{Y_1Y_2}$ are the corresponding CDFs.

Theorem 3

Assume that $u : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a monotonically increasing function, with $\frac{\partial u(x_1,x_2)}{\partial x_1} \geq 0$, $\frac{\partial u(x_1,x_2)}{\partial x_2} \geq 0$ and $\frac{\partial^2 u(x_1,x_2)}{\partial x_1 \partial x_2} \leq 0$. If, for random vectors $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X} \succeq_{FSD} \mathbf{Y}$, then $\mathbf{X}$ is preferred to $\mathbf{Y}$ by all decision makers, i.e.,

$$\mathbf{X} \succeq_{FSD} \mathbf{Y} \implies \mathbb{E}(u(\mathbf{X})) \geq \mathbb{E}(u(\mathbf{Y}))$$

Proof. As $\mathbf{X} \succeq_{FSD} \mathbf{Y}$, $\forall t, z$ we have

$$F_{\mathbf{X}}(t,z) \leq F_{\mathbf{Y}}(t,z),$$
$$\text{or } \Delta_F(t,z) = F_{\mathbf{X}}(t,z) - F_{\mathbf{Y}}(t,z) \leq 0.$$

The expected utility of the random variable $\mathbf{X}$ can be written as follows:

$$\mathbb{E}\, u(\mathbf{X}) = \int_0^\infty \int_0^\infty u(t,z) f_{\mathbf{X}}(t,z) dt dz,$$

where f is the probability density function of $\mathbf{X}$. Note that

$$\begin{aligned}\Delta_f(t,z) &= f_{\mathbf{X}}(t,z) - f_{\mathbf{Y}}(t,z)\\ &= \frac{\partial^2 \Delta_F(t,z)}{\partial t \partial z}.\end{aligned}$$

Using integration-by-parts (I), and the fact that $\Delta_F(t,0) = \frac{\partial \Delta_F(0,z)}{\partial z} = 0$ (Z), the following can be obtained:

$$\begin{aligned}
&\mathbb{E}\, u(\mathbf{X}) - \mathbb{E}\, u(\mathbf{Y}) \\
&= \int_0^\infty \int_0^\infty u(t,z)\Delta_f(t,z)dtdz \\
&\overset{(I)}{=} \int_0^\infty \left[u(t,z)\frac{\partial \Delta_F(t,z)}{\partial z}\right]_{t=0}^\infty dz - \int_0^\infty \int_0^\infty \frac{\partial u(t,z)}{\partial t}\frac{\partial \Delta_F(t,z)}{\partial z} dtdz \\
&\overset{(I)}{=} \int_0^\infty \left[u(t,z)\frac{\partial \Delta_F(t,z)}{\partial z}\right]_{t=0}^\infty dz - \int_0^\infty \left[\frac{\partial u(t,z)}{\partial t}\Delta_F(t,z)\right]_{z=0}^\infty dt + \\
&\int_0^\infty \int_0^\infty \frac{\partial^2 u(t,z)}{\partial t \partial z}\Delta_F(t,z)dtdz \\
&\overset{(Z)}{=} \int_0^\infty \lim_{t\to\infty} u(t,z)\frac{\partial \Delta_F(t,z)}{\partial z}dz - \int_0^\infty \lim_{z\to\infty} \frac{\partial u(t,z)}{\partial t}\Delta_F(t,z)dt + \\
&\int_0^\infty \int_0^\infty \frac{\partial^2 u(t,z)}{\partial t \partial z}\Delta_F(t,z)dtdz.
\end{aligned}$$

Given that $\frac{\partial^2 u(t,z)}{\partial t \partial z} \leq 0$, $\frac{\partial u(t,z)}{\partial t} \geq 0$ and $\Delta_F(t,z) \leq 0$, the last two terms are positive. Therefore, the following can be stated:

$$\begin{aligned}
&\mathbb{E}\, u(\mathbf{X}) - \mathbb{E}\, u(\mathbf{Y}) \\
&= \int_0^\infty \lim_{t\to\infty} u(t,z)\frac{\partial \Delta_F(t,z)}{\partial z}dz - \int_0^\infty \lim_{z\to\infty} \frac{\partial u(t,z)}{\partial t}\Delta_F(t,z)dt + \\
&\int_0^\infty \int_0^\infty \frac{\partial^2 u(t,z)}{\partial t \partial z}\Delta_F(t,z)dtdz \geq \int_0^\infty \lim_{t\to\infty} u(t,z)\frac{\partial \Delta_F(t,z)}{\partial z}dz.
\end{aligned}$$

According to Lemma 2 (see Section A.3), as $u(t,z)F(t,z)$ is a positive monotonically increasing function in both t and z, the following is known:

$$\int_0^\infty \lim_{t\to\infty} u(t,z)\frac{\partial F(t,z)}{\partial z}dz = \lim_{t\to\infty}\int_0^\infty u(t,z)\frac{\partial F(t,z)}{\partial z}dz.$$

Using integration-by-parts (I), and the fact that $\Delta_F(t,0) = 0$ (Z):

$$\begin{aligned}
&\mathbb{E}\, u(\mathbf{X}) - \mathbb{E}\, u(\mathbf{Y})\\
&\geq \lim_{t\to\infty} \int_0^\infty u(t,z) \frac{\partial \Delta_F(t,z)}{\partial z} dz\\
&\overset{(I)}{=} \lim_{t\to\infty} \left[u(t,z)\Delta_F(t,z)\right]_0^\infty - \lim_{t\to\infty} \int_0^\infty \frac{\partial u(t,z)}{\partial z} \Delta_F(t,z) dz\\
&\overset{(Z)}{=} \lim_{t\to\infty} \lim_{z\to\infty} u(t,z)\Delta_F(t,z) - \lim_{t\to\infty} \int_0^\infty \frac{\partial u(t,z)}{\partial z} \Delta_F(t,z) dz.
\end{aligned}$$

Finally, given that $\frac{\partial u(t,z)}{\partial z} \geq 0$ and $\Delta_F(t,z) \leq 0$, the following is known:

$$\begin{aligned}
&\mathbb{E}\, u(\mathbf{X}) - \mathbb{E}\, u(\mathbf{Y})\\
&\geq \lim_{t\to\infty} \lim_{z\to\infty} u(t,z)\Delta_F(t,z) - \lim_{t\to\infty} \int_0^\infty \frac{\partial u(t,z)}{\partial z} \Delta_F(t,z) dz\\
&\geq 0
\end{aligned}$$

□

Using the results from Theorem 3, Equation 4.4 can be updated to include random vectors,

$$P(u(\mathbf{X}) > u(\mathbf{z})) \geq P(u(\mathbf{Y}) > u(\mathbf{z})). \tag{4.5}$$

Definition 17

For random vectors $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X}$ is preferred over $\mathbf{Y}$ by all decision makers with a monotonically increasing utility function if, and only if, the following is true:

$$\mathbf{X} \succeq_{FSD} \mathbf{Y} \Leftrightarrow$$
$$\forall u : (\forall \mathbf{v} : P(u(\mathbf{X}) > u(\mathbf{v})) \geq P(u(\mathbf{Y}) > u(\mathbf{v}))$$

Using the results from Theorem 3 and Definition 17, it is possible to extend FSD to MODeM. As defined in Section 4.1, for MODeM under the ESR criterion, the return distribution, $\mathbf{z}^\pi$, is considered to be the full distribution of the returns of a random vector received when executing a policy, π. Return distributions can be used to represent policies, and it is possible to use FSD to obtain a partial ordering over policies. For example, consider two policies, π and π', where each policy has the underlying return distribution $\mathbf{z}^\pi$ and $\mathbf{z}^{\pi'}$. If $\mathbf{z}^\pi \succeq_{FSD} \mathbf{z}^{\pi'}$ then π will be preferred over π'.

Definition 18

Policies π and π' have return distributions $\mathbf{z}^{\pi}$ and $\mathbf{z}^{\pi'}$. Policy π is preferred over policy π' by all decision makers with a utility function, u, that is monotonically increasing if, and only if, the following is true:

$$\mathbf{z}^{\pi} \succeq_{FSD} \mathbf{z}^{\pi'}.$$

Now that a partial ordering over policies has been defined under the ESR criterion for the unknown utility function scenario, it is possible to define a set of optimal policies.

4.3 Solution Sets for the Expected Scalarised Returns

Section 4.2 defines a partial ordering over policies under the ESR criterion for unknown utility functions using FSD. In the unknown utility function scenario, it is infeasible to learn a single optimal policy [Roijers et al., 2013]. When a user's utility function is unknown, multi-policy MODeM algorithms must be used to compute a set of optimal policies. To apply MODeM to the ESR criterion in scenarios with unknown utility, a set of optimal policies under the ESR criterion must be defined. In Section 4.3, FSD is used to define multiple sets of optimal policies for the ESR criterion.

First, a set of optimal policies, known as the undominated set, is defined. The undominated set is defined using FSD, where each policy in the undominated set has an underlying return distribution that is FSD dominant. The undominated set contains at least one optimal policy for all possible monotonically increasing utility functions.

Definition 19

The undominated set, $U(\Pi)$, is a sub-set of all possible policies for where there exists some utility function, u, where a policy's return distribution is FSD dominant.

$$U(\Pi) = \left\{ \pi \in \Pi \;\middle|\; \exists u, \forall \pi' \in \Pi : \mathbf{z}^{\pi} \succeq_{FSD} \mathbf{z}^{\pi'} \right\}$$

However, the undominated set may contain excess policies. For example, under FSD, if two dominant policies have return distributions that are equal, then both policies will be in the undominated set. Given both return distributions are equal, a user with a monotonically increasing utility function will not prefer one policy over the other. In this case, both policies have the same expected utility. To reduce the number of policies that must be considered at execution time, for each possible utility function it is possible to keep just one corresponding FSD dominant policy; such a set of policies is called a coverage set (CS).

Definition 20

The coverage set, $CS(\Pi)$, is a subset of the undominated set, $U(\Pi)$, where, for every utility function, u, the set contains a policy that has a FSD dominant return distribution,

$$CS(\Pi) \subseteq U(\Pi) \land \Big(\forall u, \exists \pi \in CS(\Pi), \forall \pi' \in \Pi : \mathbf{z}^{\pi} \succeq_{FSD} \mathbf{z}^{\pi'} \Big)$$

In practice, a decision maker may aim to learn the smallest possible set of optimal policies. However, the FSD relation considered does not have a strict inequality condition. Moreover, the undominated set generated using FSD may contain excess policies. Therefore, to compute a coverage set in practice where each optimal policy has a unique return distribution, a new dominance relation called expected scalarised returns dominance (ESR dominance) is defined. In contrast to FSD, ESR dominance ensures that an explicitly strict inequality exists.

Definition 21

For random vectors $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X} \succ_{ESR} \mathbf{Y}$ for all decision makers with a monotonically increasing utility function if, and only if, the following is true:

$$\mathbf{X} \succ_{ESR} \mathbf{Y} \Leftrightarrow$$

$$\forall u : (\forall \mathbf{v} : P(u(\mathbf{X}) > u(\mathbf{v})) \geq P(u(\mathbf{Y}) > u(\mathbf{v}))$$
$$\land \exists\, \mathbf{v} : P(u(\mathbf{X}) > u(\mathbf{v})) > P(u(\mathbf{Y}) > u(\mathbf{v}))).$$

ESR dominance (Definition 21) extends FSD, however, FSD is a more strict dominance criterion. For FSD, policies that have equal return distributions are considered dominant policies, which is not the case under ESR dominance. Therefore, if a random vector is ESR dominant, the random vector has a greater expected utility than all ESR dominated random vectors. Theorem 4 proves that if a random vector $\mathbf{X}$ ESR dominates a random vector $\mathbf{Y}$, $\mathbf{X}$ has a greater expected utility than $\mathbf{Y}$. Theorem 4 focuses on random vectors $\mathbf{X}$ and $\mathbf{Y}$ where each random vector has two random variables, such that $\mathbf{X} = [X_1, X_2]$ and $\mathbf{Y} = [Y_1, Y_2]$. $F_{\mathbf{X}}$

and $F_{\mathbf{Y}}$ are the corresponding CDFs and $\mathbf{v} = [t, z]$. However, Theorem 4 can easily be extended for random vectors with n random variables ($\mathbf{X} = [X_1, X_2, ..., X_n]$).

> **Theorem 4**
>
> A random vector, $\mathbf{X}$, is preferred to a random vector, $\mathbf{Y}$, by all decision makers with a monotonically increasing utility function if, and only if, $\mathbf{X} \geq_{ESR} \mathbf{Y}$:
>
> $$\mathbf{X} \succ_{ESR} \mathbf{Y} \implies \mathbb{E}(u(\mathbf{X})) > \mathbb{E}(u(\mathbf{Y}))$$

Proof. $\mathbf{X}$ and $\mathbf{Y}$ are random vectors with n random variables. If $\mathbf{X} \succ_{ESR} \mathbf{Y}$ the following two conditions must be met for all u:

1. $\forall \mathbf{v} : P(u(\mathbf{X}) > u(\mathbf{v})) \geq P(u(\mathbf{Y}) > u(\mathbf{v}))$
2. $\exists\, \mathbf{v} : P(u(\mathbf{X}) > u(\mathbf{v})) > P(u(\mathbf{Y}) > u(\mathbf{v}))$

From Definition 17, if $\mathbf{X} \succeq_{FSD} \mathbf{Y}$ then the following is true:

$$\forall u : \forall \mathbf{v} : P(u(\mathbf{X}) > u(\mathbf{v})) \geq P(u(\mathbf{Y}) > u(\mathbf{v}))$$

If $\mathbf{X} \succeq_{FSD} \mathbf{Y}$, then, from Theorem 3, the following is true:

$$\mathbb{E}(u(\mathbf{X})) \geq \mathbb{E}(u(\mathbf{Y}))$$

If condition 1 is satisfied, the expected utility of $\mathbf{X}$ is at least equal to the expected utility of $\mathbf{Y}$, then:

$$\mathbb{E}(u(\mathbf{X})) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} u(\mathbf{z}) f_{\mathbf{X}}(t, z)\, dt\, dz$$

$$\mathbb{E}(u(\mathbf{Y})) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} u(\mathbf{z}) f_{\mathbf{Y}}(t, z)\, dt\, dz$$

In order to satisfy condition 2, some limits must exist to give the following,

$$\int_{a}^{b} \int_{c}^{d} u(t, z) f_{\mathbf{X}}(t, z)\, dt\, dz > \int_{a}^{b} \int_{c}^{d} u(t, z) f_{\mathbf{Y}}(t, z)\, dt\, dz$$

The minimum requirement to satisfy condition 1 is:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} u(t, z) f_{\mathbf{X}}(t, z)\, dt\, dz = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} u(t, z) f_{\mathbf{Y}}(t, z)\, dt\, dz$$

If condition 1 is satisfied, to satisfy condition 2 some limits must exist:

$$\int_a^b \int_c^d u(t,z) f_{\mathbf{X}}(t,z)\, dt\, dz > \int_a^b \int_c^d u(t,z) f_{\mathbf{Y}}(t,z)\, dt\, dz.$$

Therefore,

$$\int_{-\infty}^a \int_{-\infty}^c u(t,z) f_{\mathbf{X}}(t,z)\, dt\, dz + \int_a^b \int_c^d u(t,z) f_{\mathbf{X}}(t,z)\, dt\, dz +$$

$$\int_b^\infty \int_d^\infty u(t,z) f_{\mathbf{X}}(t,z)\, dt\, dz > \int_{-\infty}^a \int_{-\infty}^c u(t,z) f_{\mathbf{Y}}(t,z)\, dt\, dz +$$

$$\int_a^b \int_c^d u(t,z) f_{\mathbf{Y}}(t,z)\, dt\, dz + \int_b^\infty \int_d^\infty u(t,z) f_{\mathbf{Y}}(t,z)\, dt\, dz.$$

Finally,

$$\int_{-\infty}^\infty \int_{-\infty}^\infty u(t,z) f_{\mathbf{X}}(t,z)\, dt\, dz > \int_{-\infty}^\infty \int_{-\infty}^\infty u(t,z) f_{\mathbf{Y}}(t,z)\, dt\, dz$$

if $\mathbf{X} \succ_{ESR} \mathbf{Y}$, then,

$$\mathbb{E}(u(\mathbf{X})) > \mathbb{E}(u(\mathbf{Y})).$$

$\square$

In the ESR dominance criterion defined in Definition 21, the utility of different vectors is compared. However, it is not possible to calculate the utility of a vector when the utility function is unknown. In this case, Pareto dominance [Pareto, 1896] can be used instead to determine whether one of the vectors being compared is guaranteed to give a higher utility.

Definition 22

A Pareto dominates ($\succ_p$) **B** if the following is true:

$$\mathbf{A} \succ_p \mathbf{B} \Leftrightarrow (\forall i : \mathbf{A}_i \geq \mathbf{B}_i) \wedge (\exists i : \mathbf{A}_i > \mathbf{B}_i). \tag{4.6}$$

For monotonically increasing utility functions, if the value of an element of the vector increases, then the scalar utility of the vector also increases. Therefore, using Definition 22, if vector **A** Pareto dominates vector **B**, for a monotonically increasing utility function, **A** has a higher utility than **B**. To make ESR comparisons between return distributions, Pareto dominance can be used.

Definition 23

For random vectors $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X} \succ_{ESR} \mathbf{Y}$ for all monotonically increasing utility functions if, and only if, the following is true:

$$\mathbf{X} \succ_{ESR} \mathbf{Y} \Leftrightarrow$$

$$\forall \mathbf{v} : P(\mathbf{X} >_P \mathbf{v}) \geq P(\mathbf{Y} >_P \mathbf{v}) \wedge \exists \mathbf{v} : P(\mathbf{X} >_P \mathbf{v}) > P(\mathbf{Y} >_P \mathbf{v}).$$

It is also possible to calculate ESR dominance by comparing the CDFs of random vectors. Using the CDF, also guarantees a higher expected utility. Using the CDF it is possible to compare the cumulative probabilities for a given vector where a lower cumulative probability is preferred. ESR dominance with the CDF does not require any information about the utility function of a user and, therefore, can be used in the unknown utility function scenario.

Definition 24

For random vectors $\mathbf{X}$ and $\mathbf{Y}$, $\mathbf{X} \succ_{ESR} \mathbf{Y}$ for all monotonically increasing utility functions if, and only if, the following is true:

$$\mathbf{X} \succ_{ESR} \mathbf{Y} \Leftrightarrow$$
$$\forall \mathbf{v} : F_{\mathbf{X}}(\mathbf{v}) \leq F_{\mathbf{Y}}(\mathbf{v}) \wedge \exists \mathbf{v} : F_{\mathbf{X}}(\mathbf{v}) < F_{\mathbf{Y}}(\mathbf{v}).$$

Therefore, either Definition 23 or Definition 24 can be used to calculate ESR dominance to give a partial ordering over policies.

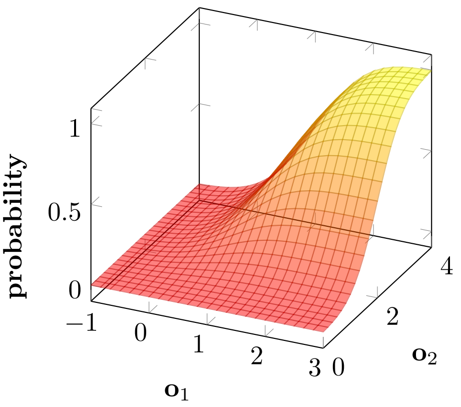
Definition 25

For return distributions $\mathbf{z}^{\pi}$ and $\mathbf{z}^{\pi'}$ for policies π and π', π is preferred over π' by all decision makers with a monotonically increasing utility function if, and only if, the following is true:

$$\mathbf{z}^{\pi} \succ_{ESR} \mathbf{z}^{\pi'}$$

To illustrate the ESR dominance relation and how a partial ordering over return distributions is determined, consider the example outlined in Figure 4.1 and Figure 4.2. As already highlighted, to determine ESR dominance, the CDF, $F_{\mathbf{X}}$ of a return distribution $\mathbf{X}$ must be compared with the CDF, $F_{\mathbf{Y}}$, of a return distribution $\mathbf{Y}$. To illustrate the example, it is possible to rewrite Definition 24 to give the following condition, which must be true:

$$\forall \mathbf{v} : F_{\mathbf{X}}(\mathbf{v}) - F_{\mathbf{Y}}(\mathbf{v}) \leq 0 \wedge \exists \mathbf{v} : F_{\mathbf{X}}(\mathbf{v}) - F_{\mathbf{Y}}(\mathbf{v}) < 0.$$



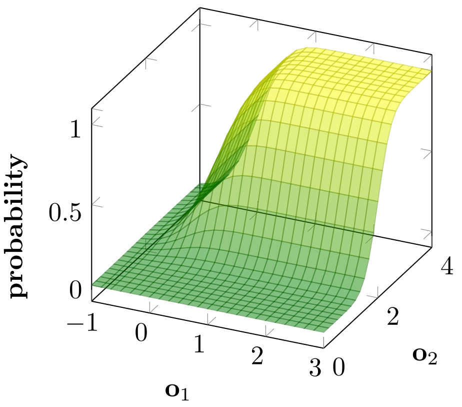


Figure 4.1: **(left)** The CDF, $F_{\mathbf{X}}$, of a return distribution $\mathbf{X}$. **(right)** The CDF, $F_{\mathbf{Y}}$, of a return distribution $\mathbf{Y}$.

Figure 4.2 highlights the difference in probability for $F_{\mathbf{X}} - F_{\mathbf{Y}}$. The dotted line in Figure 4.2, labelled (a), highlights that, for at least one point, $F_{\mathbf{X}} - F_{\mathbf{Y}} > 0$. Therefore, the return distribution $\mathbf{X}$ cannot ESR dominate the return distribution $\mathbf{Y}$.

Finally, by using ESR dominance it is possible to define a set of optimal policies, known as the *ESR set*.

Definition 26

The *ESR set*, $ESR(\Pi)$, is a sub-set of all policies where each policy in the *ESR set* is ESR dominant,

$$ESR(\Pi) = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \mathbf{z}^{\pi'} \succ_{ESR} \mathbf{z}^{\pi}\}. \tag{4.7}$$

The *ESR set* is a set of non-dominated policies, where each policy in the *ESR set* is ESR dominant. The *ESR set* can be considered a coverage set, when no excess policies exist in the *ESR set.* It is viable for a multi-policy MODeM method to use ESR dominance to construct the *ESR set.*

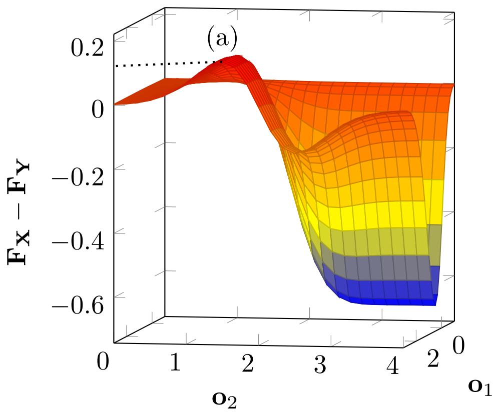


Figure 4.2: The difference in probability mass for $F_{\mathbf{X}} - F_{\mathbf{Y}}$, which is used to visualise the requirements for ESR dominance. A dotted line (a) is drawn to highlight that $F_{\mathbf{X}} - F_{\mathbf{Y}} > 0$ for least at one point. Therefore, $\mathbf{X}$ does not ESR dominate $\mathbf{Y}$.

4.4 Related Work

The various orders of SD have been used extensively as a method to determine the optimal decision when making decisions under uncertainty in economics [Choi and Johnson, 1988], finance [Ali, 1975; Bawa, 1978], game theory [Fishburn, 1978], and various other real-world scenarios [Bawa, 1982]. However, SD has largely been overlooked in systems that learn. Cook and Jarrett [2018] use various orders of SD and Pareto dominance with genetic algorithms to compute optimal solution sets for an aerospace design problem with multiple objectives when constrained by a computational budget. Martin et al. [2020] use second-order stochastic dominance (SSD) with a single-objective distributional reinforcement learning (distRL) algorithm [Bellemare et al., 2017]. Martin et al. [2020] use SSD to determine the optimal action to take at decision time, and this approach is shown to learn good policies during experimentation.

4.5 Summary

This chapter has added several major contributions to the MODeM literature. Prior to this work, the ESR criterion for the unknown utility function scenario had not been explored, and how best to compute sets of optimal policies for the ESR criterion remained an open question. The work presented in this chapter outlines how MODeM problems can be solved under the ESR criterion with unknown utility functions and provides a theoretical methodology for doing so.

An important aspect of this chapter was the identification of the limitations of SER value vector methods for the ESR criterion. Prior to this work, the SER criterion was the only optimality criteria that had been explored for settings with unknown utility functions. The findings presented in Section 4.1 outline that dedicated algorithms must be developed to compute sets of optimal policies under the ESR criterion by taking a distributional approach.

Taking a distributional approach to MODeM means new methods must be used to determine a partial ordering over policies. Section 4.2 and Section 4.3 outlined and proved how SD and ESR dominance can be used under the ESR criterion to determine a partial ordering over policies. Having tractable methods to determine a partial ordering over policies is crucial when computing sets of optimal policies when the utility function is unknown. Therefore, Section 4.2 and Section 4.3 outline major contributions to the MODeM literature.

Finally, by following the utility-based approach, a user is always assumed to be part of the decision making process. Therefore, the optimal policies which have been computed must be returned to the user for selection. Again, under the ESR criterion, such methods had not been explored. In Section 4.3, an undominated set, a coverage set, and the *ESR set* were defined. The defined sets contain policies that are optimal for all monotonically increasing utility functions and, therefore, can be returned to the user during the decision making process. By comparing distributions from an *ESR set*, rather than expected value vectors, users can get a better intuition of the range of possible outcomes for a decision. For example, a user could avoid selecting policies or actions that that have an unacceptable probability of an undesirable outcome; this is impossible with SER methods and expected value vectors.

5 | Algorithms for Unknown Utility Functions[1]

In Chapter 4 multiple solution concepts were defined for the expected scalarised returns (ESR) criterion that can be used to determine a partial ordering over policies. Additionally, relations to determine sets of optimal policies for the ESR criterion were also defined. As previously highlighted, multi-policy methods have not been explored for the ESR criterion. However, now that a partial ordering over policies and sets of optimal solutions for the ESR criterion have been defined, it is possible to design multi-policy algorithms for the ESR criterion by taking a distributional approach. Chapter 5 proposes several multi-policy multi-objective decision making (MODeM) algorithms that can compute sets of optimal polices under the ESR criterion in different MODeM settings. Each algorithm utilises distributions over the returns to compute a set of optimal policies for the ESR criterion. During the empirical evaluation for each algorithm, the *ESR set* is computed. The contributions of Chapter 5 are as follows:

1. First, Section 5.1 proposes a new pruning algorithm for the ESR criterion known as ESRPrune. The pruning algorithm proposed is necessary to compute sets of optimal policies under the ESR criterion.

[1]The contributions presented in Chapter 5 are published in the following papers: [Hayes et al., 2022a,b,d,e].

2. Second, Section 5.2 proposes a novel *multi-objective distributional tabular reinforcement learning (MOTDRL)* algorithm that computes a set of optimal policies under the ESR criterion in a multi-objective multi-armed bandit (MOMAB) setting. Furthermore, to evaluate MOTDRL a novel evaluation metric for the ESR criterion is proposed. MOTDRL is evaluated in multiple MOMAB problem domains.

3. Next, Section 5.3 defines a new *multi-objective distributional value iteration (MODVI)* algorithm to compute a set of optimal policies (*ESR set*) in a multi-objective Markov decision process (MOMDP). MODVI is evaluated using several MOMDP benchmark problems from the literature.

4. Finally, Section 5.4 describes a novel *distributional multi-objective variable elimination (DMOVE)* algorithm that computes the *ESR set* for multi-agent settings, specifically in multi-objective coordination graphs (MO-CoGs). DMOVE is evaluated using several benchmark MO-CoGs from the literature.

5.1 A Pruning Algorithm for the Expected Scalarised Returns

To compute the *ESR set*, comparisons of return distributions must be made using ESR dominance. In settings with a large number of policies, many comparisons between return distributions must be made to determine optimality. Therefore, to optimise the computation of the *ESR set*, a pruning algorithm known as ESRPrune is defined. ESRPrune can be used to reduce the nessesary comparisons of return distributions in multi-objective settings under the ESR criterion. ESRPrune is particularly useful in settings with a large number of policies.

For the SER criterion, multiple pruning operators exist. For example, PPrune [Roijers, 2016] can be utilised to compute the Pareto front or CPrune [Roijers et al., 2015] can be used to compute the convex coverage set. However, such pruning operators utilise expected value vectors which are fundamentally incompatible with the ESR criterion, as shown in Chapter 4. Therefore, in order to apply a pruning algorithm to the ESR criterion, ESRPrune takes return distributions into consideration. Algorithm 14 presents the ESRPrune algorithm. ESRPrune can be used to compute the *ESR set* for multi-objective settings under the ESR criterion.

ESRPrune utilises ESR dominance and, like Pareto dominance, ESR dominance is transitive [Wolfstetter, 1999]. Therefore, ESRPrune can be applied in sequence. To compute ESR dominance, the cumulative distribution function (CDF) of each return distribution in the given set must be calculated. ESRPrune iterates over the given set of return distributions and compares the CDFs of the return distributions

Algorithm 14: ESRPrune

```
1  Input: $\mathbf{Z}$ ← A set of return distributions
2  $\mathbf{Z}^*$ ← $\emptyset$
3  while $Z \neq \emptyset$ do
4      $\mathbf{z}$ ← the first element of $Z$
5      for $z' \in Z$ do
6          if $z' >_{ESR} z$ then
7              $\mathbf{z} \leftarrow \mathbf{z}'$
8          end
9      end
10     Remove $z$ and all return distributions ESR-dominated by $\mathbf{z}$ from $\mathbf{Z}$
11     Add $\mathbf{z}$ to $\mathbf{Z}^*$
12 end
13 Return $\mathbf{Z}^*$
```

to determine which are ESR non-dominated. The return distributions that are ESR dominated are removed from the set. ESRPrune is utilised in Section 5.3 and Section 5.4 to compute the *ESR set* in MOMDPs and MO-CoGs.

5.2 Solving Multi-Objective Multi-Armed Bandits for the Expected Scalarised Returns

To compute a set of optimal policies under the ESR criterion in MOMAB settings, a new *multi-objective distributional tabular reinforcement learning (MOTDRL)* algorithm is proposed in Section 5.2.1. To evaluate the performance of MOTDRL, Section 5.2.2 proposes a new evaluation metric for the ESR criterion. In Section 5.2.3 MOTDRL is evaluated using several MOMAB problems and computes the *ESR set* for each problem domain.

5.2.1 Multi-Objective Tabular Distributional Reinforcement Learning

In this section, a novel multi-objective distributional tabular reinforcement learning (MOTDRL) algorithm is presented that learns an optimal set of policies for the ESR criterion, also known as the *ESR set*, for MOMAB problems. MOTDRL learns the return distribution for a policy by sampling each available arm in a MOMAB setting. Given MOTDRL only considers MOMAB problem domains, MOTDRL

Z	$x_2 = 0$	$x_2 = 1$	$x_2 = 2$	$x_2 = 3$	$x_2 = 4$	$x_2 = 5$
$x_1 = 0$	0	0	0	0	0	0
$x_1 = 1$	0	0	0	0	0	0
$x_1 = 2$	0	0	0	0	0	0
$x_1 = 3$	0	0	0	0	0	0
$x_1 = 4$	0	0	0	0	0	0
$x_1 = 5$	0	0	0	0	0	0

Figure 5.1: An illustration of an initialised Z-table for a problem domain with two objectives, x_1 and x_2, with each index value set to 0.

maintains a distribution per arm and updates the distribution after each timestep with the return vector received from executing the sampled arm.

When optimising under the ESR criterion, it is crucial that a MODeM method learns the underlying distribution over the returns. Other distributional MODeM methods, such as bootstrap Thompson sampling (BTS) (see Chapter 3), cannot be used to learn a set of optimal policies under the ESR criterion when the utility function is unknown. Such methods learn a distribution over the mean returns. In scenarios where the utility function is unknown or unavailable, computing ESR dominance would not be possible using the outlined methods.

MOTDRL can learn the underlying return distribution for an arm by maintaining a tabular representation of the underlying multivariate distribution. To maintain a tabular representation of a multivariate distribution, a Z-table for each arm is initialised where the Z-table has an axis per objective. The Z-table maintains a count of the number of times a return vector is received for a given arm. The size of each Z-table is initialised using the parameters $\mathbf{R}_{min}$ and $\mathbf{R}_{max}$, which are the minimum and maximum returns obtainable for any of the objectives in the given environment. Therefore, each axis in the Z-table will use $\mathbf{R}_{min}$ and $\mathbf{R}_{max}$ to define the length of the axis, where each index value of the Z-table is initialised to 0. Using $\mathbf{R}_{min}$ and $\mathbf{R}_{max}$ as initialisation parameters, a Z-table can be constructed that contains an index for all possible return vectors in a given problem domain. Figure 5.1 visualises an initialised Z-table for a multi-objective problem with two objectives where $\mathbf{R}_{min} = 0$ and $\mathbf{R}_{max} = 5$.

Each Z-table can be used to calculate the return distribution of an arm, $\mathbf{z}^{\pi}$, that can be used to represent a policy π. At each timestep, t, the returns, $\mathbf{R}$, received from pulling arm, i, are used to update the Z-table. The Z-table is used to maintain a count of the number of times the return, $\mathbf{R}$, is received. In MOMAB problem domains, the returns received from the execution of an arm represent the full returns of the execution of a policy. To update the Z-table, the value at the

Algorithm 15: Z-table Update

```
1 Input - arm, $i$
2 Require - $Z$-table for arm, $i$, $Z_i$
3 Pull arm, $i$, and observe return, $\mathbf{R}$.
4 $Z_i(\mathbf{R}) = Z_i(\mathbf{R}) + 1$
5 $N_i = N_i + 1$
6 return $Z$-table, $Z_i$.
```

index corresponding to the return $\mathbf{R}$ is incremented by one. To correctly calculate the probability of receiving return $\mathbf{R}$ when pulling arm i, a counter, N_i, which represents the number of times arm i has been pulled, must be maintained. Each time arm i is pulled, the counter N_i is incremented by one. Algorithm 15 outlines how the Z-table for each arm is updated.

MOTDRL is a multi-policy algorithm that can learn the *ESR set* using ESR dominance. Algorithm 16 outlines how MOTDRL learns the *ESR set* when the utility function of a user in unknown in a MOMAB problem domain. In Algorithm 16, $\mathcal{A}$ is defined as a set of available arms, the *ESR set* is defined as E, D is the number of objectives, n is the total number of pulls across all arms, N_j and N_i are the total pulls of arms j and i, and $|E^*|$ is the cardinality of the *ESR set*, which is known a priori. When learning, the MOTDRL algorithm has priori knowledge of $\mathcal{A}$, $\mathbf{R}_{max}$ and $\mathbf{R}_{min}$. The agent must have knowledge of $\mathbf{R}_{max}$ and $\mathbf{R}_{min}$ so the Z-table can be correctly initialised, and the agent must know the number of arms in $\mathcal{A}$ for action selection. A suitable stopping condition criteria for Algorithm 16 is a fixed number of episodes.

Algorithm 16: Multi-Objective Tabular Distributional Reinforcement Learning

```
1 Pull each arm $i$ in $\mathcal{A}$, $\beta$ times
2 $Z$-table Update(i) $\forall$ i $\in \mathcal{A}$
3 repeat
4     Find $E$ such that $\forall$ i $\in E$, $\forall$ $j$
5     $\mathbf{z}^j + \sqrt{\frac{2ln(n\sqrt[4]{D|E^*|})}{N_j}} \not\succ_{ESR} \mathbf{z}^i + \sqrt{\frac{2ln(n\sqrt[4]{D|E^*|})}{N_i}}$
6     Pull $i$ uniform randomly chosen from $E$
7     Z-table Update($i$)
8 until stopping condition is met;
```

On initialisation each arm is pulled β times. The hyperparameter β is selected to ensure each arm is pulled sufficiently to build an initial distribution. For good performance β is set to greater than 1. For β greater than 1, MOTDRL can build a sufficient initial distribution and can then efficiently explore each arm with the upper confidence bound (UCB) statistic. At each timestep, the return distribution of the policies associated with the execution of each arm is calculated. The *ESR set*, E, is then calculated from the resulting return distributions. Therefore, for all the non-optimal arms $l \notin E$, there exists an ESR dominant arm $i \in ESR$ that ESR dominates the arm l.

To calculate ESR dominance in Algorithm 16 at Line 5, it is nessesary to compute both the probability density function (PDF) and cumulative distribution function (CDF) of the underling return distribution of a policy. The PDF can be calculated by computing the probability of receiving individual returns. Combining the Z-table and N for an arm, i, it is possible to compute the probability of receiving each return in a given problem domain, since the following is true:

$$f_{\mathbf{X}}(x_1, x_2, ..., x_n) = P(\mathbf{X} = x_1, \mathbf{X} = x_2, ..., \mathbf{X} = x_n) = \frac{Z_i(x_1, x_2, ..., x_n)}{N_i} \tag{5.1}$$

Once the PDF has been computed using Equation 5.1, it is possible to compute the CDF. Since the following is true:

$$\begin{aligned} F_{\mathbf{X}}(x_1, x_2, ..., x_n) =& P(\mathbf{X} \leq x_1, \mathbf{X} \leq x_2, ..., \mathbf{X} \leq x_n) \\ =& \sum_{x_a \leq x_1} \sum_{x_b \leq x_2} ... \sum_{x_k \leq x_n} P(\mathbf{X} = x_a, \mathbf{X} = x_b, ..., \mathbf{X} = x_k) \\ =& \sum_{x_a \leq x_1} \sum_{x_b \leq x_2} ... \sum_{x_k \leq x_n} \frac{Z_i(x_a, x_b, ..., x_k)}{N_i} \end{aligned} \tag{5.2}$$

Using the PDF and the CDF of a return distribution, it is possible to determine if arms are ESR dominated using Definition 23 or Definition 24.

To efficiently explore all available arms, MOTDRL uses the UCB statistic presented by Drugan and Nowe [2013]. MOTDRL uses UCB to transform the PDF of the underlying return distribution, by adding the UCB statistic, computed at Line 5 in Algorithm 16, to the PDF. By summing the UCB statistic and the PDF, the PDF is shifted relative to the value of the computed UCB statistic. The CDF can then calculated based on the transformed PDF and ESR dominance can then be calculated.

Transforming the PDF using the UCB statistic ensures that there is sufficient exploration of all available arms during experimentation. However, as the number of pulls of a given arm increases, the UCB statistic decreases, which decreases exploration. Over time the UCB statistic's effect on the PDF and CDF becomes negligible. At such a point, MOTDRL can exploit the return distributions learned during exploration and compute the *ESR set*.

Given MOTDRL is a multi-policy algorithm, MOTDRL can be used in the unknown utility function scenario. During learning, MOTDRL learns the *ESR set* by utilising the steps in Algorithm 16. In Section 5.2.3 MOTDRL is evaluated using two MOMAB settings. However, the state-of-the-art MODeM evaluation methods are only useful when optimising under the SER criterion. Therefore, in Section 5.2.2, a new metric is proposed that can be used to evaluate MODeM algorithms under the ESR criterion.

5.2.2 Evaluation Metrics

The standard metrics for MODeM [Vamplew et al., 2011; Zintgraf et al., 2015; Yang et al., 2019] are not suitable to evaluate a multi-policy method under the ESR criterion because they are designed to specifically evaluate SER multi-policy methods. To evaluate MODeM algorithms under the ESR criterion, the coverage ratio metric used by Yang et al. [2019] is adapted for the ESR criterion. The coverage ratio evaluates the agent's ability to recover optimal solutions in the *ESR set* (E). If $\mathcal{F} \subseteq R^m$ is the set of solutions found by the agent, the following can be defined:

$$\mathcal{F} \cap_\epsilon E := \{\mathbf{z}^\pi \in \mathcal{F} \mid \exists \mathbf{z}^{\pi'} \in E \, s.t \sup_{\mathbf{x}} |F_{\mathbf{z}^\pi}(\mathbf{x}) - F_{\mathbf{z}^{\pi'}}(\mathbf{x}) \,| \leq \epsilon\}, \tag{5.3}$$

where $\mathbf{x} = [x_1, x_2, ..., x_D]$ and D is equal to the number of objectives. Equation 5.3 uses the Kolmogorov–Smirnov statistic [Darling, 1957] (Equation 5.4), where $\sup_{\mathbf{x}}$ is the supremum of the set of distances. The Kolmogorov–Smirnov statistic takes the largest absolute difference between the two CDFs across all $\mathbf{x}$ values,

$$\sup_{\mathbf{x}} |F_{\mathbf{z}^\pi}(\mathbf{x}) - F_{\mathbf{z}^{\pi'}}(\mathbf{x})|. \tag{5.4}$$

The Kolmogorov–Smirnov statistic returns a minimum value of 0 and a maximum value of 1. If two CDFs are equal, then the Kolmogorov–Smirnov statistic will return a value of 0.

The coverage ratio is then defined as:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{5.5}$$

where $precision = |\mathcal{F} \cap_\epsilon E|/|\mathcal{F}|$ indicating the fraction of optimal solutions among the retrieved solutions, and the $recall = |\mathcal{F} \cap_\epsilon E|/|E|$ indicating the fraction

arm_1

P(Arm 1 = **R**)	**R**
0.4	(0, 1)
0.6	(5, 4)

arm_2

P(Arm 2 = **R**)	**R**
0.85	(1, 0)
0.15	(3, 2)

arm_3

P(Arm 3= **R**)	**R**
0.75	(2, 0)
0.25	(4, 2)

arm_4

P(Arm 4 = **R**)	**R**
0.8	(0, 1)
0.2	(1, 2)

arm_5

P(Arm 5 = **R**)	**R**
0.7	(2, 0)
0.3	(4, 5)

Table 5.1: A MOMAB with 5 arms where selecting an arm has two outcomes and two objectives.

of optimal instances that have been retrieved over the total amount of optimal solutions [Yang et al., 2019].

5.2.3 Empirical Evaluation

MOTDRL is evaluated using two newly defined MOMAB settings. First, a traditional MOMAB setting with stochastic rewards is defined and is used to evaluate MOTDRL. Second, a newly proposed MOMAB problem domain, known as the Vaccine Recommender System (VRS) environment, is defined and used to evaluate MOTDRL.

5.2.3.1 Multi-Objective Multi-Armed Bandit Environment

To evaluate MOTDRL, a bi-objective MOMAB with five arms is considered. Table 5.1 outlines the number of possible outcomes obtainable when selecting a given arm and the corresponding probabilities. Table 5.1 is unknown to the agent, and the agent aims to learn each distribution per arm and prune the ESR dominated arms from consideration. In the MOMAB setting, the *ESR set* is known a priori where the return distributions for arm_1 and arm_5 are ESR dominant. Therefore, the *ESR set* only contains arm_1 and arm_5. To evaluate MOTDRL in a MOMAB environment, the following hyperparameters are set: $R_{min} = 0$, $R_{max} = 10$, $D = 2$, $\beta = 5$ and $|E^*| = 2$. To compute the coverage ratio, ϵ is selected as follows: $\epsilon = 0.01$. All experiments in this setting are averaged over 10 runs, where each experiment lasts for $200,000$ episodes.

MOTDRL is able to learn the underlying return distributions for each arm in the MOMAB setting. Using the return distributions for each arm, MOTDRL can learn the *ESR set* in the MOMAB environment. In Figure 5.2, the coverage ratio is presented as the F_1 score. MOTDRL converges to the optimal F_1 score of 1.

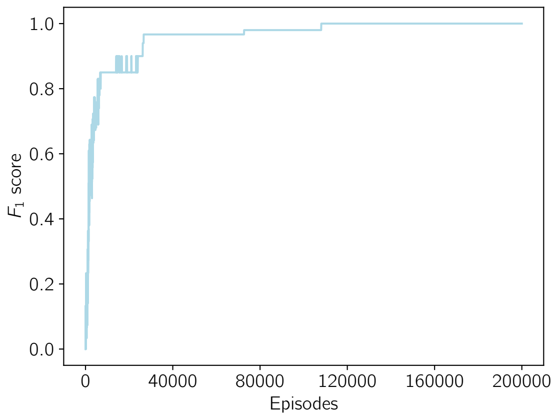


Figure 5.2: Results from the MOMAB environment. MOTDRL is able to learn the *ESR set* as MOTDRL converges to the optimal coverage ratio since the F_1 score reaches the maximum possible value of 1.

MOTDRL converges to the optimal F_1 score after $150,000$ episodes. An optimal F_1 score can only be achieved when all policies in the *ESR set* have been learned by the agent.

The learned *ESR set* contains two arms: arm_1 and arm_5. Both arm_1 and arm_5 are ESR dominant and, therefore, any user with a monotonically increasing utility function would prefer arm_1 or arm_5 over all other available arms in the MOMAB problem. Given the utility-based perspective is followed, MOTDRL will return the *ESR set* to the user during the selection phase. In practice, a user will select a policy form the *ESR set* which best reflects their preferences and the selected policy will be executed.

Given ESR dominance is a new solution concept, Figure 5.3, Figure 5.4, and Figure 5.5 are utilised to give the reader some intuition about ESR dominance. Figure 5.3 displays the return distributions in the *ESR set* learned by MOTDRL as heatmaps. Each heatmap in Figure 5.3 corresponds to the probabilities highlighted for arm_1 (left) and arm_5 (right) in Table 5.1.

Figure 5.4 displays the CDFs for each return distribution in the *ESR set* learned by MOTDRL. The CDF is used to calculate ESR dominance and the CDFs in Figure 5.4 correspond to the CDFs of arm_1 (left) and arm_5 (right) in Table 5.1.

Figure 5.5 describes how $arm_1 \not\succ_{ESR} arm_5$ and $arm_5 \not\succ_{ESR} arm_1$ given the CDFs for arm_1 and arm_5 intersect at multiple points.

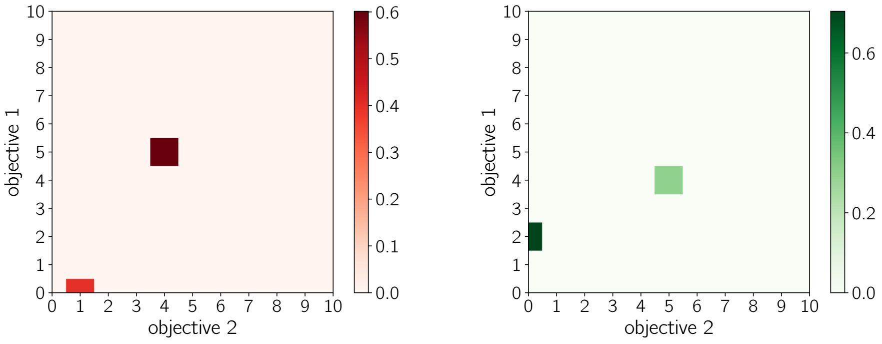


Figure 5.3: Heatmaps for each return distribution in the *ESR set* learned by MOTDRL in the MOMAB environment. The left heatmap describes the return distribution for arm_1 learned by MOTDRL and the right heatmap describes the return distribution for arm_5 learned by MOTDRL.

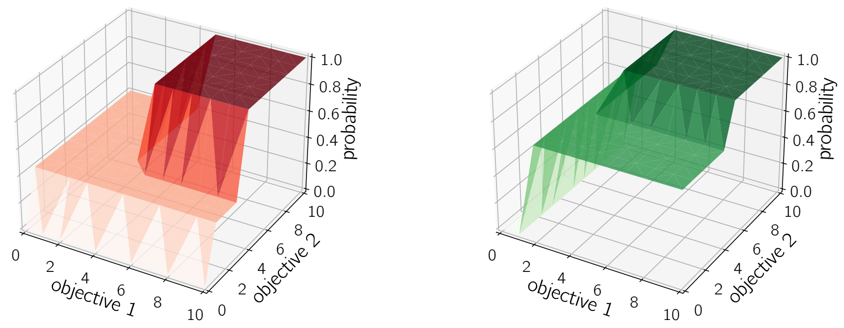


Figure 5.4: CDFs for each policy in the ESR set learned by MOTDRL in the MOMAB environment. The left figure describes the CDF for arm_1 learned by MOTDRL and the right figure describes the CDF for arm_5 learned by MOTDRL.

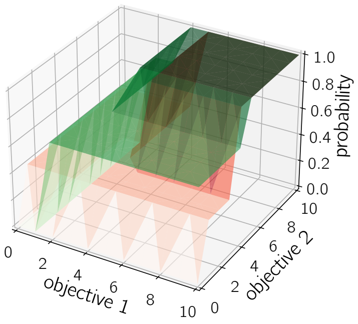


Figure 5.5: The CDFs for arm_1 and arm_5 intersect at multiple points. Therefore, as per Definition 21: $arm_1 \not\succ_{ESR} arm_5$ and $arm_5 \not\succ_{ESR} arm_1$.

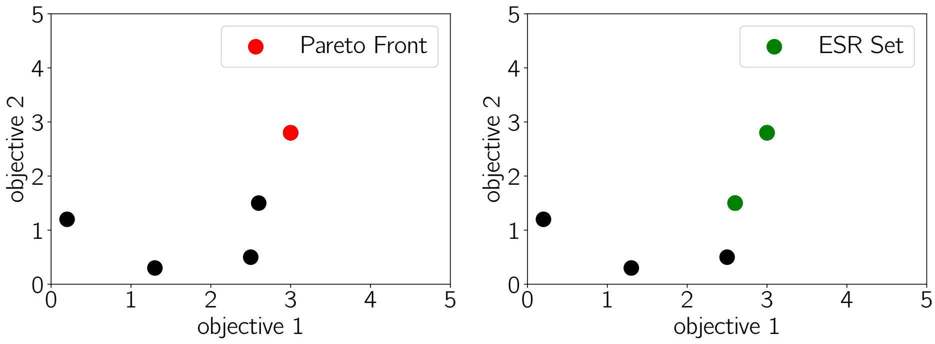


Figure 5.6: The policies on the Pareto front (left) are different from the policies in the *ESR set* (right). In this case, a policy that is in the *ESR set* is not on the Pareto front. This figure illustrates why SER methods cannot be used to learn the *ESR set*.

Figure 5.6 highlights why the choice of optimality criteria must be taken into consideration for MODeM when the utility function of the user is unknown. A number of SER methods use Pareto dominance to determine a partial ordering over policies. The Pareto dominant policies, or Pareto front, are then returned to the user. To determine the Pareto front [Pareto, 1896], the expectations of each arm in the MOMAB setting are calculated and the Pareto dominant policies are determined.

In Figure 5.6 the policies on the Pareto front (left) have been highlighted in red; all other policies are Pareto dominated. In the MOMAB environment outlined in Table 5.1, the Pareto front consists of a single policy. Figure 5.6 (right) displays the expected value vectors of the policies in the *ESR set*, highlighted in green. By comparing both plots in Figure 5.6, it is clear that the *ESR set* contains an extra policy. Therefore, in some settings, certain policies that are optimal under the ESR criterion are dominated under the SER criterion. Figure 5.6 highlights the importance of selecting the correct optimality criterion when learning. If SER methods are used to compute a set of optimal policies in scenarios where the ESR criterion should be used, it is possible a sub-optimal policy may be selected by the user at decision time. This may have adverse affects when applying multi-policy multi-objective methods in real-world decision making settings.

5.2.3.2 Vaccine Recommender System

To illustrate a potential real-world use case for the ESR criterion and ESR dominance, a new MOMAB environment known as the Vaccine Recommender System (VRS) is defined. For example, in a medical setting a doctor may only have one opportunity to select a treatment for a patient. In this case, it is necessary to optimise under the ESR criterion.

Consider the following scenario: a patient is travelling to another country where it is required to be vaccinated for a specific disease to gain entry to the country. There are five available vaccines, however, each vaccine will have varying side effects (safety rating) and effectiveness. This problem has two objectives: safety and effectiveness. Both objectives are ranked from 0 to 5, with 0 being the worst rating and 5 being the best rating. None of the available vaccines are 100% effective at preventing the disease. When taking each vaccine, there is a chance of different outcomes occurring. For example, there is a chance of having severe side effects (low safety rating) and a chance of the vaccine providing the required immunity to the disease (high effectiveness rating). Table 5.2 outlines each vaccine and the probability of each outcome occurring after taking the vaccine. Table 5.2 is unknown to the agent, and the agent aims to learn each distribution per vaccine and prune the ESR dominated vaccines from consideration.

Vaccine 1 (V_1)

P(V_1= **R**)	**R**
0.05	(2, 0)
0.05	(2, 1)
0.1	(3, 2)
0.8	(4, 2)

Vaccine 2 (V_2)

P(V_2= **R**)	**R**
0.1	(0, 0)
0.1	(1, 1)
0.5	(2, 0)
0.3	(2, 1)

Vaccine 3 (V_3)

P(V_3= **R**)	**R**
0.1	(1, 0)
0.1	(1, 3)
0.2	(3, 4)
0.6	(5, 4)

Vaccine 4 (V_4)

P(V_4= **R**)	**R**
0.1	(1, 0)
0.4	(2, 1)
0.4	(3, 1)
0.1	(3, 2)

Vaccine 5 (V_5)

P(V_5= **R**)	**R**
0.8	(0, 0)
0.05	(1, 1)
0.05	(1, 2)
0.1	(4, 0)

Table 5.2: A group of available vaccines that have varying outcomes. Some vaccines have a higher chance of side effects (low safety rating), while others are more effective at providing immunity. The objectives are ordered as follows: $\mathbf{R}$ = (safety, effectiveness).

Given the utility function of the user is unknown, the MOTDRL algorithm is used to learn the underlying return distributions for each vaccine in Table 5.2 and determine the *ESR set*. Once MOTDRL has finished learning a set of optimal polices, in this case the *ESR set* is returned to the user. When the user's utility function becomes known, a vaccine that maximises the user's utility function can be selected from the *ESR set* by the user.

The *ESR set* for the VRS environment is known a priori. The return distributions for V_1 and V_3 are ESR dominant and, therefore, V_1 and V_3 are the only distributions in the *ESR set*. The VRS environment, has five arms where each arm corresponds to a vaccine in Table 5.2. To evaluate MOTDRL in a VRS environment, the following hyperparameters are used: $\mathbf{R}_{min} = 0$, $\mathbf{R}_{max} = 10$, $D = 2$, $\beta = 5$ and $|E^*| = 2$. All experiments in this setting are averaged over 10 runs and each experiment lasts $200,000$ episodes. To compute the coverage ratio, the ϵ parameter is set as follows: $\epsilon = 0.01$.

After sufficient sampling, MOTDRL is able to learn the underlying return distributions for each arm in the VRS environment. Given return distributions can be used to give a partial ordering over policies, MOTDRL can use the return distributions for each arm to compute the *ESR set* in the VRS environment. In Figure 5.7, the coverage ratio as the F_1 score is presented. MOTDRL converges to the optimal F_1 score after $120,000$ episodes. Given MOTDRL converges to the optimal F_1 score, it is clear MOTDRL is able to learn the *ESR set.*

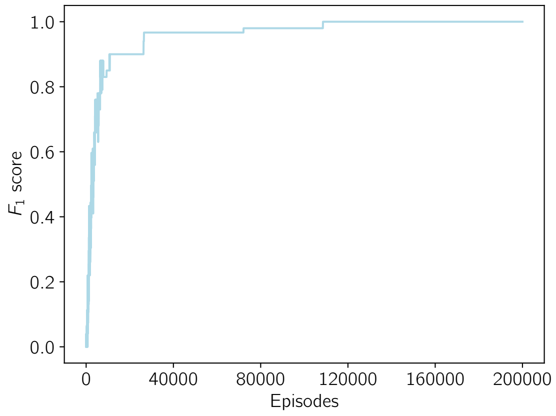


Figure 5.7: Results from the VRS environment. MOTDRL is able to learn the full *ESR set* as it converges the optimal F_1 score of 1.

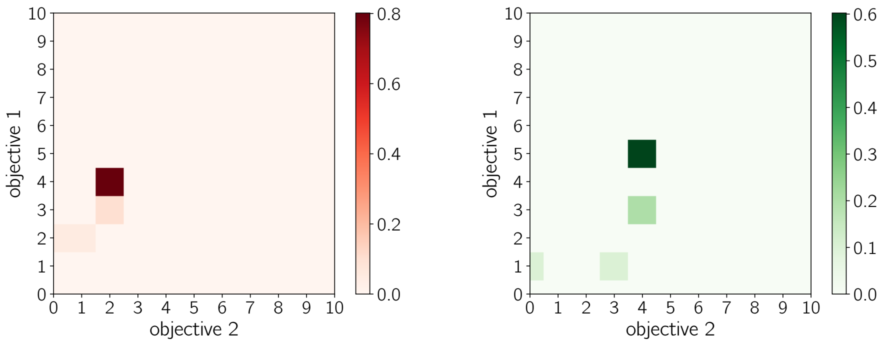
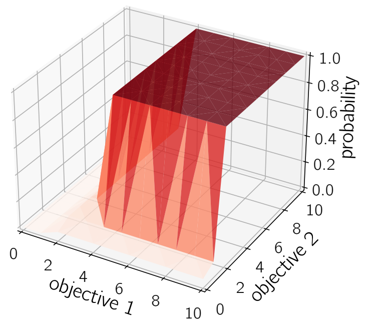


Figure 5.8: Heatmaps for each policy in the ESR set learned by MOTDRL. The left heatmap describes the distribution for V_1 learned by MOTDRL and the right heatmap describes the distribution for V_3 learned by MOTDRL.



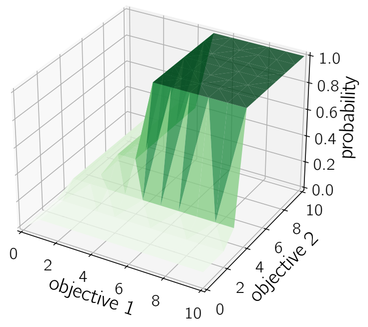


Figure 5.9: CDFs for each policy in the ESR set learned by MOTDRL in the VRS environment. The left figure describes the CDF for V_1 learned by MOTDRL and the right figure describes the CDF for V_3 learned by MOTDRL.

In practice, once learning has completed, MOTDRL returns the learned *ESR set* for the VRS environment to the user. The learned *ESR set* contains two vaccines; V_1 and V_3. Both vaccines in the *ESR set* are ESR dominant. Moreover, a user with a monotonically increasing utility function will prefer either V_1 or V_3 over all other vaccines in the VRS environment.

Figure 5.8 and Figure 5.9 are presented to give the reader some intuition about ESR dominance. Figure 5.8 presents heatmaps to represent the policies in the *ESR set* learned by MOTDRL. Each heatmap represents a return distribution learned by MOTDRL and shows the return vectors and the corresponding probabilities. Each heatmap in Figure 5.8 corresponds to the probabilities highlighted for V_1 (left) and V_3 (right) in Table 5.2. Figure 5.9 displays the policies in the *ESR set* learned by MOTDRL and their corresponding CDFs. Each CDF in Figure 5.9 corresponds to the CDFs of the underlying return distributions of V_1 and V_3 in Table 5.2.

5.2.4 Discussion

In this section, a novel *multi-objective distributional tabular reinforcement learning (MOTDRL)* algorithm is proposed that can compute a set of optimal policies for the ESR criterion. MOTDRL learns a distribution over the returns for each arm in MOMAB settings. By utilising a distributional approach, MOTDRL is able to learn the *ESR set* in each evaluation setting. MOTDRL is the first algorithm that can learn the *ESR set* in MOMAB settings.

During experimentation, it was found that the Pareto front and the *ESR set* can contain different policies. Therefore, the choice of optimality criterion is important, given policies that are sub-optimal for the SER criterion may be optimal under the ESR criterion. Selecting the wrong optimality criterion can lead to the user selecting a sub-optimal solution at decision time.

Furthermore, a novel evaluation metric for the ESR criterion is also defined. However, the proposed evaluation metric can only be used in settings where the *ESR set* is known a priori, and therefore its application is limited to settings where the full *ESR set* is known.

5.3 Solving Multi-Objective Markov Decision Processes for the Expected Scalarised Returns

In this section a novel *multi-objective distributional value iteration (MODVI)* algorithm is proposed that can compute a set of optimal policies for the ESR criterion in sequential decision making settings.

5.3.1 Multi-Objective Distributional Value Iteration

To compute a set of optimal policies for sequential MODeM scenarios under the ESR criterion when the utility function of a user is unknown, a novel *multi-objective distributional value iteration (MODVI)* algorithm is proposed. MODVI maintains sets of return distributions for each state and uses ESR dominance to compute a set of non-dominated return distributions, known as the *ESR set*.

As previously outlined, the state-of-the-art sequential MODeM algorithms use expected value vectors to compute sets of optimal policies [Wang and Sebag, 2012; Wiering and de Jong, 2007; White, 1982]. However, expected value vectors can only be used when optimising for the SER criterion. Therefore, to compute a set of optimal polices for the ESR criterion, expected value vectors must be replaced with return distributions. Generally, expected value MODeM algorithms utilise the Bellman operator [Bellman, 1957a] to compute the expected value vectors for each state in sequential settings. Given the proposed approach is distributional, the distributional Bellman operator [Bellemare et al., 2017], $\mathcal{T}_D^\pi$, is adopted to update the return distribution for each state-action pair:

$$\mathcal{T}_D^\pi \mathbf{z}(s,a) \overset{D}{=} \mathbf{r}_{s,a} + \gamma \ \mathbf{z}(s',a'). \tag{5.6}$$

To represent a return distribution in multi-objective settings, a multivariate categorical distribution similar to the distributions used by Reymond et al. [2021]

and Bellemare et al. [2017] is used. The categorical distribution is paramaterised by a number of atoms, $N \in \mathbb{N}$, where the distribution has a dimension per objective, n. The atoms outline the width of each category and are bounded by the minimum returns, $\mathbf{R}_{min}$, and maximum returns, $\mathbf{R}_{max}$. The multivariate categorical distribution has a set of atoms defined as follows [Reymond et al., 2021]:

$$\{\mathbf{z}_{i \dots k} = (\mathbf{R}_{\min_0} + i\Delta\mathbf{z}_0, \dots, \mathbf{R}_{\min_n} + k\Delta\mathbf{z}_n) : 0 \leq i < N, \dots, 0 \leq k < N\}, \quad (5.7)$$

where each objective, n, has a separate $\mathbf{R}_{\min_b}, \mathbf{R}_{\max_b}$ for $0 < b \leq n$ and $\Delta\mathbf{z} = \frac{\mathbf{R}_{max} - \mathbf{R}_{min}}{N-1}$. The distribution is a set of discrete categories, N, where each category, p_i, represents the probability of receiving a return [Reymond et al., 2021]. To ensure the distribution is an accurate representation of the returns of the execution of a policy, it is crucial a number of atoms are selected to sufficiently cover the range of values from $\mathbf{R}_{\min}$ and $\mathbf{R}_{\max}$. For example, if $\gamma = 1$ and reward values are expected to be integers in the range $\mathbf{R}_{\min} = [0, 0]$ to $\mathbf{R}_{\max} = [1, 10]$, $N = 11$ is the required value to ensure that the distribution is represented without aliasing between different reward levels.

To update the multivariate categorical distribution, the state space, action space and reward function of the model are utilised. During an update of the multivariate categorical distribution, each atom, j, is evaluated for each objective. To update the return distribution, $\mathbf{z}_s$, for state s, the distributional Bellman update $\hat{\mathcal{T}}\mathbf{z}_{s,j} = \mathbf{r}_{s,a,s'} + \gamma\mathbf{z}_{s',j}$ is computed for each atom j, for a given reward $\mathbf{r}_{s,a,s'}$ and return distribution, $\mathbf{z}_{s'}$, for state s'. Next, the probability, p, for the atom, j, of the return distribution, $p_j(\mathbf{z}_{s'})$, in state s', is distributed to the corresponding atom of the updated return distribution, z_s, for state s. Therefore, the return distribution, $\mathbf{z}_s$, for state s is equivalent to the return distribution, $\mathbf{z}_{s'}$, in state s', shifted relative to the reward, $\mathbf{r}_{s,a,s'}$.

At each iteration, k, of MODVI, for each state, s, and action, a, a set of optimal return distributions is backed up once. In Equation 5.8, the Bellman operator has been replaced with the distributional Bellman operator [Bellemare et al., 2017],

$$\mathbf{Q}_{k+1}(s,a) \leftarrow \bigoplus_{s'} T(s'|s,a)[\mathbf{r}_{s,a,s'} + \gamma\mathbf{Z}_k(s')] \quad (5.8)$$

where $\mathbf{Q}_{k+1}(s,a)$ and $\mathbf{Z}_k(s')$ represent sets of return distributions, $\oplus$ denotes the cross-sum between sets of return distributions, and $T(s'|s,a)$ represents the probability of transitioning to state s' from state s after taking action a.

During a distributional Bellman backup, each return distribution, $\mathbf{z}_{s'}$, in the set $\mathbf{Z}_k(s')$, is updated with the reward, $\mathbf{r}_{s,a,s'}$, for action, a, in state, s, as follows: $\{\mathbf{r}_{s,a,s'} + \gamma\mathbf{z}_{s'} : \forall \mathbf{z}_{s'} \in \mathbf{Z}_k(s')\}$. Each updated return distribution in the set for state s' is then multiplied by the transition probability, $T(s'|s,a)$. The cross sum for each resulting set of updated return distributions is computed for each next possible next state, s'. The cross sum between two sets of return distributions, $\mathbf{X} \bigoplus \mathbf{Y}$, is defined as follows: $\{\mathbf{x} + \mathbf{y} : \mathbf{x} \in \mathbf{X} \wedge \mathbf{y} \in \mathbf{Y}\}$, where $\mathbf{x}$ and $\mathbf{y}$ are *return distributions*.

To compute a set of ESR non-dominated policies for each state, the ESRPrune algorithm (Algorithm 14) is utilised which computes a set of ESR non-dominated policies by removing ESR dominated return distributions from a given set.

$$\mathbf{Z}_{k+1}(s) \leftarrow \text{ ESRPrune}\left(\bigcup_a \mathbf{Q}_{k+1}(s,a)\right) \tag{5.9}$$

Equation 5.9 calculates the set of return distributions for a given state, s, by taking the union of each set of return distributions over each action, a. The resulting set of return distributions is then passed to the ESRPrune algorithm as input.

Algorithm 17: Multi-Objective Distributional Value Iteration

```
1 Initialise all return distributions and sets
2 while not converged do
3     for $s \in S$ do
4         for $a \in A$ do
5             $\mathbf{Q}_{k+1}(s,a) \leftarrow \bigoplus_{s'} T(s'|s,a)[\mathbf{R}(s,a,s') + \gamma \mathbf{Z}_k(s')]$
6         end
7         $\mathbf{Z}_{k+1}(s) \leftarrow$ ESRPrune $\left(\bigcup_a \mathbf{Q}_{k+1}(s,a)\right)$
8     end
9 end
```

Algorithm 17 describes the MODVI algorithm[2]. On initialisation of MODVI, a set of return distributions is generated for each state-action pair. For infinite horizon settings, each set contains a single return distribution that is randomly initialised, where an atom is selected at random and a probability mass of 1.0 is assigned to that atom. In finite horizon settings each return distribution is initialised by assigning a probability mass of 1.0 to the atom which corresponds to the return $[0,0]$. During each iteration of MODVI, a set of return distributions is computed (Algorithm 17, Line 5) for each state, s and action, a. The union of the resulting sets of return distributions is then passed to the ESRPrune algorithm to remove the dominated return distributions. Once ESRPrune (Algorithm 17, Line 7) has been executed for the given iteration of MODVI, a set of non-dominated return

[2]Algorithm 17 describes MODVI for infinite horizon settings. However, it is trivial to alter MODVI for finite horizon settings.

distributions is backed up for the state s. Once MODVI has converged, a set of ESR non-dominated policies, or the *ESR set*, is available at the start state, s_0.

5.3.2 Empirical Evaluation

In this section, MODVI is evaluated using two multi-objective benchmark problems form the literature.

5.3.2.1 Space Traders

First, MODVI is evaluated using a multi-objective benchmark problem known as Space Traders [Vamplew et al., 2020, 2021a]. Space Traders is a problem with nine policies and a small number of returns per policy. Therefore, it is possible to visualise each policy in the *ESR set*, illustrating how policies can be returned to a user during the selection phase in practice. Of course, for larger problems, the user could select subsets of the policies to visualise and compare.

Space Traders has two timesteps, two non-terminal states, and three available actions per state. In Space Traders, an agent must deliver cargo from its home planet (state A) to some destination planet (state B) and then return home. While delivering the cargo, the agent must avoid being intercepted by space pirates. An agent acting in the Space Traders environment aims to complete the mission and minimise time. An agent receives a reward of 1 for returning home to planet (state A) and completing the mission, and at all other states the agent receives a reward of 0 for mission success. After each action, the agent receives a negative reward corresponding to the time taken to reach the next planet. Finally, after taking each action there is a probability the agent will be intercepted by space pirates. If the agent is intercepted by space pirates, the agent will receive a reward of 0 for mission success, a negative time penalty, and the episode will terminate. All remaining implementation details for the Space Traders environment are available in the Appendix A.4.

MODVI is run using the following hyperparameters: $\gamma = 1$, $N = 23$, $\mathbf{R}_{min} = [0, -22]$ and $\mathbf{R}_{max} = [1, 0]$. Table 5.3 outlines the six return distributions in the computed *ESR set*. Figure 5.10 plots the expected value vectors of each return distribution in the *ESR set* and also plots the expected value vectors for the Pareto front [Vamplew et al., 2021a]. It is important to note that the *ESR set* for Space Traders contains a policy that is not present on the Pareto front. The Pareto front is a set of optimal policies for the SER criterion. As demonstrated earlier, certain policies that are optimal under the ESR criterion are not optimal under the SER criterion. In real-world decision making, incorrectly selecting an optimality

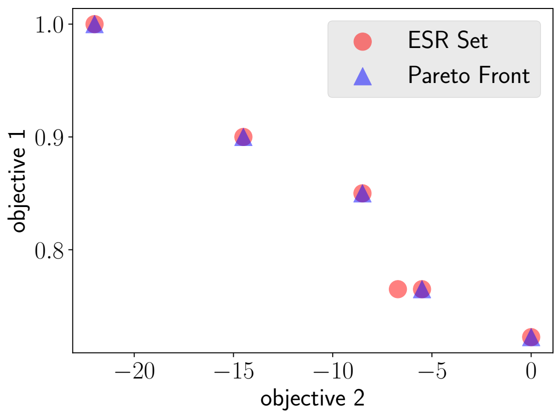


Figure 5.10: The expected value vectors of the return distributions in the *ESR set* (red) are plotted against the expected value vectors of the Pareto front (blue).

criterion can lead to sub-optimal performance, given some optimal policies may not be returned to the user.

During the selection phase, visualisations, like Figure 5.10, are returned to the user to aid in their decision making. However, in Figure 5.10, the details of the return distributions for each policy in the *ESR set* are lost. Computing expected value vectors for each return distribution reduces the information available about a policy, given the information about each individual return of a policy is no longer available. As already highlighted, under the ESR criterion the utility of a user is derived from a single execution of a policy. Therefore, it is crucial a user has sufficient information available at decision time, given a policy may only be executed once. Figure 5.11 visualises each potential return and the corresponding probability of the return distributions in the *ESR set*. In Figure 5.11, each return distribution has a shape, where the position of each shape corresponds to a return and the colour of each shape corresponds to the probability of receiving the return. Figure 5.12 presents the individual return distributions for each policy. In practice, a user would be able to choose which return distributions in the *ESR set* to display at a given moment, allowing the user to compare and contrast different policies individually. Figure 5.11 provides an intuitive aid which can be returned to a user when making decisions under the ESR criterion.

π	r_1	r_2	$P(r_1, r_2)$
π_1	1	-22	1.0
π_2	0	-1	0.1
	1	-16	0.9
π_3	0	-7	0.085
	0	0	0.15
	1	-8	0.765
π_4	0	0	0.15
	1	-10	0.85
π_5	0	0	0.2775
	1	0	0.7225
π_6	0	-6	0.135
	0	-1	0.1
	1	-6	0.765

Table 5.3: The return distributions in the *ESR set* for the Space Traders environment, with $\gamma = 1$.

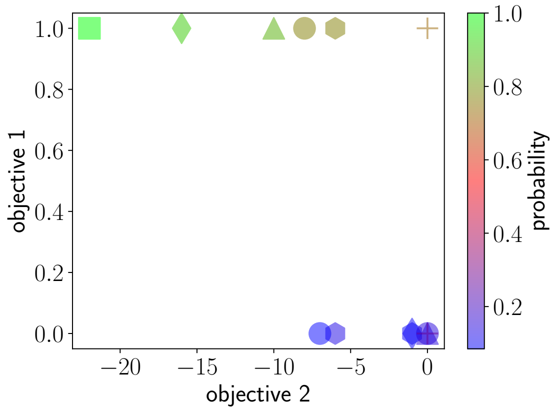


Figure 5.11: The return distributions in the *ESR set* computed by MODVI. Each shape corresponds to a computed policy in the *ESR set*, where the location of the shape corresponds to a return in the policy. Colours correspond to the probability of receiving the specific return when executing the policy.

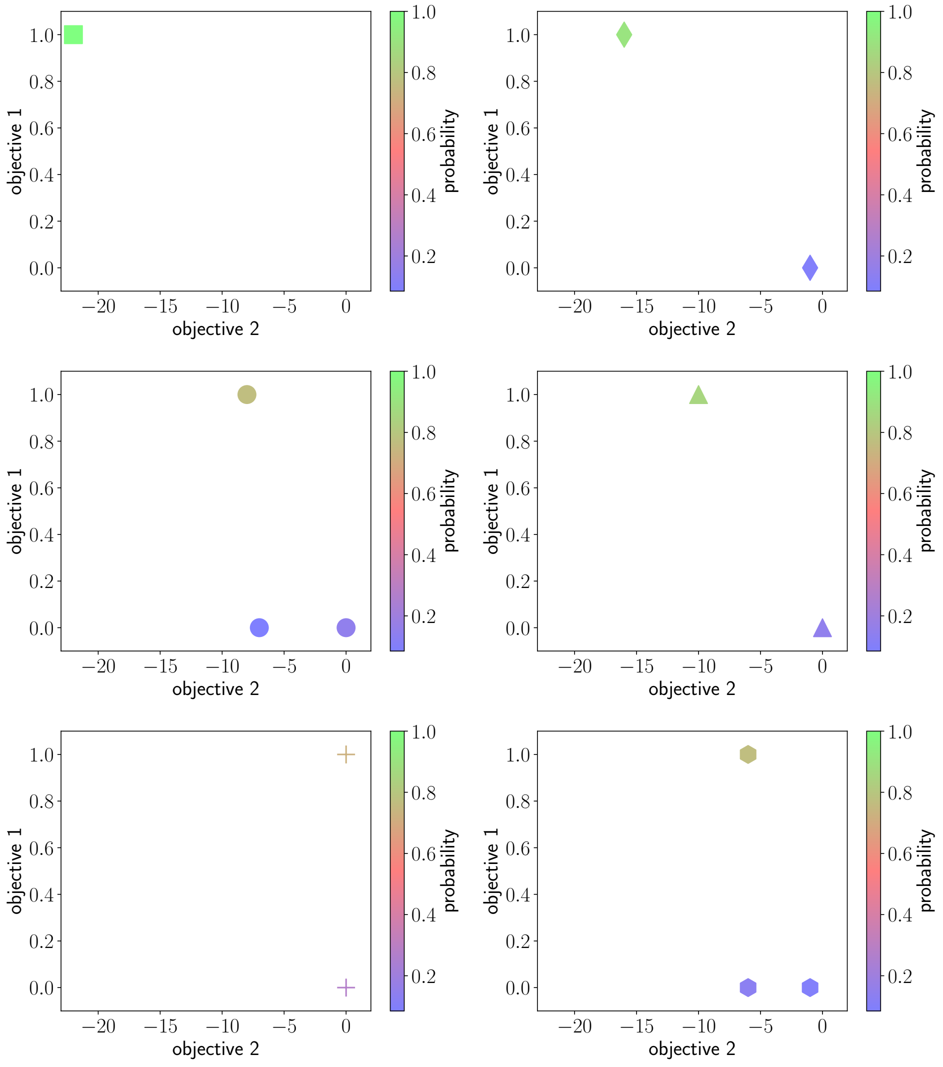


Figure 5.12: The return distributions for policy π_1, π_2, π_3, π_4, π_5, and π_6 computed by MODVI for Space Traders.

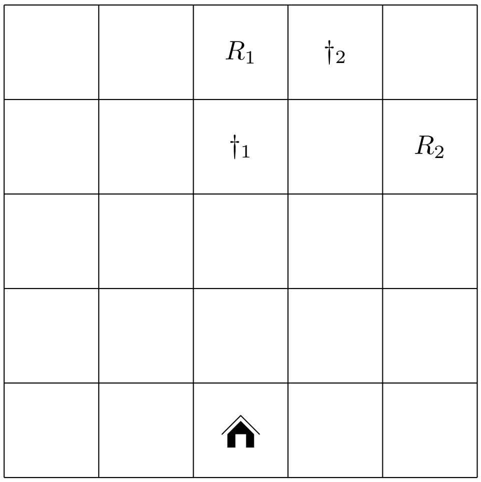


Figure 5.13: The grid for the Resource Gathering environment. $\dagger_1$ and $\dagger_2$ are enemy states. R_1 and R_2 are the resources that need to be gathered, before returning to the home state.

5.3.2.2 Resource Gathering

Next, MODVI is evaluated using the Resource Gathering benchmark [Barrett and Narayanan, 2008]. Resource Gathering is a multi-objective benchmark problem with intuitive trade-offs between objectives, motivating the need to consider the ESR criterion in real-world decision making. MODVI is evaluated on a four-objective version of Resource Gathering, where time is added as an objective. The Resource Gathering environment is shown in Figure 5.13. The agent starts in a home state and navigates the grid environment to collect the available resources (R_1 and R_2), while avoiding the enemy states ($\dagger_1$ and $\dagger_2$) before returning home again. At each timestep, the agent receives a reward of $[-1, 0, 0, 0]$. If the agent returns to the home state having gathered the available resources, the agent receives one of the following rewards: $[-1, 0, 10, 0]$ for collecting R_1, $[-1, 0, 0, 10]$ for collecting R_2, and $[-1, 0, 10, 10]$ for collecting R_1 and R_2. The agent must avoid the enemy states. If the agent enters an enemy state, there is a 0.1 chance the agent will be attacked. If the agent is attacked in an enemy state, the agent receives a reward of $[-10, -10, 0, 0]$. In this case, the agent also receives a time penalty for being attacked and the episode terminates.

π	r_1	r_2	r_3	r_4	$\mathrm{P}(r_1, r_2, r_3, r_4)$
π_1	-18	0	10	10	1.0
π_2	-12	0	10	0	1.0
π_3	-16	-10	0	0	0.1
	-14	0	10	10	0.9
π_4	-12	-10	0	0	0.1
	-16	0	10	10	0.9
π_5	-12	-10	0	0	0.1
	-10	0	10	0	0.9
π_6	-14	-10	0	0	0.09
	-12	-10	0	0	0.1
	-12	0	10	10	0.81
π_7	-14	-10	0	0	0.09
	-12	-10	0	0	0.1
	-8	0	10	0	0.81
π_8	-10	0	0	10	1.0

Table 5.4: The return distributions in the *ESR set* for the Resource Gathering environment, with $\gamma = 1$.

For Resource Gathering, the following hyperparameters were set for MODVI: $\gamma = 1$, $N = 25$, $\mathbf{R}_{min} = [-24, -24, -14, -14]$ and $\mathbf{R}_{max} = [0, 0, 10, 10]$. Table 5.4 outlines the return distributions in the *ESR set* for Resource Gathering. The *ESR set* contains eight policies, where each policy gathers one or both resources before returning home. An important aspect of the distributional approach applied by MODVI is that a user will have sufficient information about the trade-offs between each objective for each policy in the *ESR set*. For example, there is a clear trade-off between objectives in π_3 and π_6 in Table 5.4. When considering π_3, fourteen timesteps are taken to gather both resources and the agent enters one enemy state with a 0.1 chance of being attacked. When considering π_6, twelve timesteps are taken to gather both resources, but the agent must enter both enemy states, which poses 0.09 chance and 0.1 chance of being attacked.

Using a distributional approach ensures a user has sufficient information to understand the trade-offs between objectives across different policies. In Resource Gathering, a user looking to minimise time, while also being indifferent about being attacked, may select π_6 having fully understood the probabilities of being attacked. Therefore, having sufficient critical information available at decision time enables the user to make more informed decisions that could potentially better reflect

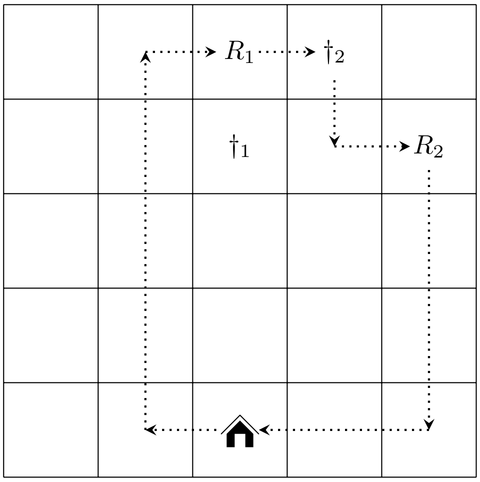


Figure 5.14: A potential path for policy π_3. The agent obtains both resources (R_1, R_2) and crosses enemy state $\dagger_2$ before returning home.

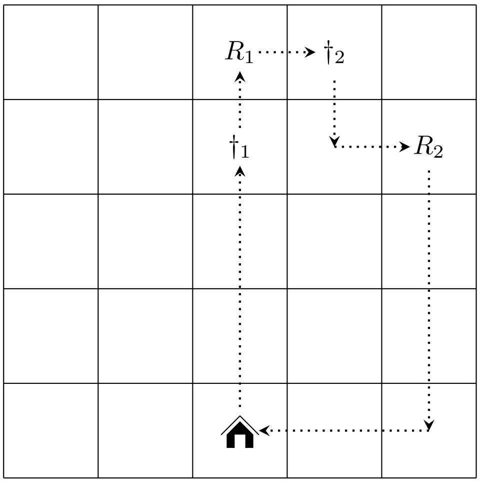


Figure 5.15: A potential path for policy π_6. The agent obtains both resources (R_1, R_2) and crosses both enemy states $(\dagger_1, \dagger_2)$ before returning home.

their preferences over objectives, when compared to expected value vector based methods.

Additionally, visual aids like those outlined in Figure 5.14 and Figure 5.15 can be presented to a user at decision time. Both visualisations illustrate a potential route the agent can take to achieve the return distribution associated with the highlighted policy. In Figure 5.14, a potential route for policy π_3 is outlined. When following this route the agent can collect both resources, while the agent avoids one enemy state but enters the other enemy state. Furthermore, Figure 5.15 presents a potential route the agent can take to obtain the return distribution for policy π_6. As previously highlighted, the agent obtains both resources but enters both enemy states. It is important to remember that the agent can execute the route for policy π_6 in less time when compared to π_3. However, there is a higher chance of being attacked. Using visual aids, like those presented in Figure 5.14 and Figure 5.15, and having the return distributions available can help the user select an appropriate policy at decision time. The visualisations presented in Figure 5.14 and Figure 5.15 differ from those presented in Section 5.3.2.1 and in certain circumstances may provide the user with a more intuitive understanding of the potential outcomes a policy may have.

5.3.3 Discussion

In this section, a novel *multi-objective distributional value iteration (MODVI)* algorithm is proposed that can compute a set of optimal policies for the ESR criterion. MODVI utilises return distributions, which replace expected value vectors in MODeM. MODVI is the first algorithm that can compute a set of optimal policies under the ESR criterion in sequential MODeM settings. MODVI is evaluated using two benchmark MOMDPs from the MODeM literature, and MODVI is shown to compute the *ESR set* in each setting. Because it is the first of its kind, MODVI opens up decision-theoretic planning for a key range of real-world problems.

As shown in the Space Traders environment (Figure 5.10) the *ESR set* and the Pareto front can be different for MOMDPs. Therefore, the choice of optimality criterion must be considered when computing sets of optimal policies. In this case, some policies that are optimal under the ESR criterion are sub-optimal under the SER criterion. When applying MODeM to real-world decision making problems, it is important the choice of optimality criterion is carefully considered.

5.4 Solving Multi-Objective Coordination Graphs for the Expected Scalarised Returns

In this section a novel *distributional multi-objective variable elimination (DMOVE)* algorithm is proposed that can compute a set of optimal policies for the ESR criterion in multi-agent settings, like multi-objective coordination graphs (MO-CoGs). However, before DMOVE can be applied, MO-CoGs must first be extended for the ESR criterion. DMOVE is then evaluated using benchmark MO-CoGs from the literature.

5.4.1 Multi-Objective Coordination Graphs for the Expected Scalarised Returns

The current literature on MO-CoGs focuses exclusively on the SER criterion [Rollón and Larrosa, 2006; Roijers et al., 2015]. As previously shown, methods that compute solution sets for the SER criterion cannot be used under the ESR criterion. Therefore, a set of optimal policies under the ESR criterion for MO-CoGs must be defined.

To determine a partial ordering over policies under the ESR criterion, ESR dominance can be used. To calculate ESR dominance a return distribution for each local payoff function must be calculated. For MO-CoGs, a return distribution, $\mathbf{z}$, is defined as the distribution over the returns of a local payoff function. Therefore, $\mathcal{Z}$ where $\mathcal{Z} = \{\mathbf{z}^1, ..., \mathbf{z}^l\}$ is a set of l, d-dimensional return distributions of local payoff functions. The joint payoff for all agents is the sum of local payoff return distributions: $\mathbf{z}(\mathbf{a}) = \sum_{e=1}^{l} \mathbf{z}^e(\mathbf{a}^e)$. The set of all possible joint action return distributions is denoted by $\mathcal{V}$.

By utilising ESR dominance a set of optimal policies under the ESR criterion for a MO-CoG can be defined as a set of ESR non-dominated global joint actions $\mathbf{a}$ and associated return distributions of local payoff functions, $\mathbf{z}(\mathbf{a})$, known as the *ESR set*:

Definition 27

The *ESR set* (ESR) of a MO-CoG, is the set of all joint actions and associated payoff return distributions that are ESR non-dominated,

$$ESR(\mathcal{V}) = \{\mathbf{z}(\mathbf{a}) \in \mathcal{V} \mid \nexists\, \mathbf{z}'(\mathbf{a}) \in \mathcal{V} : \mathbf{z}'(\mathbf{a}) \succ_{\text{ESR}} \mathbf{z}(\mathbf{a})\}. \tag{5.10}$$

5.4.2 Distributional Multi-Objective Variable Elimination

To solve MO-CoGs for the ESR criterion a novel *distributional multi-objective variable elimination (DMOVE)* algorithm is defined. DMOVE utilises return distributions and ESR dominance to compute the *ESR set*.

Generally, multi-objective variable elimination (MOVE) methods translate the problem to a set of value set factors [Roijers et al., 2013, 2015]. Given that under the ESR criterion return distributions must be utilised, the problem must first be translated to a set of return distribution set factors (RSF), f, where each RSF f^e is a function mapping local joint actions to set of payoff return distributions. On initialisation, each RSF is a singleton set containing a local actions payoff return distribution and is defined as follows:

$$f^e(\mathbf{a}^e) = \{\mathbf{z}^e(\mathbf{a}^e)\} \tag{5.11}$$

It is possible to describe the coordination graph as a bipartite graph whose nodes, $\mathcal{D}$, are both agents and components of a factored RSF, and an edge $(i, f^e) \in \mathcal{E}$, if and only if agent i influences component f^e. It is important to note an agent node is joined by an edge to a factored RSF component if the agent influences the RSF. Therefore, the dependencies can be described by setting $\mathcal{E} = \{(i, f^e) | i \in \mathcal{D}^e\}$. To compute an *ESR set*, DMOVE treats a MO-CoG as a series of local sub-problems. DMOVE manipulates the set of RSFs by computing local *ESR set*s (LESRs) when eliminating agents. To calculate LESRs, a set of neighbouring RSFs, f_i, must first be defined.

Definition 28

The set of neighbouring RSFs f_i of agent i is the subset of all RSFs which agent i influences.

Each agent i has a set of neighbour agents, n_i, where each agent in n_i influences one or more RSFs in f_i. To compute a LESR, all return distributions for the sub-problem must first be considered, $\mathcal{V}_i$, by calculating the following:

$$\mathcal{V}_i(f_i, \mathbf{a}^{n_i}) = \bigcup_{a^i} \bigoplus_{f^e \in f_i} f^e(\mathbf{a}^e), \tag{5.12}$$

where $\oplus$ is the cross sum of sets of return distributions. In Equation 5.12 all actions in $\mathbf{a}^{n_i}$ are fixed, apart from a^i, and $\mathbf{a}^e$ is formed from a^i and the appropriate part of $\mathbf{a}^{n_i}$. Once $\mathcal{V}_i(f_i, \mathbf{a}^{n_i})$ has been computed for agent i, a LESR can be calculated by applying an ESR pruning operator. Therefore, a LESR is defined as follows:

Definition 29

A local ESR set, a LESR, is the ESR non-dominated subset of $\mathcal{V}_i(f_i, \mathbf{a}^{n_i})$:

$$LESR_i(f_i, \mathbf{a}^{n_i}) = ESR\ (\ \mathcal{V}_i(f_i, \mathbf{a}^{n_i})\ ), \tag{5.13}$$

When calculating a LESR, a new RSF, f_{new}, is generated, which is conditioned on the actions of the agents in n_i:

$$\forall\ \mathbf{a}^{n_i}\ \ f_{new}(\mathbf{a}^{n_i}) = LESR_i(f_i, \mathbf{a}^{n_i}). \tag{5.14}$$

The set of RSFs, f, must then be updated with f_{new}. Therefore, the RSFs in f_i are removed from f and f is updated with f_{new}. To do so, a new operator, known as the eliminate operator, is defined as follows:

$$\mathsf{eliminate}(f, i) = (f \setminus f_i) \cup \{f_{new}(\mathbf{a}^{n_i})\}. \tag{5.15}$$

Computing Equation 5.14 and Equation 5.15 removes agent i from the coordination graph. Therefore, the nodes and edges of the coordination graph are updated, where the edges for each agent in n_i are now connected to the new RSF, f_{new}.

Utilising the steps outlined above, the DMOVE algorithm (Algorithm 18) can be defined. DMOVE first translates the problem into a set of RSFs and removes agents in a predefined order, $\mathbf{q}$. DMOVE calls the algorithm eliminate which computes local *ESR sets* by pruning, and updates the set of RSFs. Once all agents have been eliminated, the final factor from the set of RSFs, f, is retrieved, pruned, and returned. The resulting set, $\mathcal{S}$, contains ESR non-dominated return distributions, known as the *ESR set*, and the associated joint actions. DMOVE only executes a forward pass and calculates joint actions using a tagging scheme [Roijers et al., 2015].

Algorithm 18: DMOVE ($\mathcal{P}$, f, prune1, prune2, prune3, q)

```
1 Input: $\mathcal{P}$ ← A set of local payoff functions; $\mathbf{q}$ ← an elimination order
2 while $a^{n_i} \in \mathcal{A}^{n_i}$ do
3     $i$ ← q.dequeue()
4     $f$ ← eliminate($f, i$, prune1, prune2)
5 end
6 $f$ ← retrieve final factor from $f$
7 $\mathcal{S}$ ← prune3($f(a_\emptyset)$)
8 Return $\mathcal{S}$
```

The eliminate algorithm (Algorithm 19) calculates the LESRs and updates the set of RSFs, f. To calculate the LESRs, the function CalculateLESR is utilised. CalculateLESR$_i$ is defined as follows:

$$\text{CalculateLESR}_i(f_i, \mathbf{a}^{n_i}, \text{prune1,prune2}) = \text{prune2}\left(\bigcup_{a^i} \overset{*}{\bigoplus_{f^e \in f_i}} f^e(\mathbf{a}^e)\right), \tag{5.16}$$

where $\overset{*}{\bigoplus}$ is the prune and cross-sum operator defined by Roijers et al. [2015]. The CalculateLESR$_i$ function prunes at two different stages; prune1 is applied after the cross-sum has been computed and prune2 is applied after the union over all a^i, which results in incremental pruning [Cassandra et al., 1997]. Both

Algorithm 19: eliminate(f, i, prune1, prune2)

```
1 Input: $f$ ← A set of RSFs; $i$ ← an agent
2 $n_i$ ← the set of neighbouring agents of $i$; $f_i$ ← the subset of $f$ that $i$
    influences; $f_{new}(a^{n_i})$ ← a new RSF
3 for $a^{n_i} \in \mathcal{A}^{n_i}$ do
4     $f_{new}(a^{n_i}) \leftarrow \text{CalculateLESR}_i(f_i, a^{n_i}, \text{prune1,prune2})$
5 end
6 $f \leftarrow (f \setminus f_i) \cup \{f^{new}\}$
7 Return $f$
```

DMOVE (Algorithm 18) and eliminate (Algorithm 19) are paramaterised by pruning operators. To compute the *ESR set*, the pruning algorithm ESRPrune (see Algorithm 14) is used as each pruning operator.

To represent the distribution for each RSF, f^e, the multivariate categorical distribution defined in Equation 5.7 is utilised. Given DMOVE is a planning algorithm, the returns and the corresponding probabilities for each joint local action are known a priori. Therefore, it is possible to initialise a multi-variate return distribution for each RSF, f^e, with the relevant probabilities for each potential return. As previously defined, a multivariate categorical distribution is parameterised by the maximum returns and minimum returns which must be specified before initialisation. Also, a number of atoms must be selected, where each atom maintains the probability for the corresponding return. By utilising this approach, the DMOVE algorithm can be used to compute the *ESR set* for MO-CoGs.

5.4.3 Empirical Evaluation

To evaluate DMOVE, two MO-CoGs from the literature are used. First, DMOVE is evaluated using a random MO-CoG [Roijers et al., 2015] in which it is possible to directly control all variables. Second, DMOVE is evaluated using a MO-CoG inspired by a more realistic scenario known as Mining Day [Roijers et al., 2013; Verstraeten et al., 2020].

5.4.3.1 Random MO-CoG

A random MO-CoG was first introduced by Roijers et al. [2015]. For random MO-CoGs it is possible to control all variables. Therefore, a random MO-CoG takes a number of parameters as input. To construct a random MO-CoG, the number of agents, n, the number of payoff dimensions, d, the number of local payoff functions, p, and the size of the action space for each agent, $|\mathcal{A}_i|$, are specified. However, the rewards for the random MO-CoG introduced by Roijers et al. [2015] were deterministic, therefore, the random MO-CoG must be extended to include stochastic rewards. To make the returns stochastic, an underlying probability distribution for each local action is generated that, when sampled, returns a vector reward. The number of outcomes per local action, o, is also configurable and the corresponding probabilities for each outcome are randomly selected, and naturally sum to 1. The values for the different objectives for each outcome in the probability distribution for each local payoff function are real numbers that are drawn independently and uniformly from the interval $[0, 5]$.

To evaluate DMOVE in a random MO-CoG, the random MO-CoG is initialised with the following hyperparameters: $n = 8$, $d = 2$, $p = 7$, $|\mathcal{A}|_i = 4$, and $o = 5$. Furthermore, DMOVE is initialised with the following hyperparameters: $\mathbf{R}_{min} = [0, 0]$, $\mathbf{R}_{max} = [50, 50]$, and $N = 51$.

The *ESR set* computed by DMOVE for the random MO-CoG contains 156 policies. Figure 5.16 plots the expected value vectors for each return distribution in the *ESR set*. Figure 5.16 contains a number of policies that would be sub-optimal under the SER criterion. Figure 5.16 shows similar results to those presented in Figure 5.10. However, in this instance, the results presented are for multi-agent settings, not single-agent sequential decision making problems. Therefore, selecting the correct optimality criterion in MODeM is also very important in multi-objective multi-agent decision making settings.

Plotting the expected value vectors for the *ESR set* removes the information about the distribution over the returns for each policy, but it is still important to outline how the policies in the *ESR set* could be presented to a user. A user could take a first look at the expected value vectors for the *ESR set* and then select policies of interest. The full distribution over the returns could be shown to the user for the

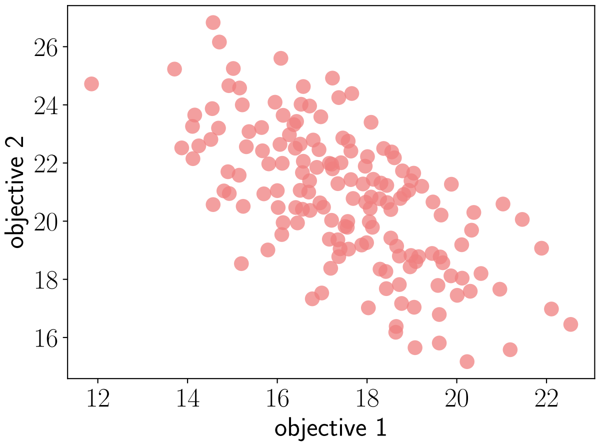


Figure 5.16: The expected value vectors of the return distributions in the *ESR set* for a random MO-CoG with stochastic rewards.

selected policies of interest. This could provide a more intuitive selection process when compared to showing the user all distributions individually.

Using DMOVE for the random MO-CoG, the Pareto front is also computed. To compute the Pareto front the PPrune algorithm [Roijers et al., 2015] is used as the prune1, prune2 and prune3 parameters for DMOVE. Therefore, DMOVE still computes the distribution over the returns, however, when pruning the expected value vector for each distribution is computed and the Pareto dominated expected value vectors are removed from consideration by PPrune. The PPrune algorithm is presented in Section A.5. In Figure 5.17 both the *ESR set* and the Pareto front are shown. As observed in Figure 5.10, the *ESR set* contains the full Pareto front along with excess policies that are Pareto dominated. Therefore, certain policies that are optimal under the ESR criterion are sub-optimal under the SER criterion. Figure 5.17 highlights how selecting the correct optimality criterion is important given optimal policies under the ESR criterion can be sub-optimal under the SER criterion. Such an observation was also made in Figure 5.6 and Figure 5.10 when using the MOTDRL and MODVI algorithms, which also holds in multi-agent settings. Figure 5.17 also highlights the flexibility of distributional methods given the Pareto front can be computed by simply switching the pruning operators. Furthermore, using DMOVE it may be possible to compute the convex hull by using the CPrune pruning algorithm [Roijers et al., 2015].

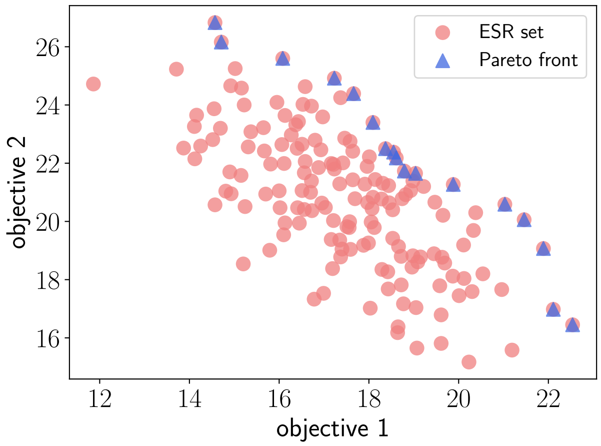


Figure 5.17: The expected value vectors of the return distributions in the *ESR set* (red) and the Pareto front (blue) for a random MO-CoG with stochastic rewards.

Additionally, two return distributions from the *ESR set* are shown in Figure 5.18 and Figure 5.19. While a random MO-CoG is not a realistic setting, it provides a good baseline to show the *ESR set* in multi-agent settings. Each distribution in the *ESR set* displays each individual outcome which is possible from executing a joint global action. Having such information available at decision time is essential for ESR settings. Furthermore, having such information available at decision time can also aid in real-world decision making, allowing certain undesirable outcomes can be avoided, given in certain real-world settings even a small probability of an undesirable event occurring is unacceptable. Having such a capability is essential when making decisions in the real world where laws, regulations, and social consequences must be taken into consideration. All remaining return distributions from the *ESR set* for the random MO-CoG are displayed in Section A.6.1.

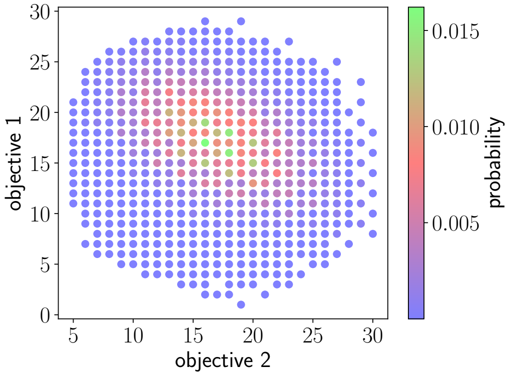


Figure 5.18: The return distribution, π_1, which is present in the *ESR set* for a random MO-CoG instance.

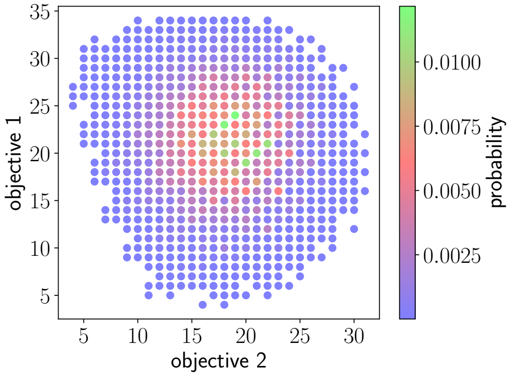


Figure 5.19: The return distribution, π_{14}, which is present in the *ESR set* for a random MO-CoG instance.

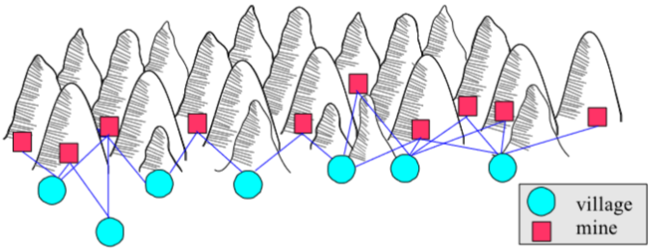


Figure 5.20: An example of a Mining Day instance [Roijers et al., 2015].

5.4.3.2 Mining Day

The Mining Day problem is inspired by a realistic scenario where in order to maximise their chances of excavating gold and silver, a mining company must decide which mines to send their workers to. The goal of the mining company is to maximise the total amount of gold and silver excavated, where mining gold and mining silver are the objectives. The motivating scenario for Mining Day is outlined as follows: A mining company sells excavated gold and silver (objectives) from a set of mines located in the mountains nearby (local payoff functions). All mine workers live locally in villages which are located at the foot of the mountains. However, each village only has one van for transporting its workers. Therefore, every morning each village must determine which mine to send their van of workers to (local actions). A further complicating factor is a van can only travel to nearby mines (coordination graph). There is a higher chance of finding gold or silver if more workers are at a mine. For an instance of Mining Day there is a 3% efficiency bonus per worker. Therefore, the amount of gold and silver mined per worker is defined as follows:

$$r_{gold} = b_{gold} \cdot 1.03^{w-1}, \tag{5.17}$$

$$r_{silver} = b_{silver} \cdot 1.03^{w-1}, \tag{5.18}$$

where b_{gold} is the base rate per worker for gold, b_{silver} is the base rate per worker for silver, and w is the number of workers at a mine. The rewards are defined as follows:

$$\mathbf{R} = [r_{gold}, r_{silver}]. \tag{5.19}$$

The base rate of gold and silver are properties of the mine. Given the company aims to mine and sell gold and silver, the company aims to maximise their profits. Therefore, the best strategy depends on the fluctuating prices of gold and silver. For example, on a given day gold may be worth far more than silver. In this case, it would be optimal to send more workers to mines that have a higher chance of

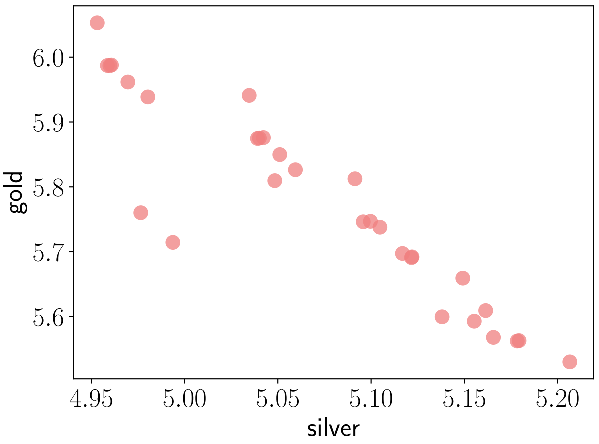


Figure 5.21: The expected value vectors of the return distributions in the *ESR set* for the Mining Day problem.

excavating gold when compared to silver. However, the worth of gold and silver may fluctuate from day to day. As such, by the time a single policy is computed using the most recent price information, the worth of gold and silver may have changed. As a result the mining company may lose out on some profits. Therefore, time would be wasted calculating the most optimal strategy with the latest possible information. Instead it is more useful to compute a set of optimal policies, where a policy can be selected that best reflects the current price information.

To evaluate DMOVE, a Mining Day instance is generated. The Mining Day instance has 7 villages (agents), then $2-5$ workers are randomly assigned to each village, and the village is connected to $2-4$ mines. It is important to note that each village is only connected to mines with a greater or equal index[3]. The last village is connected to 4 mines. The base rates per worker for gold and silver at each mine are drawn uniformly from the interval $[0, 1.0]$. To make the reward stochastic, uncertainty is added to the problem by generating the rewards from a multivariate normal distribution, where the mean $\mu = \mathbf{R}$ and a covariance matrix $\Sigma = \left(\begin{smallmatrix} 0.0001 & 0 \\ 0 & 0.0001 \end{smallmatrix}\right)$. Therefore, the rewards, $\mathbf{r}$, can be generated as follows: $\mathbf{r} \sim \mathcal{N}(\mu, \Sigma)$. DMOVE is initialised with the following hyperparameters: $\mathbf{R}_{min} = [0, 0]$, $\mathbf{R}_{max} = [10, 10]$, and $N = 1001$.

[3]For example, village 2 can only be connected to mines $2, 3, 4, ...,$ etc. and cannot be connected to mine 1.

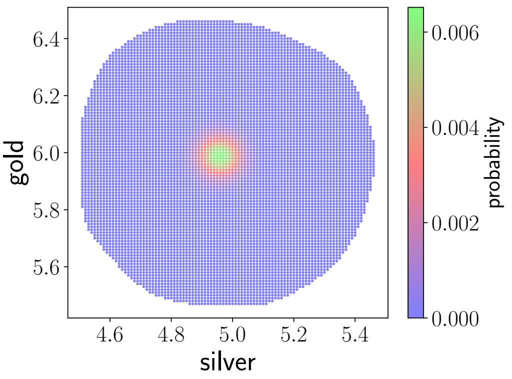


Figure 5.22: The return distribution, π_1, which is present in the *ESR set* for a Mining Day instance.

Figure 5.21 presents the expected value vectors for the return distributions in the *ESR set*. The *ESR set* contains a total of 30 policies. Figure 5.21 contains a number of policies that are Pareto dominated. This further highlights how selecting the correct optimality criterion in multi-objective multi-agent decision making is important given different sets of policies can be returned for different optimality criteria.

Given the mining company aims to maximise their profits, it is crucial that the company has sufficient information about the potential outcomes that a policy execution may have and the likelihood of each outcome. Therefore, using the distributions presented in Figure 5.18 and Figure 5.19, a manager can perform a more informed analysis of potential policies. Efficient planning of miners' schedules is important to both the mining company and the miners themselves, and thus each policy execution is important. Having the outlined distributions available can help a manager avoid selecting policies which may have some likelihood of occurring losses. A further advantage of DMOVE is the local distributions may also be extracted from the algorithm. Therefore, a manager can investigate the potential outcomes for each mine and can understand the distribution of possibilities that can occur at each mine. Such information can also be useful when making decisions globally. For example, some mines may have certain constraints about the number of workers that can work there at any given time. Such an issue may not be entirely visible at the global level and therefore having local distributions can be an aid to a decision maker.

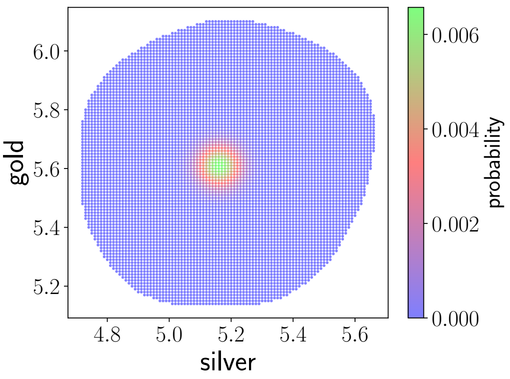


Figure 5.23: The return distribution, π_7, which is present in the *ESR set* for a Mining Day instance.

5.4.4 Discussion

In this section, a novel *distributional multi-objective variable elimination (DMOVE)* algorithm is proposed. DMOVE is evaluated using two benchmark problems from the multi-objective literature. DMOVE can compute the *ESR set* in MO-CoGs, and the Pareto front by simply switching pruning operator. DMOVE is the first multi-objective variable elimination algorithm that can compute a distribution over the returns for multi-agent settings.

The work presented in this section shows that the *ESR set* and the Pareto front can be different for multi-agent settings. Therefore, polices that are optimal under the ESR criterion may be sub-optimal under the SER criterion. As before, the choice of optimality criterion can have an impact on the set of optimal policies computed in multi-agent settings.

5.5 Related Work

In recent years, using distributions in decision making has become an active area of research for both single and multi-objective problem domains. For example, Martin et al. [2020] use a single-objective distributional C51 algorithm with stochastic dominance (SD) to make risk-aware decisions. Abdolmaleki et al. [2020] take a distributional approach to MODeM to compute a set of optimal policies for the SER criterion. However, it is important to note, taking a distributional approach to decision making is not a new idea and methods like conditional value-at-risk

(CVAR) [Rockafellar et al., 2000] and value-at-risk (VAR) [Duffie and Pan, 1997] have been used extensively in finance [Rockafellar and Uryasev, 2002; Engle and Manganelli, 2001] to make decisions under uncertainty. However, it is difficult to apply both VAR and CVAR to multi-objective settings because the computed CVAR and VAR values would not give a total ordering over policies.

Beyond a distributional approach, many algorithms can compute a set of optimal policies for the SER criterion for MOMABs, MOMDPs, and MO-CoGs. For MOMABs, the UCB algorithm has been extended to compute the Pareto front [Drugan and Nowe, 2013]. Furthermore, TS has also been extended to compute the Pareto front for MOMABs [Yahyaa and Manderick, 2015]. For MOMDPs, multi-objective Monte Carlo tree search (MOMCTS) [Wang and Sebag, 2012], Pareto value iteration (PVI) [White, 1982], convex hull value iteration (CHVI) [Barrett and Narayanan, 2008], and CON-MODP [Wiering and de Jong, 2007; Wiering et al., 2014] can be used to compute the Pareto front. For MO-CoGs, many methods like PMOVE [Rollón and Larrosa, 2006], and CMOVE [Roijers et al., 2015] are used to compute the Pareto front and convex coverage set respectively.

5.6 Summary

In this chapter a series of distributional multi-objective algorithms are presented and evaluated. Each presented algorithm computes a distribution over the returns. Each algorithm also uses ESR dominance to compute the *ESR set*. In Section 5.2, MOTDRL is presented which can be used to compute the *ESR set* for MOMABs. In Section 5.3, an algorithm known as MODVI is presented that can compute the *ESR set* in MOMDPs. In Section 5.4, DMOVE is presented, and the *ESR set* is defined for MO-CoGs. DMOVE is shown to compute the *ESR set* in two MO-CoGs. Furthermore, an evaluation metric for the ESR criterion is defined in Section 5.2, however the proposed metric can only be used in settings where the *ESR set* is known a priori. Finally, in Section 5.1 a pruning operator for the ESR criterion is defined. The pruning algorithm, known as ESRPrune, is used by MODVI and DMOVE to compute the *ESR set*.

During experimentation, it was demonstrated that the *ESR set* and the Pareto front can be different for MOMABs, MOMDPs, and MO-CoGs. In each setting it was shown that policies that are optimal under the ESR criterion may be sub-optimal under the SER criterion. Therefore, it is important to carefully consider the optimality criteria for MODeM.

Furthermore, each proposed algorithm computes a distribution over the returns for policies. When optimising for the ESR criterion, a user may only execute a policy once. By have a distribution over the returns available, the user can carefully evaluate each outcome that may occur when executing a policy. Having such information available is important as a user can then choose a policy that avoids undesirable outcomes. As such, taking a distributional approach is essential when making decisions for the ESR criterion.

6 | Conclusion

This chapter concludes the work presented with a summary of the main contributions and further discussions on the impact, limitations, and potential future work arising from this thesis.

As stated in Chapter 1, the core research questions this body of work aimed to explore were:

1. Is it necessary to design algorithms specifically for the ESR criterion in single-agent settings where the utility function is known? (RQ1)
2. When the utility function is unknown, what methodologies can be used to derive a partial ordering over policies for the ESR criterion? (RQ2)
3. How can multi-policy algorithms be designed for the ESR criterion for different multi-objective settings (e.g., multi-armed bandits, Markov decision processes and coordination graphs)? (RQ3)

Following from the investigations presented in Chapter 3, Chapter 4, and Chapter 5, the outlined research questions can be answered as follows:

1. The investigation presented in Chapter 3 showed that, for single-agent settings, the policies computed under the SER criterion and the ESR criterion can be different for nonlinear utility functions. Therefore, multi-objective algorithms that can compute policies explicitly for the ESR criterion must be developed. In Chapter 3, two multi-objective Monte Carlo tree search (MCTS) algorithms are proposed that can compute policies for the ESR criterion.

2. The investigation presented in Chapter 4 showed that the methods that determine optimality based on expected value vectors (SER criterion) cannot be used to compute a set of optimal policies under the ESR criterion. As a result, in Chapter 4, new dominance relations are proposed that can determine a partial ordering over policies under the ESR criterion. Using these dominance relations, an optimal set of solutions for ESR settings, known as the *ESR set*, was defined.

3. Three new multi-objective multi-policy algorithms are proposed in Chapter 5 that can compute sets of optimal policies under the ESR criterion in various multi-objective settings. The algorithms proposed are the first algorithms to compute sets of optimal policies under the ESR criterion.

6.1 Summary of Contributions

In summary, the main contributions of this thesis are:

6.1.1 Analysis of Multi-Objective Optimality Criteria for Nonlinear Utility Functions in Single-Agent Settings

The analysis presented in Chapter 3 showed for the first time in single-agent settings that the policies for the SER criterion and the ESR criterion can be different when the utility function of a user is nonlinear. The results of the analysis reflect the results presented by Rădulescu et al. [2020] in multi-agent settings and address RQ1.

Based on the findings of this analysis, dedicated multi-objective algorithms that can compute policies for the ESR criterion must be developed. In Chapter 3, two novel multi-objective MCTS algorithms are proposed. The algorithms, known as NLU-MCTS and DMCTS, are able to compute good policies for an array of different nonlinear utility functions in various MOMDPs. Both algorithms are empirically evaluated using multi-objective benchmarks. Furthermore, DMCTS achieves state-of-the-art performance under the ESR criterion.

6.1.2 Theoretical Analysis of Multi-Policy Methods under the Expected Scalarised Returns Criterion

Prior to this thesis, very little was understood about the use of multi-policy methods for the ESR criterion. In Chapter 4, it was shown that methods that use expected value vectors to determine optimality cannot be used to compute sets of optimal policies under the ESR criterion, given these methods compute the utility of the expectation. As a result, the state-of-the-art solution sets and dominance relations for the SER criterion cannot be used under the ESR criterion. In Chapter 4, it is demonstrated that, under the ESR criterion, a distributional approach must be taken to compute sets of optimal policies and determine a partial ordering over policies.

Based on the findings of this investigation, multiple new dominance relations are proposed that can be utilised to determine a partial ordering over policies for the ESR criterion. First, stochastic dominance (SD) is extended to MODeM and it is shown that SD can be used to determine a partial ordering over policies under the ESR criterion. As a result, an undominated set and a coverage set for the ESR criterion are also proposed. Second, another dominance relation, known as ESR dominance, is outlined. Finally, a set of ESR dominant policies is defined, known as the *ESR set*. Each of the methodologies outlined in Chapter 4 are theoretically analysed. Furthermore, it is proven that each method can be used to compute a set of optimal polices under the ESR criterion. Therefore, the work presented in Chapter 4 addresses RQ2.

6.1.3 Distributional Multi-Policy Algorithms for the Expected Scalarised Returns Criterion

The final major contribution of this thesis addresses RQ3. In Chapter 5, three novel multi-objective distributional algorithms are proposed to compute sets of optimal policies under the ESR criterion. The algorithms utilise the methodologies presented in Chapter 4. Each algorithm maintains categorical distributions over the returns and uses ESR dominance to compute the *ESR set*. The algorithms, known as MOTDRL, MODVI, and DMOVE, compute the *ESR set* in multiple MOMABs, MOMDPs, and MO-CoGs.

Furthermore, the empirical evaluations presented in Chapter 5 show that the *ESR set* and the Pareto front can be different. Moreover, policies that are sub-optimal under the SER criterion can be optimal under the ESR criterion. Therefore, if the correct optimality criterion is not selected when learning or planning, then sub-optimal solutions may be returned to the user. Moreover, computing sub-optimal solutions in real-world settings could have negative implications.

The algorithms proposed in Chapter 5 are the first algorithms presented that can compute a set of optimal policies for the ESR criterion.

6.2 Impact

Following the contributions outlined above, this thesis will influence future research in a number of significant ways. The contributions of this thesis answer many significant open questions regarding the ESR criterion. The work presented in this thesis has shown that dedicated single-policy and multi-policy algorithms must be developed in order to optimise under the ESR criterion. Moreover, this thesis introduces a theoretical framework that utilises a distributional approach to MODeM for the ESR criterion and shows that such an approach is necessary to optimise under the ESR criterion. Therefore, this thesis has established a new area of research, known as *distributional multi-objective reinforcement learning and planning*, that can aid in applying multi-objective methods to real-world settings. Given policies under the ESR criterion may only be executed once, the ESR criterion aligns with many real-world decision making scenarios. For example, in a medical setting a user may only have one opportunity to select a treatment. Prior to this work, it was not possible to accurately optimise for such scenarios in MODeM settings under the ESR criterion. The work outlined in this thesis provides a starting point for researchers and AI practitioners to develop algorithms that can compute a single policy, or sets of polices, in real-world settings under the ESR criterion.

6.3 Limitations

Despite the contributions outlined above, this work does have limitations, as outlined below:

6.3.1 Categorical Distributions

The distributional algorithms presented in this thesis utilise categorical distributions. While categorical distributions have been used extensively in the distRL literature [Reymond et al., 2021; Bellemare et al., 2017], they have many limitations. In multi-objective settings, a categorical distribution must maintain categories to represent the probability of each possible return. However, the number of categories scales poorly with the number of objectives. Therefore, utilising categorical distributions can have an impact on memory requirements, especially in settings with a large range between the maximum and minimum returns. In settings where a large number of categories is required, categorical distributions can also be computationally inefficient to update. Therefore, categorical distributions do not scale efficiently and potentially cannot be used in the most complex real-world settings.

6.3.2 Benchmarks and Evaluation Metrics

In this thesis, all evaluations were completed using toy problems or altered benchmarks. As the ESR criterion had not previously been extensively studied, the majority of multi-objective benchmark problems had been designed for the SER criterion. Furthermore, the majority of benchmark problems have deterministic state transitions and reward functions. Designing dedicated benchmarks for the ESR criterion where the *ESR set* is known a priori would be a significant challenge and, hopefully, this thesis inspires future work in this area.

Furthermore, the work presented in this thesis regarding multi-policy methods lacks evaluation metrics. As previously noted, the majority of research for MODeM has focused on the SER criterion. Therefore, each multi-objective evaluation metric can only be used to evaluate SER algorithms. As a result, an evaluation metric for the ESR criterion was designed in Chapter 4. However, this method requires the *ESR set* to be known a priori which limits its use. Therefore, to accurately evaluate multi-policy ESR algorithms, multi-objective evaluation metrics must be developed.

6.3.3 Discrete States & Actions

Each multi-objective problem domain presented during the empirical evaluations of this thesis had a finite set of states and actions that were represented discretely. Focusing on discrete states and actions was useful when exploring single-policy methods and multi-policy methods for the ESR criterion for the first time. However, in real-world settings the state and action spaces for a given problem may be continuous. The methods proposed in this thesis cannot be used to compute solutions for problems with continuous state or actions spaces. Therefore, the proposed methods are limited in their applicability to real-world problem domains.

6.3.4 Computational Analysis

While this thesis proposes novel algorithms that can tackle new problem settings for the first time, no mathematical analysis of the computational requirements of these algorithms was undertaken. Each of the multi-policy methods must maintain a distribution over the returns to determine optimality. It can be assumed that computing a distribution over the returns is more computationally inefficient when compared to expected value vector methods used under the SER criterion. Therefore, before multi-policy methods that compute policies under the ESR criterion can be applied to real-world settings, a computational analysis of these methods must be undertaken. Performing such analyses can determine the feasibility of these methods in larger problem domains so that future researchers and practitioners can better understand how well these methods will scale to larger problems. The results of such analyses will be informative when trying to determine if these methods can feasibly be applied to a particular real-world problem.

6.4 Future Work

Following from the investigations in this thesis, there are a number of promising avenues for future research:

6.4.1 Approximating Return Distributions

The algorithms presented in this thesis utilise categorical distributions to represent return distributions. A limiting factor of categorical distributions is, when the range of potential returns increases, maintaining a sufficient number of categories for the categorical distribution requires a large amount of memory. It is expected that, in scenarios with large state-action spaces, like [Abrams et al., 2021], the range of possible returns would be difficult to maintain using a categorical distribution. A potential solution would be to approximate the distributions using various methods.

Recently, a class of machine learning models, known as generative models [Tomczak, 2022], have become widely used in the literature. Generative models, like generative adversarial networks [Goodfellow et al., 2020], generative flow models [Papamakarios et al., 2021; Kobyzev et al., 2020], and diffusion models [Ho et al., 2020] have shown excellent performance in modelling high dimensional data distributions. One model that shows promise is a generative flow model known as real-NVP [Dinh et al., 2016]. Using real-NVP to model return distributions is a promising starting point for some future work that could improve the scalability of these algorithms in larger and more realistic problems.

6.4.2 Trustworthy AI

Given the rise in deployment of artificial intelligence systems in real-world settings, it is important that these systems are explainable, trustworthy, and safe [Mannion et al., 2021]. Distributional multi-policy methods, like MOTDRL, DMOVE, and MODVI, can be used to provide explainability for MODeM problems. For example, consider a cleaning robot operating in a household and, one day, the robot breaks a vase [Hayes et al., 2022c]. In this situation, it is important to be able to review the robot's behaviour to understand why the robot acted in a way that resulted in the vase being damaged. Having a distribution over the returns available can aid system experts in understanding how and why an adverse event occurred, as such an event could be present in the distribution of returns. Additionally, distributional methods can also bring outlier events to the attention of a system before a decision is made, allowing unsafe events to be avoided. Therefore, it may be possible to utilise distributional multi-objective approaches for expanding the field of trustworthy and safe artificial intelligence [Mannion et al., 2021].

As already highlighted, the state-of-the-art MODeM approaches focus on computing policies using expected value vectors [Roijers et al., 2015; Bryce et al., 2007]. Making decisions based on expected value vectors reduces the explainability and safety of such algorithms, given the distribution over the potential outcomes is lost when the expectation is computed. For example, for a given action, there may be a very small probability that the robot will come into contact with a vase and knock it over. An expected value vector may not effectively capture uncommon events, given their probability of occurring may be small. Expected value vectors can also be difficult to interpret, especially in environments where outcomes can be stochastic. In contrast, return distributions can easily be interpreted by a system expert. Therefore, distributional methods could be utilised as a decision making aid in complex real-world settings. As such, distributional methods may be a fruitful direction for future trustworthy AI research.

6.4.3 Further Theoretical Investigations

From the empirical evaluations presented in Chapter 5, it appears that there is a relationship between the *ESR set* and the Pareto front. In all experimental results so far, it has been observed that policies that are on the Pareto front are also in the *ESR set*. Therefore, the Pareto optimal set could be a subset of the *ESR set*. However, no theoretical investigations have been undertaken to determine this potential relationship. Furthermore, the theoretical work presented in Chapter 4 is limited to specific settings. In order to fully understand the relationships between the ESR criterion, the SER criterion, and the rest of the MODeM literature, further theoretical investigations must be undertaken. Therefore, additional theoretical investigations are needed to expand the literature for ESR criterion. This is an exciting avenue for future research.

Finally, ESR dominance is a strict dominance criterion. In many settings, ESR dominance may produce very large sets of policies that would be optimal for all decision makers. It would be possible to relax the ESR dominance requirements by using *almost stochastic dominance* to generate smaller solution sets, where each policy in the set is optimal for most decision makers [Leshno and Levy, 2002].

6.4.4 Further Optimality Criteria

In this thesis, the SER criterion and the ESR criterion are studied. However, in certain scenarios, a user's utility may be derived from a mixture of the SER criterion and the ESR criterion across different objectives. For example, consider a user who is planning their daily commute to work. The user needs to arrive on time. Therefore, the user must optimise for the ESR criterion for the time objective. However, the user may still want to have an average cost for commuting, therefore, the the user must optimise for the SER criterion for the cost objective. In this case, a user has a mixture of different optimality criteria for a given problem. Such mixed optimality criteria have not been explored in the literature and the implications of mixed optimality criteria is an open question. Developing a theoretical methodology to compute policies for mixed optimality criteria is an interesting starting point for future research.

6.5 Final Remarks

As RL and planning methods continue to be deployed in real-world settings, the case for MODeM grows in importance, given many real-world problems have multiple objectives that need to be optimised. However, some limitations have restricted the applicability of MODeM in certain real-world settings. As described throughout this thesis, the ESR criterion aligns with many practical problems. Despite the contributions to the ESR criterion presented in this thesis, the ESR criterion must be investigated further. However, this thesis has provided a starting point for aspiring researchers who are interested in developing theoretical and algorithmic methods for the ESR criterion. I hope the work presented in this thesis inspires future researchers to continue work in this field.

Glossary

BTS bootstrap Thompson sampling. 26, 27, 70, 72–76, 84, 85, 108

C51 categorical deep Q-networks. 30, 31, 72, 77–86, 142

CDF cumulative distribution function. 32, 91, 93, 98, 100, 101, 106, 110, 111, 113–115, 119, 168

CHMCTS convex hull Monte Carlo tree search. 51

CHVI convex hull value iteration. 143

CMOVE convex hull variable elimination. 49, 143

CoG coordination graph. 27–29, 41, 42

CVAR conditional value-at-risk. 142, 143

DEED Dynamic Economic Emissions Dispatch. 79, 159, 161, 162, 165

distRL distributional reinforcement learning. 102, 148

DMCTS distributional Monte Carlo tree search. 56, 67–70, 72, 73, 75–84, 86, 146, 158, 165–167

DMOVE distributional multi-objective variable elimination. 106, 131–136, 140–143, 147, 150, 171, 172

DQN deep Q-networks. 30, 31, 50

ESR expected scalarised returns. 3, 15–17, 51–59, 61, 64, 65, 68, 70, 72, 76, 77, 80, 81, 83, 86–91, 93, 95–97, 99–103, 105–120, 122–125, 127, 128, 130–138, 140–152, 158, 172, 173, 186

EUPG expected utility policy gradient. 53, 54, 72, 77–82

FSD first-order stochastic dominance. 32, 91–93, 95–97

LESR local *ESR set*. 132–134

MAB multi-armed bandit. 25, 26, 29, 41, 49

MCTS Monte Carlo tree search. 17, 23, 24, 29, 51, 59–61, 78, 81, 84–86, 145, 146

MDP Markov decision process. 20–23, 25, 29, 37, 38, 44, 49

MO-CoG multi-objective coordination graph. 42, 49, 106, 107, 131, 132, 134–138, 142, 143, 147, 171–173

MODeM multi-objective decision making. 3, 14–17, 37–39, 41, 44, 45, 51–59, 72, 87, 88, 90, 93, 95, 96, 101, 103, 105, 108, 111, 116, 120, 130, 135, 142, 143, 147–152, 161, 165, 168

MODVI multi-objective distributional value iteration. 106, 120, 122, 123, 125–128, 130, 136, 143, 147, 150

MOMAB multi-objective multi-armed bandit. 41, 42, 49, 106–109, 111–114, 116, 119, 143, 147

MOMCTS multi-objective Monte Carlo tree search. 51, 143

MOMDP multi-objective Markov decision process. 37, 38, 44, 49, 51, 58, 59, 73, 75, 76, 86, 106, 107, 130, 143, 146, 147, 169

MOTDRL multi-objective distributional tabular reinforcement learning. 106–114, 117–119, 136, 143, 147, 150

MOVE multi-objective variable elimination. 49, 132

NLU-MCTS Monte Carlo tree search for nonlinear utility functions. 56, 59–65, 67, 68, 72, 76–82, 84, 86, 146, 158

PDF probability density function. 110, 111

PMOVE Pareto multi-objective variable elimination. 49, 143

POMDP partially observable Markov decision process. 29

PVI Pareto value iteration. 143

REDEED Renewable Energy Dynamic Economic Emissions Dispatch. 76, 79–82, 159, 161–163, 165–167

RL reinforcement learning. 13, 14, 19–25, 29–31, 34, 35, 37, 44, 49–51, 58, 72, 77, 85, 86, 152, 161

RSF return distribution set factors. 132–134

SD stochastic dominance. 16, 31, 32, 34, 88, 102, 103, 142, 147, 168

SER scalarised expected returns. 3, 14–17, 51–59, 86–89, 103, 106, 111, 115, 116, 120, 123, 130, 131, 135, 136, 142, 143, 145–147, 149, 151, 152, 158

SSD second-order stochastic dominance. 33, 102

TS Thompson sampling. 24, 26, 27, 49, 56, 67, 84, 85, 143

UCB upper confidence bound. 24, 25, 49, 61, 67, 84, 85, 110, 111, 143

UCT upper confidence trees. 24, 85

VAR value-at-risk. 143

VE variable elimination. 49

VRS Vaccine Recommender System. 112, 116–119

A | Appendices

A.1 Monte Carlo Tree Search Algorithms

Under the ESR criterion it is also possible to backpropagate the utility of the cumulative returns, $u(\mathbf{R}_t)$, during the backpropagation phase. Therefore, the relevant statistics are updated using the backpropagated utility of the cumulative returns. Figure A.1 outlines how the utility of the cumulative returns can be backpropagated to each node. However, this method makes the DMCTS and NLU-MCTS algorithms less general as they cannot be as easily adapted for the SER criterion.

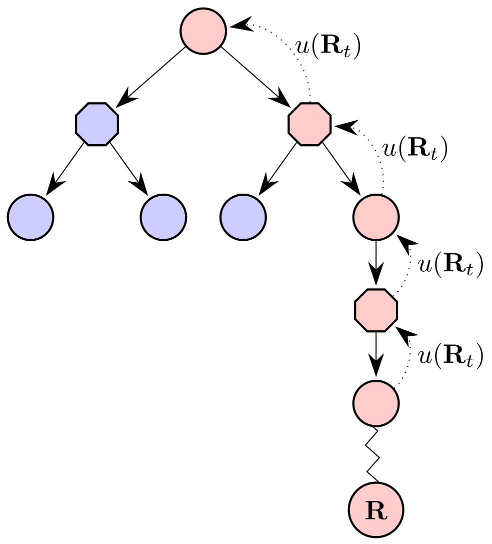


Figure A.1: To compute polices under the ESR criterion, the utility of the cumulative returns, $u(\mathbf{R}_t)$, can also be backpropagated to each node visited during the planning phase.

A.2 Renewable Energy Dynamic Economic Emissions Dispatch Implementation Details

Renewable Energy Dynamic Economic Emissions Dispatch (REDEED) is a variant of the Dynamic Economic Emissions Dispatch (DEED) problem [Basu, 2008]. In Chapter 3 some relevant implementation details for the REDEED problem domain were not included. In this section, all nessesary details to implement the REDEED problem domain are included. Before the REDEED domain is introduced, the DEED problem is discussed in detail for completeness.

A.2.1 Dynamic Economic Emissions Dispatch

DEED is a multi-objective optimisation problem first introduced by Basu [2008]. For the DEED problem, a number of electrical generators must be run in order to provide power to a population in a town. The objective of the DEED problem proposed by Basu [2008] is to minimise both the cost and emissions of running a required number of generators.

Both cost and emissions of operating a generator are defined by equations. The following equation represents the fuel cost for each generator:

$$f_1 = \sum_{m=1}^{M} \sum_{n=1}^{N} [a_n + b_n P_{nm} + c_n (P_{nm})^2 + |d_n sin\{e_n (P_n^{min} - P_{nm})\}|] \tag{A.1}$$

where $M = 24$ is the number of operating hours and $N = 10$ is the number of generators, a_n, b_n, c_n, d_n and e_n are the cost coefficients associated with each generator n, P_{nm} is the power output from generator n at time m, and P_n^{min} is the minimum permissible power output of generator n.

The emissions from each generator can be defined by the following equation:

$$f_2 = \sum_{m=1}^{M} \sum_{n=1}^{N} [\alpha_n + \beta_n P_{nm} + \gamma_n (P_{nm})^2 + \eta \exp \delta P_{nm}] \tag{A.2}$$

where α_n, β_n, γ_n, η_n and δ_n are the emission coefficients associated with each generator n.

$$\sum_{n=1}^{N} P_{nm} - P_{Dm} - P_{Lm} = 0 \tag{A.3}$$

Mannion [2017] defined the total power output in a given hour must be equal to the sum of power demand and transmission losses. Equation A.3 can be modified to define the following:

$$\sum_{n=1}^{N} P_{nm} = P_{Dm} + P_{Lm} \qquad \forall m \in M \tag{A.4}$$

where P_{Dm} is the power demand over hour m and P_{Lm} is the transmission loss over hour m.

There are two constraints for real power operating limits and generating unit ramp rate limits [Basu, 2008]. Real power operating limits can be defined as follows:

$$P_n^{min} \leq P_{nm} \leq P_n^{max} \tag{A.5}$$

The real power operating limits outline the minimum and maximum possible power output of a generator. The ramp rate limits can be defined by the following equations:

$$P_{nm} - P_{n(m-1)} \leq UR_n \tag{A.6}$$

$$P_{n(m-1)} - P_{nm} \leq DR_n \tag{A.7}$$

The ramp rate limits determine the maximum or minimum possible power output of a generator from one hour to the next. These equations determine the range in which a generator's power output can fluctuate over time.

For equations A.5, A.6 and A.7: P_n^{min} and P_n^{max} refer to the minimum and maximum power output of each generator, P_{nm} is the power output for $n \in N$ and $m \in M$, and UR_n and DR_n are the ramp up and ramp down limits for generator n.

To satisfy Equation A.4 Mannion [2017] outlines that the first generator, $n = 1$, must be treated as a slack generator. Therefore, if the power output of the other generators does not satisfy the required demand, the slack generator is able to generate more energy to meet the demands if required. The power output of the slack generator for a single hour, P_{1m}, can be calculated as follows:

$$P_{1m} = P_{Dm} + P_{Lm} - \sum_{n=2}^{N} P_{nm} \tag{A.8}$$

The transmission loss P_{Lm} is a function of all the generators, including the slack generator, and can be defined with the following equation:

$$P_{Lm} = \sum_{n=2}^{N}\sum_{j=2}^{N} P_{nm}B_{nj}P_{jm} + 2P_{1m}(\sum_{n=2}^{N} B_{1n}P_{nm}) + B_{11}(P_{1m})^2 \tag{A.9}$$

where B is the matrix of transmission line loss coefficients. Therefore, by expanding Equation A.8 and rearranging with respect to Equation A.9, the following equation can be defined:

$$0 = B_{11}(P_{1m})^2 + (2\sum_{n=2}^{N} B_{1n}P_{nm} - 1)P_{1m} + (P_{Dm} + \sum_{n=2}^{N}\sum_{j=2}^{N} P_{nm}B_{nj}P_{nm} - \sum_{n=2}^{N} P_{nm}) \tag{A.10}$$

The loading of the slack generator at each hour can be found by solving the quadratic Equation A.10. All required values for the cost coefficients, emission coefficients, ramp limits, generator capacity limits, power demands, and transmission line loss coefficients can be found in the work of Basu [2008].

A.2.2 Renewable Energy Dynamic Economic Emissions Dispatch

Mannion [2017] reformulated the DEED problem as a multi-objective stochastic game. Mannion et al. [2017] applied multi-agent RL to the DEED problem and achieved good results [Mannion et al., 2016, 2017]. In order to create a version of the DEED problem suitable for singe-agent MODeM, further changes to the work of Mannion [2017] must be made.

In the DEED problem there are 9 controlled generators ($n = \{2, ..., 10\}$) and an additional slack generator, $n = 1$. However, for REDEED, generator $n = 4$ is replaced by a wind turbine. Additionally, generator $n = 3$ is controlled by the agent[1]. The power for all other generators in the REDEED problem is fixed. The values for each generator are derived from the work of Mannion [2017] and can be found in Table A.1[2]. The REDEED problem is a sequential decision making problem, where each hour $m \in M$ is a timestep.

The setting for REDEED covers a period of 24 hours and for each hour a weather forecast is received for a city. For hours $1 - 15$, the weather is predictable and the optimal power values derived by Mannion [2017] can be used to generate power (see Table A.1). 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

[1]Generally, when the DEED problem is utilised in multi-agent settings each agent controls its directly assigned generator. However, given REDEED is reformulated for single-agent settings, the agent controls the generator $n = 3$.

[2]It is important to note minor changes from the power values derived by Mannion [2017] have been made to Table A.1.

wind turbine will produce 25% more power than optimal. At each hour the values for P_4 in Table A.1 can be utilised to implement the power outputs from the wind turbine. In the REDEED problem the aim is to compute a policy that can ensure the required power is met over the entire day while reducing the cost, emissions, and penalty violations created by all generators.

The objectives of the REDEED problem are the same as those outlined in the DEED problem proposed by Mannion [2017], whereby cost, emissions, and penalty violations must be minimised.

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

$$f_c^L(n,m) = a_n + b_n P_{nm} + c_n (P_{nm})^2 + |d_n sin\{e_n(P_n^{min} - P_{nm})\}|. \tag{A.11}$$

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

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

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

$$f_e^L(n,m) = E(a_n + b_n P_{nm} + \gamma_n (P_{nm})^2 + \eta \exp \delta P_{nm}). \tag{A.13}$$

where $E = 10$ is the emissions scaling factor, chosen so that the magnitude of the local emissions $f_e^L(n,m)$ is the same as that of the local cost function $f_c^L(n,m)$. Therefore the global emissions for all generators can be defined as:

$$f_e^G(m) = \sum_{n=1}^{N} f_e^L(n,m). \tag{A.14}$$

It is important to note the emissions the wind turbine, $n = 4$, are set to 0.

The power limits and the ramp limits for the slack generator must be taken into consideration. Mannion [2017] developed a global penalty function f_p^G that captures the violations of these constraints:

$$f_p^G(m) = \sum_{v=1}^{V} C(|h_v + 1|\delta_v) \tag{A.15}$$

$$h_1 = \begin{cases} P_{1m} - P_1^{max} & \text{if } P_{1m} > P_1^{max} \\ P_1^{min} - P_{1m} & \text{if } P_{1m} < P_1^{min} \\ 0 & \text{otherwise} \end{cases} \tag{A.16}$$

$$h_2 = \begin{cases} (P_{1m} - P_{1(m-1)}) - UR_1 & \text{if } (P_{1m} - P_{1(m-1)}) > UR_1 \\ (P_{1m} - P_{1(m-1)}) + DR_1 & \text{if } (P_{1m} - P_{1(m-1)}) < -DR_1 \\ 0 & \text{otherwise} \end{cases} \tag{A.17}$$

where V is determined by the number of constraints handled in the given problem. In this case $V = 2$ given slack generator power and the ramp limits can be violated. $C = 10E6$ is the violation constant and h_v is the violation of each constraint. Further more $\delta_v = 0$ if there are no violations for a given constraint and $\delta_v = 1$ if the constraint is violated.

Therefore the agent receives the following reward at each timestep t:

$$\mathbf{r}_t = [-f_c^G, -f_e^G, -f_p^G] \tag{A.18}$$

The next state for the agent is a vector that contains the change in power demand from the previous timestep, ΔP_D, and the previous power output of the Generator 3, P_{nm}. The change in power demand at time m can be calculated as follows:

$$\Delta P_{Dm} = P_{Dm} - P_{D(m-1)} \tag{A.19}$$

Therefore the state vector for the agent (Generator 3) at hour m is:

$$s_{im} = [\Delta P_{Dm}, P_{n(m-1)}] \tag{A.20}$$

The action chosen by the agent at each timestep determines the power output of Generator 3. The power outputted by Generator 3 has to meet the constraints set in equation A.5. To overcome the high variability of the policies learned, Mannion [2017] created an abstraction A^* of the action space with 101 actions. However, for REDEED the action space is reduced further to include 11 actions. Therefore, in A^* the agent has a set of 11 possible actions, $A^* = \{0, 10, 20, ..., 100\}$. Each action represents a different percentage value of the operating range of Generator 3 controlled by the agent. The power output from generator n for action a_i^* can be calculated as follows:

$$P_n = P_n^{min} + a_i^* (\frac{P_n^{max} - P_n^{min}}{100}) \tag{A.21}$$

where $i = n$. Using the action abstraction A^* ensures the agent is still subject to the ramp limits outlined in equations A.5, A.6, and A.7.

$Hour(m)$	P_2	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
1	245	115	145	86	73	80	51	26
2	245	124	152	95	80	91	64	30
3	245	165	187	101	94	101	70	44
4	251	174	219	114	120	111	75	51
5	314	179	226	131	122	118	75	52
6	291	214	237	157	122	117	75	52
7	294	219	241	158	122	117	79	54
8	337	250	241	158	126	119	79	55
9	380	295	241	159	127	119	79	55
10	407	298	241	159	128	119	79	55
11	430	298	241	159	129	119	79	55
12	384	298	241	159	129	119	79	55
13	354	298	241	159	129	119	79	55
14	314	298	241	159	129	119	79	55
15	241	293	206	154	129	119	79	42
16	218	276	206	131	127	109	79	52
17	261	267	214	137	127	109	79	53
18	337	250	224	148	127	109	79	54
19	414	269	236	154	128	112	79	54
20	344	269	236	156	128	116	79	54
21	341	238	209	108	125	95	68	52
22	264	188	199	92	94	83	65	43
23	211	186	152	115	65	68	52	34
24	274	160	122	69	62	53	40	13

Table A.1: Power outputs derived from the work of Mannion [2017] for generators $2, 4, 5, 6, 7, 8, 9, 10$. For the REDEED problem domain the generator P_4 is replaced with a wind turbine. However, the power outputs specified for P_4 can be used to derive the power outputs for the wind turbine. The power outputs for generator P_1 are determined by the slack generator equation.

A.2.3 Sample Solutions

Typically in MODeM, single policy algorithms compute policies for a known utility function. Therefore, the goal of the agent is to compute a policy that maximises utility. In Section 3.4, DMCTS is evaluated using the REDEED problem with a specified nonlinear utility function. However, typically the DEED and REDEED problems evaluate an algorithm's performance based on the algorithm's ability to optimise for the individual objectives. Generally, the cost, emissions, and penalty violations of the policies computed by the agent(s) is reported. Given utility is only reported in Section 3.4, further experiments are presented for DMCTS where the cost, emissions, and penalty violations are also reported. To ensure the results are in line with the literature, the power output of the slack generator, the wind turbine, and the generator the agent controls are also included.

For the REDEED environment DMCTS is run using the following parameters: $J = 100$ and $n_{exec} = 2$. To present the results for DMCTS, the algorithm is run for $10,000$ episodes. Table A.2 and Table A.3 present two sample solutions computed by DMCTS. Each table outlines the power for the Slack generator, the wind turbine and the generator powered by the DMCTS agent. Furthermore, the cost, emissions, and penalty violations are also reported.

It is important to note that the results presented in Table A.2 and Table A.3 are not directly comparable to the original DEED problem implementations [Basu, 2008; Mannion, 2017], given the problem has changed considerably. The factors that have influenced this are as follows: The REDEED problem is a single-agent problem, whereas the DEED problem used by Mannion [2017] is a multi-agent problem. With the addition of the wind turbine the REDEED problem is stochastic, whereas the outcomes of actions and state transitions in the DEED problem are deterministic. The wind turbine also has 0 emissions, which is not the case in other DEED implementation. Furthermore, the action space of the REDEED problem has been reduced to 11 actions compared to the version used by Mannion [2017] which has 101 actions.

Hour	P_1 (Slack)	P_3 (Agent)	P_4 (Turbine)
1	219.19	99.0	115.0
2	259.92	153.0	124.0
3	263.26	179.0	165.0
4	179.12	233.0	174.0
5	211.8	259.0	179.0
6	214.94	286.0	214.0
7	223.14	340.0	219.0
8	304.3	340.0	250.0
9	323.81	340.0	295.0
10	387.03	340.0	298.0
11	413.32	340.0	298.0
12	423.43	286.0	298.0
13	369.32	206.0	298.0
14	300.02	153.0	298.0
15	245.1	99.0	293.0
16	202.56	126.0	276.0
17	259.15	179.0	267.0
18	313.63	206.0	250.0
19	456.15	233.0	201.0
20	490.30	206.0	201.0
21	329.61	126.0	238.0
22	272.80	73.0	188.0
23	260.05	73.0	186.0
24	159.63	73.0	200.0
Cost ($\$x10^6$)	Emissions (lbx10^5)	Violations (x10^6)	Utility
-2.731874279	-2.902131774	-200.4386492	-1.59^{21}

Table A.2: A sample solution computed by DMCTS for the REDEED problem domain, where the values for P_2, P_5, P_6, P_7, P_8, P_9, and P_{10} can be found in Table A.1.

Hour	P_1 (Slack)	P_3 (Agent)	P_4 (Turbine)
1	219.19	99.0	115.0
2	259.92	153.0	124.0
3	263.26	179.0	165.0
4	179.12	233.0	174.0
5	211.8	259.0	179.0
6	214.94	286.0	214.0
7	223.14	340.0	219.0
8	304.3	340.0	250.0
9	323.81	340.0	295.0
10	387.03	340.0	298.0
11	413.32	340.0	298.0
12	423.43	286.0	298.0
13	369.32	206.0	298.0
14	300.02	153.0	298.0
15	245.1	99.0	293.0
16	202.56	126.0	276.0
17	326.09	179.0	200.0
18	313.63	206.0	250.0
19	387.01	233.0	269.0
20	448.87	179.0	269.0
21	357.58	99.0	238.0
22	226.12	73.0	235.0
23	188.03	99.0	232.0
24	199.19	73.0	160.0
Cost (\$x10^6)	Emissions (lbx10^5)	Violations (x10^6)	Utility
-2.721039042	-2.757148637	-171.7684759	-1.29^{21}

Table A.3: A second sample solution computed by DMCTS for the REDEED problem domain, where the values for P_2, P_5, P_6, P_7, P_8, P_9, and P_{10} can be found in Table A.1

A.3 Further Theory for ESR Dominance

In Section 4.2, Theorem 3 proves that stochastic dominance (SD) can be extended to MODeM, Lemma 2 is used in Theorem 3.

Lemma 1

(Beppo Levi's lemma [Schappacher, 1996]) Consider a point-wise non-decreasing sequence of positive functions $f_n : X \to [0, +\infty]$, i.e., for every $k \geq 1$ and every $x \in X$.

$$0 \leq f_n(x) \leq f_{n+1}(x) \leq +\infty$$

Set the point-wise limit of the sequence $\{f_i\}$ to be f. That is, for every $x \in X$,

$$\lim_{n \to +\infty} f_n(x) = f(x)$$

Then f is measurable and:

$$\lim_{n \to +\infty} \int f_n(x)dx = \int \lim_{n \to +\infty} f_n(x)dx$$

Lemma 2

(Monotone convergence) Let u be a non-negative monotonically increasing utility function in x and y, and F the CDF of a random variables X and Y. Then,

$$\int \lim_{y \to +\infty} u(x,y)F(x,y)dx = \lim_{y \to +\infty} \int u(x,y)F(x,y)dx.$$

Proof. Let $g_n(x) = u(x,n)F(x,n)$. As u and F are positive monotonically increasing functions in n, the function g_n is also positive and monotonically increasing, i.e.,

$$0 \leq g_n(x) \leq g_{n+1}(x) \leq +\infty$$

According to Beppo Levi's lemma (see Lemma 1), the limit of the integral of $g_n(x)$ in x is the integral of its limit, i.e.,

$$\lim_{n \to +\infty} \int g_n(x)dx = \int \lim_{n \to +\infty} g_n(x)dx.$$

State	Action	$P(success)$	Reward on success	Reward on failure
A	Indirect	1.0	(0, -12)	n/a
	Direct	0.9	(0, -6)	(0, -1)
	Teleport	0.85	(0, 0)	(0, 0)
B	Indirect	1.0	(1, -10)	n/a
	Direct	0.9	(1, -8)	(0, -7)
	Teleport	0.85	(1, 0)	(0, 0)

Table A.4: The probability of success and reward values for each state-action pair in the Space Traders MOMDP.

□

A.4 Space Traders

Space Traders is a finite horizon problem with a horizon of 2 timesteps that was first introduced by Vamplew et al. [2021a]. Space Traders has 5 states, with 2 non-terminal states, and 3 available actions in each state. The goal of the agent is to travel from its home planet (State A) to another planet (State B) to deliver a shipment of cargo and then return to its home planet with the payment. Space Traders has two objectives: mission success and time. The agent receives a reward of 1 for returning to its home planet with the cargo and a reward of 0 for mission success at all other states. The agent receives a negative reward for time for each action taken. It is important to note the time values and reward for the outward (State A to State B) and return (State B to State A) journeys for each action vary (see Table A.4).

There are three possible pathways between each of the planets. The agent can take a direct path, which is short. However, there is a chance the agent will be intercepted by space pirates, which results in a terminal state. The agent can also take the indirect path, which avoids the pirates and guarantees completion of the mission. However, the indirect path takes a long time. Finally, the agent can also teleport from one state to the next instantaneously. However, this action has a high chance of failure, resulting in a terminal state.

Table A.4 summarises the transition probabilities and rewards of the Space Traders MOMDP. Table A.5 presents the distribution over the returns for each possible deterministic policy in the Space Traders MOMDP. Utilising the details presented above along with Table A.4 and Table A.5, it is possible to fully implement the Space Traders problem.

π	Action in state A	Action in state B	r_1	r_2	$P(r_1, r_2)$
II	Indirect	Indirect	1	-22	1.0
ID	Indirect	Direct	0	-19	0.1
			1	-20	0.9
IT	Indirect	Teleport	1	-12	0.85
			0	-12	0.15
DI	Direct	Indirect	0	-1	0.1
			1	-16	0.9
DD	Direct	Direct	1	-14	0.81
			0	-13	0.09
			0	-1	0.1
DT	Direct	Teleport	1	-6	0.765
			0	-6	0.135
			0	-1	0.1
TI	Teleport	Indirect	1	-10	0.85
			0	0	0.15
TD	Teleport	Direct	1	-8	0.765
			0	-7	0.085
			0	0	0.15
TT	Teleport	Teleport	1	0	0.7225
			0	0	0.1275
			0	0	0.15

Table A.5: The return distributions for each policy in the Space Traders MOMDP.

A.5 PPrune

Below the PPrune algorithm presented by Roijers et al. [2015] is outlined. PPrune is used as the pruning operator for DMOVE in Section 5.4.3.1 to compute the Pareto front for a Random MO-CoG instance.

Algorithm 20: PPrune

```
1  Input: $\mathcal{V} \leftarrow$ A set of value vectors
2  $\mathcal{V}^* \leftarrow \emptyset$
3  while $\mathcal{V} \neq \emptyset$ do
4      $\mathbf{V} \leftarrow$ the first element of $\mathcal{V}$
5      for $\boldsymbol{V}' \in \mathcal{V}$ do
6          if $\boldsymbol{V}' \succ_p \boldsymbol{V}$ then
7              $\mathbf{V} \leftarrow \mathbf{V}'$
8          end
9      end
10     Remove $\mathbf{V}$, and all vectors Pareto-dominated by $\mathbf{V}$, from $\mathcal{V}$
11     Add $\mathbf{V}$ to $\mathcal{V}^*$
12 end
13 Return $\mathcal{V}^*$
```

A.6 Exhaustive List of Results for DMOVE

In Section 5.4.3, DMOVE is evaluated using two MO-CoGs from the literature. The return distributions for the policies in the *ESR set* computed by DMOVE during evaluation are presented below. For the Random MO-CoG, 156 return distributions are presented. For the Mining Day instance 30 return distributions are presented. It is important to note, each return distributions represents the distribution over possible outcomes (rewards) a policy may have.

A.6.1 Random MO-CoG

Below the return distributions for each policy in the *ESR set* for a random MO-CoG instance are presented.

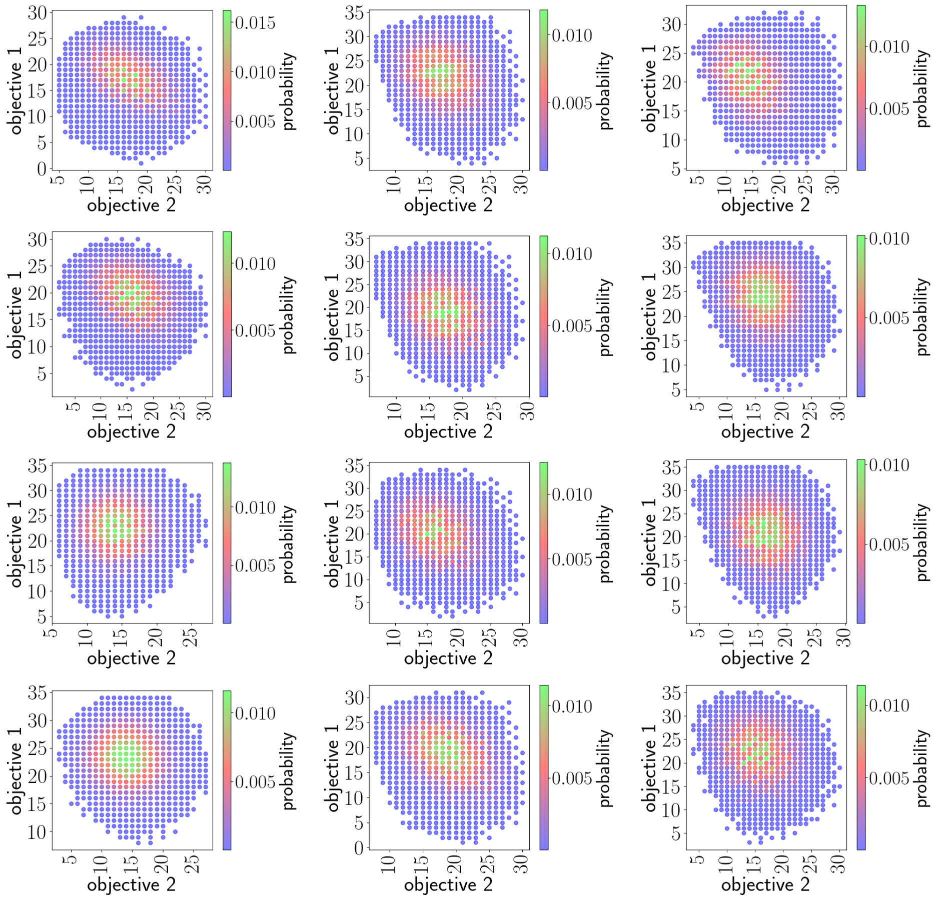


Figure A.2: The return distributions for policies π_1 - π_{12} computed by DMOVE for Random MO-CoG.

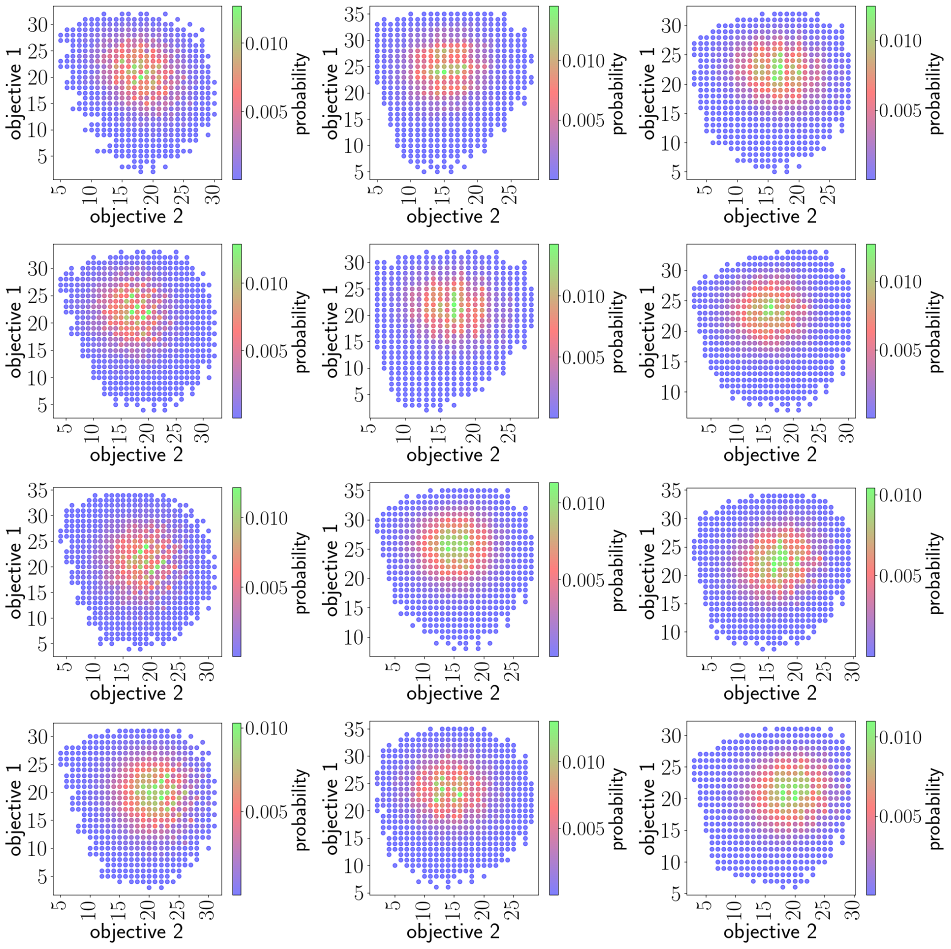


Figure A.3: The return distributions for policies π_{13} - π_{24} computed by DMOVE for Random MO-CoG.

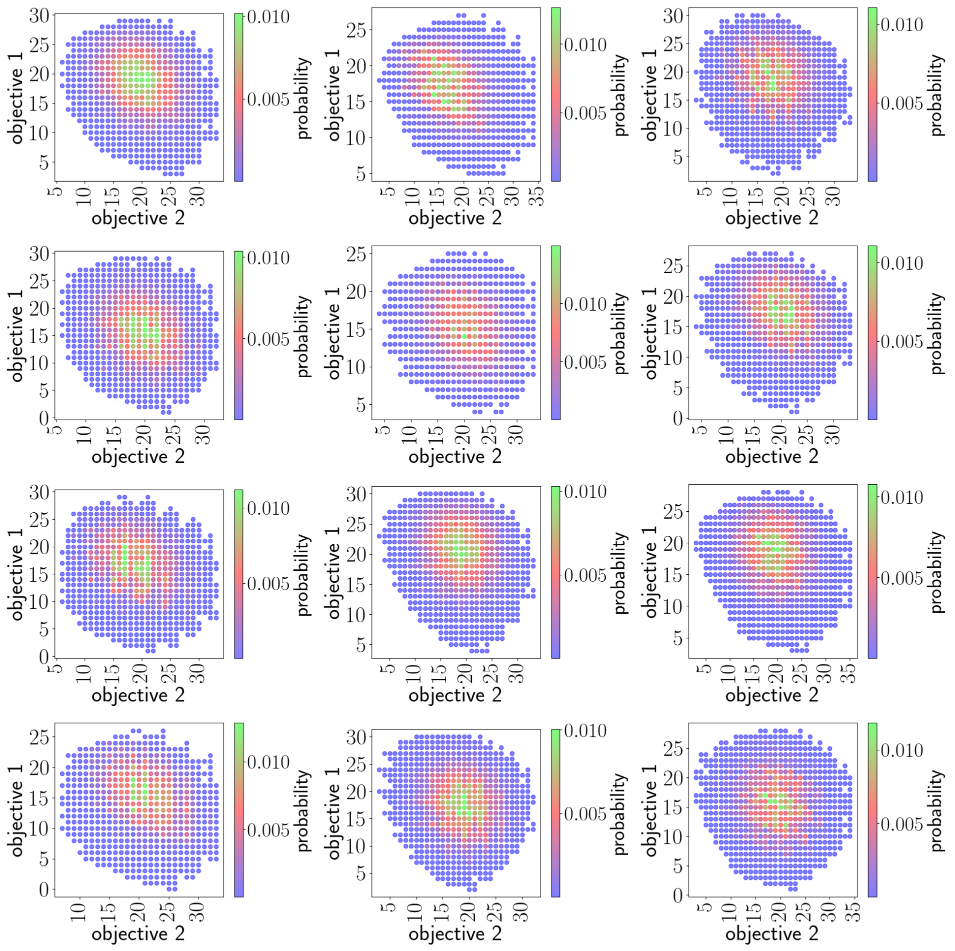


Figure A.4: The return distributions for policies π_{25} - π_{36} computed by DMOVE for Random MO-CoG.

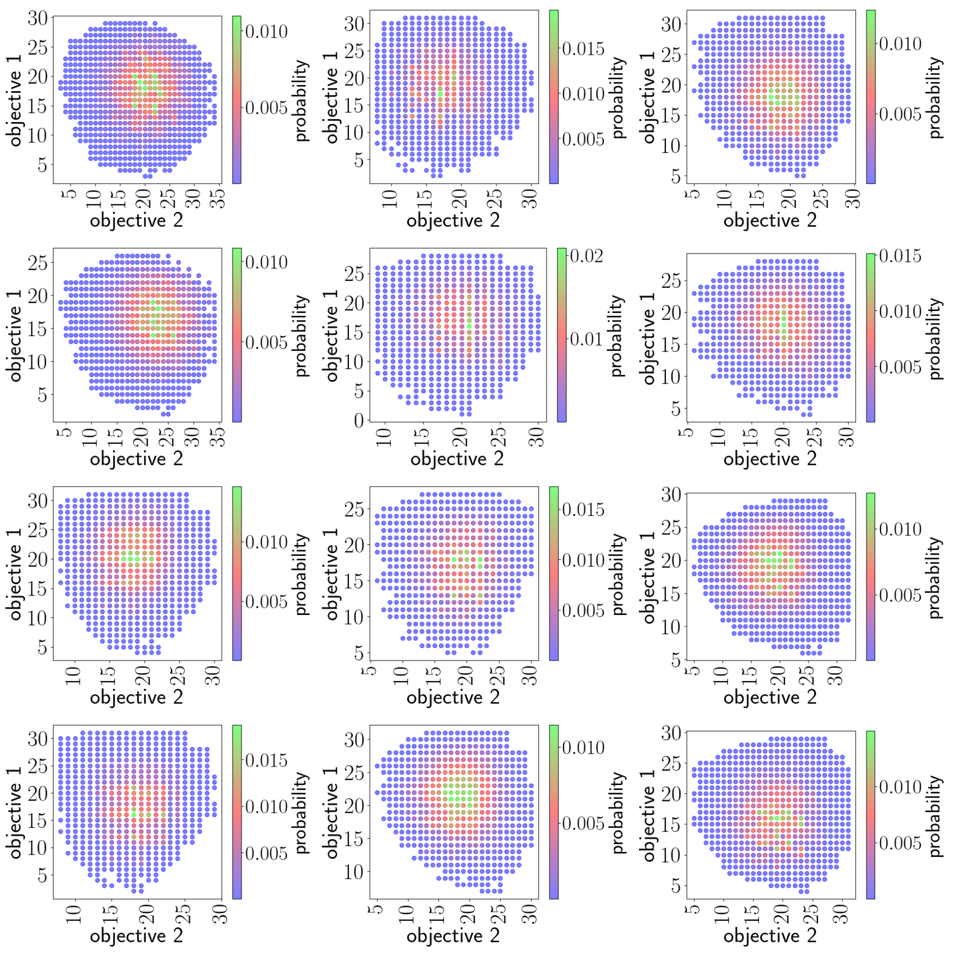


Figure A.5: The return distributions for policies π_{37} - π_{48} computed by DMOVE for Random MO-CoG.

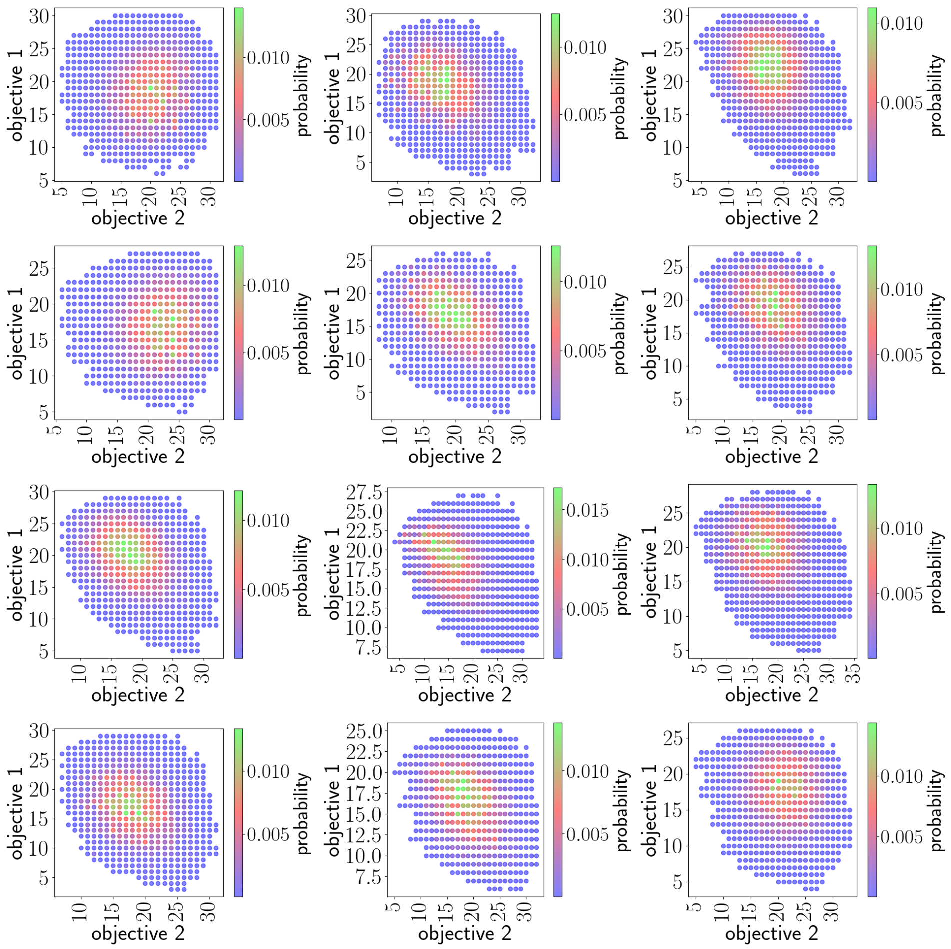


Figure A.6: The return distributions for policies π_{49} - π_{60} computed by DMOVE for Random MO-CoG.

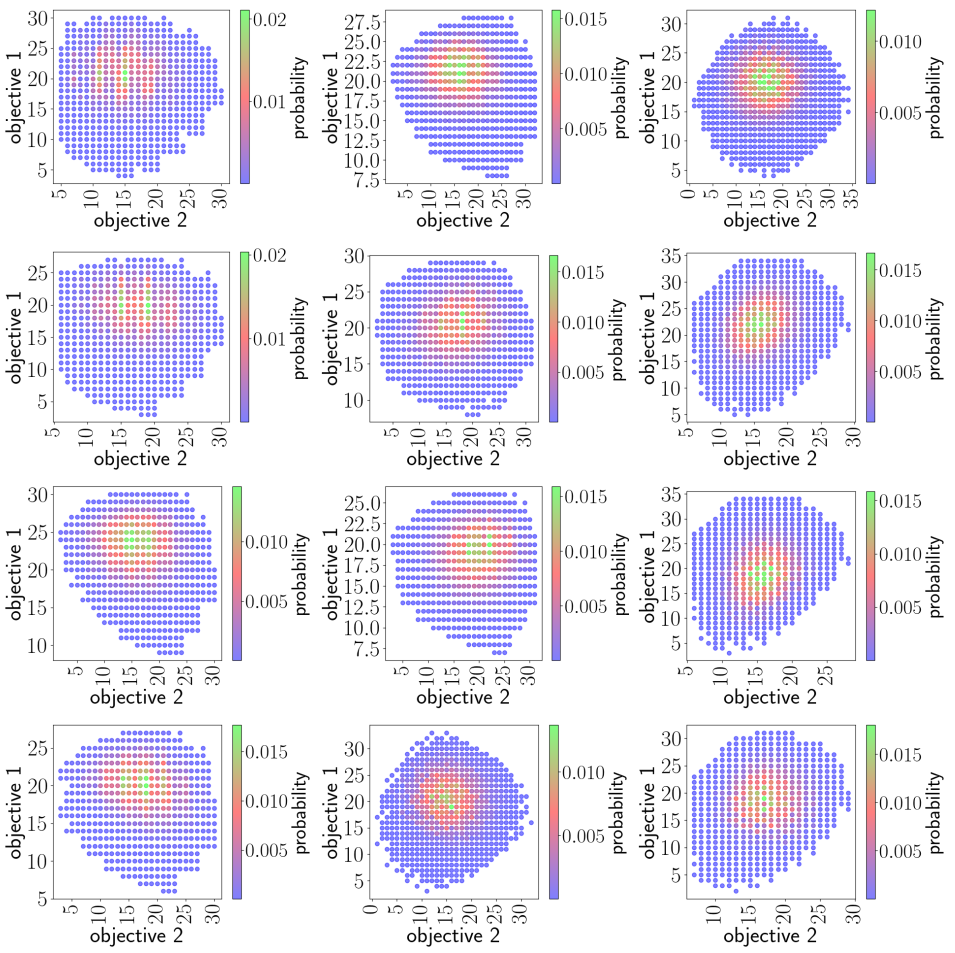


Figure A.7: The return distributions for policies π_{61} - π_{72} computed by DMOVE for Random MO-CoG.

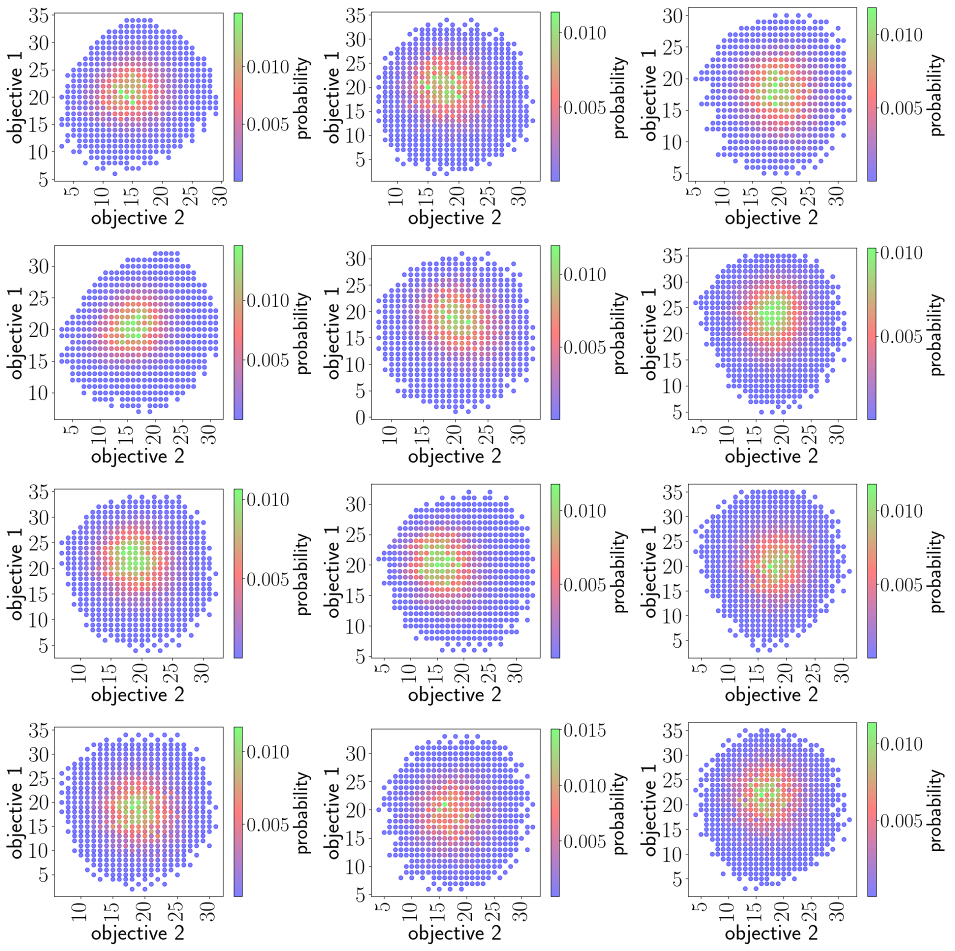


Figure A.8: The return distributions for policies π_{73} - π_{84} computed by DMOVE for Random MO-CoG.

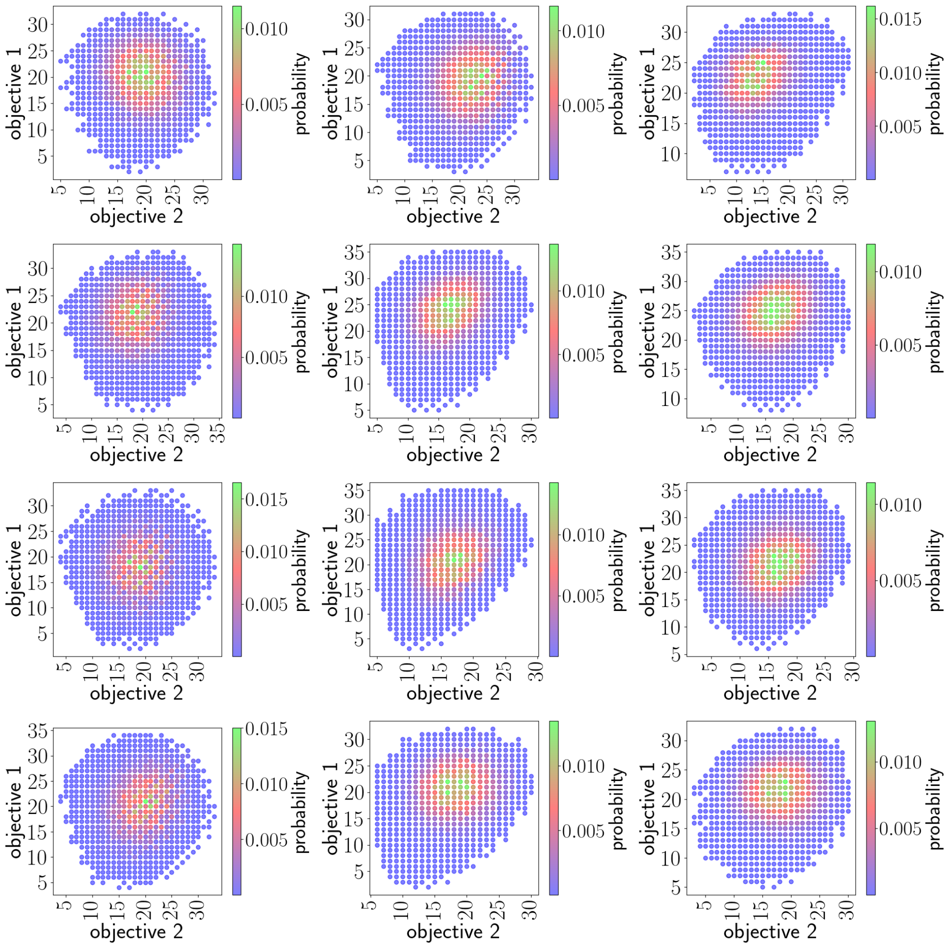


Figure A.9: The return distributions for policies π_{85} - π_{96} computed by DMOVE for Random MO-CoG.

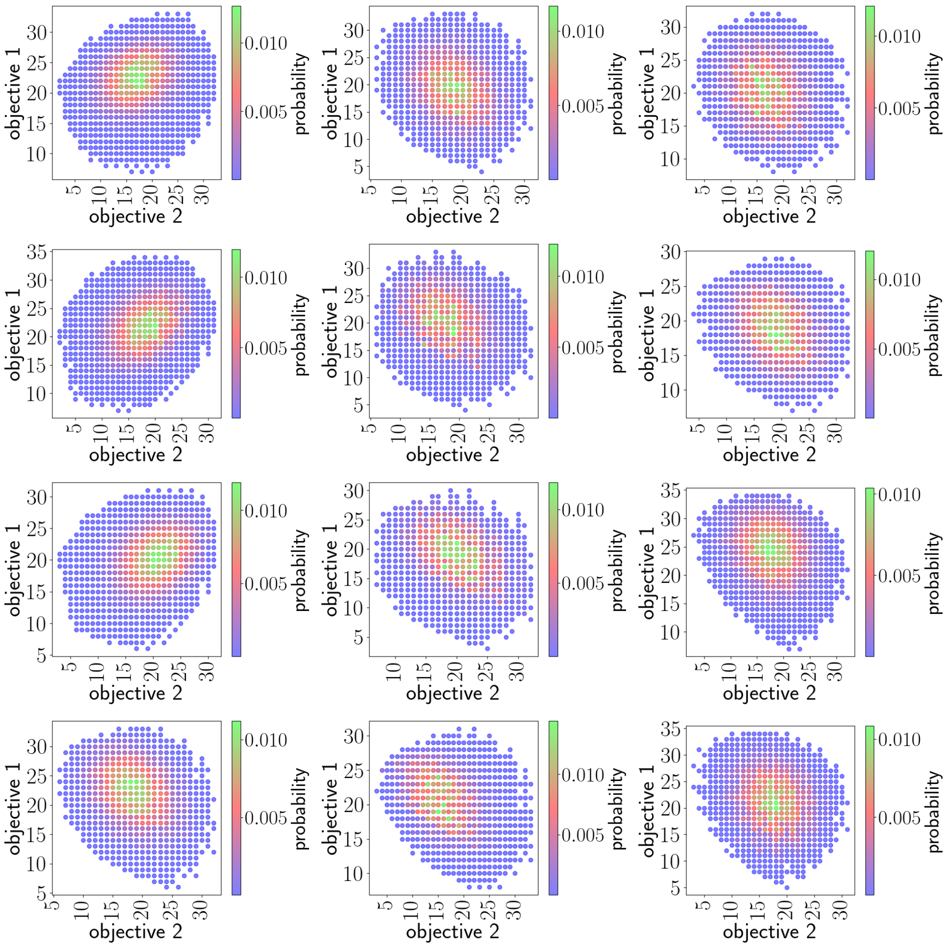


Figure A.10: The return distributions for policies π_{97} - π_{108} computed by DMOVE for Random MO-CoG.

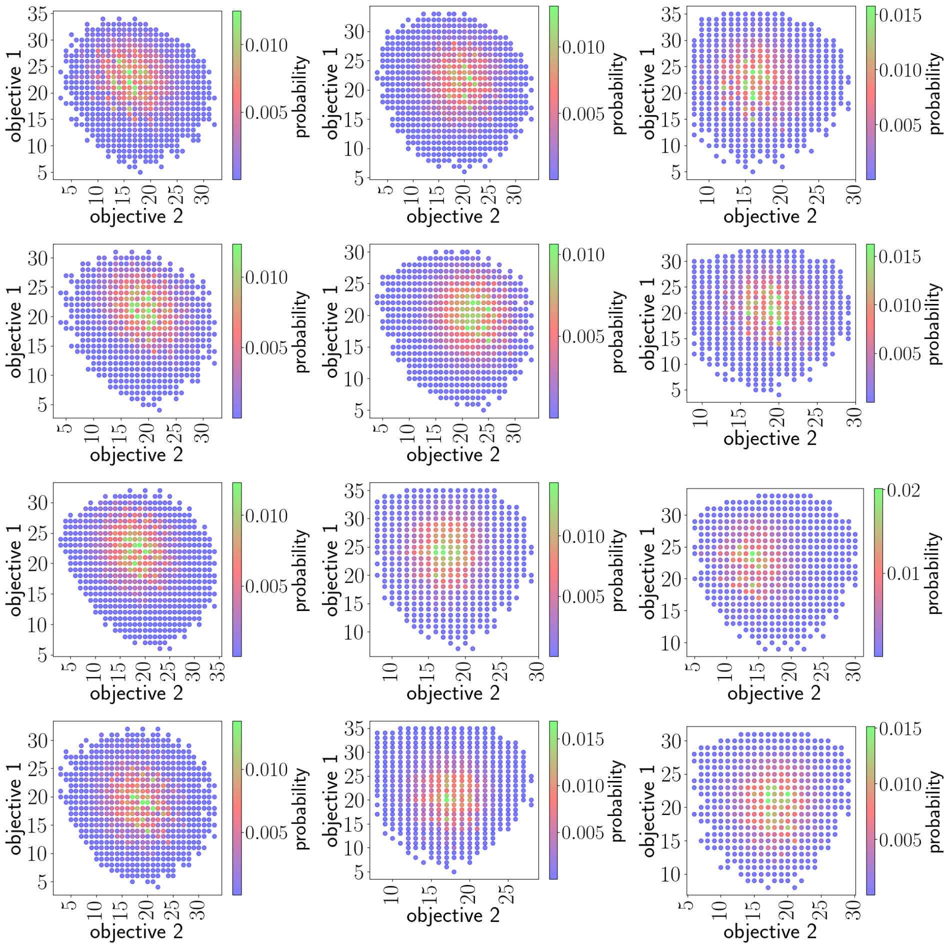


Figure A.11: The return distributions for policies π_{109} - π_{120} computed by DMOVE for Random MO-CoG.

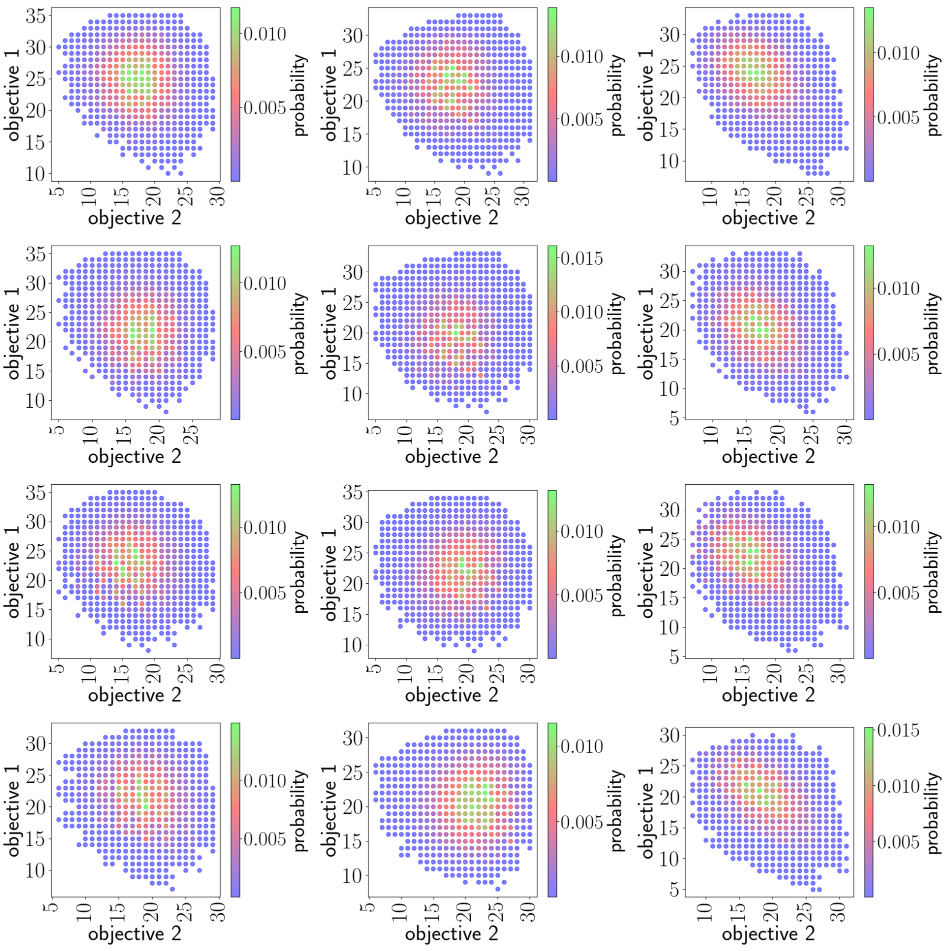


Figure A.12: The return distributions for policies π_{121} - π_{132} computed by DMOVE for Random MO-CoG.

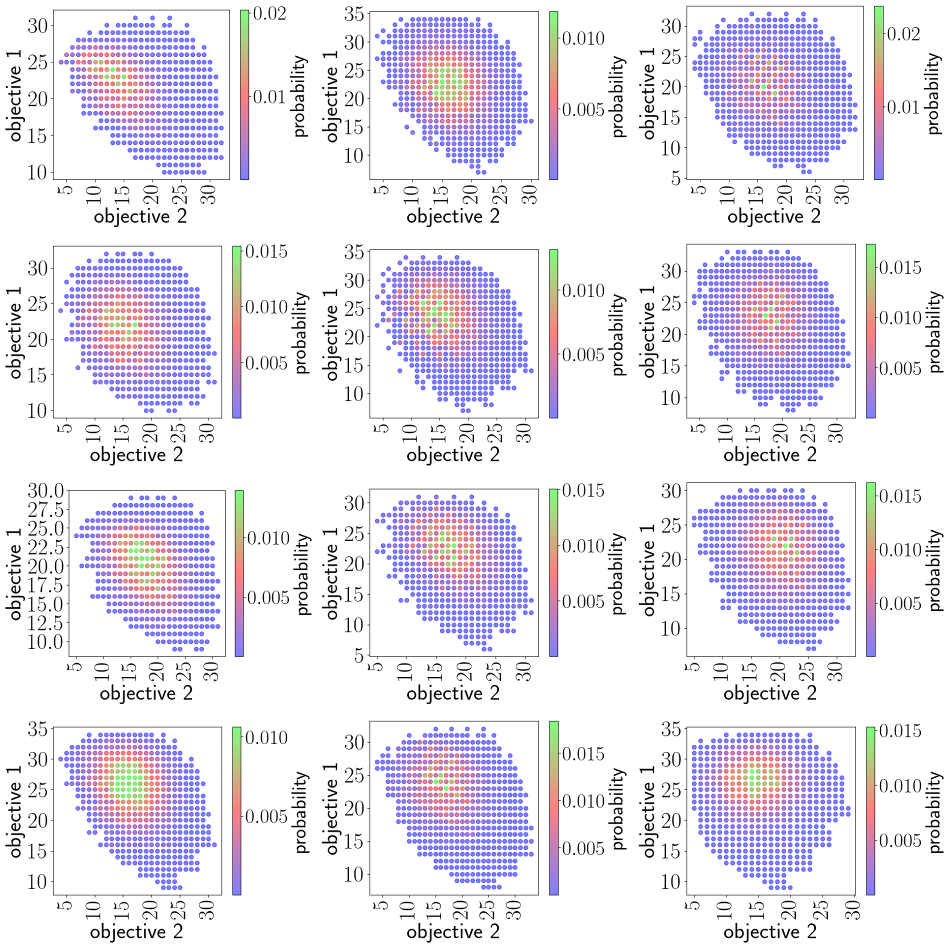


Figure A.13: The return distributions for policies π_{133} - π_{144} computed by DMOVE for Random MO-CoG.

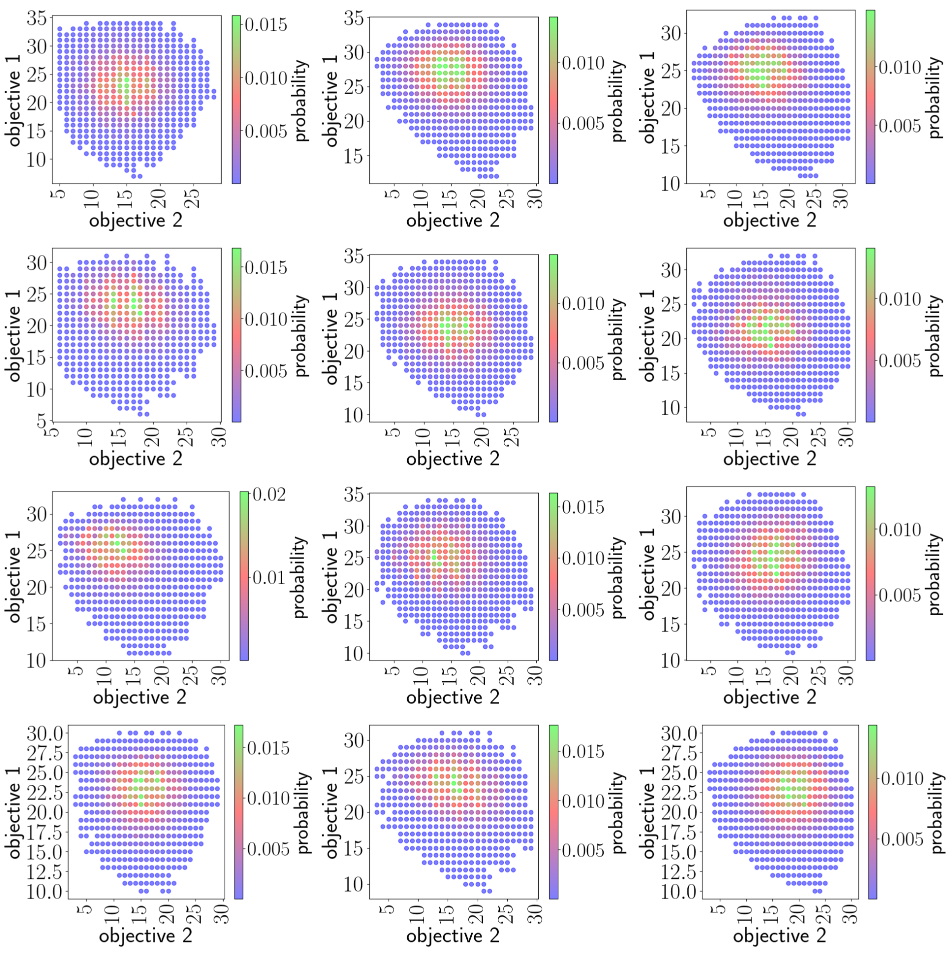


Figure A.14: The return distributions for policies π_{145} - π_{156} computed by DMOVE for Random MO-CoG.

A.6.2 Mining Day

Below the return distributions for each policy in the *ESR set* for a Mining Day instance are presented.

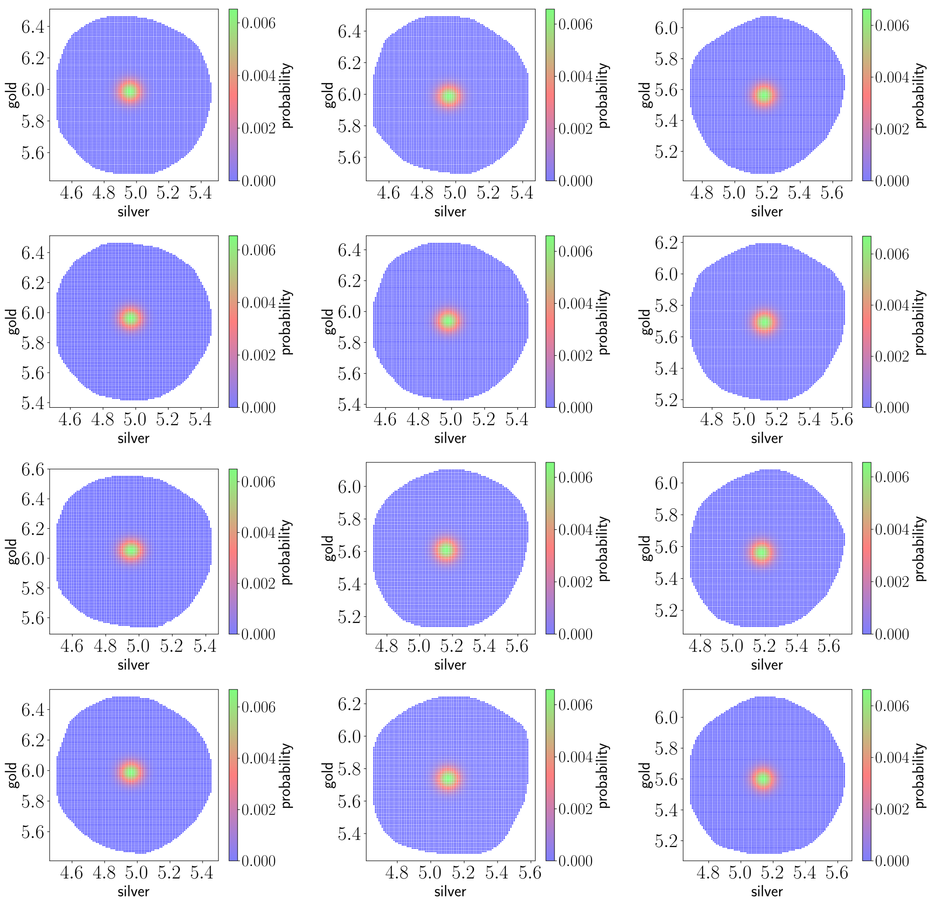


Figure A.15: The return distributions for policies π_1 - π_{12} computed by DMOVE for Mining Day.

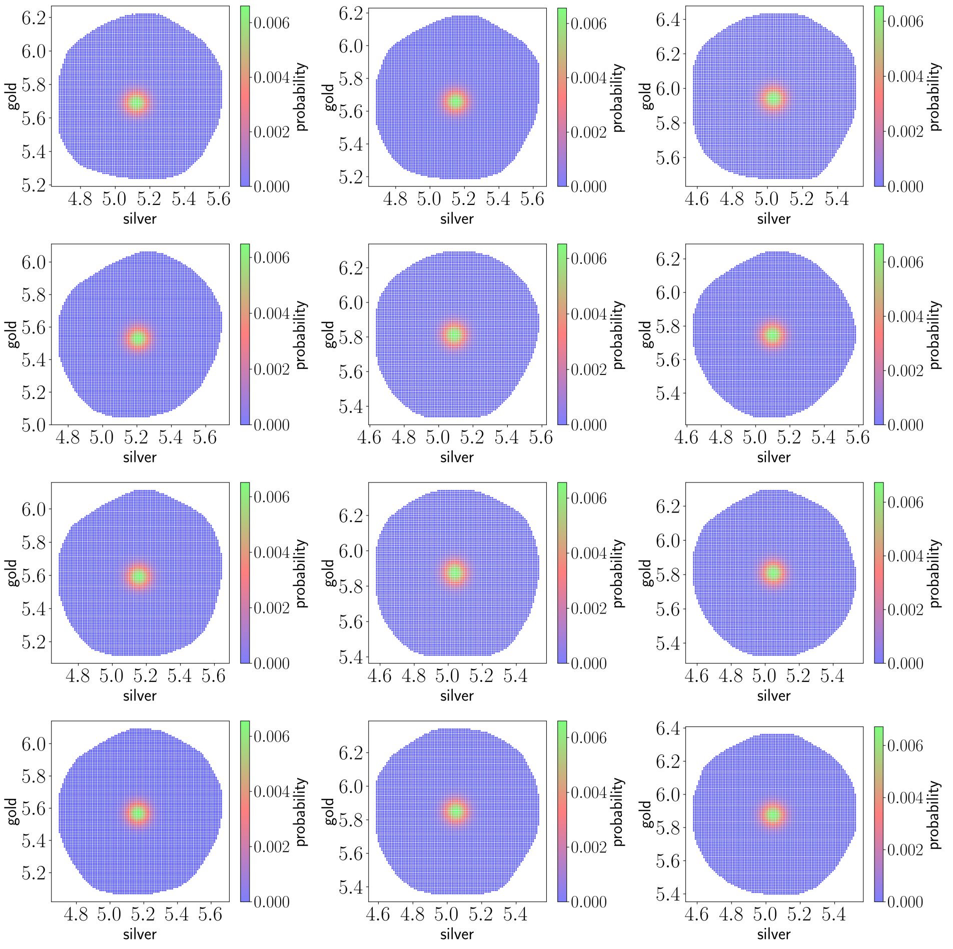


Figure A.16: The return distributions for policies π_{13} - π_{24} computed by DMOVE for Mining Day.

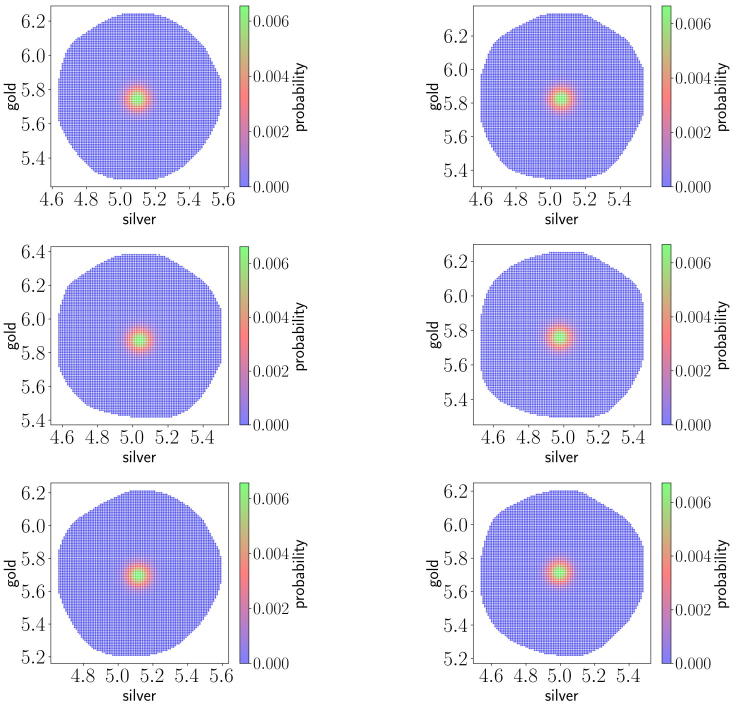


Figure A.17: The return distributions for policies π_{25} - π_{30} computed by DMOVE for Mining Day.

Bibliography

Abdolmaleki, A., S. Huang, L. Hasenclever, M. Neunert, F. Song, M. Zambelli, M. Martins, N. Heess, R. Hadsell, and M. Riedmiller
2020. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*, Pp. 11–22. PMLR.

Abels, A., D. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher
2019. Dynamic weights in multi-objective deep reinforcement learning. In *International Conference on Machine Learning*, Pp. 11–20. PMLR.

Abrams, S., J. Wambua, E. Santermans, L. Willem, E. Kuylen, P. Coletti, P. Libin, C. Faes, O. Petrof, S. A. Herzog, P. Beutels, and N. Hens
2021. Modelling the early phase of the belgian covid-19 epidemic using a stochastic compartmental model and studying its implied future trajectories. *Epidemics*, 35:100449.

Abramson, B.
1987. *The expected-outcome model of two-player games.* PhD thesis, Columbia University.

Aissani, N., B. Beldjilali, and D. Trentesaux
2008. Efficient and effective reactive scheduling of manufacturing system using sarsa-multi-objective agents. In *MOSIM'08: 7th Conference Internationale de Modelisation et Simulation*, Pp. 698–707.

Alegre, L. N., A. Bazzan, and B. C. Da Silva
2022. Optimistic linear support and successor features as a basis for optimal policy transfer. In *International Conference on Machine Learning*, Pp. 394–413. PMLR.

Ali, M. M.
1975. Stochastic dominance and portfolio analysis. *Journal of Financial Economics*, 2(2):205–229.

Atkinson, A. B. and F. Bourguignon
1982. The Comparison of Multi-Dimensioned Distributions of Economic Status. *The Review of Economic Studies*, 49(2):183–201.

Auer, P.
2002. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422.

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

Bai, A., S. Srivastava, and S. Russell
2016. Markovian state and action abstractions for mdps via hierarchical mcts. In *IJCAI*, Pp. 3029–3039.

Bai, A., F. Wu, Z. Zhang, and X. 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*, ICAPS'14, P. 29–37. AAAI Press.

Bargiacchi, E., T. Verstraeten, D. Roijers, A. Nowé, and H. Hasselt
2018. Learning to coordinate with coordination graphs in repeated single-stage multi-agent decision problems. In *International conference on machine learning*, Pp. 482–490. PMLR.

Barrett, L. and S. Narayanan
2008. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*, Pp. 41–47.

Basu, M.
2008. Dynamic economic emission dispatch using nondominated sorting genetic algorithm-ii. *International Journal of Electrical Power and Energy Systems*, 78:140–149.

Bawa, V. S.
1975. Optimal rules for ordering uncertain prospects. *Journal of Financial Economics*, 2(1):95 – 121.

Bawa, V. S.
1978. Safety-first, stochastic dominance, and optimal portfolio choice. *The Journal of Financial and Quantitative Analysis*, 13(2):255–271.

Bawa, V. S.
1982. Research bibliography-stochastic dominance: A research bibliography. *Manage. Sci.*, 28(6):698–712.

Bellemare, M. G., S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang
2020. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82.

Bellemare, M. G., W. Dabney, and R. Munos
2017. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, Pp. 449–458. JMLR. org.

Bellemare, M. G., W. Dabney, and M. Rowland
2023. *Distributional Reinforcement Learning*. MIT Press. http://www.distributional-rl.org.

Bellemare, M. G., Y. Naddaf, J. Veness, and M. Bowling
2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R.
1957a. *Dynamic programming*. Courier Corporation.

Bellman, R.
1957b. A markovian decision process. *Journal of mathematics and mechanics*, Pp. 679–684.

Bonet, B. and H. Geffner
2006. Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to mdps. In *ICAPS*, volume 6, Pp. 142–151.

Bonet, B. and H. Geffner
2012. Action selection for mdps: Anytime ao* versus uct. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, Pp. 1749–1755.

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

Bryce, D., W. Cushing, and S. Kambhampati
2007. Probabilistic planning is multi-objective. *Arizona State University, Tech. Rep. ASU-CSE-07-006.*

Cassandra, A., M. L. Littman, and N. L. Zhang
1997. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, P. 54–61, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Castelletti, A., F. Pianosi, and M. Restelli
2012. Tree-based fitted q-iteration for multi-objective markov decision problems. In *The 2012 international joint conference on neural networks (IJCNN)*, Pp. 1–8. IEEE.

Castelletti, A., F. Pianosi, and M. Restelli
2013. A multiobjective reinforcement learning approach to water resources systems operation: Pareto frontier approximation in a single run. *Water Resources Research*, 49(6):3476–3486.

Cazenave, T. and A. Saffidine
2010. Score bounded monte-carlo tree search. In *International Conference on Computers and Games*, Pp. 93–104. Springer.

Chang, H. S., M. C. Fu, J. Hu, and S. I. Marcus
2005. An adaptive sampling algorithm for solving markov decision processes. *Oper. Res.*, 53(1):126–139.

Chapelle, O. and L. Li
2011. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems*, volume 24.

Choi, E. and S. Johnson
1988. Stochastic dominance and uncertain price prospects. *Center for Agricultural and Rural Development (CARD) at Iowa State University, Center for Agricultural and Rural Development (CARD) Publications*, 55.

Coello, C. C.
2000. Handling preferences in evolutionary multiobjective optimization: A survey. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, volume 1, Pp. 30–37. IEEE.

Cook, L. and J. Jarrett
2018. Using stochastic dominance in multi-objective optimizers for aerospace design under uncertainty. In *American Institute of Aeronautics and Astronautics Journal.*

Coulom, R.
2006. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, Pp. 72–83. Springer.

Dabney, W., G. Ostrovski, D. Silver, and R. Munos
2018a. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, Pp. 1096–1105. PMLR.

Dabney, W., M. Rowland, M. Bellemare, and R. Munos
2018b. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Darling, D. A.
1957. The kolmogorov-smirnov, cramer-von mises tests. *The Annals of Mathematical Statistics*, 28(4):823–838.

Dearden, R., N. Friedman, and S. Russell
1998. Bayesian q-learning. *Aaai/iaai*, 1998:761–768.

Dinh, L., J. Sohl-Dickstein, and S. Bengio
2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803.*

Drugan, M. M. and A. Nowe
2013. Designing multi-objective multi-armed bandits algorithms: A study. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Pp. 1–8.

Duffie, D. and J. Pan
1997. An overview of value at risk. *Journal of derivatives*, 4(3):7–49.

Dulac-Arnold, G., N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester
2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468.

Eckles, D. and M. Kaptein
2014. Thompson sampling with the online bootstrap. *CoRR*, abs/1410.4009.

Eckles, D. and M. Kaptein
2019. Bootstrap thompson sampling and sequential decision problems in the behavioral sciences. *SAGE Open*, 9(2).

Efron, B.
2012. Bayesian inference and the parametric bootstrap. *Ann. Appl. Stat.*, 6(4):1971–1997.

Engle, R. and S. Manganelli
2001. Value at risk models in finance. Technical report, European Central Bank.

Eriksson, H., D. Basu, M. Alibeigi, and C. Dimitrakakis
2022. Sentinel: taming uncertainty with ensemble based distributional reinforcement learning. In *Uncertainty in Artificial Intelligence*, Pp. 631–640. PMLR.

Fawzi, A., M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli
2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53.

Fishburn, P. C.
1978. Non-cooperative stochastic dominance games. *International Journal of Game Theory*, 7(1):51–61.

Goodfellow, I., Y. Bengio, and A. Courville
2016. *Deep learning.* MIT press.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio
2020. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Greenwald, A., K. Hall, R. Serrano, et al.
2003. Correlated q-learning. In *ICML*, volume 3, Pp. 242–249.

Guestrin, C., D. Koller, and R. Parr
2001. Multiagent planning with factored mdps. *Advances in neural information processing systems*, 14.

Guo, Y., A. Zeman, and R. Li
2009. A reinforcement learning approach to setting multi-objective goals for energy demand management. *International Journal of Agent Technologies and Systems (IJATS)*, 1(2):55–70.

Haarnoja, T., A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al.
2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hadar, J. and W. R. Russell
1969. Rules for ordering uncertain prospects. *The American Economic Review*, 59(1):25–34.

Hayes, C. F., E. Howley, and P. Mannion
2020. Dynamic thresholded lexicograpic ordering. In *Adaptive and Learning Agents Workshop (at AAMAS 2020)*.

Hayes, C. F., M. Reymond, D. M. Roijers, E. Howley, and P. Mannion
2021a. 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*, Pp. 1530–1532.

Hayes, C. F., M. Reymond, D. M. Roijers, E. Howley, and P. Mannion
2021b. Risk-aware and multi-objective decision making with distributional monte carlo tree search. *In: Proceedings of the Adaptive and Learning Agents workshop at AAMAS 2021)*.

Hayes, C. F., D. M. Roijers, E. Howley, and P. Mannion
2022a. Decision-theoretic planning for the expected scalarised returns. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, Pp. 1621–1623.

Hayes, C. F., D. M. Roijers, E. Howley, and P. Mannion
2022b. Multi-objective distributional value iteration. In *Adaptive and Learning Agents Workshop (AAMAS 2022)*.

Hayes, C. F., R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz, E. Howley, A. A. Irissappane, P. Mannion, A. Nowé, G. Ramos, M. Restelli, P. Vamplew, and D. M. Roijers
2022c. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26.

Hayes, C. F., T. Verstraeten, D. M. Roijers, E. Howley, and P. Mannion
2021c. Dominance criteria and solution sets for the expected scalarised returns. In *Proceedings of the Adaptive and Learning Agents workshop at AAMAS 2021.*

Hayes, C. F., T. Verstraeten, D. M. Roijers, E. Howley, and P. Mannion
2022d. Expected scalarised returns dominance: A new solution concept for multi-objective decision making. *Neural Computing and Applications*, Pp. 1–21.

Hayes, C. F., T. Verstraeten, D. M. Roijers, E. Howley, and P. Mannion
2022e. Multi-objective coordination graphs for the expected scalarised returns with generative flow models. *European Workshop on Reinforcement Learning (EWRL).*

Ho, J., A. Jain, and P. Abbeel
2020. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851.

Huang, R., M. M. Ajallooeian, C. Szepesvári, and M. Müller
2017. Structured best arm identification with fixed confidence. In *Proceedings of the 28th International Conference on Algorithmic Learning Theory*, S. Hanneke and L. Reyzin, eds., volume 76 of *Proceedings of Machine Learning Research*, Pp. 593–616. PMLR.

Jumper, J., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al.
2021. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

Kaufmann, E. and W. M. Koolen
2017. Monte-carlo tree search by best arm identification. *Advances in Neural Information Processing Systems*, 30.

Kiran, B. R., I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez
2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems.*

Kobyzev, I., S. J. Prince, and M. A. Brubaker
2020. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979.

Kocsis, L. and C. Szepesvári
2006. Bandit based monte-carlo planning. In *Machine Learning: ECML*, volume 2006, Pp. 282–293.

Kok, J. R. and N. Vlassis
2004. Sparse cooperative q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, P. 61.

Koller, D. and N. Friedman
2009. *Probabilistic graphical models: principles and techniques.* MIT press.

Kompella, V., R. Capobianco, S. Jong, J. Browne, S. Fox, L. Meyers, P. Wurman, and P. Stone
2020. Reinforcement learning for optimization of covid-19 mitigation policies. *arXiv preprint arXiv:2010.10560.*

Konda, V. and J. Tsitsiklis
1999. Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Kumar, A., X. B. Peng, and S. Levine
2019. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465.*

LeCun, Y., Y. Bengio, and G. Hinton
2015. Deep learning. *nature*, 521(7553):436–444.

Leshno, M. and H. Levy
2002. Preferred by "all" and preferred by "most" decision makers: Almost stochastic dominance. *Management Science*, 48(8):1074–1085.

Levhari, D., J. Paroush, and B. Peleg
1975. Efficiency analysis for multivariate distributions. *The Review of Economic Studies*, 42(1):87–91.

Levy, H.
1992. Stochastic dominance and expected utility: Survey and analysis. *Management Science*, 38(4):555–593.

Levy, H. and M. Robinson
2006. *Stochastic dominance: Investment decision making under uncertainty*, volume 34. Springer.

Li, L., W. Chu, J. Langford, and R. E. Schapire
2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, Pp. 661–670.

Li, Y.
2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Lin, L.-J.
1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321.

Lyle, C., M. G. Bellemare, and P. S. Castro
2019. A comparative analysis of expected and distributional reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, Pp. 4504–4511.

Malerba, F. and P. Mannion
2021. Evaluating tunable agents with non-linear utility functions under expected scalarised returns. In *Multi-Objective Decision Making Workshop (MODeM 2021)*.

Mannion, P.
2017. Knowledge-based multi-objective multi-agent reinforcement learning. *PhD Thesis, NUI, Galway.*

Mannion, P., S. Devlin, J. Duggan, and E. Howley
2018. Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning. *The Knowledge Engineering Review*, 33:e23.

Mannion, P., S. Devlin, K. Mason, J. Duggan, and E. Howley
2017. Policy invariance under reward transformations for multi-objective reinforcement learning. *Neurocomputing*, 263:60–73.

Mannion, P., F. Heinz, T. G. Karimpanal, and P. Vamplew
2021. Multi-objective decision making for trustworthy ai. *Multi-Objective Decision Making Workshop (MODeM 2021).*

Mannion, P., K. Mason, S. Devlin, E. Howley, and J. Duggan
2016. Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS).*

Marinescu, R.
2009. Exploiting problem decomposition in multi-objective constraint optimization. In *International Conference on Principles and Practice of Constraint Programming*, Pp. 592–607. Springer.

Marinescu, R., A. Razak, and N. Wilson
2012. Multi-objective influence diagrams. *arXiv preprint arXiv:1210.4911.*

Marinescu, R., A. Razak, and N. Wilson
2017. Multi-objective influence diagrams with possibly optimal policies. In *Thirty-First AAAI Conference on Artificial Intelligence.*

Martin, J., M. Lyskawinski, X. Li, and B. Englot
2020. Stochastically dominant distributional reinforcement learning. In *International Conference on Machine Learning*, Pp. 6745–6754. PMLR.

Mas-Colell, A., M. D. Whinston, J. R. Green, et al.
1995. *Microeconomic theory*, volume 1. Oxford university press New York.

Mason, K. and S. Grijalva
2019. A review of reinforcement learning for autonomous building energy management. *Computers & Electrical Engineering*, 78:300–312.

Mavrin, B., H. Yao, L. Kong, K. Wu, and Y. Yu
2019. Distributional reinforcement learning for efficient exploration. In *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, eds., volume 97 of *Proceedings of Machine Learning Research*, Pp. 4424–4434, Long Beach, California, USA. PMLR.

Mirhoseini, A., A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, et al.
2020. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746.*

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu
2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, Pp. 1928–1937. PMLR.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller
2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602.*

Moerland, T. M., J. Broekens, and C. M. Jonker
2020. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712.*

Monahan, G. E.
1982. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16.

Mossalam, H., Y. M. Assael, D. M. Roijers, and S. Whiteson
2016. Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707*.

Nakayama, H., T. Tanino, and Y. Sawaragi
1981. Stochastic dominance for decision problems with multiple attributes and/or multiple decision-makers. *IFAC Proceedings Volumes*, 14(2):1397 – 1402. 8th IFAC World Congress on Control Science and Technology for the Progress of Society, Kyoto, Japan, 24-28 August 1981.

Natarajan, S. and P. Tadepalli
2005. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, Pp. 601–608.

Newton, M. and A. Raftery
1994. Approximate bayesian inference by the weighted likelihood bootstrap. *Journal of the Royal Statistical Society Series B-Methodological*, 56:3 – 48.

O'Callaghan, D. and P. Mannion
2021. Exploring the impact of tunable agents in sequential social dilemmas. In *Proceedings of the Adaptive and Learning Agents Workshop (ALA-21) at AAMAS*.

Osband, I., C. Blundell, A. Pritzel, and B. Van Roy
2016. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29.

Owen, A. B. and D. Eckles
2012. Bootstrapping data arrays of arbitrary order. *The Annals of Applied Statistics*, 6(3):895–927.

Oza, N. C. and S. J. Russell
2001. Online bagging and boosting. In *International Workshop on Artificial Intelligence and Statistics*, Pp. 229–236. PMLR.

Pan, A., W. Xu, L. Wang, and H. Ren
2020. Additional planning with multiple objectives for reinforcement learning. *Knowledge-Based Systems*, 193:105392.

Papamakarios, G., E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan
2021. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64.

Pareto, V.
1896. *Manuel d'Economie Politique*, volume 1. Giard, Paris.

Parisi, S., M. Pirotta, and J. Peters
2017. Manifold-based multi-objective policy search with sample reuse. *Neurocomputing*, 263:3–14.

Perez, J., C. Germain-Renaud, B. Kégl, and C. Loomis
2009. Responsive elastic computing. In *Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, Pp. 55–64.

Petersen, B. K., M. L. Larma, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim
2020. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*.

Pineau, J., G. Gordon, S. Thrun, et al.
2003. Point-based value iteration: An anytime algorithm for pomdps. In *Ijcai*, volume 3, Pp. 1025–1032. Citeseer.

Puterman, M. L.
1990. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434.

Rădulescu, R., P. Mannion, D. M. Roijers, and A. Nowé
2020. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems*, 34(10).

Rao, A. and T. Jelvis
2022. *Foundations of Reinforcement Learning with Applications in Finance*. CRC Press.

Reymond, M., E. Bargiacchi, and A. Nowé
2022a. Pareto conditioned networks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, Pp. 1110–1118.

Reymond, M., C. F. Hayes, D. M. Roijers, D. Steckelmacher, and A. Nowé
2021. Actor-critic multi-objective reinforcement learning for non-linear utility functions. In *Multi-Objective Decision Making Workshop (MODeM 2021)*.

Reymond, M., C. F. Hayes, L. Willem, R. Rădulescu, S. Abrams, D. M. Roijers, E. Howley, P. Mannion, N. Hens, A. Nowé, et al.
2022b. Exploring the pareto front of multi-objective covid-19 mitigation policies using reinforcement learning. *arXiv preprint arXiv:2204.05027*.

Reymond, M. and A. Nowé
2019. Pareto-dqn: Approximating the pareto front in complex multi-objective decision problems. In *Proceedings of the Adaptive and Learning Agents Workshop (ALA-19) at AAMAS*.

Richard, S. F.
1975. Multivariate risk aversion, utility independence and separable utility functions. *Management Science*, 22(1):12–21.

Rockafellar, R. T. and S. Uryasev
2002. Conditional value-at-risk for general loss distributions. *Journal of banking & finance*, 26(7):1443–1471.

Rockafellar, R. T., S. Uryasev, et al.
2000. Optimization of conditional value-at-risk. *Journal of risk*, 2(3):21–41.

Roijers, D., L. Zintgraf, P. Libin, and A. Nowe
2018a. Interactive multi-objective reinforcement learning in multi-armed bandits for any utility function. In *Proceedings of the Adaptive and Learning Agents Workshop (ALA-18) at AAMAS*.

Roijers, D. M.
2016. Multi-objective decision-theoretic planning: Abstract. *AI Matters*, 2(4):11–12.

Roijers, D. M., W. Röpke, A. Nowé, and R. Rădulescu
2021. On following pareto-optimal policies in multi-objective planning and reinforcement learning. In *Proceedings of the Multi-Objective Decision Making (MODeM) Workshop*.

Roijers, D. M., D. Steckelmacher, and A. Nowé
2018b. Multi-objective reinforcement learning for the expected utility of the return. In *Proceedings of the Adaptive and Learning Agents workshop at FAIM 2018*.

Roijers, D. M., P. Vamplew, S. Whiteson, and R. Dazeley
2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113.

Roijers, D. M. and S. Whiteson
2017. Multi-objective decision making. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 11(1):1–129.

Roijers, D. M., S. Whiteson, and F. A. Oliehoek
2015. Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, 52:399–443.

Roijers, D. M., L. M. Zintgraf, P. Libin, M. Reymond, E. Bargiacchi, and A. Nowé
2020. Interactive multi-objective reinforcement learning in multi-armed bandits with gaussian process utility models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Pp. 463–478. Springer.

Rollón, E.
2008. *Multi-objective optimization in graphical models.* PhD thesis, Universitat Politècnica de Catalunya.

Rollón, E. and J. Larrosa
2006. Bucket elimination for multiobjective optimization problems. *Journal of Heuristics*, 12(4):307–328.

Röpke, W., R. Radulescu, D. M. Roijers, and A. Nowe
2021. Communication strategies in multi-objective normal-form games. In *Adaptive and Learning Agents Workshop 2021.*

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

Rădulescu, R., P. Mannion, Y. Zhang, D. M. Roijers, and A. Nowé
2020. A utility-based analysis of equilibria in multi-objective normal form games. *The Knowledge Engineering Review*, 35(e32).

Rubin, D. B.
1981. The bayesian bootstrap. *The Annals of Statistics*, 9(1):130–134.

Ruiz-Montiel, M., L. Mandow, and J.-L. Pérez-de-la Cruz
2017. A temporal difference method for multi-objective reinforcement learning. *Neurocomputing*, 263:15–25.

Russell, S. J. and P. Norvig
2010. *Artificial intelligence a modern approach.* Pearson Education, Inc.

Russo, D. and B. Van Roy
2014. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243.

Russo, D. J., B. Van Roy, A. Kazerouni, I. Osband, Z. Wen, et al.
2018. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96.

Scarsini, M.
1988. Dominance conditions for multivariate utility functions. *Management Science*, 34(4):454–460.

Schappacher, N.
1996. Beppo levi and the arithmetic of elliptic curves. *The Mathematical Intelligencer*, 18(1):57—69.

Schmidhuber, J.
2019. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875.*

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov
2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.*

Shabani, N.
2009. *Incorporating flood control rule curves of the Columbia River hydroelectric system in a multireservoir reinforcement learning optimization model.* PhD thesis, Citeseer.

Shen, W., F. Trevizan, S. Toyer, S. Thiébaux, and L. Xie
2019. Guiding mcts with generalized policies for probabilistic planning. *HSDIP 2019*, P. 63.

Siddique, U., P. Weng, and M. Zimmer
2020. Learning fair policies in multi-objective (deep) reinforcement learning with average and discounted rewards. In *International Conference on Machine Learning*, Pp. 8905–8915. PMLR.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al.
2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al.
2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D., S. Singh, D. Precup, and R. S. Sutton
2021. Reward is enough. *Artificial Intelligence*, 299:103535.

Silver, D. and J. Veness
2010. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds., volume 23. Curran Associates, Inc.

Slivkins, A. et al.
2019. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286.

Somani, A., N. Ye, D. Hsu, and W. S. Lee
2013. Despot: Online pomdp planning with regularization. *Advances in neural information processing systems*, 26.

Spaan, M. T.
2012. Partially observable markov decision processes. In *Reinforcement Learning*, Pp. 387–414. Springer.

Sriboonchitta, S., W.-K. Wong, s. Dhompongsa, and H. Nguyen
2009. *Stochastic Dominance and Applications to Finance, Risk and Economics.* Chapman and Hall/CRC.

Sunberg, Z. N. and M. J. Kochenderfer
2018. Online algorithms for pomdps with continuous state, action, and observation spaces. In *Twenty-Eighth International Conference on Automated Planning and Scheduling.*

Sutton, R. S. and A. G. Barto
2018. *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book.

Tamar, A., Y. Wu, G. Thomas, S. Levine, and P. Abbeel
2016. Value iteration networks. *Advances in neural information processing systems*, 29.

Tesauro, G., V. T. Rajan, and R. Segal
2010. Bayesian inference in monte-carlo tree search. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, P. 580–588, Arlington, Virginia, USA. AUAI Press.

Thomas, M. and A. T. Joy
2006. *Elements of information theory*. Wiley-Interscience.

Thompson, W. R.
1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294.

Tomczak, J. M.
2022. *Deep Generative Modeling*. Springer Nature.

Turgay, E., D. Oner, and C. Tekin
2018. Multi-objective contextual bandit problem with similarity information. In *International Conference on Artificial Intelligence and Statistics*, Pp. 1673–1681. PMLR.

Vamplew, P., R. Dazeley, A. Berry, R. Issabekov, and E. Dekker
2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84:51–80.

Vamplew, P., R. Dazeley, C. Foale, S. Firmin, and J. Mummery
2018. Human-aligned artificial intelligence is a multiobjective problem. *Ethics and Information Technology*, 20(1):27–40.

Vamplew, P., C. Foale, and R. Dazeley
2020. A demonstration of issues with value-based multi objective reinforcement learning under stochastic state transitions. In *Adaptive and Learning Agents Workshop (AAMAS 2020)*.

Vamplew, P., C. Foale, and R. Dazeley
2021a. The impact of environmental stochasticity on value-based multiobjective reinforcement learning. In *Neural Computing and Applications*.

Vamplew, P., C. Foale, R. Dazeley, and A. Bignold
2021b. Potential-based multiobjective reinforcement learning approaches to low-impact agents for ai safety. *Engineering Applications of Artificial Intelligence*, 100:104186.

Vamplew, P., B. J. Smith, J. Källström, G. Ramos, R. Rădulescu, D. M. Roijers, C. F. Hayes, F. Heintz, P. Mannion, P. J. Libin, et al.
2022. Scalar reward is not enough: A response to silver, singh, precup and sutton (2021). *Autonomous Agents and Multi-Agent Systems*, 36(2):1–19.

Vamplew, P., J. Yearwood, R. Dazeley, and A. Berry
2008. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *AI 2008: Advances in Artificial Intelligence*, W. Wobcke and M. Zhang, eds., Pp. 372–378, Berlin, Heidelberg. Springer Berlin Heidelberg.

Van Hasselt, H., A. Guez, and D. Silver
2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Van Moffaert, K. and A. Nowé
2014a. Multi-objective reinforcement learning using sets of Pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512.

Van Moffaert, K. and A. Nowé
2014b. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512.

Veness, J., K. S. Ng, M. Hutter, W. Uther, and D. Silver
2011. A monte-carlo aixi approximation. *J. Artif. Int. Res.*, 40(1):95–142.

Verstraeten, T., E. Bargiacchi, P. J. Libin, J. Helsen, D. M. Roijers, and A. Nowé
2020. Multi-agent thompson sampling for bandit applications with sparse neighbourhood structures. *Scientific reports*, 10(1):1–13.

Verstraeten, T., P.-J. Daems, E. Bargiacchi, D. M. Roijers, P. J. Libin, and J. Helsen
2021. Scalable optimization for wind farm control using coordination graphs. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, Pp. 1362–1370.

Wakuta, K. and K. Togawa
1998. Solution procedures for multi-objective markov decision processes. *Optimization*, 43(1):29–46.

Wang, W. and M. Sebag
2012. Multi-objective Monte-Carlo tree search. In *Proceedings of Machine Learning Research*, S. C. H. Hoi and W. Buntine, eds., volume 25, Pp. 507–522, Singapore Management University, Singapore. PMLR.

Wang, Z., T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas
2016. Dueling network architectures for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, eds., volume 48 of *Proceedings of Machine Learning Research*, Pp. 1995–2003, New York, New York, USA. PMLR.

Watkins, C. J. and P. Dayan
1992. Q-learning. *Machine learning*, 8(3):279–292.

White, D.
1982. Multi-objective infinite-horizon discounted markov decision processes. *Journal of mathematical analysis and applications*, 89(2):639–647.

Wiering, M. A. and E. D. de Jong
2007. Computing optimal stationary policies for multi-objective markov decision processes. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, Pp. 158–165.

Wiering, M. A., M. Withagen, and M. M. Drugan
2014. Model-based multi-objective reinforcement learning. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Pp. 1–6. IEEE.

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

Wolfstetter, E.
1999. *Topics in Microeconomics: Industrial Organization, Auctions, and Incentives*. Cambridge University Press.

Wray, K. H., S. Zilberstein, and A.-I. Mouaddib
2015. Multi-objective mdps with conditional lexicographic reward preferences. In *Twenty-ninth AAAI conference on artificial intelligence*.

Wurman, P. R., S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, et al.
2022. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228.

Xu, J., Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik
2020. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *International Conference on Machine Learning*, Pp. 10607–10616. PMLR.

Yahyaa, S. and B. Manderick
2015. Thompson sampling for multi-objective multi-armed bandits problem. In *Proceedings of the 23rd European Symposium on Artificial Neural Network, Computational Intelligence and Machine Learning*, P. 47. Presses universitaires de Louvain.

Yahyaa, S. Q., M. M. Drugan, and B. Manderick
2014. Annealing-pareto multi-objective multi-armed bandit algorithm. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Pp. 1–8. IEEE.

Yang, R., X. Sun, and K. Narasimhan
2019. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *Advances in Neural Information Processing Systems*, Pp. 14636–14647.

Yu, C., J. Liu, S. Nemati, and G. Yin
2021. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36.

Zhang, N. L. and D. Poole
1996. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328.

Zhang, P., X. Chen, L. Zhao, W. Xiong, T. Qin, and T.-Y. Liu
2021. Distributional reinforcement learning for multi-dimensional reward functions. *Advances in Neural Information Processing Systems*, 34:1519–1529.

Zintgraf, L. M., T. V. Kanters, D. M. Roijers, F. Oliehoek, and P. Beau
2015. Quality assessment of morl algorithms: A utility-based approach. In *Benelearn 2015: Proceedings of the 24th Annual Machine Learning Conference of Belgium and the Netherlands.*

Zintgraf, L. M., D. M. Roijers, S. Linders, C. M. Jonker, and A. Nowé
2018. Ordered preference elicitation strategies for supporting multi-objective decision making. In *17th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2018*, Pp. 1477–1485. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).