

Multi-Objective Categorical Deep Q-Networks

Farès Chouaki^{a,*}, Aurelie Beynier^a, Nicolas Maudet^a and Paolo Viappiani^b

^aLIP6, Sorbonne Université, CNRSF-75005 ParisFrance

^bLAMSADE, CNRS, Université Paris Dauphine - PSL Paris-75016France

ORCID (Farès Chouaki): <https://orcid.org/0009-0005-2280-2324>, ORCID (Aurelie Beynier):

<https://orcid.org/0000-0002-3812-6755>, ORCID (Nicolas Maudet): <https://orcid.org/0000-0002-4232-069X>,

ORCID (Paolo Viappiani): <https://orcid.org/0000-0002-7922-3877>

Abstract. Motivated by recent advances in distributional reinforcement learning on one hand and multi-objective reinforcement learning (MORL) on the other, we are interested in the design of value based algorithms that, given a possibly non-linear scalarization function, learn the policy with maximal expected scalarized return. Our approach is then extended to propose a first value-based multi-policy algorithm for solving MORL problems under the expected scalarized return criterion. The proposed algorithms are tested on environments from the MO-gymnasium benchmark. The results are promising, on the one hand they show that by leveraging the recent advances in value based deep reinforcement learning our algorithm learns policies as good as those obtained by existing approaches in the literature while requiring fewer interactions with the environments. On the other hand, given a set of scalarization functions our multi-policy takes advantage of its off-policy nature to successfully optimize several policies concurrently and efficiently provide a set of policies each one optimal for a given scalarization function.

1 Introduction

Many complex real-world sequential decision making problems such as electric plant control and autonomous driving involve compromising between several possibly conflicting objectives. To solve such problems, algorithms from the Multi-Objective Reinforcement Learning (MORL) sphere can be used. On the one hand, when no information on the preferences over the objectives is available, multi-policy algorithms can be used to learn a set of policies that ensure non-dominated returns. On the other hand, when such preferences are available in the form of a scalarization function transforming the vector returns to scalar ones, single-policy algorithms can learn a policy maximizing the scalarized returns. A majority of single-policy algorithms assume the scalarization function to be a weighted sum. Such a simplification allows the use of state-of-the-art algorithms from mono-objective reinforcement learning to solve such problems, however, it is often too restrictive and unrealistic. Some single-policy algorithms address this expressivity issue through the use of more complex, non-linear scalarization functions. These algorithms can be categorized according to the criteria they optimize: some use the Scalarized Expected Return (SER) criterion [17] aiming to identify a policy that maximizes a score defined as the aggregation of the obtained average return vectors, while others [15, 7, 16] choose to

optimize the Expected Scalarized Return (ESR), learning the policy that maximizes the average of the aggregated returns.

The Expected Scalarized Return is concerned with finding the policy that maximizes the expected scalarized vectorial return instead of the Scalarized Expected Return. As a result, ESR, unlike SER, penalizes policies that prioritize different objectives inconsistently across executions and ensures more balanced performance across the objectives over a single execution. The distinction between ESR and SER becomes crucial when dealing with non-linear preferences such as those commonly exhibited by humans. Under such preferences, ESR is particularly well-suited for scenarios where a policy is executed only once due to the nature of the application. For example, rescue rovers or exploration robots typically operate in a single environment and must perform effectively during that one deployment. In other cases, the cost of implementing a policy may be prohibitively high, making repeated execution impractical, for instance, in government budget allocations, where funding becomes non-recoverable once implemented. It is also the desirable way to solve scenarios where the implementation of a policy is irreversible such as medical treatment or the choice of response to the current global environmental crisis, where greenhouse gas emissions have to be reduced while still ensuring that the global economy stays stable. It is clear that no matter what global policy is chosen, it will have lasting impacts on humanity. Under such circumstances, a single execution of the policy determines its value. Due to the critical nature of these tasks, it is important to have efficient algorithms capable of finding the optimal policy under ESR.

Contributions This paper addresses two issues of the MORL literature. First, we propose a value-based algorithm that extends the categorical deep-Q-networks algorithm to solve multi-objective sequential decision making problems. We illustrate through examples that the return distribution learned by our algorithm for each state-action pair converges to the true distribution of the returns. Given a scalarization function, value-based distributional reinforcement learning is then used to find optimal policies under ESR. We also show that the proposed algorithm can leverage existing research in single-objective value-based reinforcement learning to improve its performances. Our algorithm is then extended to leverage off-policy experience to provide a first solution to multi-policy learning under ESR. Comparisons of our algorithm against existing state-of-the-art algorithms show that ours learns better policies more efficiently.

* Corresponding Author. Email: fares.chouaki@lip6.fr

2 Background

This section introduces the background required to understand our approach. First, the Multi-Objective Markov Decision Process framework used to model multi-objective sequential decision-making problems is detailed. Then, optimization criteria in the context of Multi-Objective RL are presented. Finally, foundations of univariate categorical reinforcement learning are presented.

2.1 Multi-Objective Markov Decision Processes (MO-MDPs)

Multi-objective sequential decision-making problems can be modeled using Multi-Objective Markov Decision Processes (MO-MDPs) [11], which generalize the classical Markov Decision Process framework. An MO-MDP is defined as a tuple $\langle d, S, A, T, \mu, R, \gamma \rangle$, where d denotes the number of objectives in the decision problem, S is the state space, and A is the action space. The transition model $T : S \times A \rightarrow \mathcal{P}(S)$ maps each state-action pair (s, a) to a probability distribution over the next states, specifying the likelihood of transitioning to each state $s' \in S$. The initial state distribution is given by $\mu : \mathcal{P}(S)$, which represents the probability distribution over the starting states. The reward function $R : S \times A \rightarrow \mathbb{R}^d$ assigns a d -dimensional real-valued reward vector to each state-action pair, capturing the multi-objective nature of the problem. Finally, $\gamma \in [0, 1)^d$ is a vector of discount factors, one for each objective.

This model can be extended to model partial observability [22] and the presence of multiple agents in the environment [20].

2.2 Optimization criteria in Multi-Objective Reinforcement Learning

Multi-objective RL algorithms try to either learn a front of policies that approximates the Pareto front of the returns, or the policy that maximizes a given aggregation function of the objectives. When provided with such a function, single-policy multi-objective RL algorithms learn the policy maximizing the aggregated return. At first glance, such a procedure is no different from what single-objective RL algorithms do. However, when the given scalarization function is non-linear, two optimization criteria can be distinguished.

- **Scalarized Expected Return (SER):** The value of a policy is obtained by applying the scalarization function over the expected return vector it obtains, thus, given a scalarization function $u : \mathbb{R}^d \rightarrow \mathbb{R}$, the value V_u^π of a policy π is given by: $V_u^\pi = u(\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi, s_0 \sim \mu])$.
- **Expected Scalarized Return (ESR):** The value of a policy is defined as the expected value of the scalarized returns it obtains, therefore given a scalarization function $u : \mathbb{R}^d \rightarrow \mathbb{R}$ the value V_u^π of a policy π is given by: $V_u^\pi = \mathbb{E}[u(\sum_{t=0}^{\infty} \gamma^t r_t) \mid \pi, s_0 \sim \mu]$.

In practice, a policy maximizing the Scalarized Expected Return would be used in scenarios where the policy's value is derived over several executions. A policy maximizing the Expected Scalarized Return, in contrast, is more suited to scenarios where the policy's value is derived after a single execution or when the policy needs to satisfy certain properties at each execution.

2.3 Distributional reinforcement learning

Our approach leverages categorical RL to address multi-objective problems. To provide a comprehensive understanding of our method,

this section outlines the representation of the value function used by categorical RL algorithms, the mechanism used for action selection based on the distribution of returns, and the update procedure of the learned value function based on experiences collected from interactions with the environment.

How is the return distribution represented in categorical RL?

Given a behavior policy π , categorical RL aims to learn for each state-action pair (s, a) the distribution of future returns $Z^\pi(s, a)$ that an agent would get from performing action a in state s and then following π . This return distribution is represented by the agent using a categorical representation. In the univariate case a categorical representation is parametrized by N , V_{MIN} , and V_{MAX} . A continuous probability distribution defined on an interval $[V_{MIN}, V_{MAX}]$ is represented with a categorical distribution using N evenly spaced support points $S = \{z_i = V_{MIN} + i\Delta z \mid i \in \{0, \dots, N-1\}\}$ where $\Delta z = \frac{V_{MAX} - V_{MIN}}{N-1}$ and each point $z_i \in S$ is associated with a probability mass p_i .

How is the value function used to take actions? To take an action using the learned return distributions, comparisons between return distributions are necessary. Indeed, at each state s , each action a has an associated return distribution. In order to choose an action, the Q -value of each action is computed as the expected value of each return distribution i.e., $Q(s, a) = \mathbb{E}[Z(s, a)] = \sum_{z_i \in S} z_i p_i$. The action with the maximum Q -value is taken.

How is the value function updated from experience? The categorical distribution uses a fixed support S to represent a probability distribution. This presents a challenge when these distributions are learned using the Bellman equation. This equation has been extended to the distributional case by [3], who introduced the following distributional Bellman equation.

$$Z(x, a) \stackrel{D}{=} R(x, a) + \gamma Z(X', A') \quad (1)$$

Hence, the return distribution of performing action a in state x is the mixture of the reward distribution associated with performing a in x and the return distribution of the next state-action (X', A') . When all three of these distributions are represented with a categorical representation with the same support, the contraction operation $\gamma Z(X', A')$ results in a new distribution with a new support $S' = \{\gamma z_i \mid z_i \in S\}$. To aggregate the two distributions $R(x, a)$ and $\gamma Z(X', A')$ it is necessary to project the latter into the same support as the other two distributions. A projection operator is introduced by [3] to achieve this and allows learning the return distribution of performing action a in state s .

3 Literature Review

This section introduces the existing research and algorithms in the field of Value-based RL, Multi-Objective RL, and distributional RL.

3.1 Value-based deep reinforcement learning

Value-based reinforcement learning algorithms estimate the value of state-action pairs to derive a policy. The first major success in Deep value-based RL was the Deep Q-Network (DQN) algorithm [13], which introduced two key innovations: a replay buffer to decorrelate training samples and a target network updated less frequently to stabilize learning. Double DQN [24] further improved stability by using one network to select actions and another to evaluate them, reducing overestimation bias. Dueling DQN [26] enhanced performance by separately estimating state values and advantages. Prioritized Experience Replay (PER) [21] improved sample efficiency by

sampling transitions based on temporal-difference error. Finally, soft Q-learning [9] extended value-based RL to continuous actions and stochastic policies, encouraging better exploration by maximizing policy entropy.

3.2 Distributional Reinforcement learning

Distributional Reinforcement Learning (RL) focuses on learning the full return distribution for each state-action pair, rather than just the expected return. This can be done using either categorical or quantile-based representations. C51-DQN [3], the first distributional RL algorithm, extends DQN by modeling returns as a categorical distribution and updates these return distributions using the projected Bellman operator. However, it requires prior knowledge of the return range, limiting its flexibility. QR-DQN [6] addresses this by representing returns using fixed quantiles, which improves accuracy and performance. Still, fixing the number of quantiles is suboptimal, as shown by IQN [5], which learns the full quantile function dynamically. More recently, [29] proposed MD3QN, allowing agents to learn the joint return distribution in multi-reward settings. However, their algorithm is focused on settings where a primitive reward function can be decomposed into a sum of several components.

3.3 Multi-Objective Reinforcement learning

We distinguish between two types of multi-objective reinforcement learning algorithms:

- Single-policy algorithms assume that the preferences over the objectives are given in the form of a scalarization function. The goal of such algorithms is then to learn a single policy that maximizes one of the optimization criteria presented in 2.2.
- Multi-policy algorithms assume no information about the preference over the objectives and aim to learn a portfolio of potentially optimal policies which can then be used to adapt a policy to the preferences of users.

When a scalarization function is not provided, two classes of algorithms can be distinguished: Inner-loop algorithms learn multiple policies in parallel. Instead, outer-loop algorithms assume a model of the scalarization function and then learn the optimal policy under a given parametrization of that model. This parametrization model is then varied for a predefined number of times to produce a set of policies that are optimal for each parametrization of the considered model.

Most of Multi-Objective RL research has been focused on learning one or several policies maximizing the Scalarized Expected Return of a policy [2, 28, 14, 25, 19, 1]. Recently, however, more effort has been put into learning policies that optimize the Expected Scalarized Return. The following section gives a review of such algorithms. **Single-policy algorithms for optimizing ESR:** The first algorithm designed to learn optimal policies for non-linear scalarization functions under ESR is the Expected Utility Policy Gradient (EUPG) algorithm [16]. It extends the standard policy gradient method by conditioning the policy on both the agent’s current observation and the accumulated returns, and by incorporating these accumulated returns into state-action value estimates. To address EUPG’s limitations in sample efficiency and stability, [15] proposed the Multi-Objective Categorical Actor-Critic (MOCAC), which uses a distributional critic to estimate the full return distribution, allowing bootstrapping mid-episode and supporting more stable, frequent updates. In parallel,

[12] introduced *Non-Linear Utility MCTS (NLU-MCTS)* and *Distributional MCTS*, which extend Monte Carlo Tree Search to compute ESR-optimal policies given a scalarization function. Finally, to promote fairness across objectives, [7] proposed *NSW-Q-learning*, an adaptation of Q-learning that optimizes policies under the Nash Social Welfare scalarization.

Multi-policy algorithms for optimizing ESR: The value-iteration algorithm was extended by [10] to learn a front of possibly optimal policies under ESR. [18] provides an extension of the algorithm presented in [25] that learns the set of undominated policies following a novel dominance criterion they introduce. Both approaches are inner-loop algorithms and are limited to problems with a discrete state and action spaces. To learn several policies under ESR, [4] propose a two step algorithm that first generates a set of N diverse non-linear scalarization functions and then transforms the MO-MDP into N scalarized MDPs using the generated scalarization functions. A policy-gradient algorithm is finally applied on each MDP to find the optimal policy.

3.4 Research Gap

The previous sections presented existing research in the fields of value-based RL, Multi-Objective RL and distributional RL. This review has shown that although the bulk of research in Multi-Objective RL is focused on optimizing SER, interest for optimizing ESR is growing. However, existing solutions still suffer from limitations highlighted below:

1. Policy-based algorithms like EUPG and MOCAC greedily exploit experiences and cannot generalize to other utility functions.
2. Single-policy value-based algorithms like NSW-QL can only handle scenarios with small state and action spaces.
3. Single-objective algorithms like the ones proposed in [13, 21, 3] cannot treat multi-objective problems and require adaptation.
4. Inner-loop multi-policy algorithms optimizing ESR can only be used on tasks with discrete state and action spaces.
5. Outer-loop multi-policy algorithms optimizing ESR, such as [4] require retraining the agent several times with different scalarization functions to achieve a decent approximation of all the possibly optimal policies under ESR.

In this paper, we first address limitations 1, 2, and 3 by introducing MO-CDQN, a value-based multi-objective distributional algorithm extending C51-DQN. MO-CDQN finds optimal policies under ESR when given a scalarization function. Then, to handle limitations 4 and 5, we propose a multi-policy algorithm leveraging the off-policy nature of MO-CDQN. This property of the algorithm enables the simultaneous optimization of multiple non-linear scalarization functions in tasks with large state spaces, while efficiently reusing experience across functions.

4 Multi-Objective Categorical Deep-Q-Networks (MO-CDQN)

We first present the inner-workings of the proposed solution and how our algorithm can be used to find the policy maximizing the Expected Scalarized Return when given a single scalarization function. The multi-policy extension of the proposed algorithm is presented in the second part of this section.

4.1 Optimizing a single scalarization function under ESR

We propose the Multi-Objective Categorical Deep-Q-Networks (MO-CDQN) algorithm, which extends the C51-DQN algorithm to the multi-objective case. To achieve this, we adapt both the action selection and update rules of C51-DQN, and modify the network architecture to predict multivariate categorical distributions. In this section, we first describe how the return distributions are represented by our algorithm. We then detail the action selection rule, followed by the update procedure.

Return distribution representation MO-CDQN learns to predict the full multivariate return distribution for each state–action pair in the environment. For each objective, we represent the return distribution using a categorical approximation. In the multi-objective setting, this results in a multivariate categorical representation. The support of this distribution is defined as the Cartesian product of the supports of each objective’s categorical distribution: $S = \times_{j \in \{0..d\}} S_j$ where $S_j = \{V_{MIN_j} + \Delta z_j \cdot i \mid 0 \leq i \leq N_j - 1\}$, V_{MIN_j} is the minimal value of the return the agent can achieve on the j -th objective and $\Delta z_j = \frac{V_{MAX_j} - V_{MIN_j}}{N_j - 1}$ is the size of the intervals of the categorical representation of the j -th objective. Note that each item of S is a tuple of size d , each tuple in S is called *atom* and we write $S = \{\mathbf{s}_i \mid 0 \leq i \leq \prod_j N_j\}$

In order to estimate these distributions our algorithm uses two neural networks: a policy network conditioned by a set of parameters θ and a target network conditioned by the parameters θ' . The policy network is used at decision time to estimate the return distributions $Z_\theta(s, a)$ obtained from performing action a in state s . The target network is only used when updating the policy, it allows to stabilize learning and it estimates the target return distribution $\hat{T}Z(s, a)$.

Action selection To optimize ESR, [16] showed that the policy must be conditioned on the return accumulated up to the current timestep. MO-CDQN therefore selects actions using both the current state s_t and the accumulated return \mathbf{G}_{acc} . For each action a , the algorithm first predicts the multivariate future return $Z(s, a)$ (Line 1 of Algorithm 2). The accumulated return \mathbf{g}_{acc} is then used to compute a translated support S' , defined as:

$$S' = \times_{j \in \{0..d\}} S'_j$$

$$\text{with } S'_j = \{\mathbf{G}_{acc_j} + V_{MIN_j} + \Delta z_j \cdot i \mid 0 \leq i \leq N_j - 1\}$$

(Line 2). The atoms of S' are scalarized (Line 3) and combined with their probabilities to compute Q-values (Line 4). The action is finally selected by sampling from the softmax distribution over these Q-values (Lines 5–7).

Update Rule Training is based on temporal-difference learning [23]. At each step, a minibatch of transitions $(s, \mathbf{G}_{acc}, a, \mathbf{r}, s')$ is sampled from the replay buffer. For each transition, the target return distribution is computed as: $\hat{T}Z(s, a) = r + \gamma \cdot Z_{\theta'}(s', a^*)$, where a^* is the greedy action (Line 5). This corresponds to shrinking the support of $Z_{\theta'}(s', a^*)$ by γ and translating it by r [3, 15]. The resulting distribution is projected back onto the original support S , yielding $\Phi \hat{T}Z(s, a)$ (Line 6). The cross-entropy between $Z_\theta(s, a)$ and $\Phi \hat{T}Z(s, a)$ is used as the loss. Gradients are accumulated across samples, and both the policy and target networks are updated accordingly (Lines 12–13, Algorithm 3).

4.2 Optimizing several scalarization functions with off-policy experience under ESR

We now present the multi-policy extension of MO-CDQN. This extension optimizes multiple scalarization functions simultaneously. This addresses the gap in multi-policy ESR optimization by reusing off-policy experience across learners targeting different scalarization functions. We assume a finite set of scalarization functions $U = \{u_1, \dots, u_M\}$ is known beforehand. Practically, this assumption is verified by two-steps algorithms where the first step consists of generating a set of diverse scalarization functions that are later used to provide a diverse set of solutions like [4]. More generally this assumption is often valid in decision-support scenarios [11], where designers can optimize various utility functions to better understand user preferences. It also applies when users aim to enforce a desired behavior through a family of parameterized scalarization functions but are unsure of the best parameter values. For example, a user seeking fairness across objectives might use the Generalized Gini Index (GGI), yet still struggle to select parameters that balance fairness and efficiency.

The intuition behind our second algorithm (called Distributed MO-CDQN) resides in the off-policy nature of MO-CDQN algorithm introduced in the previous section. Suppose we are given two scalarization functions $u, u' : \mathbb{R}^d \rightarrow \mathbb{R}$ and let $\pi_u^{t_i}$ and $\pi_{u'}^{t_i}$ be the policies of two agents optimizing u and u' respectively. It is possible to use experience collected under π_u to update $\pi_{u'}$ and vice versa. Given a finite set of possible scalarization functions that can be considered online by the decision-making agent, our approach aims at learning policies that can fit these different scalarization functions without requiring retraining. We thus extend our single-policy algorithm MO-CDQN to account for a set of M scalarization functions $U = \{u_1, \dots, u_m\}$ by introducing M artificial agents (referred to as learners) where each agent l_i is assigned the scalarization function u_i and searches for the policy $\pi_{u_i}^* = \operatorname{argmax}_\pi \mathbb{E}[u_i(\sum_{t=0}^{\infty} \gamma^t r^t) \mid \pi, s_0 \sim \mu]$. However, these learners are not trained separately. Instead, we propose to share environment experiences between the learners. More precisely, at each environment timestep t , all learners share the same observation s_t . Each learner then selects an action following its policy¹ $a_i^t \sim \pi_{u_i}(s_t)$. The algorithm then randomly selects one action from the set of actions proposed by the learners. The action is executed, resulting in a reward \mathbf{r}_t and a next state s_{t+1} . The observed experience tuple $(s_t, a_t, \mathbf{r}_t, s_{t+1})$ is added to the replay buffer of each learner after which the policy of each learner is updated using the update rule described in Algorithm 3.

Validity of the algorithm

We briefly discuss the theoretical validity of the multi-policy algorithm. Specifically, we address the following question: how can the policy of learner l_i in charge of optimizing the scalarization function u_i converge to its corresponding optimal policy while exploiting experience collected from other learners?

The answer to this question resides in the off-policy nature of the algorithm. Thus, an agent’s policy can be updated from experience that was collected from any other policy. A popular example of this is the use of ϵ -greedy policies to improve exploration in Q-learning [27]. With this in mind, when updating the policy of the i -th learner using experience tuple $(s_t, acc_r_t, a_t, \mathbf{r}, s'_t, t)$ we can distinguish two cases. If the algorithm picked the action proposed by l_i or by another learner with the same greedy action then $a_t = \pi_{u_i}$ is considered as a greedy action from l_i ’s perspective. When the algo-

¹ The action of learner l_i is sampled from the distribution $\pi_{u_i}(a \mid s_t)$ as described in Algorithm 2.

Algorithm 1 Distributed MO-CDQN

Require: Number of objectives d , Set of scalarization functions to optimize $U = \{u_1, \dots, u_M\}$, Training budget B ,

```
1: for  $i = 1$  to  $M$  do
2:   Create learner  $l_i$  with replay buffer  $D_i$ , value network  $Z_\theta^i$ 
   and target  $Z_{\theta'}^i$ .
3: end for
4: for  $t = 1$  to  $B$  do
5:    $\mathbf{acc\_r} \leftarrow \vec{0}$ 
6:    $c \leftarrow 0$ 
7:   Initialize environment and receive initial state  $s_0$ 
8:    $s_t \leftarrow s_0$ 
9:   while  $\neg$  done do
10:    proposed_actions  $\leftarrow \{\}$ 
11:    for  $i = 1$  to  $M$  do
12:       $a_i \leftarrow$  Distributional_Action_Selection( $s_t, \mathbf{acc\_r}, Z_\theta^i, u_i$ )
13:      Add  $a_i$  to proposed_actions
14:    end for
15:     $a_t \leftarrow$  RandomChoice(proposed_actions)
16:    Execute action  $a_t$  in the environment
17:    Observe reward  $\mathbf{r}_t$ , next state  $s_{t+1}$ , and done flag
18:    for  $i = 1$  to  $M$  do
19:      Add ( $s_t, \mathbf{acc\_r}, a_t, \mathbf{r}_t, s_{t+1}, c$ ) to  $D_i$ 
20:      Update  $Z_\theta^i$  using distributional Bellman update de-
described in Algorithm 3
21:    end for
22:     $\mathbf{acc\_r} \leftarrow \mathbf{acc\_r} + \gamma^c \mathbf{r}_t$ 
23:     $s_t \leftarrow s_{t+1}$ 
24:     $c \leftarrow c + 1$ 
25:  end while
26: end for
```

Algorithm 1 picks an action different from the one l_i proposed i.e. $a_t \neq \pi_{u_i}$ then the action a_t can be considered as an exploratory action. Thus, the action selection procedure of the algorithm does not hinder its convergence properties but also improves each learners' exploration. The algorithm also allows for concurrent policy updates, making it more efficient than applying a single policy algorithm sequentially on different scalarization functions. Algorithm 1 uses this property to reduce the number of environment interactions needed to learn multiple policies.

5 Experiments

To validate our approach, we first show that the proposed MO-CDQN algorithm learns an accurate approximation of the true distribution of returns by testing it on a small toy MO-MDP. Then, we move on to comparing the performance of the algorithm with state-of-the-art algorithms proposed in [16, 15] on the Fruit-Tree-Navigation environment of the MO-gymnasium suite of benchmarks. This section describes both the validation and testing environments. We then detail the experimental protocol that was followed to train and evaluate the approach. Finally, we present the results obtained on the validation task and compare the proposed algorithm to the considered baseline algorithms.

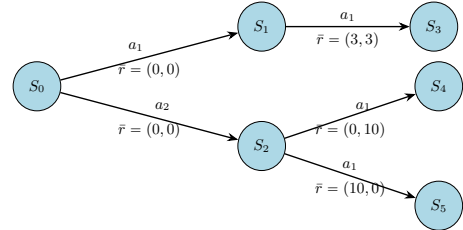


Figure 1: Toy MO-MDP used for validating the algorithm

5.1 Evaluation Scenarios

5.1.1 Validation scenario

To validate our algorithm and confirm that the return distribution it converges to is the true distribution of returns, we train our algorithm to solve a toy MO-MDP illustrated in Figure 1. In this problem, a single agent has to solve a bi-objective stochastic problem. Each action leads to a vector of rewards: one reward per objective; hence, in this example a couple of values. The agent starts at state s_0 and has to choose between two actions a_1 and a_2 leading to states s_1 and s_2 and receiving rewards $(2, 0)$ and $(1, 1)$ respectively. From state s_1 , the agent can perform action a_1 which leads to terminal state s_3 and gives a reward of $(3, 3)$. From state s_2 , the agent can perform action a_1 which is stochastic and leads the agent to states s_4 and s_5 with probability 0.5 each. When the agent arrives at state s_4 it receives a reward of $(10, 0)$ and when it arrives at state s_5 it receives a reward of $(0, 10)$.

5.1.2 Testing environments

Our algorithm is compared with other existing multi-objective RL algorithms that optimize ESR on the Fruit-Tree-Navigation, and the Deep-Sea-Treasure environments that are available in the MO-gymnasium suite of MORL benchmark environments [8]. In the following, a brief description of each environment is given:

Fruit-Tree-Navigation (FTN): This environment is a binary tree where at each step the agent needs to choose between going to the right or left sub-tree. The episode ends when the agent reaches a leaf node and receives the reward associated with that leaf node. The depth of the tree is fixed to 7 for all the experiments. The environment gives the agent a reward over six objectives; however, for scaling purposes the reward is limited to 3 objectives in the presented results.

Deep-Sea-Treasure (DST): This environment consists of a bi-objective gridworld environment where a submarine agent needs to explore the grid to find a treasure while minimizing fuel consumption. After each action where the agent did not find a treasure it receives a reward of $(0, -1)$ indicating that no treasure has been found (value 0 for the first objective) and the fuel consumption leads to a cost of -1 (second objective). The episode ends when the agent moves to a cell containing a treasure chest at which point it receives a reward of $(treasure_value, -1)$. The positions and values of treasures are given by the map described in Appendix B.

5.2 Baselines

Our MO-CDQN algorithm and its multi-policy extension are compared to state-of-the-art multi-objective RL algorithms that learn optimal policies for ESR. Namely, the EUPG [16] and MOCAC [15] algorithms are used as baselines. To point out the limitations of the approach proposed by [7], their algorithm is tested on the validation task.

5.3 Experimental Protocol

5.3.1 Validation

On the validation task, the agent is trained over 1250 episodes in the environment, and the average scalarized return of the policy is recorded after each episode. At the end of the training, the categorical representation of the return distribution the agent converged to is presented. The scalarization function used to collect experience is the product (i.e. $f(\vec{r}) = \prod_i r_i$).

5.3.2 Single policy algorithms

Fruit-Tree-Navigation: On this task, the goal of the agent is to learn a policy maximizing the product of the objectives. Our algorithm is trained over 500 000 environment steps, whereas baseline algorithms are allowed twice that budget. The policy of each agent is evaluated on the same task periodically every 5 000 steps in the environment. Since the policy learned by the agent is stochastic every evaluation step consists of running the agent’s policy on the tree-navigation task for 100 episodes. After each evaluation step, the average scalarized return achieved by the policies of each agent is reported. To alleviate the impact of random initialization of the policies, all agents are trained 11 times and the median as well as the first and third quartile scores are reported.

Deep-Sea-Treasure: To motivate the use of this environment, we use the same scenario as the one studied in [15]. Consider that the submarine crew is in debt d and is given a certain amount of time t to fully reimburse the debt. If the crew manages to find a treasure that allows them to fully reimburse the debt on time then the crew is set free and keeps the rest of the treasure. If the crew is delayed but finds a treasure sufficient to cover the debt, they are required to pay a late fee f and are penalized in proportion to the number of days overdue. The remainder of the treasure is retained by the crew. Finally, if the crew comes back with a treasure that does not fully cover the debt then they are imprisoned in which case they receive the maximal penalty e . This scenario is formalized by the following scalarization function:

$$u(r_0, r_1) = \begin{cases} r_0 - d & \text{if } r_0 \geq d \wedge r_1 \leq t \\ r_0 - d - ((r_1 - t)^2 + f) & \text{if } r_0 \geq d \wedge r_1 > t \\ e & \text{if } r_0 < d \end{cases}$$

To highlight the efficiency of our algorithm, it is only allowed a budget of 100 000 timesteps while the rest of the baselines are trained for 200 000 timesteps. The policy of each algorithm is periodically evaluated after 1 000 steps.

5.3.3 Multi-policy algorithm

To evaluate our multi-policy algorithm, the Fruit-Tree-Navigation task limited to 2 objectives is used with 5 hand-picked non-linear scalarization functions. The considered scalarization functions (u_1 to u_5) are the following. Note that r^\uparrow is the vector obtained from reordering the components of the reward vector increasingly.

$$\begin{aligned} u_1(r_0, r_1) &= \max(r_0, r_1) & u_3(r_0, r_1) &= r_0 r_1 \\ u_2(r_0, r_1) &= \min(r_0, r_1) & u_4(r_0, r_1) &= w_0 r_0^\uparrow + w_1 r_1^\uparrow \end{aligned}$$

$$u_5(r_0, r_1) = -(w_0(i_0 - r_0) + w_1(i_1 - r_1))^2$$

The performance of our multi-policy algorithm is compared with the that obtained by the single-policy variant of MO-CDQN, EUPG

and MOCAC. The on-policy algorithms are allowed a budget of 1 000 000 environment timesteps while the single policy MO-CDQN algorithm is given 500 000 environment timesteps to optimize each scalarization function. The multi-policy algorithm is provided the same budget to find the optimal policy for all the scalarization functions at once. Each experiment is repeated 5 times. We then compare the scores obtained by the best policies found by each algorithm.

5.4 Experimental results

This section presents the experimental results that were obtained by our algorithm and the considered baselines.

Validation We present the results obtained by our algorithm on the validation task. Figure 2 shows the categorical representations of the return distributions the algorithm converges to for each action in state s_0 . Each square is identified with the coordinates of its bottom left point (x_{min}, y_{min}) and its upper right point (x_{max}, y_{max}) in the figure. A square represents the probability of getting a vector return $r = (r_0, r_1)$ such that $r_0 \in [y_{min}, y_{max}[\wedge r_1 \in [x_{min}, x_{max}[$ while its color indicates the intensity of the probability. For example, the yellow square at the top-left side of Figure 2b indicates that the probability of getting a vector return of $(1, 11)$ is ≈ 0.5 . We can see that for both actions the predicted return distribution is accurate since the agent predicts that it will receive a return of $(3, 3)$ with probability ≈ 1 when taking action a_0 and predicts that it will receive returns $(11, 1)$ and $(1, 11)$ with equal probability ≈ 0.5 .

Figure 3a shows the results of training our single-policy MO-DQN algorithm, MOCAC, EUPG and NSW-QL on the toy MO-MDP used for validation with the same scalarization described in 5.3.1. We can see that our algorithm and both policy-based algorithms are able to solve the task whereas NSW-Q-learning is unable to. This proves the novelty of our approach and shows that representing the entire multivariate distribution is necessary to design a value-based MORL algorithm for ESR optimization.

Comparison to baselines We now turn our focus to showing how our algorithm can yield better results than existing state-of-the-art algorithms optimizing ESR. Figures 3c and 3b show the evolution of the median discounted scalarized return obtained by each agent. The shaded regions represent the fourth and sixth deciles of the returns obtained by the agents. The figures clearly illustrate the superiority of our algorithm compared to the baselines. For instance, from figure 3b, it can be observed that our algorithm requires less timesteps to converge to a better policy than the one found by the baselines. Figure 3c shows how our algorithm can scale to a higher number of objectives, however, it also shows that the considered categorical algorithms (ours and MOCAC) can only compete with EUPG when the number of objectives remains small.

Results of the multi-policy algorithm Table 1 shows the best expected scalarized return obtained by the policy computed by each algorithm. From the table, we can see that the multi-policy variant of our algorithm is not only competitive with the rest of the baselines while requiring less environment steps, it also manages to outperform them on three out of five of the considered scalarization functions without requiring re-training. We also observe that for u_2 and u_4 the multi-policy variant of MO-CDQN obtains better results than the single-policy counterpart.

6 Discussion

In this section, we discuss the performance of our algorithm and the limitations that our approach still suffers from.

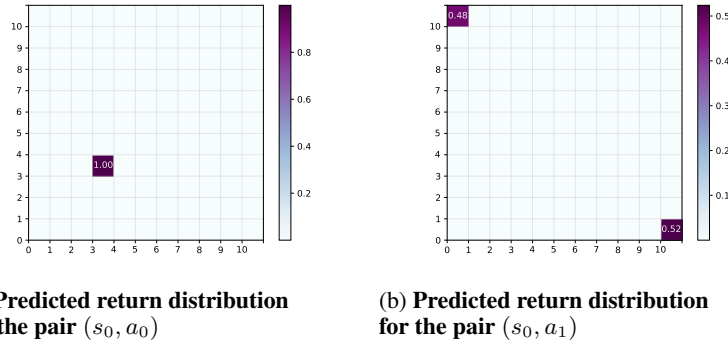


Figure 2: Predicted multivariate return distributions for different actions in state s_0 . Y axis shows returns on the first objective and X axis returns on the second objective

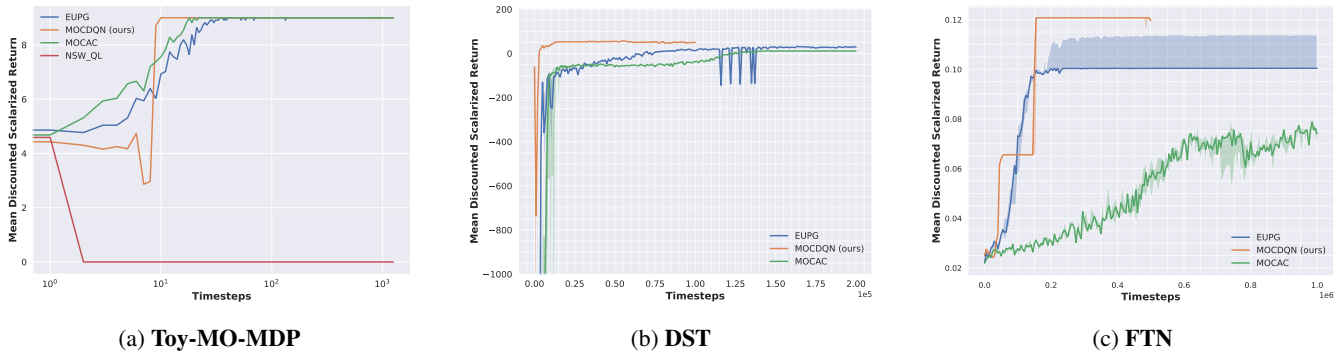


Figure 3: Comparison of the performances of the proposed algorithms to the baselines on the validation and evaluation tasks

	MO-CDQN	MOCAC	EUPG	Distributed MO-CDQN
u_1	0.8548	0.8852	0.8852	0.8521
u_2	0.4785	0.2849	0.4708	0.4805
u_3	0.3604	0.1348	0.2963	0.3604
u_4	0.4861	0.4808	0.4817	0.4890
u_5	-4.3306	-4.3781	-4.3524	-4.5129

Table 1: Performance of each algorithm on each scalarization functions

Sample efficiency Our results demonstrate that our algorithm achieves superior sample efficiency compared to state-of-the-art policy-based methods such as EUPG and MOCAC. As shown in Figure 3c, our approach converges to better policies than the ones learned by MOCAC and EUPG while requiring fewer timesteps. Moreover, our multi-policy algorithm illustrates how experience can be reused to train policies optimizing different scalarization functions, further improving the sample-efficiency of our approach. Both algorithms highlight the advantages of our method, which relies on a categorical representation of returns instead of directly optimizing policies in a high-dimensional parameter space. By leveraging value-based methods, our algorithm can more efficiently approximate the Expected Scalarized Return, leading to faster convergence.

Scaling The categorical return representation can become computationally expensive as the number of objectives increases, since the size of the support of the representation increases exponentially with the number of objectives. Another limitation of our algorithm is its inability to effectively handle continuous action spaces. Since our approach relies on a discrete set of actions, applying it to continuous control problems would require discretization, which may lead to suboptimal performance due to the loss of fine-grained control. In contrast, policy-based methods like EUPG and MOCAC inherently support continuous actions through parameterized policies.

Choice of scalarization in the multi-policy context One flaw of the multi-policy algorithm is that it assumes that the set of possible scalarization functions is known. This assumption is nonetheless addressed within the decision support scenario presented in [11]. To alleviate this problem, in the future our algorithm, could be extended to a two step approach, like [4], where the first step consists of generating a set of diverse non-linear scalarization functions that are then fed to the multi-policy algorithm to optimize each function to generate a diverse front of possibly optimal policies under ESR.

7 Conclusion

This paper tackles multi-objective sequential decision-making problems where the goal is to optimize non-linear scalarization functions under ESR. A novel value-based algorithm based on learning a categorical representation of the multivariate return distribution is introduced. We validate our approach by empirically showing that the return distribution to which our algorithm converges provides an accurate estimate of the ground truth. Our algorithm is compared to other existing solutions on multi-objective sequential decision-making tasks and the results prove that the proposed solution can learn better policies faster. The algorithm is then extended to propose a first value-based multi-policy algorithm able to optimize several non-linear scalarization functions under ESR. The contributed algorithms still suffer from scaling issues that will be addressed in future work. We also plan on investigating better action-selection strategies for the multi-policy algorithm and exploring how to automatically select a set of scalarization functions that achieve the best coverage of the Pareto front.

References

- [1] M. Agarwal, V. Aggarwal, and T. Lan. Multi-objective reinforcement learning with non-linear scalarization. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '22, page 9–17, Richland, SC, 2022. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450392136.
- [2] T. Basaklar, S. Gumussoy, and U. Y. Ogras. Pd-morl: Preference-driven multi-objective reinforcement learning algorithm, 2023. URL <https://arxiv.org/abs/2208.07914>.
- [3] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 449–458. JMLR.org, 2017.
- [4] X.-Q. Cai, P. Zhang, L. Zhao, J. Bian, M. Sugiyama, and A. J. Llorens. Distributional pareto-optimal multi-objective reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.
- [5] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit quantile networks for distributional reinforcement learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1096–1105. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/dabney18a.html>.
- [6] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.
- [7] Z. Fan, N. Peng, M. Tian, and B. Fain. Welfare and fairness in multi-objective reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, page 1991–1999, Richland, SC, 2023. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450394321.
- [8] F. Felten, L. N. Alegre, A. Nowé, A. L. C. Bazzan, E. G. Talbi, G. Danoy, and B. C. da. Silva. A toolkit for reliable benchmarking and research in multi-objective reinforcement learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.
- [9] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1352–1361. JMLR.org, 2017.
- [10] C. F. Hayes, D. M. Roijers, E. Howley, and P. Mannion. Decision-theoretic planning for the expected scalarised returns. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '22, page 1621–1623, Richland, SC, 2022. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450392136.
- [11] C. F. Hayes, 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. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1), Apr. 2022. ISSN 1573-7454. doi: 10.1007/s10458-022-09552-y. URL <http://dx.doi.org/10.1007/s10458-022-09552-y>.
- [12] C. F. Hayes, M. Reymond, D. M. Roijers, E. Howley, and P. Mannion. Monte carlo tree search algorithms for risk-aware and multi-objective reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 37(2), Apr. 2023. ISSN 1387-2532. doi: 10.1007/s10458-022-09596-0. URL <https://doi.org/10.1007/s10458-022-09596-0>.
- [13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. URL <https://api.semanticscholar.org/CorpusID:205242740>.
- [14] M. Reymond, E. Bargiacchi, and A. Nowé. Pareto conditioned networks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '22, page 1110–1118, Richland, SC, 2022. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450392136.
- [15] M. Reymond, C. Hayes, D. Steckelmacher, D. Roijers, and A. Nowé. Actor-critic multi-objective reinforcement learning for non-linear utility functions. *Autonomous Agents and Multi-Agent Systems*, 37, 04 2023. doi: 10.1007/s10458-023-09604-x.
- [16] D. Roijers, D. Steckelmacher, and A. Nowé. Multi-objective reinforcement learning for the expected utility of the return. July 2020. 2018 Adaptive Learning Agents, ALA 2018 - Co-located Workshop at the Federated AI Meeting, FAIM 2018 ; Conference date: 14-07-2018 Through 15-07-2018.
- [17] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, Oct. 2013. ISSN 1076-9757. doi: 10.1613/jair.3987. URL <http://dx.doi.org/10.1613/jair.3987>.
- [18] W. Röpke, C. F. Hayes, P. Mannion, E. Howley, A. Nowé, and D. M. Roijers. Distributional multi-objective decision making. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023. ISBN 978-1-956792-03-4. doi: 10.24963/ijcai.2023/634. URL <https://doi.org/10.24963/ijcai.2023/634>.
- [19] M. Ruiz-Montiel, L. Mandow, and J.-L. P. de-la Cruz. A temporal difference method for multi-objective reinforcement learning. *Neurocomputing*, 263:15–25, 2017. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2016.10.100>. URL <https://www.sciencedirect.com/science/article/pii/S0925231217310998>. Multiobjective Reinforcement Learning: Theory and Applications.
- [20] R. Rădulescu, P. Mannion, D. M. Roijers, and A. Nowé. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems*, 34(1), Dec. 2019. ISSN 1573-7454. doi: 10.1007/s10458-019-09433-x. URL <http://dx.doi.org/10.1007/s10458-019-09433-x>.
- [21] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015. URL <https://api.semanticscholar.org/CorpusID:13022595>.
- [22] H. Soh and Y. Demiris. Evolving policies for multi-reward partially observable markov decision processes (mr-pomdps). In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, page 713–720, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450305570. doi: 10.1145/2001576.2001674. URL <https://doi.org/10.1145/2001576.2001674>.
- [23] G. Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, Mar. 1995. ISSN 0001-0782. doi: 10.1145/203330.203343. URL <https://doi.org/10.1145/203330.203343>.
- [24] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016. doi: 10.1609/aaai.v30i1.10295. URL <https://ojs.aaai.org/index.php/AAAI/article/view/10295>.
- [25] K. Van Moffaert and A. Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *J. Mach. Learn. Res.*, 15(1): 3483–3512, Jan. 2014. ISSN 1532-4435.
- [26] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1995–2003. JMLR.org, 2016.
- [27] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL <https://doi.org/10.1007/BF00992698>.
- [28] R. Yang, X. Sun, and K. Narasimhan. *A generalized algorithm for multi-objective reinforcement learning and policy adaptation*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [29] P. Zhang, X. Chen, L. Zhao, W. Xiong, T. Qin, and T.-Y. Liu. Distributional reinforcement learning for multi-dimensional reward functions. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

A Algorithms

This appendix provides detailed pseudocode for the core components of the proposed distributional multi-objective reinforcement learning approach. These algorithms complement the descriptions provided in the main body of the paper and are included here for reproducibility and clarity.

Action selection This procedure describes how to select an action given a state and an accumulated return using a scalarized distributional representation of multi-objective returns.

Algorithm 2 Distributional_Action_Selection

Require: State s , accumulated return \mathbf{r}_{acc} , value network Z_θ , scalarization function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, discount factor γ , time of the decision t .

- 1: Retrieve categorical distribution $Z_\theta(s, a) = \{(p_i, \mathbf{s}_i)\}_{i=0}^{\prod_j (N_j + 1)}$ for each action a in state s using the value network.
- 2: Compute the support of the new distribution:

$$S' = \bigtimes_{j \in \{0..d\}} \{\mathbf{r}_{\text{acc}_j} + \gamma^{t+1}(V_{\text{MIN}_j} + i\Delta z_j) | 0 \leq i \leq N_j\}$$

- 3: Apply scalarization to the atoms of S' :

$$\text{scalarized}_{S'} = \{f(\mathbf{s}), \quad \forall \mathbf{s} \in S'\}$$

- 4: Compute expected scalarized value for each action:

$$Q(s_t, a) = \sum_{\text{atom} \in \text{scalarized}_{S'}} p_i * \text{atom}$$

- 5: Compute action probabilities using softmax:

$$\pi(a|s) = \frac{\exp(Q(s, a))}{\sum_b \exp(Q(s, b))}$$

- 6: Sample action $a^* \sim \pi(a|s)$.
 - 7: Return selected action a^* .
-

Update This algorithm outlines the update step of the distributional value-based learning algorithm. It computes the projected target distribution for each transition in the sampled batch and minimizes the cross-entropy loss between this target and the current estimated distribution.

B Testing environments description

This section provides a detailed description of the environments used to evaluate the performance of the proposed algorithms. These environments are taken from the MO-Gymnasium benchmark suite [8] and are designed to assess the ability of multi-objective reinforcement learning algorithms to learn under complex reward trade-offs.

Fruit-Tree-Navigation (FTN)

The Fruit-Tree-Navigation environment is structured as a binary tree of depth d , where each internal node corresponds to a decision point for the agent. At each step, the agent chooses whether to move left or right. Once a leaf node is reached, the episode ends and the agent receives a reward vector associated with the chosen leaf.

In our experiments, the tree has a fixed depth of 7, resulting in $2^7 = 128$ possible terminal nodes. Each leaf node is associated with a d -dimensional reward vector where d is the number of objectives.

Algorithm 3 Distributional_Update

Require: Batch size b , Replay buffer \mathcal{D} , value network Z_θ , target network $Z_{\theta'}$, learning rate α , discount factor γ , update rate τ .

- 1: Sample *batch* of size b from \mathcal{D}
- 2: **for** $j = 1$ to b **do**
- 3: $(s, G_{\text{acc}}, a, r, s', t) \leftarrow \text{batch}[j]$
- 4: Select action $a^* = \arg \max_a \mathbb{E}_{z \sim Z(s', a)} f(r + \gamma z)$
- 5: Compute target distribution using $Z_{\theta'}(s', a^*)$, γ and r .

$$\hat{T}Z(s, a) = r + \gamma \cdot Z_{\theta'}(s', a^*)$$

- 6: Project $\hat{T}Z(s, a)$ into S resulting in $\Phi \hat{T}Z(s, a)$.
 - 7: Retrieve current action distribution from $Z_\theta(s, a)$
 - 8: Compute cross-entropy loss between the projected target distribution $\Phi \hat{T}Z(s, a)$ and the current distribution $Z_\theta(s, a)$
 - 9: Compute total loss: $L_j = L_{\text{CE}}$
 - 10: Accumulate gradient update $\Delta \leftarrow \Delta + \nabla_\theta L_j$
 - 11: **end for**
 - 12: Update network parameters $\theta \leftarrow \theta - \eta \Delta$,
 - 13: Soft update target network: $\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$
-

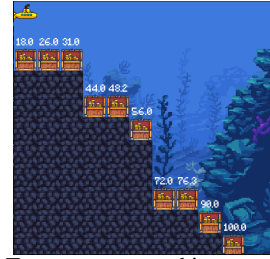


Figure 4: Treasure map used in our experiments

The reward vectors are assigned such that different paths through the tree lead to different trade-offs between the objectives, requiring the agent to make strategic decisions early in the episode to reach favorable leaves.

While the original FTN environment supports up to 6 objectives, we limit ourselves to the first 3 objectives for the single-policy experiments and to first 2 objectives for the multi-policy setting in order to ensure scalability of the categorical algorithms.

Deep-Sea-Treasure (DST)

Deep-Sea-Treasure is a classic multi-objective grid world problem that models the trade-off between finding valuable treasures and minimizing resource consumption. The agent controls a submarine and navigates a two-dimensional grid to reach a cell containing a treasure.

Each cell in the grid either contains no treasure (yielding a reward of $(0, -1)$) or a treasure of some positive value v (yielding a reward of $(v, -1)$). The second component represents a fixed cost per action due to fuel consumption. The episode terminates when the agent reaches a cell containing a treasure.

The treasure map used in our experiments is the standard one proposed in [15], where several treasures of varying values are placed at increasing depths. This setup introduces a trade-off between quickly reaching low-value treasures and spending more fuel to reach higher-value ones. The treasure locations and their corresponding values are shown in Figure 4.

The values of the parameters of the scalarization function used for this environment are the following: $d = 20$; $t = 19$; $e = -100$