Actor-Critic Multi-Objective Reinforcement Learning for Non-Linear Utility Functions

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

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

Diederik M. Roijers
Vrije Universiteit Brussel &
HU Utrecht Univ. of Applied Sciences
Brussels (BE) & Utrecht (NL)
diederik.roijers@vub.be

Denis Steckelmacher
Vrije Universiteit Brussel
Brussels, Belgium
dsteckel@ai.vub.ac.be

Ann Nowé
Vrije Universiteit Brussel
Brussels, Belgium
ann.nowe@vub.be

ABSTRACT

We propose a novel multi-objective reinforcement learning algorithm that successfully learns the optimal policy even for nonlinear utility functions, a class of utility functions that pose a challenge for SOTA approaches, both in term of learning efficiency as well as the solution concept. A key insight is that, by proposing a critic that learns a multi-variate distribution over the returns, which is then combined with accumulated rewards, we can directly optimize on the utility function, even if it is non-linear. This allows us to vastly increase the range of problems that can be solved compared to those which can be handled by single-objective methods or multi-objective methods requiring linear utility functions, yet avoiding the need to learn the full Pareto front. We demonstrate our method on multiple multi-objective benchmarks, and show that it learns effectively where baseline approaches fail.

KEYWORDS

Reinforcement Learning, Multi-Objective Reinforcement Learning, Non-Linear Utility Functions, Expected Scalarized Return

ACM Reference Format:
Mathieu Reymond, Conor Hayes, Diederik M. Roijers, Denis Steckelmacher, and Ann Nowé. 2021. Actor-Critic Multi-Objective Reinforcement Learning for Non-Linear Utility Functions. In *Proc.of the 1st Workshop on Multi-Objective Decision Making (MODeM), The Internet, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

Multi-objective decision problems are ubiquitous, as many real-world problems require trading off different goals. For example, we may want to maximise the power output of a hydroelectric power plant while minimising the risk of flooding further downstream [6], or we may want to maximise the radiation used to kill cancer cells, while minimising the damage to healthy surrounding tissues in radiotherapy [8]. In multi-objective reinforcement learning (MORL), agents learn to balance different objectives by interacting with the environment.

Of course, depending on the decision maker, some trade-offs are preferred above others. The utility function – the function that converts a multi-objective return to a single, scalar preference score – can help the MORL process find the policy that leads to the preferred outcome. However, when user preferences are non-linear – as human preferences often are – the utility function cannot be used to straightforwardly reduce a multi-objective problem to a single-objective one. Dedicated multi-objective models and methods are required, even if the utility function is known a priori [16]. After all, reinforcement learning (RL) relies heavily on the assumption that optimizing the (discounted) sum of individual rewards will also optimize the overall problem. But with non-linear utility functions, the sum of individual utilities is *not* equal to the utility over the return. This makes our setting a hard-to-solve problem, as the vast majority of RL algorithms cannot be straightforwardly applied: they use the Bellman equation, which takes advantage of this sum-of-rewards assumption. Moreover, most work on MORL applies the utility function on the expected returns, not directly on the discounted sum of rewards (we explain in details the differences between these two optimality criterions in Section 2.2). We focus on the latter, which implies that each policy evaluation is relevant to the user in terms of utility. This also means that the utility function cannot be applied on Q-values, as they estimate expected returns. Perhaps this is why, even though identified as an open challenge in the seminal survey on MORL [16], this setting is still severely understudied in the MORL literature, most body of work instead bypassing this issue by assuming that the utility function is a weighted sum over the objectives.

In this paper, we overcome this challenge by proposing a novel algorithm that explicitly keeps track of the different objectives and correctly applies the non-linear utility function on estimates of the overall multi-objective return. Our key insight is that, if the agent learns a multi-variate distribution over the future returns, we can use this distribution to bootstrap, enabling us to exploit the Bellman equation in MORL. Using this insight, we propose an actor-critic method we call Multi-Objective Categorical Actor-Critic (MOCAC), that uses such bootstrapping in its critic. We implement and demonstrate our methods on multiple multi-objective

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Proc.of the 1st Workshop on Multi-Objective Decision Making (MODeM), Hayes, Mannion, Vamplew (eds.), May 3–7, 2021, The Internet. © 2021 Copyright held by the owner/author(s).

benchmarks, and show that they learn effectively where single-objective baselines fail. To the best of our knowledge, this is the first MORL algorithm that exploits a priori known non-linear utility functions to optimize the expected utility.

2 BACKGROUND

2.1 Multi-Objective Reinforcement Learning

In *reinforcement learning (RL)*, an agent learns to optimise its behaviour by interacting with the environment. In this paper, we deal with decision problems with multiple objectives, and model this as a *multi-objective Markov decision process (MOMDP)*. A MOMDP is a tuple, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \vec{\mathcal{R}})$, where $\mathcal{S}, \mathcal{A}$ are the respective state and action spaces, $\mathcal{T}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a probabilistic transition function, γ is a discount factor determining the importance of future rewards and $\vec{\mathcal{R}}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^n$ is an n-dimensional vector-valued immediate reward function. In single-objective RL, $n = 1$ while in *multi-objective reinforcement learning (MORL)*, $n > 1$.

When $n = 1$, the goal is to find a policy π that maximizes the expected sum of discounted rewards, i.e. $\pi^* = \arg\max_\pi E[\sum_{t=0}^{h} \gamma^t r_t | \pi, s_0]$. However, when $n > 1$, these sums can lead to returns for which, without any additional information, there is no clear winner (e.g., we cannot decide which return is optimal between $(0, 10)$, $(5, 5)$ or $(10, 0)$). We thus assume the existence of a utility function u which, given a vectorial return, outputs a preference score (the utility of the decision maker) that can be used to rank the said vectorial returns. For example, given $u = \min(r_0, r_1)$, return $(5, 5)$ is the best choice between $(0, 10)$, $(5, 5)$ and $(10, 0)$ as they have utilities of $0, 5, 0$ respectively. MORL distincts two criterions to optimize a MOMDP under: the expected scalarized return (ESR) and the scalarized expected return (SER). Depending on the criterion, the resulting optimal policy can be vastly different.

2.2 ESR versus SER

In most MORL research the agent aims to compute a policy π that optimises the utility of the expected return, i.e.,

$$\pi^* = \arg\max_\pi u\left(E\left[\sum_{t=0}^{h} \gamma^t \vec{r}_t | \pi, s_0\right]\right) \quad (1)$$

where the utility function, u, can be any monotonically increasing function in all objectives. This is known as the scalarized expected return (SER) optimisation criterion. The particularity of this criterion is that the utility is only optimal on the average of multiple executions of the learned policy. While this can be useful for some problems, most of the time the utility should be optimal for a single policy execution (the utility of the policy thus depends on a single roll-out).

Therefore the agent should maximise the expected utility over single policy executions, i.e.,

$$\pi^* = \arg\max_\pi E\left[u\left(\sum_{t=0}^{h} \gamma^t \vec{r}_t\right) | \pi, s_0\right] \quad (2)$$

This is the expected scalarized return (ESR) criterion. We illustrate the difference between those two solution concepts using a small example. Consider a single-step, 2-objective MOMDP with two possible actions a_0, a_1. Action a_0 always results in reward $\vec{r} = (3, 3)$, while action a_1 results with equal probability in either $\vec{r} = (0, 10)$ or $\vec{r} = (10, 0)$. Under ESR, given $u = \min(r_0, r_1)$, the optimal policy is to always take a_0 since it results in the highest utility $u = 3$ (compared to $u = 0$ for a_1). However, *in expectation*, i.e., over many policy runs, $E[\vec{r}|a_1] = (5, 5)$. Under the SER criterion, given the same utility function, the optimal policy is to always take a_1.

Most body of work in MORL focusses on linear utility functions, i.e., the utility is a weighted sum over the objectives. A nice property is that for this class of utility functions SER is equivalent to ESR [14]. Moreover, if the linear utility function is known, the MOMDP can be reduced to a single-objective MDP on which we can apply single-objective methods [16].

In contrast, dedicated methods are required for non-linear utility functions. However, under SER, we can still take advantage of classic value-based methods [22], but this is not possible under ESR. The lack of methods for ESR MORL algorithms was identified as an important open problem in the seminal survey on MOMDPs. In this paper, we address this problem.

3 MULTI-OBJECTIVE CATEGORICAL ACTOR-CRITIC

The actor-critic framework has been used to produce many state-of-the-art deep reinforcement learning algorithms. At its essence, it leverages two components. On the one hand, the actor learns a probability distribution over the actions in $\mathcal{A}$, conditioned on a given state. These probabilities are updated regularly, increasing the probabilities of actions that lead to high returns by using the negative log-likelihood loss [24].

On the other hand, the critic learns to estimate the future expected returns for a given state-action pair. This estimate is commonly known as the Q-value (and V-value if conditioned on the state only). Using a critic allows the actor to be updated at every timestep, by estimating the future returns using the critic instead of waiting for them to be played out. Additionally, the use of the Advantage metric $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$ introduced by the Advantage Actor-Critic algorithm [9] has been shown to improve the stability of the learning process by reducing variance. Using this metric, we can formalize the policy-update rule as follows:

$$\begin{aligned}\mathcal{L}(\pi) &= -\sum_{t=0}^{T} A(s_t, a_t) \log(\pi_\theta(a_t|s_t)) \\ &= -\sum_{t=0}^{T} (r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t)) \log(\pi_\theta(a_t|s_t))\end{aligned} \quad (3)$$

where s_t, a_t, r_t are respectively the state, action, reward at timestep t. The actor is represented by the policy π parametrized by θ, while the critic is represented by V parametrized by ψ. This allows the actor be updated at every timestep, improving for sample efficiency.

In this paper we generalize the core of the actor-critic framework (Equation 3) to the ESR, known utility setting. This allows us learn a policy that directly optimizes on the user utility, while using a critic to increase sample efficiency. To this end, two main challenges need to be overcome:

(1) the utility function u can only be applied on the total episodic return, while the action probability distribution for state s_t is updated according to the *future* returns.
(2) by using V in Equation 3, the policy is updated based on an estimate of the *expectation* over the returns. Using u to scalarize a vectorial V-value would amount to a scalarization over the expectation of the returns (SER), which is different from our setting. We showcase why this difference is crucial in Section 5, and how it can lead to drastically different policies.

3.1 Accrued Rewards

The utility function can only be applied on whole episodic returns. Thus, at any timestep t, it is essential to not only estimate the future rewards, but also the accrued rewards: the rewards accumulated from timestep 0 until timestep t.

Consider a 2-objective MOMDP where, at timestep $t > 0$, the agent has already accumulated a total reward $\vec{r} = (5, 0)$. The preferences of the decision maker are formalized as the non-linear utility function $u = \min(r_0, r_1)$. Let us say the agent has two possible actions, each leading it to a final state, thus ending the episode. The first action a_0 results in reward $\vec{r}_t = (2, 2)$, while the second action a_1 results in reward $\vec{r}_t = (0, 5)$. If only future rewards are considered, executing a_0 will result in $u = 2$, while a_1 will result in $u = 0$ (a_0 is optimal). However, if we take into account the accrued rewards, a_0 will result in a total episodic return of $(7, 2)$ ($u = 2$), while a_1 will result in $(5, 5)$ ($u = 5$). In this case, the action leading to the highest user utility is a_1. We thus propose to incorporate the accrued reward in order to make correct use of u. Given, at timestep t, the future discounted return $\vec{R}_t = \sum_{k=t}^{H} \gamma^{k-t} \vec{r}_k$, we define the accrued reward as:

$$\vec{R}_t^- = \sum_{k=0}^{t-1} \gamma^k \vec{r}_k \tag{4}$$

The episodic return is then simply the sum of the accrued rewards and $\vec{R}_t$:

$$\vec{R} = \vec{R}_t^- + \gamma^t \vec{R}_t \tag{5}$$

By incorporating the accrued reward into the actor-update equation (Equation 3), we can directly optimize the policy on the user utility. This would lead to the the following equation:

$$\begin{aligned}\mathcal{L}(\pi) = -\sum_{t}^{T} \big(u(\vec{R}_t^- + \gamma^t(\vec{r}_t + \gamma V_\psi(s_{t+1}))) - \\ u(\vec{R}_t^- + \gamma^t V_\psi(s_t))\big) \log(\pi_\theta(a_t|s_t))\end{aligned} \tag{6}$$

with the output of V_ψ an n-dimensional vector.

Note that by using a history of past rewards to update a policy conditioned on state s_t we break the Markov property. This, however, can be prevented by conditioning the state on the accrued rewards. We note that conditioning can be necessary to reach optimal utility in some environments, but for the environments in this paper it was not necessary in practice.

3.2 Distributional Critic

Since the goal of our setting is to learn a policy that leads to the highest user utility for every single evaluation, u needs to be applied on the episodic return, not on the expected return. In light of this, we will incorporate distributional reinforcement learning into our algorithm, by creating a critic that estimates a multi-variate distribution Z over return values, inspired by the work by [3].

In single-objective RL, the expected return from a state s_t, $V(s_t)$, can be decomposed into a distribution p_i over future returns:

$$V(s_t) = \sum_i z_i p_i(s_t), \tag{7}$$

where p_i represents the probability of having a return z_i from state s_t. Learning the whole distribution instead of just the expected return leads to more stable learning. To solve our multi-criteria setting, we build upon this idea, and extend it to the multivariate case, allowing us to learn the distribution over n-dimensional future returns. This is essential, as we need to sum the accrued return with future returns *before* applying the utility function and taking the expectation.

Bellemare et al. use a discrete distribution parametrised by $N \in \mathbb{N}$, as it is computationally friendly and highly representative. Its support is the set of atoms $\{z_i = V_{\text{MIN}} + i\Delta z : 0 \le i < N\}$, with $\Delta z := \frac{V_{\text{MAX}} - V_{\text{MIN}}}{N-1}$, where $V_{\text{MIN}}, V_{\text{MAX}} \in \mathbb{R}$ represent the smallest and largest return values of the distribution, respectively. Please note that V_{MIN} and V_{MAX} are bounds on the returns, not on the value, but we use the V-notation to remain consistent with [3]. We can see this distribution as a discrete set of N categories, where each category p_i represents the probability of ending with a return $R_t \in [z_i, z_{i+1}[$.

For our algorithm, we require a multivariate distribution where z_i represents a vectorial return. As each objective has a separate $V_{\text{MIN}}, V_{\text{MAX}}$, the set of atoms becomes :

$$\begin{aligned}\{\vec{z_{i\ldots k}} = (V_{\text{MIN}_1} + i\Delta z_1, \ldots, V_{\text{MIN}_n} + k\Delta z_n) : \\ 0 \le i < N, \ldots, 0 \le k < N\}.\end{aligned} \tag{8}$$

where we assume the same number of categories N for each objective-dimension, resulting in a discrete distribution parametrized by N^n. Regardless, $V(s_t)$ is computed in the same manner as the single-objective case (Equation 7).

Because the critic now represents a full distribution instead of an expected value, we overcome the second challenge of our setting: since $\vec{z_{i\ldots k}}$ is defined as a return (with its associated probability $p_{i\ldots k}$), it can be converted into a preference score using u under the ESR criterion. This results in the following equation:

$$u_t = \sum_j u(\vec{R}_t^- + \gamma^t \vec{z}_j) p_j(s_t), \tag{9}$$

where, for conciseness, the index j represents each combination of indexes $i \ldots k$.

Note that we include the accrued reward as defined in Section 3.1 to correctly compute the utility.

Using this equation, we propose a novel actor-critic algorithm for multi-objective optimization whose parametric policy optimizes the user utility directly by performing gradient descent on the following loss function:

$$\mathcal{L}(\pi) = -\sum_{t=0}^{T} A(s_t, a_t)\log(\pi_\theta(a_t|s_t))$$
$$= -\sum_{t=0}^{T}\Big(\sum_j u(\vec{R}_t^- + \gamma^t(r_t + \gamma \vec{z}_j))p_j(s_{t+1}) - \sum_j u(\vec{R}_t^- + \gamma^t \vec{z}_j)p_j(s_t)\Big)\log(\pi_\theta(a_t|s_t)). \quad (10)$$

As for the critic, we compute the distributional Bellman update $\hat{\mathcal{T}}z_j := \vec{r}_t + \gamma\vec{z}_j$ for each atom $\vec{z}_j$, for a given sample transition $(s_t, a_t, \vec{r}_t, s_{t+1})$. We then distribute its probability $p_j(s_{t+1})$ to the immediate neighbours of $\hat{\mathcal{T}}z_j$. Each of the components of the projected update is:

$$(\Phi\hat{\mathcal{T}}Z(s_t))_i = \sum_j \left[1 - \frac{|[\hat{\mathcal{T}}z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}} - \vec{z}_i|}{\Delta z}\right]_0^1 p_j(s_{t+1}), \quad (11)$$

with $[.]_b^a$ bounding the argument between $[a, b]$.

We use the cross-entropy term of the KL-divergence as the loss function for the critic:

$$D_{KL}(\Phi\hat{\mathcal{T}}Z(s)||Z(s)). \quad (12)$$

Thus, we propose Multi-Objective Categorical Actor-Critic (MOCAC), an algorithm that optimizes the utility under the ESR criterion and is able to take advantage of any kind of monotonically increasing utility function. To the best of our knowledge, it is the first reinforcement learning algorithm to cope with this setting. Moreover, we show in the experimental section that it is also stable and sample-efficient.

3.3 Policy Gradient as a Baseline

Since no other RL algorithms that cope with our setting currently exist, we propose to adapt the classic policy gradient algorithm, Reinforce [24], to multi-objective optimization. Reinforce uses the discounted return to update its policy using the negative log-likelihood loss. Since Reinforce is an actor-only method, the policy is updated at the end of every episode, once the full discounted return is known. This means that, by also keeping track of the accrued rewards (as explained in Section 3.1) the utility function can be used. Thus, as a baseline, we propose a multi-objective variant of Reinforce called MOReinforce in the experimental section, that optimizes the user utility under ESR using the following loss equation:

$$\mathcal{L}(\pi) = -\sum_{t=0}^{T} u(\vec{R}_t^- + \gamma^t \vec{R}_t)\log(\pi_\theta(a_t|s_t)) \quad (13)$$

4 ABLATIVE STUDY

To demonstrate the need of a distributional critic, we first perform an ablative study on a (new) simple environment we call Split. We compare our method with a modified version. This version takes into account accrued rewards, but uses a critic that outputs $V(s_t)$. Concretely this version – called MOAC in this experiment – uses Equation 6 to update its actor. MOAC's critic improves according

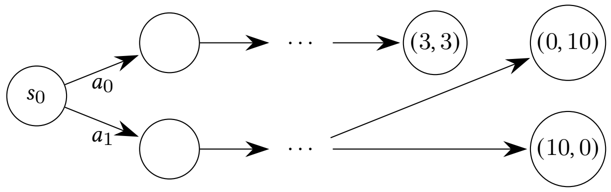

Figure 1: A visual representation of the Split environment. In our experiment, the length of each hallway is 10.

to a vectorial version of Temporal-Difference (each objective is updated independently):

$$V(s_t) \leftarrow V(s_t) + \alpha(\vec{r}_t + \gamma V(s_{t+1}) - V(s_t)) \quad (14)$$

where α is the learning rate. Finally, MO Reinforce is used as a baseline.

The Split environment, depicted in Figure 1, is defined as follows: in the start-state, the agent can choose between two hallways of equal length. The first one leads to a reward $(3, 3)$, while the second one leads to either a reward $(10, 0)$ or $(0, 10)$.

We use a synthetic utility function that multiplies both rewards together[1]:

$$u = r_0 r_1 \quad (15)$$

As such, the utilities for reaching the first and second hallway are 9 and 0, respectively.

Results are shown in Figure 2. Both MOCAC and MO Reinforce learn policies that lead to the optimal utility. MO Reinforce doesn't use a critic, and applies u directly on the episodic return. We thus expect it to reach optimality. MOAC however, learns to take the wrong hallway leading to a utility of 0. The reason behind this behaviour lies in the critic. It learns the expected values of each objective rather than a distribution over the returns. These expected values are $(3, 3)$ and $(5, 5)$ for the first and second hallway, respectively. Applying the utility on the resulting V-values leads to an incorrect estimate of $u = 25$ for the second path, which in turn leads to corrupted advantages – and thus a poor performance – for the actor gradient.

Finally, as we can see in Figure 2, even though MOCAC has many more parameters to learn than MO Reinforce, it reaches the optimal utility at an earlier stage, since the actor-critic algorithm makes an update at every timestep, while the baseline only updates its estimators at the end of each episode.

5 EXPERIMENTS

In order to test the effectiveness and sample-efficiency of MOCAC, we evaluate it on two different MOMDP benchmarks from the MORL literature. All experiments compare with our proposed baseline algorithm, MO Reinforce, as well as single-objective methods. All experiments are averaged over 5 runs. All hyperparameters and network architectures are reported in the Appendix.

We note that, while each experiment uses a utility function that matches a plausible real-life scenario, early experiments showed

[1]To ensure positive rewards, it is implemented as $u = \max(0, r_0)\max(0, r_1)$.

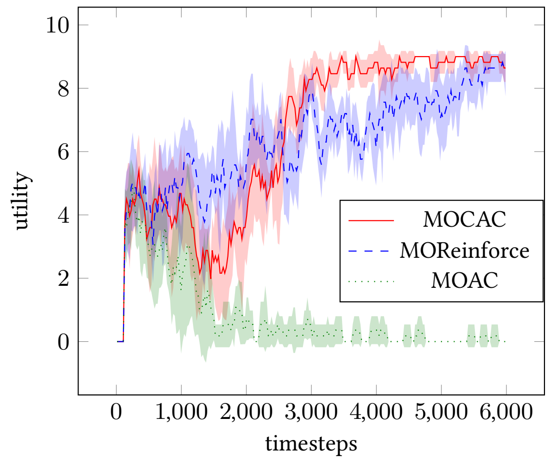

Figure 2: Results for the Split environment. Using a distribution over returns (MOCAC) is key for learning optimal policies when using a critic. When this is not the case (MOAC), the ESR criterion is not met.

that our algorithm can cope with many types of utility functions and our conclusions remain valid.

5.1 Deep-Sea-Treasure

Deep-Sea-Treasure is a classic multi-objective benchmark [21] in which the agent controls a submarine in search for treasure. Deeper in the ocean lies higher-valued treasures, but it will take longer to reach them. Thus, there is a trade-off between treasure-value and time.

Linear Utility Function. We consider two scenarios for this environment. Although our focus lies in the non-linearity of utility functions, we first show that MOCAC performs just as well on linear utility functions. In this scenario, we use a linearly weighted sum as utility function, i.e., $u = wr_0 + (1-w)r_1$ and $w \in 0, 0.1, \ldots, 1$.

As explained in Section 2.2, in such a setting ESR is equivalent to SER. The utility function u can be applied on the individual rewards, reducing the MOMDP to a MDP. Thus, at each timestep, the scalar reward $r_t = u(\vec{r}_t)$. As a baseline, we evaluate the well-known single-objective Advantage Actor-Critic (A2C) algorithm [9] on this MDP, as it is the one closest to our method. We call this baseline *A2C (timestep)*.

Non-linear Utility Function. In a second scenario we consider a non-linear utility function since a key contribution of our proposed method is that it can cope with this class of functions. In this setting, a debt-ridden crew is seeking treasure to pay back their creditors before some deadline. If they are late, they have to pay a late-fee penalty, as well as interests for every additional timestep. We translate this scenario in the following utility function:

$$u = \begin{cases} \ln(1 + e^{(r_0 - d_0)}) & \texttt{if } r_1 \leq d_1 \\ \ln(1 + e^{(r_0 - d_0)}) - (r_1 - d_1)^2 - p & \texttt{if } r_1 > d_1, \end{cases} \quad (16)$$

where d_0 is the debt, d_1 the deadline, and p the penalty. The first term is a softplus function, meaning that any treasure with a lower value than the debt will yield a zero reward, but the crew is of course free to keep any additional spoils. The other terms represent the interests and penalty. In this case, $d_0 = 45, d_1 = 10, p = 10$, resulting in the sixth treasure being optimal (out of ten).

For this scenario, we include A2C (timestep) as well. While this approach is not theoretically sound for non-linear utility functions, it provides an insight into what happens if the multi-objective aspect is ignored.

Since in this case timestep scalarization does not lead to the correct user utility, we propose another MOMDP to MDP transform, such that single-objective methods can be applied, even with a non-linear u. During any non-terminal timestep, the MDP returns a zero-reward $r_t = 0$. However, the actual vectorial rewards are accumulated in the background until a terminal state is reached (also taking into account γ). During the last, terminal timestep, the accumulated vectorial rewards correspond to the episodic return. The scalar reward which is then returned by the MDP corresponds to the utility of this return, i.e., $r_t = u(\vec{R}_t)$.

Two issues arise with this transform. Firstly, it reduces the problem to a single-objective MDP with a highly sparse reward function. Such sparse problems are notoriously hard to learn, especially for long episodes (a well-known example is the Atari 2600 game *Montezuma's Revenge*) [4]. Secondly, similarly as with accrued rewards, this transform breaks the Markov property. Still, we noticed that augmenting the state-space to recover the Markov property worsened the agent's performance. It is thus omitted, leaving only the sparsity problem.

As a baseline, we train A2C on this transformed MDP, which we call *A2C (terminal)*. As, to our knowledge, no specific known-utility ESR methods presently exist, this is the only method – with our proposed algorithm MOCAC and baseline MO Reinforce – to cope with this setting.

5.1.1 *Results.*

We first discuss the linear utility function. Regardless of the value for the weight w, MO Reinforce is not able to learn well because, unable to explore beyond the first treasure, it sticks to the easily obtainable reward the treasure provides. In contrast, both the A2C (timestep) and A2C (terminal) baselines learn the optimal policy. However, differences in learning speed appear depending on the weight value.

For example, low values for w result in lower convergence speeds for A2C (terminal). Since it only receives non-zero rewards at the end of the episode, it spends more time exploring compared to A2C (timestep). On the other hand, with high values for w, A2C (timestep)'s convergence speed decreases compared to the other algorithms, even though rewards are provided at every timestep. In this case, exploration is beneficial for A2C (terminal). It reaches later treasures earlier than A2C (timestep).

Regardless, MOCAC proves to be robust to the differences in weights, as it systematically learns the optimal policy. Moreover, its convergence speed is on par with or faster than the best performing baseline for each w.

With a non-linear utility function, the story becomes quite different: only the explicitly multi-objective methods consistently reach the optimal solution. As expected, A2C (timestep) is unable to learn

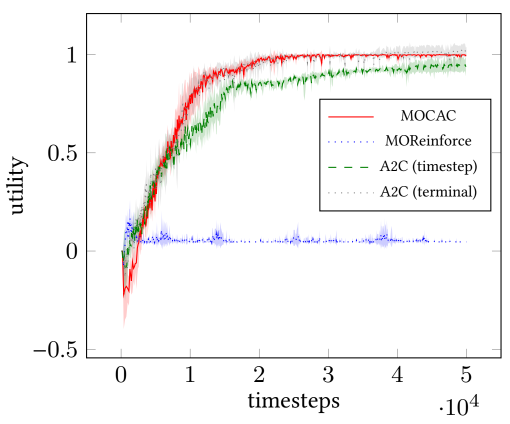

Figure 3: Results for Deep-Sea-Treasure, using linear utility with weight $w = 0.9$. A2C (timestep) displays a slower convergence speed than MOCAC and A2C (terminal). As for most weights, is stuck at the first treasure.

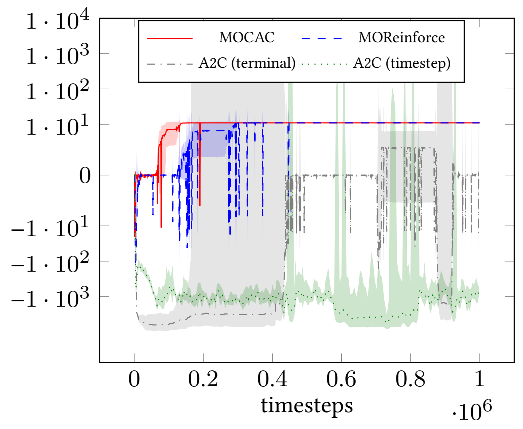

Figure 4: Results for Deep-Sea-Treasure, using a non-linear utility function. Only the dedicated multi-objective approaches learn the optimal policy. A2C (terminal) shows unstable learning curves and A2C (timestep) fails to learn any decent policy.

any decent behaviour. Since scalarisation occurs at every timestep, u never receives the total time spent to find the treasure, meaning the deadline penalty is never applied. Because time is incorrectly taken into account, this baseline learns a policy that seeks the biggest treasure, but is also the one that is the furthest away. This yields a poor episodic utility.

Secondly, although the terminal scalarisation does receive the correct utility, learning quickly stagnates and the algorithm is unable to reach optimality. The harsh time constraints (the quadratic time factor as well as the penalty term) make exploration difficult. Moreover, since scalarisation is applied, it occurs that high treasure values get negated by the time penalties, especially when the episode contains unnecessary steps. After 1.5×10^5 timesteps, the learnt policy leads the submarine to the nearest treasure and is incapable of learning better behaviour.

In contrast, both and MOCAC reach the optimal treasure. Even though the time constraint penalises the total utility, keeping track of the different objectives separately benefits the learning process. In terms of convergence speeds, MOCAC reaches the optimal treasure first but, as the episodes are short, the number of additional updates compared to the number of episodes amounts to an average of only 0.2%. Thus, it is not so much the sample efficiency but the reduction in variance – due to the advantage – that helps MOCAC perform better than MO Reinforce.

5.2 Minecart

Finally, we perform experiments on a complex environment with a high-dimensional state-space: Minecart [1]. Starting at a base station, the agent controls a cart whose goal is to mine diverse ores from the mines scattered in the environment, and go back to the base to sell the ores.

The agent can execute 6 possible actions: it can accelerate, decelerate and rotate the cart to the left or right. It can mine ores, which will only be effective if it is located in a mine. It can also simply do nothing. There are a total of 2 objectives: the amount of ores mined, and the fuel consumption.

We perform two sets of experiments. In the first, the agent is trained on the 6-dimensional, continuous state-space from the environment (containing features such as cart position, velocity, ...). In the second, the agent is trained on 84×84 pixel frames, using the same image pre-processing and convolutional network architecture as in [10].

Non-linear Utility Function. As with the Deep-Sea-Treasure environment, we imagine a setting where a known non-linear utility function needs to be applied. The agent is a mining company with a contract to provide a specific amount of ores at an agreed-upon price. The leftover ores can be sold at market price. If there is a breach of contract due to insufficient amount of ores, a compensation penalty will be applied. Finally, fuel is seen as an additional expense. This can be formalised in the following utility function:

$$u = \begin{cases} t_0 p_0 + (r_0 - t_0) p_1 - r_1/20 & \texttt{if } t_0 \leq r_0 \\ r_0 p_0 - c - r_1/20 & \texttt{if } t_0 > r_0, \end{cases} \tag{17}$$

where t_0, p_0, p_1, c are the request ore amount, contract price, market price, and compensation penalty, respectively. In our scenario, $t_0 = 0.7, p_0 = 5, p_1 = 7, c = 2$.

5.2.1 Results. As can be seen in Figure 5 and Figure 6, for both sets of experiments MOCAC reaches the highest utility. The agent almost consistently fills its cart to capacity, and goes back to the base station to sell the minerals.

Looking at the baselines, we observe different behavior depending on the state-space used for training. For the 6-dimensional state-space, MO Reinforce on average performs on-par with A2C (terminal), despite the higher variance and worse sample efficiency than its actor-critic counterpart. Both agents do not always fill their cart to full capacity, resulting in a lower utility.

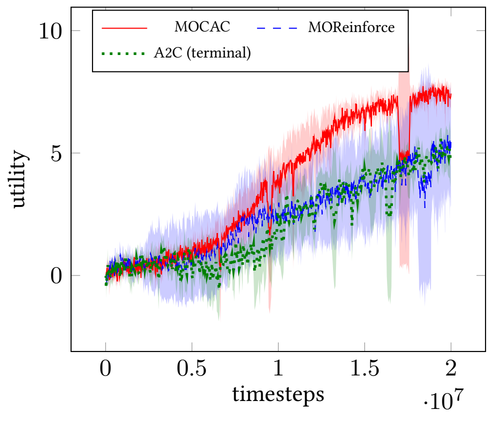

Figure 5: Results for Minecart using a non-linear u with a 6-dimensional continuous state-space. MOCAC outperforms all baselines. MO Reinforce performs on-par with A2C (terminal).

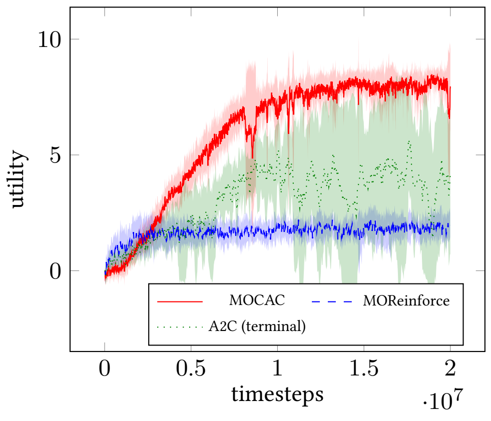

Figure 6: Results for Minecart using a non-linear u with 84×84 pixel frames, similar to the Atari 57 suite. MOCAC outperforms all other algorithms.

With pixel frames their behavior is different. Although the network architecture is (except for the output) the same as MOCAC, some A2C (terminal) runs never learn to mine ores. MO Reinforce only partially fills its cart, usually just enough to reach the quota, but with nothing left to sell at market price. All in all, combining the actor with a distributional critic in MOCAC is key to obtain good utility.

6 RELATED WORK

Current MORL methods focus almost exclusively on SER. They can be broadly divided into two main categories: *single-policy* and *multi-policy* algorithms [21]. In the first case, one tries to learn a single, optimal policy for a given set of preferences, i.e., for a known utility function. In the second case, the utility function is unknown (or uncertain) and the goal is to learn a set of policies that cover all possible utility functions, i.e., a coverage set.

Most of the recent work on MORL falls into the second category and assumes an unknown, but linear utility function. The goal is then to train an agent such that the optimal policy can be recovered for any preference weights. [17] propose Optimistic Linear Support (OLS), a generic method that iteratively selects different sets of weights and calls a single-objective subroutine to find the corresponding optimal policy. [11] extend this method for Deep RL. Another approach, taken by [2, 7], is to directly optimise on the coverage set without single-objective subroutine, by modifying the Bellman equation. In [5], Fitted Q-Iteration (FQI) is extended to use a modified Q-network conditioned on preference weights. Similarly, [1] use such a conditioned Q-network to extend Deep Q-Networks (DQN). Moreover, a similar network is used in [25], in combination with a multi-objective Bellman operator. All these approaches aim to learn a convex coverage set [16], which is the coverage set for linear utility functions, but can also be used to construct a coverage set for unknown non-linear utility functions when policies are allowed to be stochastic, employing mixture policies [20].

In the setting where the utility function is decided upon beforehand, [12, 19] use linear utility functions, with the known issue that a small change in weights might lead to completely different policies [21]. Non-linear utility has been investigated in a tabular setting by [22], who use a Chebyshev function. Moreover, monotonically increasing utility functions in general have been investigated in the (much simpler) bandit setting [18], by modelling them using a Gaussian process and interacting with the user to obtain preference information.

Finally, when the utility function is unknown, can be non-linear, but only deterministic policies are allowed, the coverage set is the Pareto front of deterministic (possibly non-stationary) policies. [13, 15, 23] learn such a Pareto front using metrics such as hypervolume or non-dominance.

7 CONCLUSION

We proposed Multi-Objective Categorical Actor-Critic (MOCAC). To our knowledge, this is the first actor-critic RL algorithm that can handle MORL under the expected scalarized returns criterion, where the utility function can be non-linear. MOCAC takes into account accrued rewards and, in contrast to single-objective actor-critic RL algorithms, its critic only works if it learns a multivariate distribution over future returns, rather than an expected value over future returns. We show empirically that MOCAC can successfully learn in MOMDPs under ESR with a known utility function. Furthermore, we show that it is much more sample-efficient and stable than all the proposed alternatives, clearly indicating that learning a distribution over the vectorial returns can convey important benefits in this class of problems.

In future work, we aim to learn the non-linear utility function by interaction with the user (querying its preferences).

Table 1: Hyperparameters for the Split, Deep-sea-treasure and Minecart environments.

	Split	Deep-Sea-Treasure	Minecart
	Common		
lr	0.001	0.001	0.0003
γ	1.00	0.95	1.00
timesteps	$6,000$	$1,000,000$	$20,000,000$
neurons (actor)	$(26, 20, 2)$	$(132, 50, 4)$	$(6, 20, 20, 6)$
non-linearity	Tanh	Tanh	Tanh
clip-grad-norm	None	50	50
	MOCAC		
value-coef	0.5	0.5	0.5
entropy-coef	0.01	0.1	0.1
update every	1	10	200
neurons (critic)	$(26, 50, 121)$	$(132, 50, 50, 121)$	$(6, 20, 20, 20, 121)$
c	11	11	11
V_{MIN}	$(-1, -1)$	$(0, -20)$	$(0, -4)$
V_{MAX}	$(10, 10)$	$(100, 0)$	$(1.5, 0)$
	MOAC	**A2C**	
value-coef	0.5	0.5	0.5
entropy-coef	0.01	0.1	0.1
update every	1	10	200
neurons (critic)	$(26, 50, 1)$	$(132, 50, 50, 1)$	$(26, 20, 20, 1)$

A EXPERIMENTAL DETAILS

A.1 Split

In the Split environment, the agent chooses between two hallways of length 11 to traverse. Including the start- and end-states, there are 26 states. The states are one-hot encoded, resulting in a 26-sized vector that is given as input to any of the actor and critic estimators.

A.2 Deep sea treasure

Deep sea treasure is a grid-world environment, where the submarine moves on a 11×12, resulting in 132 different states. The state number is one-hot encoded, resulting in a 132-sized vector that is given as input to any of the actor and critic estimators. The rewards provided by each treasure are made in such a way that every optimal `treasure` $\times$ `fuel` combination is evenly spread out on the convex coverage set. Treasure values are displayed on Figure 7.

A.3 Minecart

In Minecart, the agent moves a cart in a continuous 2-dimensional space. The state is represented by a 6-dimensional continuous vector.

Additionally, we perform experiments using the pixel frames as observations. The frame pre-processing follows [10]: they are rescaled to 84×84, converted to grayscale and normalized.

All hyperparameters used for all these experiments, including neural network architectures are listed in Table 1.

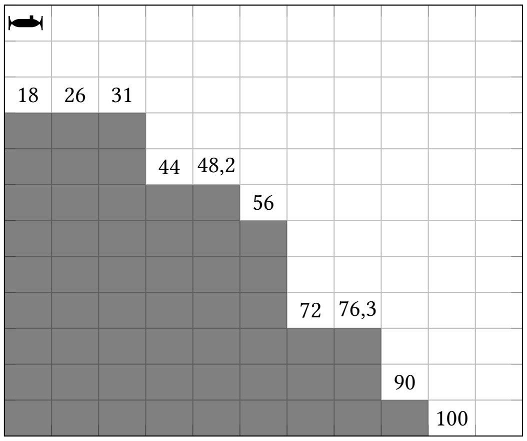

Figure 7: The *Deep Sea Treasure* environment. The agent starts on the top-left corner and tries to reach any of the treasures. Further treasures are worth more.

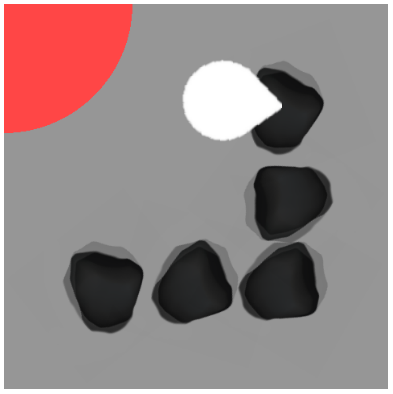

Figure 8: The *Minecart* environment. The base is located at the top-left corner, while the 5 mines are spread around the environment.

REFERENCES

[1] Axel Abels, Diederik Marijn Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. 2019. Dynamic Weights in Multi-Objective Deep Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, Long Beach, California, USA, 11–20.

[2] Leon Barrett and Srini Narayanan. 2008. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*. ACM, 41–47.

[3] Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887* (2017).

[4] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2019. Exploration by random network distillation. In *International Conference on Learning Representations*.

[5] Andrea Castelletti, Francesca Pianosi, and Marcello Restelli. 2012. Tree-based fitted Q-iteration for multi-objective Markov decision problems. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[6] A Castelletti, Francesca Pianosi, and Marcello Restelli. 2013. A multiobjective reinforcement learning approach to water resources systems operation: Pareto frontier approximation in a single run. *Water Resources Research* 49, 6 (2013), 3476–3486.

[7] Kazuyuki Hiraoka, Manabu Yoshida, and Taketoshi Mishima. 2009. Parallel reinforcement learning for weighted multi-criteria model with adaptive margin. *Cognitive neurodynamics* 3, 1 (2009), 17–24.

[8] Ammar Jalalimanesh, Hamidreza Shahabi Haghighi, Abbas Ahmadi, Hossein Hejazian, and Madjid Soltani. 2017. Multi-objective optimization of radiotherapy: distributed Q-learning and agent-based simulation. *Journal of Experimental & theoretical artificial intelligence* 29, 5 (2017), 1071–1086.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[11] Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. 2016. Multi-Objective Deep Reinforcement Learning. *CoRR* abs/1610.02707 (2016). arXiv:1610.02707 http://arxiv.org/abs/1610.02707

[12] Daniel Neil, Marwin Segler, Laura Guasch, Mohamed Ahmed, Dean Plumbley, Matthew Sellwood, and Nathan Brown. 2018. Exploring deep recurrent models with reinforcement learning for molecule design. In *6th International Conference on Learning Representations (ICLR), Workshop Track*.

[13] Simone Parisi, Matteo Pirotta, and Marcello Restelli. 2016. Multi-objective reinforcement learning through continuous pareto manifold approximation. *Journal of Artificial Intelligence Research* 57 (2016), 187–227.

[14] Roxana Rădulescu, Patrick Mannion, Diederik M Roijers, and Ann Nowé. 2020. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems* 34, 1 (2020), 10.

[15] Mathieu Reymond and Ann 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*.

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

[17] Diederik Marijn Roijers, Shimon Whiteson, and Frans A Oliehoek. 2015. Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research* 52 (2015), 399–443.

[18] Diederik M Roijers, Luisa M Zintgraf, Pieter Libin, and Ann Nowé. 2018. Interactive Multi-Objective Reinforcement Learning in Multi-Armed Bandits for Any Utility Function. In *ALA workshop at FAIM*, Vol. 8.

[19] Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey Kephart, David Levine, Freeman Rawson, and Charles Lefurgy. 2008. Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems*. 1497–1504.

[20] Peter Vamplew, Richard Dazeley, Ewan Barker, and Andrei Kelarev. 2009. Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. In *Australasian joint conference on artificial intelligence*. Springer, 340–349.

[21] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. 2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning* 84, 1-2 (2011), 51–80.

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

[23] Kristof Van Moffaert and Ann Nowé. 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research* 15, 1 (2014), 3483–3512.

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

[25] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 14610–14621.