Model-Based Multi-Objective Reinforcement Learning with Dynamic Utility Functions

Johan Källström
Linköping University
Linköping, Sweden
johan.kallstrom@liu.se

Fredrik Heintz
Linköping University
Linköping, Sweden
fredrik.heintz@liu.se

ABSTRACT

Many real-world problems require a trade-off between multiple conflicting objectives. Decision-makers' preferences over solutions to such problems are determined by their utility functions, which convert multi-objective values to scalars. In some settings, utility functions change over time, and the goal is to find methods that can efficiently adapt an agent's policy to changes in utility. Previous work on learning with dynamic utility functions has focused on model-free methods, which often suffer from poor sample efficiency. In this work, we instead propose a model-based actor-critic, which explores with diverse utility functions through imagined rollouts within a learned world model between interactions with the real environment. An experimental evaluation shows that by learning a model of the environment the performance of the agent can be improved compared to model-free algorithms.

KEYWORDS

Multiple Objectives, Reinforcement Learning, Model-Based Learning

1 INTRODUCTION

In many real-world problems, an agent needs to consider multiple conflicting objectives when making decisions. For instance, when operating a mining company the market prices of various types of ores must be considered in relation to the cost of mining each of them. Such multi-objective optimisation problems do not have a single optimal solution, but instead a set of solutions that are optimal for some trade-off among the objectives.

Multi-objective reinforcement learning (MORL) provides methods that allow agents to learn these optimal solutions by interacting with the environment [8, 18]. Multi-objective reinforcement learning problems are modelled using vector reward signals [25], with each element representing one of the objectives, in contrast to the scalar rewards used in single-objective reinforcement learning [22], resulting in vector returns. To evaluate the outcomes of different solutions in relation to each other, a utility function is used to convert the vector return to a scalar, representing a specific trade-off among the objectives.

In some scenarios, the utility function is not fixed over time. In MORL research this is referred to as the *Dynamic Utility Scenario* [8]. For instance, in our example of a mining company, as the prices of different ores and fuel change, the utility (return on investment) of the company's policy for operating its equipment in its mines will also change, and a new policy must be found to balance the objective of mining ores and the objective of managing fuel consumption. To handle such changes in utility efficiently, it is desirable to reuse information from learning with previously encountered utility functions, instead of restarting learning from scratch. It would be desirable to successively learn a set of policies and their estimated multi-objective returns. Then, when preferences over objectives change, the user can select a suitable policy from the policy set to use and improve upon.

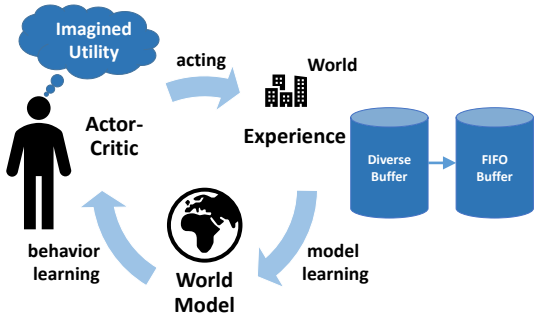


Figure 1: MO-Dreamer interacts with the environment and builds a dataset of diverse experiences, which are used to construct a model for imagination rollouts where past experienced states are revisited with experienced as well as imagined utility functions for improvement of the policy

Previous work in MORL for learning with dynamic utility functions has focused on model-free learning, which often suffers from poor sample efficiency. This is problematic in environments where only limited interaction with the real environment is available, for instance due to the cost of running experiments. Recently, model-based reinforcement learning methods have been improved, and demonstrated impressive performance on complicated tasks [5, 7, 20]. We believe that learning a model could be particularly useful for multi-objective reinforcement learning, where many different solutions need to be explored within a single environment to find one that fits the user's utility.

In this work, we therefore propose a model-based actor-critic, based on DreamerV2 [7], as illustrated in Figure 1. To stabilise learning in the dynamic utility scenario we use a form of *Diverse Experience Replay* to ensure that the replay buffer contains trajectories with diverse multi-objective returns. We then let the agent explore with diverse utility functions through imagined rollouts within the learned world model. An experimental evaluation on

Proc. of the Adaptive and Learning Agents Workshop (ALA 2023), Cruz, Hayes, Wang, Yates (eds.), May 9-10, 2023, Online, https://ala2023.github.io/. 2023.

the Minecart benchmark shows that the model-based agent significantly outperforms the model-free state-of-the-art for frequent as well as sparse utility changes. In additional experiments on the Deep Sea Treasure benchmark, the model-based agent outperforms the model-free agents overall by converging quickly, but learns a worse final policy. To the best of our knowledge this is the first study of model-based multi-objective reinforcement learning in the dynamic utility scenario.

The remainder of this paper is organised as follows. Sections 2 and 3 present relevant background information, as well as related work on learning in the dynamic utility scenario and model-based reinforcement learning. Section 4 proceeds to present the components of the MO-Dreamer agent, followed by an experimental evaluation in Section 5. Finally, section 6 provides conclusions and directions for future work.

2 BACKGROUND

In this section we present background information about reinforcement learning in Section 2.1, and multi-objective reinforcement learning in Section 2.2.

2.1 Reinforcement Learning

Reinforcement learning (RL) allows agents to find policies for sequential decision-making by interacting with their environment in a form of trial-and-error learning. RL problems are typically modeled as Markov decision processes (MDPs) defined by the tuple (S, A, T, R, γ), specifying the states, actions, transition dynamics, reward function, and discount factor of the process. In each time step the state $s_t \in S$ is observed and an action $a_t \in A$ is taken according to the agent's policy $\pi : S \times A \rightarrow [0, 1]$. After executing the action the agent enters a new state s_{t+1} according to the transition dynamics $T : S \times A \times S \rightarrow [0, 1]$, and receives a reward $r_{t+1} \in \mathbb{R}$. The goal of the agent is to maximise its future expected discounted return when starting in state s_0: $V_\pi(s) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_0 = s]$, with $\gamma \in [0, 1]$ defining the value of short-term and long-term rewards respectively.

2.2 Multi-Objective Reinforcement Learning

Multi-objective reinforcement learning is a generalisation of standard reinforcement learning to problems where multiple conflicting objectives must be considered. Such problems can be modelled as multi-objective Markov decision processes (MOMDPs) [18]. A MOMDP provides vector rewards, where each element in the vector represents the reward for one of the objectives. This results in vector returns and value functions. A solution found by an RL agent can be optimal in the sense that for at least one objective there is no other policy that produces a higher value. The set of returns produced by the set of optimal policies is known as the *Pareto front* [18].

To select a single policy we can use a utility function suitable for the intended user, and convert the vector values to scalar values for ordering of policies: $V^u_\pi = u(\boldsymbol{V}_\pi)$. The most commonly used utility function in previous work is the weighted sum [8]: $V^u_\pi = \boldsymbol{V}_\pi \cdot \boldsymbol{w}$. The weighted sum is a linear utility function, where each element w_i in the weight vector specifies the corresponding objective's relative importance compared to other objectives, and $\sum_{i=1}^{n} w_i = 1$ for n objectives. As in prior work on learning in the dynamic utility scenario, we use this form of utility function in our study.

When we model user utility with linear utility functions, we can use scalarised reinforcement learning [12] to learn policies. This means that we use the current preference weights to calculate the scalar value of vector value functions. The scalar values can then be used in the normal update rules for standard single-objective value-based or actor-critic style reinforcement learning algorithms. One disadvantage of scalarised learning with linear utility functions is that we cannot find solutions in the concave parts of the Pareto front [26]. Instead we can try to find solutions in the convex coverage set (CCS). A CCS is a set of policies that contains an optimal policy for every linear scalarisation $\boldsymbol{w}$ [18].

3 RELATED WORK

In this section we present work related to model-free multi-objective reinforcement learning in the dynamic utility scenario in Section 3.1, and model-based reinforcement learning in Section 3.2.

3.1 Learning with Dynamic Utility Functions

Natarajan & Tadepalli [13] introduced multi-objective reinforcement learning with dynamic preferences in tabular settings. Their method learns and stores un-dominated policies for encountered preference weights. When a weight change occurs, learning continues with the past policy that has the highest scalarised value for the new weight vector: $\boldsymbol{V}_\pi \cdot \boldsymbol{w}$. This improves performance compared to learning a new policy from scratch.

Abels et al. [1] extended learning with dynamic weights to multi-objective deep reinforcement learning. Instead of learning a set of individual policies, a single deep Q network (DQN [11]) was used to represent multiple policies, by conditioning the network on the preference weights of objectives. This allows the network to generalise across weight changes. In addition, a mechanism for enforcing diversity in the replay buffer was proposed, to prevent the agent from forgetting policies learned in the past. Diversity is measured as the *crowding distance* [3] of the returns of stored trajectories, and when the buffer is full trajectories that contribute the least to diversity are removed first. Conditioned networks have proven useful in the dynamic utility scenario, as well as in other settings [10, 14, 15, 17, 27, 30].

Nian et al. [14] extended the work of Abels et al. [1] to partially observable environments. They proposed a Deep Conditioned Actor-Critic (DCRAC), which can be equipped with LSTM or memory networks to form beliefs about the current state of the environment.

Wang et al. [27] proposed a new Near on-policy Experience Replay (NER) algorithm for settings where preference weights change rapidly, and the replay buffer may contain a large number of transitions that are not relevant for the current weight vector. This can result in large extrapolation errors [4]. The proposed method overcomes this problem by prioritising sampling of transitions that have states and weight vectors that are similar to the current state and weight vector.

Some model-based approaches have been proposed for MORL, e.g. variations of Monte Carlo tree search [9, 28], but to to the best of our knowledge they have not been studied in the dynamic utility

setting. We extend prior work on learning with dynamic utility functions by studying the potential benefits of online learning of a world model that can be used to train the agent.

3.2 Model-Based Reinforcement Learning

In many real-world problems, the number of interactions that a learning agent can have with the environment are limited. For example, environments populated by humans may have low availability, and environments populated by expensive vehicles may have a high operating cost. In such settings sample efficiency can be improved by learning a model of the environment dynamics, which can then support decision making by running simulations.

Recent advances in model learning have made it possible to build accurate compact representations even for complex environments with image observations. The Dreamer agent [5] uses a world model [6] consisting of three components: a representation model that provides compact vector-valued state representations of image observations, a transition model that predicts the next model state based on the current model state and action, and a reward model that predicts the reward for the current model state. The agent learns the latent dynamics of the world model by using reconstruction of the images that represent the observations of the agent as a learning objective. This results in an efficient model that can simulate several thousands of trajectories in parallel. The agent achieves impressive results on challenging visual control tasks in DeepMind Control Suite [23].

DreamerV2 [7] provides an evolution of the Dreamer agent, which significantly improves the performance on the challenging Atari benchmark [2] of environments with discrete action spaces and image-based observations. DreamerV2 uses categorical variables to represent the latent state of the world model, instead of the diagonal Gaussian distribution used by the original Dreamer agent. In addition, KL balancing is used to improve robustness to novel inputs as well as learning of long term dependencies in the environment. We use DreamerV2 as the basis for our work, and extend it to support learning in MOMDPs with dynamic utility functions.

4 METHOD

In this section we present the structure of the model-based multi-objective actor-critic, MO-Dreamer. The world model is presented in Section 4.1, followed by a description of model learning by diverse experience replay in Section 4.2. We then present the actor-critic in Section 4.3, followed by a description of policy learning through imagination rollouts in Section 4.4. We focus on the differences between MO-Dreamer and DreamerV2. For a detailed description of DreamerV2 we refer the reader to [7]. An overview of MO-Dreamer is shown in Figure 1 and pseudo code is given in Algorithm 1.

4.1 Multi-Objective World Model

We build upon the recurrent state space model (RSSM) proposed in DreamerV2 [7]. The RSSM consists of the recurrent model, representation model, and transition predictor, as shown in Equations (1) – (3).

$$h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}), \tag{1}$$

$$z_t \sim q_\phi(z_t|h_t, x_t), \tag{2}$$

$$\hat{z}_t \sim p_\phi(\hat{z}_t|h_t), \tag{3}$$

The RSSM uses the deterministic recurrent states h_t to compute distributions over the posterior state z_t and the prior state $\hat{z}_t$. The posterior state incorporates information about the image input x_t, and the prior state tries to predict the posterior state without access to x_t. The model state captures the current environment state based on observing a sequence of images of past environment states, and the model is able to predict forward in time to support "imagination" rollouts.

We also use the same predictors as DreamerV2 for state images, rewards, and discount (which is used to estimate the end of episodes), except that the reward is now a vector:

$$\hat{x}_t \sim p_\phi(\hat{x}_t|h_t, z_t), \tag{4}$$

$$\hat{\boldsymbol{r}}_t \sim p_\phi(\hat{\boldsymbol{r}}_t|h_t, z_t), \tag{5}$$

$$\hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t|h_t, z_t), \tag{6}$$

As in DreamerV2, the latent state of the model is represented with categorical variables, the image predictor is represented by a diagonal Gaussian with unit variance, and the discount predictor is represented by a Bernoulli likelihood. We make the assumption that the elements of the multi-objective reward are statistically independent, and represent them as individual univariate Gaussians with unit variance in the world model. Then the updated loss function for a world model of a MOMDP with n objectives is:

$$\begin{aligned}\mathcal{L}(\phi) \doteq E_{q_\phi(z_{1:T}|a_{1:T},x_{1:T})}\Bigg[\sum_{t=1}^{T} &-\ln p_\phi(x_t|h_t,z_t) \\ &-\sum_{i=1}^{n}\ln p_\phi(r_t^i|h_t,z_t)+\ln p_\phi(\gamma_t|h_t,z_t) \\ &+\beta KL[q_\phi(z_t|h_t,x_t)||p_\phi(z_t|h_t)]\Bigg].\end{aligned} \tag{7}$$

4.2 Diverse Experience Replay

The world model is trained with data collected from the agent's past experiences with the real environment, which have been stored in a replay buffer. In addition to sequences of image observations, actions, rewards, and discount factors, the stored experiences also contain information about the utility weights that were active in the corresponding episode. DreamerV2 randomly samples trajectories from the buffer to construct batches of sequences with fixed length. In the dynamic utility scenario it is necessary for the agent to

learn that different policies should be followed for different utility functions, and to retain that knowledge over time. Abels et al. [1] addressed this issue by splitting the replay buffer of a DQN agent into a main buffer and a secondary buffer that enforces diversity in terms of the multi-objective returns contained in the buffer. When the buffer becomes full and samples must be removed, the trajectories that contribute the least to diversity are removed first. We apply a variation of this technique to the replay buffer of MO-Dreamer.

When using model-based learning in the dynamic utility scenario, it is important to quickly learn a sufficiently accurate model of the achievable rewards in different parts of the environment. To promote learning environment features suitable for multiple utility functions we want to ensure diversity in the data used in the early stages of learning. We therefore use two replay buffers, as in [1], but enforce diversity on the main buffer. The diversity mechanism thereby comes into play as soon as the main buffer is filled (while in [1] it does not come into play until both buffers are filled). When the main buffer is full, the trajectory that contributes the least to diversity is moved to the secondary buffer, which uses a first-in-first-out (FIFO) principle. We then sample trajectories from either buffer with a probability proportional to the number of samples contained in each of them. This means that the early stages of learning will prioritise sampling trajectories with diverse outcome in terms of the multi-objective return, while in later stages of learning the diversity buffer and secondary buffer will be sampled with equal probability.

4.3 Actor-Critic for Dynamic Utility

We use an actor-critic setup to learn the behaviour of the agent, where the critic learns a multi-objective value function that guides the updates of the actor's policy. To enable single-network representations of the action distributions as well as the state value functions of multiple policies, each corresponding to a different utility function, we condition both actor and critic on the current utility weights:

$$\hat{a}_t \sim p_\psi(\hat{a}_t|\hat{z}_t, \boldsymbol{w}), \tag{8}$$

$$\boldsymbol{v}_\xi(\hat{z}_t, \boldsymbol{w}) \approx E_{p_\phi p_\psi}\left[\sum_{\tau \geq t} \hat{\gamma}^{\tau-t}\hat{\boldsymbol{r}}_\tau \middle| \boldsymbol{w}\right], \tag{9}$$

where ψ and ξ are the parameters of the actor and critic respectively. When the critic gets a certain weight vector as input in combination with the current model state, it will output the corresponding vector of optimal objective values from its current estimation of the CCS. When the actor gets a certain weight vector as input, it will select the best policy for optimising the corresponding utility.

To learn the value function of the multi-objective critic we use temporal-difference learning with the vector λ-return [22] as target value:

$$\boldsymbol{V}_t^\lambda \doteq \hat{\boldsymbol{r}}_t + \hat{\gamma}_t \begin{cases} (1-\lambda)\boldsymbol{v}_\xi(\hat{z}_{t+1}, \boldsymbol{w}) + \lambda \boldsymbol{V}_{t+1}^\lambda & \text{if } t < H \\ \boldsymbol{v}_\xi(\hat{z}_H, \boldsymbol{w}) & \text{if } t = H \end{cases}, \tag{10}$$

Algorithm 1 MO-Dreamer

```
Initialise neural network parameters randomly.
Prefill replay buffer with exploration policy.
while t < t_max do
  for c:=1 to N_train do
    p_main = N_main/N_tot : probability of sampling main buffer
    p_secondary = 1 − p_main
    Sample batch B with chunks from replay buffers.
    Train world model with B.
    Select mix of current, past and imagined weights {w_i}.
    Imagine trajectories {(z_τ, a_τ, w_i)}_{τ=t}^{t+H} from each initial
    model state-utility pair (z_t, w_i).
    Predict rewards r̂_t ~ p_φ(r̂_t|h_t, z_t) and values.
    Train actor-critic with data from imagination rollouts.
  end for
  for τ := t to t + T do
    Compute action â_τ ~ p_ψ(â_τ|ẑ_τ, w)
    Step environment o_{τ+1}, r_{τ+1}, done ← env.step(â_τ)
    if done then
      o_{τ+1} ← env.reset()
    end if
  end for
  Add new experience to main replay buffer.
  Enforce sample limit and diversity in main buffer.
  Enforce sample limit in secondary buffer.
end while
```

and optimise using the squared loss:

$$\mathcal{L}(\xi) \doteq E_{p_\phi, p_\psi}\left[\sum_{t=1}^{H-1} \frac{1}{2}\|\boldsymbol{v}_\xi(\hat{z}_t, \boldsymbol{w}) - \text{sg}(\boldsymbol{V}_t^\lambda)\|^2\right], \tag{11}$$

where *sg* refers to stopped gradients.

To learn the policy of the actor we use the scalarised advantage function as a baseline and optimise with the following Reinforce [29] loss:

$$\begin{aligned}\mathcal{L}(\xi) \doteq E_{p_\phi, p_\psi}\Bigg[\sum_{t=1}^{H-1}\Bigg(&-\rho \ln p_\psi(\hat{a}_t|\hat{z}_t, \boldsymbol{w}) \\ &\cdot \text{sg}((\boldsymbol{V}_t^\lambda - \boldsymbol{v}_\xi(\hat{z}_t, \boldsymbol{w})) \cdot \boldsymbol{w}) - \eta \text{H}[a_t|\hat{z}_t, \boldsymbol{w}]\Bigg)\Bigg],\end{aligned} \tag{12}$$

where the weight vector $\boldsymbol{w}$ is the utility function that was active when the corresponding samples of a batch was collected. The entropy term in the actor loss can help prevent over-fitting, which is of particular importance in the dynamic utility scenario, where the agent needs to do transfer learning between different preference weights.

Since the actor-critic is learning from the compact hidden state of the world model, the actor and critic are represented by simple MLPs, rather than the CNNs that would typically be used when learning directly from image input.

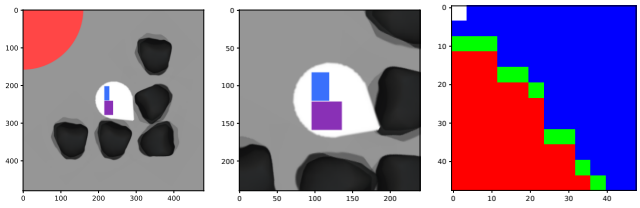


Figure 2: Minecart (left), Partially Observable Minecart (middle), and Deep Sea Treasure (right) evaluation environments

4.4 Imagination Rollouts

The MO-Dreamer actor-critic is trained through "imagination" rollouts in the learned world model. The initial state of each rollout corresponds to a time step of a sequence sampled from the replay buffer and then encoded to a compact world model state. In addition to the compact representation of the experienced real environment state, the agent is also provided with the utility function that was active in the episode where the state was experienced. Either this utility function or the current utility function of the environment are used by the agent when doing rollouts in the world model and optimising the policy with the sample batch collected. Continually revisiting past trajectories prevents the agent from forgetting which policy to use for utility functions encountered in the past, and the robustness of the agent's behaviour is improved. The diverse experience replay mechanism ensures that there remains a diverse mixture of initial states and utility functions in the replay buffer.

In the initial stages of learning the replay buffer is filled with experiences gathered by an exploration policy, and the trajectories followed may not be at all optimal for the actual utility function of the episode. Since we are learning with a model, we are able to revisit states previously encountered by the agent, but with a different utility function, to explore which parts of the environment provide most value for that function. We implement this mechanism in the exploration phase of the agent, by performing additional imagination rollouts with imagined utility functions sampled by the agent, rather than the utility functions experienced in the past. This results in a diverse mixture of initial states and utility functions, which can help the agent figure out which parts of the environment to visit when given a utility function in the future.

5 EXPERIMENTAL EVALUATION

In this section we present the experimental evaluation of MO-Dreamer. Section 5.1 describes the environments and evaluation metrics used. Section 5.2 provides a summary of the algorithms studied and their settings. Section 5.3 then presents the results.

5.1 Experiment Setup

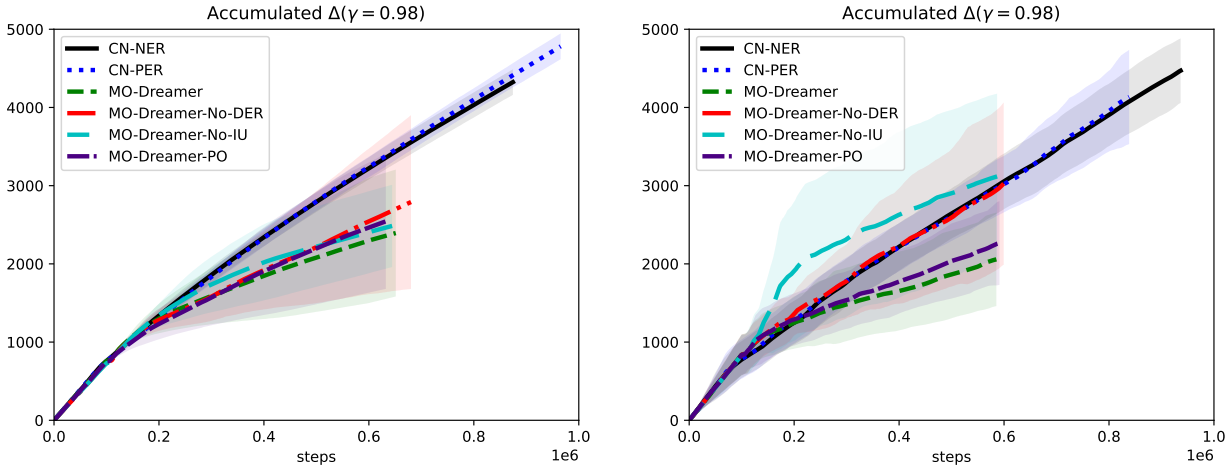
In this work, we study the dynamic utility scenario with frequent and sparse utility changes. As in previous work, we use a linear utility function represented by objective weights, which are sampled from a Dirichlet distribution ($\alpha = 1$). We perform experiments on the well-known MORL benchmarks Minecart [1] and Deep Sea Treasure [1, 24], illustrated in Figure 2. These environments were also used in prior work on learning with dynamic utility functions.

In Minecart the agent operates a minecart (white icon) to different mines (black areas) for mining different ores, which can then be brought back to the home base (red area) to be sold. The current contents of the cart is displayed as colour bars within the cart icon, as illustrated in Figure 2. The available actions allow the agent to Mine, turn Left or Right, Accelerate, Brake, or Idle. The objectives are related to the values of the different ores once sold, and the fuel cost caused by operating the minecart. Episodes end when the minecart returns to the base, or when a maximum of 1000 time steps have passed. Minecart is one of the more challenging benchmarks for multi-objective reinforcement learning, with a high-dimensional observation space in the form of an image of the environment state (as in Figure 2), stochastic state transitions when mining (based on the ore distribution specified for each mine), and delayed rewards for mining and selling ores. We use the default configuration of Minecart [1], which has two ores. In Minecart we also run experiments that study the world model's ability to handle partial observability, by only providing a 240x240 pixels observation centred on the agent (as in Figure 2) instead of the 480x480 pixels observation of the full environment. In this environment we use a discount factor of $\gamma = 0.98$.

In Deep Sea Treasure the agent operates a submarine (white square) to treasures (green squares). Deeper treasures have higher value, but there is a penalty for each time step in the episode, resulting in two objectives. The available actions allow the agent to step Left, Right, Up or Down. Episodes end when a treasure is collected, or when a maximum of 1000 time steps have passed. Observations are given as images of the environment state (as in Figure 2). We use the configuration from [1], where the value of collecting each treasure lies in the CCS. In this configuration each time step gives a penalty of -1, and the values of treasures from left to right are $\{18, 26, 31, 44, 48.2, 56, 72, 76.3, 90, 100\}$. In this environment we use a discount factor of $\gamma = 0.95$.

Ideally we would like to use regret as a metric to compare the results of different algorithms. The regret is defined as the difference between the optimal return and the actual return for a given utility function, $\Delta(\boldsymbol{g}, \boldsymbol{w}) = \boldsymbol{V}^*_{\boldsymbol{w}} \cdot \boldsymbol{w} - \boldsymbol{g} \cdot \boldsymbol{w}$, where $\boldsymbol{V^*}_{\boldsymbol{w}}$ is the optimal value in the CCS for the current weight vector, and $\boldsymbol{g}$ is the discounted return. The regret metric can consistently evaluate performance over different runs and for different weight vectors.

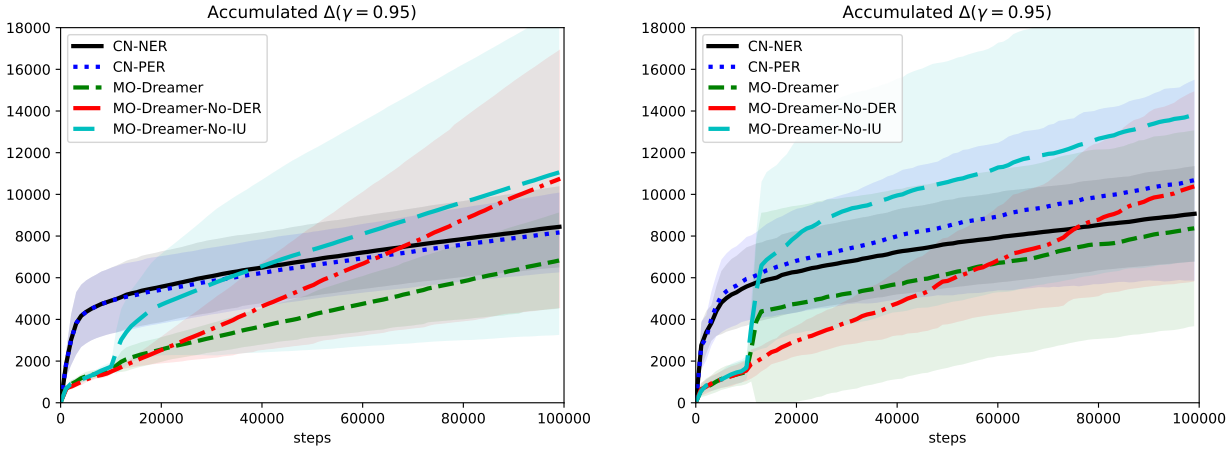
Since the optimal policy for Minecart is not known, we instead use the heuristic proposed by [1] to estimate an approximate CCS. Since we are using an estimate for the optimal utility, there is a chance that the regret estimate could become negative, if the RL agents learn policies that outperform the heuristic. To avoid this, we estimate the optimal episodic utility with the maximum of $\boldsymbol{V}^*_{\boldsymbol{w}} \cdot \boldsymbol{w}$ and the highest utility achieved by any learning agent. A negative side effect of this metric is that we can only evaluate for the fewest episodes completed by any learning agent, which is roughly 38000 episodes for CN-PER (as defined in Section 5.2). For Deep Sea Treasure the optimal policy is known, so for experiments in that environment we evaluate regret for all learning steps. Experiments in Minecart last for 1M time steps in the environment, experiments in Deep Sea Treasure last for 100k time steps, and each experiment is run for ten iterations to reduce the effects of random variations. We then evaluate the mean cumulative regret, as well as the mean episodic regret.



(a) Cumulative Δ for frequent utility changes

(b) Cumulative Δ for sparse utility changes

Figure 3: Average cumulative episodic Δ on Minecart over ten runs for frequent and sparse utility changes



(a) Cumulative Δ for frequent utility changes

(b) Cumulative Δ for sparse utility changes

Figure 4: Average cumulative episodic Δ on Deep Sea Treasure over ten runs for frequent and sparse utility changes

5.2 Algorithms

In the experiments we compare MO-Dreamer to the existing state-of-the-art model-free algorithms for learning with dynamic utility functions presented in Section 3. We also evaluate MO-Dreamer to ablated versions of itself, to see how different components of the agent affects its performance. We study the following agents:

- MO-Dreamer: Full model-based agent with utility conditioned actor-critic networks, diverse experience replay, and imagined utility functions during exploration
- MO-Dreamer-No-DER: Ablation study of MO-Dreamer without diverse experience replay and sampling
- MO-Dreamer-No-IU: Ablation study of MO-Dreamer without imagined utility functions during exploration
- MO-Dreamer-PO: MO-Dreamer with partial observability
- CN-NER: Model-free baseline with utility conditioned DQN and diverse experience replay according to [1], combined with near on-policy experience sampling (NER) according to [27]
- CN-PER: Model-free baseline with utility conditioned DQN and diverse experience replay according to [1], combined with standard prioritised experience replay (PER) [19]

We configure MO-Dreamer to train every 10 steps in the real environment for Minecart, and every 5 steps for Deep Sea Treasure. Imagination rollouts have a horizon of 15 steps in the world model. We train with batches of 10 sequences of 50 steps, sampled from the replay buffer. The replay buffer is configured with a capacity of 200k steps for Minecart and 20k steps for Deep Sea Treasure. We prefill the buffer with 100k exploration steps for Minecart and 10k exploration steps for Deep Sea Treasure (corresponding to the size

of the main buffer), before starting normal training. In Minecart we explore using a random policy. In Deep Sea Treasure we instead use model-based exploration with Plan2Explore [21], since random exploration is inefficient for finding the deeper treasures. Plan2Explore provides intrinsic motivation to the agent for exploring parts of the environment where the model quality is low.

Up until the first 20% environment steps we perform 8 training iterations between each interaction with the environment, to quickly learn a high quality world model and policy. Half of these iterations use imagined utility functions. After 20% steps we reduce the training intensity to 2 iterations between each interaction with the environment (primarily to reduce the amount of wall time required to run the experiments), and only use current and past experienced utility functions in rollouts to focus learning on experienced combinations of state and utility.

We use the default settings of DreamerV2 for the actor and critic networks. For the model networks we use the default configuration in Minecart, while in Deep Sea Treasure we use the simpler ATARI configuration, since Deep Sea Treasure has a simpler observation space. The Adam optimiser is used for training, with learning rates of $\alpha = 2 \times 10^{-4}$ for the model and critic, and $\alpha = 8 \times 10^{-5}$ for the actor.

For the model-free agents we use the reference implementation and hyperparameters of CN-PER provided by [1] as a basis, and then add the near on-policy sampling mechanism proposed by [27] with its default hyperparameters.

5.3 Results

The average cumulative regret and standard deviation over ten runs on Minecart is shown in Figure 3. It can be seen that the model-based agents need fewer steps to complete the same number of episodes as the model-free agents. It can also be seen that MO-Dreamer outperforms the model-free baselines in terms of average cumulative regret, for frequent as well as sparse utility changes, indicating improved sample-efficiency. The algorithm converges quickly after the exploration phase has ended at 100k environment steps, which is when the main replay buffer is filled. The two model-free baselines almost overlap in both settings, although they accumulate different amounts of regret after completing all episodes of learning. In our experiments, CN-PER performs better with frequently changing utility functions than what was reported by [27], but it is outperformed by CN-NER. For sparse utility changes, CN-PER has slightly better performance on average.

MO-Dreamer without diverse experience replay performs worse than the full algorithm, especially so when learning with sparse utility changes. This is likely caused by over-fitting. Qualitatively we could observe during training that in some runs an agent would have a bias for one of the ore's mined, regardless of the current utility function. The agent would often recover from this sub-optimal behaviour later in training, but at that point a lot of regret could have been accumulated. The recovery from sub-optimal behaviour indicates that it is beneficial for a learning agent to be able to revisit a previously visited area of the environment and improve upon its behaviour for a given utility function.

When learning with sparse utility changes, using imagined utility functions is essential for good performance, as illustrated by Figure 3. This mechanism allows the agent to prepare for dealing with utility functions not yet encountered in the real environment. When learning with frequently changing utility functions the mechanism is less important, since the replay buffer will still contain a sufficiently large set of diverse examples of preference weights.

Table 1: Average episodic Δ on Minecart

Algorithm	Δ overall	Δ after 200k steps
Frequent Utility Changes (every episode)		
CN-NER	0.1215 ± 0.0057	0.0959 ± 0.0053
CN-PER	0.1278 ± 0.0097	0.1030 ± 0.0089
MO-Dreamer	**0.0661** ± 0.0226	**0.0377** ± 0.0228
MO-Dreamer-No-DER	0.0786 ± 0.0342	0.0529 ± 0.0380
MO-Dreamer-No-IU	0.0682 ± 0.0132	0.0389 ± 0.0120
MO-Dreamer-PO	0.0716 ± 0.0257	0.0466 ± 0.0262
Sparse Utility Changes (every 1000 episodes)		
CN-NER	0.1313 ± 0.0131	0.1079 ± 0.0142
CN-PER	0.1291 ± 0.0199	0.1043 ± 0.0217
MO-Dreamer	**0.0605** ± 0.0130	**0.0309** ± 0.0081
MO-Dreamer-No-DER	0.1011 ± 0.0381	0.0753 ± 0.0431
MO-Dreamer-No-IU	0.0919 ± 0.0314	0.0476 ± 0.0179
MO-Dreamer-PO	0.0704 ± 0.0186	0.0414 ± 0.0146

Providing MO-Dreamer with only partial observations of the environment instead of full observations only has a small negative impact on the agent's performance. For frequent utility changes MO-Dreamer-PO's regret curve almost overlaps the regret curve of MO-Dreamer-No-DER up until 500k steps, where MO-Dreamer-PO starts performing better. For sparse utility changes MO-Dreamer-PO's performance is close to that of the full MO-Dreamer agent, while outperforming the other model-based agents. In both settings MO-Dreamer-PO outperforms the model-free baselines, even though they are learning with full observability.

Figure 3 shows that there is more variance in the results for MO-Dreamer compared to the model-free baselines. This experiment uses a high training intensity in the early stages of learning, in an attempt to reduce the accumulated regret. There is a risk that an agent will learn sub-optimal policies during the period when all relevant features are not available in the compact state of the world model, e.g., idling at the home base to minimise fuel costs instead of moving to a mine to collect ores. One thing we noted when running the experiments was that the representation of the cart's contents took time to learn. This problem of "vanishing objects" has also been noted in previous work [16]. In the early stages of learning this issue could, e.g., result in the learning agent being rewarded for bringing back ores to the home base, even though it cannot observe that there are actual ores stored in the cart.

Table 1 presents the average episodic regret overall, as well as after 200k environment steps for Minecart. MO-Dreamer significantly outperforms the model-free baselines for frequent as well as sparse utility changes.

The average cumulative regret and standard deviation over ten runs on Deep Sea Treasure is shown in Figure 4. Since the optimal policy is known in this environment, we evaluate all agents over

Table 2: Average episodic Δ on Deep Sea Treasure

Algorithm	Δ overall	Δ after 20k steps
Frequent Utility Changes (every episode)		
CN-NER	0.7885 ± 0.1616	0.3428 ± 0.0949
CN-PER	0.7705 ± 0.1629	**0.3295** ± 0.1010
MO-Dreamer	**0.6498** ± 0.1428	0.4585 ± 0.1677
MO-Dreamer-No-DER	0.8516 ± 0.3551	0.7259 ± 0.3851
MO-Dreamer-No-IU	0.9172 ± 0.4554	0.5994 ± 0.4201
Sparse Utility Changes (every 100 episodes)		
CN-NER	0.8429 ± 0.1949	**0.3385** ± 0.0854
CN-PER	0.9092 ± 0.2745	0.4123 ± 0.2138
MO-Dreamer	**0.7900** ± 0.3808	0.3822 ± 0.2136
MO-Dreamer-No-DER	0.8293 ± 0.2781	0.6535 ± 0.3339
MO-Dreamer-No-IU	1.1690 ± 0.4647	0.5756 ± 0.3617

the full 100k time steps of learning. MO-Dreamer accumulates less regret than the model-free baselines for frequent as well as sparse utility changes. However, compared to Minecart the comparison of agents is more affected by noise in the results. In a typical run of training MO-Dreamer with 10 iterations, there were a few outliers that performed significantly worse than the other iterations, and highly affected the mean and standard deviation of the experiment. This is similar to the behaviour related to overfitting that was also observed on Minecart, but more severe. One possible reason is that the replay buffer is too small, resulting in only a limited number of trajectories that reach the deeper rewards being stored in the buffer. The batches used for training are also assembled from sampled chunks of the stored trajectories, which means that for long trajectories the chunk may not contain the episode end, where the treasure is collected. Increasing the buffer size and using an improved sampling mechanism could improve performance. Other ways of improving performance could be to adjust the entropy coefficient, reduce the model capacity, or enhance the exploration policy.

Table 2 presents the average episodic regret overall, as well as after 20k environment steps for Deep Sea Treasure. MO-Dreamer has the best performance overall, but the model-free agents learn better final policies. CN-NER has slightly better total performance than CN-PER.

6 CONCLUSION

In this work, we proposed MO-Dreamer, a model-based multi-objective actor-critic for learning in environments with dynamic utility functions. MO-Dreamer enforces diversity in the returns of the trajetories stored in and sampled from the experience replay buffer, to enable high-intensity training early in the learning process without over-fitting. In addition, MO-Dreamer uses imagination rollouts with a diverse set of utility functions, to explore which policy to follow to optimise the return for a given set of objective preferences. An experimental evaluation on the Minecart benchmark with frequent as well as sparse changes in utility functions showed that MO-Dreamer significantly outperforms the model-free state-of-the-art algorithms for multi-objective reinforcement learning in the dynamic utility scenario in terms of cumulative regret and average episodic regret. On the Deep Sea Treasure benchmark, MO-Dreamer outperforms the model-free agents overall by converging quickly, but learns a worse final policy.

In future work we intend to study how learned world models can be used for various forms of transfer learning in multi-objective decision making problems. For instance, we would like to study how the world model learned when acting with a linear utility function can be used to transfer to non-linear utility functions. This might require new exploration strategies, to improve the world model in parts of the environment that are relevant for non-linear utility functions but not for linear ones. Future work should also study methods and hyperparameters for training with small datasets while maintaining robustness. Finally, experiments should be conducted on more and harder benchmark problems.

ACKNOWLEDGMENTS

This work was partially supported by the Swedish Governmental Agency for Innovation Systems (grant NFFP7/2017-04885), and the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by the supercomputing resource Berzelius provided by National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation.

REFERENCES

[1] Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. 2019. Dynamic weights in multi-objective deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 11–20.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[4] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*. PMLR, 2052–2062.

[5] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2019. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603* (2019).

[6] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Learning latent dynamics for planning from pixels. In *International conference on machine learning*. PMLR, 2555–2565.

[7] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2020. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193* (2020).

[8] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. 2022. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 1–59.

[9] Conor F Hayes, Mathieu Reymond, Diederik M Roijers, Enda Howley, and Patrick Mannion. 2021. Distributional monte carlo tree search for risk-aware and multi-objective reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 1530–1532.

[10] Johan Källström and Fredrik Heintz. 2019. Tunable dynamics in agent-based simulation using multi-objective reinforcement learning. In *Adaptive and Learning Agents Workshop (ALA-19) at AAMAS, Montreal, Canada, May 13-14, 2019*. 1–7.

[11] 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), 529–533.

[12] Hossam Mossalam, Yannis M Assael, Diederik M Roijers, and Shimon Whiteson. 2016. Multi-objective deep reinforcement learning. *arXiv preprint*

arXiv:1610.02707 (2016).

[13] Sriraam Natarajan and Prasad Tadepalli. 2005. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*. 601–608.

[14] Xiaodong Nian, Athirai A Irissappane, and Diederik Roijers. 2020. DCRAC: Deep conditioned recurrent actor-critic for multi-objective partially observable environments. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 931–938.

[15] David O'Callaghan and Patrick Mannion. 2021. Tunable behaviours in sequential social dilemmas using multi-objective reinforcement learning. In *Proceedings of the 20th international conference on autonomous agents and multiagent systems*. 1610–1612.

[16] Masashi Okada and Tadahiro Taniguchi. 2021. Dreaming: Model-based reinforcement learning by latent imagination without reconstruction. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4209–4215.

[17] Mathieu Reymond, Eugenio Bargiacchi, and Ann Nowé. 2022. Pareto Conditioned Networks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 1110–1118.

[18] 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.

[19] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[20] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

[21] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. 2020. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*. PMLR, 8583–8592.

[22] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[23] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. 2018. Deepmind control suite. *arXiv preprint arXiv:1801.00690* (2018).

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

[25] Peter Vamplew, Benjamin J Smith, Johan Källström, Gabriel Ramos, Roxana Rădulescu, Diederik M Roijers, Conor F Hayes, Fredrik Heintz, Patrick Mannion, Pieter JK Libin, et al. 2022. Scalar reward is not enough: A response to Silver, Singh, Precup and Sutton (2021). *Autonomous Agents and Multi-Agent Systems* 36, 2 (2022), 1–19.

[26] Peter Vamplew, John Yearwood, Richard Dazeley, and Adam Berry. 2008. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *Australasian joint conference on artificial intelligence*. Springer, 372–378.

[27] Shang Wang, Mathieu Reymond, Athirai A Irissappane, and Diederik M Roijers. 2022. Near On-Policy Experience Sampling in Multi-Objective Reinforcement Learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 1756–1758.

[28] Weijia Wang and Michele Sebag. 2012. Multi-objective monte-carlo tree search. In *Asian conference on machine learning*. PMLR, 507–522.

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

[30] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in Neural Information Processing Systems* 32 (2019).