Sample-Efficient Multi-Objective Learning via Generalized Policy Improvement Prioritization

Lucas N. Alegre, Ana L. C. Bazzan
Institute of Informatics - Federal University of Rio Grande do Sul
Porto Alegre - RS, Brazil
{lnalegre,bazzan}@inf.ufrgs.br

Diederik M. Roijers, Ann Nowé
Artificial Intelligence Lab - Vrije Universiteit Brussel
Brussels, Belgium
{diederik.roijers,ann.nowe}@vub.be

Bruno C. da Silva
CICS - University of Massachusetts
Amherst - MA, USA
bsilva@cs.umass.edu

ABSTRACT
Multi-objective reinforcement learning (MORL) algorithms tackle sequential decision problems where agents may have different preferences over (possibly conflicting) reward functions. Such algorithms often learn a set of policies (each optimized for a particular agent preference) that can later be used to solve problems with novel preferences. We introduce a novel algorithm that uses Generalized Policy Improvement (GPI) to define principled, formally-derived prioritization schemes that improve sample-efficient learning. They implement active-learning strategies by which the agent can (i) identify the most promising preferences/objectives to train on at each moment, to more rapidly solve a given MORL problem; and (ii) identify which previous experiences are most relevant when learning a policy for a particular agent preference, via a novel Dyna-style MORL method. We prove our algorithm is guaranteed to always converge to an optimal solution in a finite number of steps, or an $\epsilon$-optimal solution (for a bounded $\epsilon$) if the agent is limited and can only identify possibly sub-optimal policies. We also prove that our method monotonically improves the quality of its partial solutions while learning. Finally, we introduce a bound that characterizes the maximum utility loss with respect to the optimal solution incurred by the partial solutions computed by our method throughout learning. We empirically show that our method outperforms state-of-the-art MORL algorithms in challenging multi-objective tasks, both with discrete and continuous state spaces.

KEYWORDS
Multi-Objective, Model-based RL, Generalized Policy Improvement

ACM Reference Format:

1 INTRODUCTION
Reinforcement learning (RL) algorithms [43] have achieved remarkable successes in a wide range of complex tasks (e.g., [10, 41, 49]). In RL, tasks are typically modeled via a reward function. An example, consider a robot that needs to trade off between locomotion speed, battery usage, and accuracy in reaching a goal location. Multi-Objective RL (MORL) [21] algorithms tackle such a challenge. In MORL settings, a utility function exists that combines (into a single scalar value) the agent’s preferences toward optimizing each of its objectives. The goal of MORL algorithms is to rapidly identify policies that maximize the utility function, given any preferences an agent may have.

Standard RL algorithms are often sample-inefficient since they may require the agent to interact with its environment a large number of times [43]. This is often infeasible, e.g., in applications where interactions are costly or risky. The sample efficiency problem is exacerbated in MORL algorithms, since they often have to learn not a single policy, but a set of policies—each designed to optimize a particular trade-off between the agent’s preferences [52].

As we see, our novel MORL algorithm, with important theoretical guarantees, that improves sample efficiency via two novel prioritization techniques. Such principled techniques allow the agent to (i) identify the most promising preferences/objectives to train on at each moment, to more rapidly solve a MORL problem; and (ii) identify which previous experiences are most relevant when learning a policy for a particular agent preference, via a novel Dyna-style MORL method. We show formal analyses and proofs regarding our algorithm’s convergence properties, as well as the maximum utility loss (performance) incurred by the transient solutions computed by our method throughout the learning process.

To design a method with these properties, we exploit an important concept in the RL literature: Generalized Policy Improvement (GPI) [9]. GPI is a policy transfer technique that—given a set of policies—constructs a novel policy guaranteed to be at least as good as any of the available ones. First, recall that if the utility function of a MORL problem is a linear combination of the agent’s objectives, optimal solutions are sets of policies known as convex coverage sets (CCS) [38]. Given a CCS, agents can directly identify the optimal solution to any novel preferences. MORL algorithms that learn a CCS (e.g., [5, 29]) may be sample inefficient (i) due to the heuristics they use to determine which preferences to train on, at any given moment during the construction of a CCS; and (ii) because they can only improve a CCS after optimal (or near-optimal) policies are identified—which may require a large number of samples. We address the first issue via a novel GPI-based prioritization technique for selecting which preferences to train on. It is based on a lower bound on performance improvements that are formally guaranteed to be achievable, and more accurately and reliably identifies the most relevant preferences to train on (when constructing a CCS) compared to existing heuristics. To address the
second issue, we show that our method is an anytime algorithm that incrementally/monotonically improves the quality of its CCS, even if given intermediate (possibly sub-optimal) policies for different preferences. This improves sample efficiency: our method identifies intermediate CCSs with formally bounded maximum utility loss even if there are constraints on the number of times the agent can interact with its environment. Our algorithm is guaranteed to always converge to an optimal solution in a finite number of steps, or an $\varepsilon$-optimal solution (for a bounded $\varepsilon$) if the agent is limited and can only identify possibly sub-optimal policies. We prove that it monotonically improves the quality of its CCS and formally bound the maximum utility loss (with respect to an optimal solution) incurred by any partial solutions computed by our method.

A complementary approach for increasing sample efficiency is to use a model-based approach to learn policies for different preferences. Once a model is learned, it can be used to identify policies for any preferences, thus minimizing the required number of interactions with the environment. We introduce a novel Dyna-style MORL algorithm—the first model-based MORL technique capable of dealing with continuous state spaces. Dyna algorithms are based on generating simulated experiences to more rapidly update a value function or policy. An important question, however, is which artificial experiences should be generated to accelerate learning. We introduce a new, principled GPI-based prioritization technique for identifying which experiences are most relevant to rapidly learn the optimal policy for novel preferences.

In summary, we introduce two novel GPI-based prioritization techniques for use in MORL settings to improve sample efficiency. We formally show that our algorithm is supported by important theorems characterizing its convergence properties and performance bounds. We empirically show that our method outperforms state-of-the-art MORL algorithms in challenging multi-objective tasks, both with discrete and continuous state and action spaces.

2 BACKGROUND

In this section, we discuss important definitions (and corresponding notation) associated with MORL, GPI, and model-based RL.

2.1 Multi-Objective Reinforcement Learning

The multi-objective RL setting (MORL) is used to model problems where an agent needs to optimize possibly conflicting objectives, each modeled via a separate reward function. MORL problems are modeled as Multi-objective Markov decision processes (MOMDP), which differ from regular MDPs in that the reward function is vector-valued. A MOMDP is defined as a tuple $(S, A, \pi, p, \gamma, m)$, where $S$ is a state space, $A$ is an action space, $p(\cdot|s, a)$ is the distribution over next states given state $s$ and action $a$, $r: S \times A \times S \rightarrow \mathbb{R}^m$ is a multi-objective reward function with $m$ objectives, $\mu$ is an initial state distribution, and $\gamma \in [0, 1)$ is a discounting factor. Let $S_t, A_t, R_t = r(S_t, A_t, S_{t+1})$ be random variables corresponding to the state, action, and vector reward, respectively, at time step $t$. A policy $\pi: S \rightarrow A$ is a function mapping states to actions. The multi-objective action-value function of a policy $\pi$ for a given state-action pair $(s, a)$ is defined as:

$$q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_t = s, A_t = a \right],$$

where $q^\pi(s, a)$ is an $m$-dimensional vector whose $i$-th entry is the expect return of $\pi$ under the $i$-th objective, and $\mathbb{E}_\pi[\cdot]$ denotes expectation over trajectories induced by $\pi$. Let $v^\pi \in \mathbb{R}^m$ be the multi-objective value vector of $\pi$ under the initial state distribution $\mu$:

$$v^\pi = \mathbb{E}_{S_0 \sim \mu} \left[ q^\pi(S_0, \pi(S_0)) \right],$$

where $v^\pi$ is the value of $\pi$ under the $i$-th objective. For succinctness, we henceforth refer to $v^\pi$ as the value vector of policy $\pi$. A Pareto frontier is a set of nondominated multi-objective value functions $v^\pi$: $F \equiv \{v^\pi \mid \exists \pi' \text{s.t. } v^\pi' \succ v^\pi\}$, where $\succ$ is the Pareto dominance relation $v^\pi' \succ v^\pi \iff \left( \forall i : v^\pi_i \geq v^\pi'_i \land \exists i : v^\pi_i > v^\pi'_i \right)$. In general, the optimal solution to a MOMDP is a set of all policies $\pi$ such that $v^\pi$ is in the Pareto frontier.

Let a user utility function (or scalarization function) $u: \mathbb{R}^m \mapsto \mathbb{R}$ be a mapping from the multi-objective value of policy $\pi$, $v^\pi$, to a scalar. Utility functions often linearly combine the value of a policy under each of the $m$ objectives using a set of weights $w: u(v^\pi, w) = v^w_0 = v^\pi \cdot w$, where each element of $w \in \mathbb{R}^m$ specifies the relative importance of each objective. The space of weight vectors, $W^m$, is an $m$-dimensional simplex: $\sum w_i = 1, w_i \geq 0, i = 1, \ldots, m$. For any given constant $w$ (i.e., a particular way of weighting objectives), the original MOMDP collapses into an MDP with reward function $r_w(s, a') = r(s, a') \cdot w$. Given a linear utility function $u$, we can define a convex coverage set (CCS) [38] as a finite convex subset of $F$, such that there exists a policy in the set that is optimal with respect to any linear preference $w$. In other words, the CCS is the set of nondominated multi-objective value functions $v^\pi$ where the dominance relation is now defined over scalarized values:

$$\text{CCS} \equiv \{v^\pi \in F \mid \exists w \text{ s.t. } \forall v^\pi \in F, v^\pi \cdot w \geq v^\pi' \cdot w\},$$

where $w$ is a vector of weights used by the scalarization function $u$. Hence, the optimal solution to a MOMDP, under linear preferences, is a finite convex subset of the Pareto frontier.

2.2 Generalized Policy Improvement

Generalized Policy Improvement (GPI) is a generalization of the policy improvement step [36], which underlies most of the RL algorithms. The key difference is that GPI defines a new policy that improves over a set of policies, instead of a single one. GPI was originally proposed to be employed in the setting where policies are evaluated using their successor features [8, 9]. However, Alegre et al. [5] showed that GPI can also be employed with multi-objective action-value functions (Eq. (1)).

Let $\Pi = \{\pi_i\}_{i=1}^n$ be a set of previously-learned policies with corresponding multi-objective action-value functions $(q^\pi_i)^n_{i=1}$. Given any weight vector $w \in W$, we can directly evaluate all policies $\pi_i \in \Pi$ by computing the dot product of its multi-objective action-values and the weight vector: $q^w_0(s, a) = q^w(s, a) \cdot w$. This step is known as generalised policy evaluation (GPE) [9]. We define the GPI policy as a policy $\pi : S \times W \mapsto A$ that is constructed from a set of policies $\Pi$, and is conditioned on a weight vector $w \in W$:

$$\pi_{\text{GPI}}(s, w) \in \arg \max_{a \in A} \max_{\pi \in \Pi} q^w_0(s, a).$$
Let \( q_{\Pi}^{\text{GPI}}(s, a) \) be the action-value function of policy \( \pi_{\Pi}^{\text{GPI}}(\cdot; w) \). The GPI theorem [8] ensures that \( q_{\Pi}^{\text{GPI}}(s, a) \geq \max_{x \in \Pi} q_{x}(s, a) \) for all \((s, a) \in S \times A\). In other words, GPI allows for the rapid identification of a policy that is guaranteed to perform at least as well as any of the policies \( \pi_{i} \in \Pi \), for any given weight vector \( w \in W \). This result is also valid and can be extended to the scenario where \( q_{\pi_{i}} \) is replaced with estimates/approximations, \( \tilde{Q}_{\pi_{i}} \) [7].

### 2.3 Model-Based RL

In model-based RL [27], an agent learns—which is to say experiences collected while interacting with the environment—an approximate model \( p(s', r|s, a) \) of the joint distribution over the next states and rewards. This model can be employed in multiple ways, such as:

1. To perform Dyna-style planning, i.e., generate simulated experiences/weight vectors that approximate a CCS (Eq. (3)). In this Section, we first introduce a method to rapidly learn a set of policies, \( \Pi \), whose value vectors, \( V \), approximate the CCS. At each iteration, it selects a weight vector \( w \in W \) based on the formally-guaranteed improvements achievable via GPI, and learns a new policy \( \pi_{w} \) specialized in optimizing \( w \).

#### Algorithm 1: GPI Linear Support (GPI-LS)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \Pi \leftarrow { \pi_{w} }, V \leftarrow { v^{w} }, M \leftarrow { } )</td>
</tr>
<tr>
<td>2</td>
<td>while True do</td>
</tr>
<tr>
<td>3</td>
<td>( \Pi \leftarrow \Pi \cup { \pi_{w} }, V \leftarrow { v^{w} }, M \leftarrow { } )</td>
</tr>
<tr>
<td>4</td>
<td>( \Pi \leftarrow \Pi \cup { \pi_{w} }, V \leftarrow { v^{w} } )</td>
</tr>
<tr>
<td>5</td>
<td>if ( W_{\text{corner}} ) is empty then</td>
</tr>
<tr>
<td>6</td>
<td>( \Pi \leftarrow \Pi \cup { \pi_{w} }, V \leftarrow { v^{w} } )</td>
</tr>
<tr>
<td>7</td>
<td>( \pi_{w} \leftarrow \text{NewPolicy}(w, \Pi) )</td>
</tr>
<tr>
<td>8</td>
<td>( \Pi \leftarrow \Pi \cup { \pi_{w} }, V \leftarrow { v^{w} } )</td>
</tr>
<tr>
<td>9</td>
<td>if done then</td>
</tr>
<tr>
<td>10</td>
<td>( \Pi \leftarrow \Pi \cup { \pi_{w} }, V \leftarrow { v^{w} } )</td>
</tr>
</tbody>
</table>

We start by defining the corner weights [37] of a given set of value vectors, \( V \). Corner weights \( W_{\text{corner}} \subset W \) are a finite subset of the weight vector space with relevant properties we will explore.

**Definition 3.1.** Let \( V = \{ v^{w} \}_{w \in W} \) be a set of multi-objective value functions of \( n \) policies. Corner weights are the weights contained in the vertices of a polyhedron, \( P \), defined as:

\[
P = \{ x \in \mathbb{R}^{d+1} | V^{x} \mathbf{x} \leq 0, \sum_{i} w_{i} = 1, w_{i} \geq 0, \forall i \},
\]

where \( V^{x} \) is a matrix whose rows store the elements of \( V \) and is augmented by a column vector of \(-1\)'s. Each vector \( x = (w_{1}, ..., w_{d}, v^{w}) \) in \( P \) is a composed of a weight vector and its scalarized value.

Intuitively, corner weights are the weight vectors for which the policy selected in the maximization \( \max_{x \in \Pi} v^{w} \) changes. These are weight vectors for which two or more policies in \( \Pi \) share the same value with respect to the above-mentioned maximization. Extrema weights (weights where only one element is 1 and all others are 0) are special cases of corner weights. Notice that the number of vertices of the polyhedron \( P \) is finite when \( V \) is finite. The importance of corner weights comes from the following theorem:

**Theorem 3.2.** (Theorem 7 of Roijers [37]). Let \( \Pi = \{ \pi_{i} \}_{i=1}^{n} \) be a set of \( n \) policies with corresponding value vectors \( V = \{ v^{\pi_{i}} \}_{i=1}^{n} \). Let \( \Delta_{w}(w, \Pi) = v^{w} - \max_{x \in \Pi} v^{x} \) be the utility loss of weight vector \( w \in W \) given the policy set \( \Pi \); that is, the difference between the value of the optimal policy for \( w \) and the value that can be obtained if using one of the policies in \( \Pi \) for solving \( w \). Then, a weight vector \( w \in \arg \max_{w \in W} \Delta_{w}(w, \Pi) \) is one of the corner weights of \( V \).
Due to this theorem, when selecting a weight vector to train on when constructing a CCS, we only need to consider the finite set of corner weights, \( W_{\text{corner}} \), instead of the infinite weight simplex \( W \). However, this does tell us how to select a weight vector (among all the corner weights) to more rapidly learn the CCS. Let \( w_{\text{GPI}} \) be the scalarized value of the GPI policy (Eq. 4) for the weight vector \( w \). We propose to prioritize weight vectors based on the magnitude of the improvement that can be formally achieved via GPI. In particular, given the corner weights \( W_{\text{corner}} \) of the current value set, \( V \), we select the weight vector \( w \in W_{\text{corner}} \) given by:

\[
\arg \max_{w \in W_{\text{corner}}} (w_{\text{GPI}} - w^\pi).
\] (7)

Intuitively, Eq. (7) identifies the corner weight for which we are guaranteed to achieve the maximum possible improvement via GPI.

In Alg. 1 we show an iterative algorithm, GPI Linear Support (GPI-LS), that constructs a CCS in a finite number of iterations based on the above-mentioned ideas.\(^1\) Let NewPolicy(\( w, \Pi \)) be any RL algorithm that searches for a policy that optimizes a weight vector \( w \in W \), starting from an initial candidate policy \( \pi_w \) set to \( \arg \max_{w \in W_{\text{corner}}} w_{\text{GPI}} \). We assume this algorithm returns the value vector of this policy, \( v_w \bullet \), and a flag (done) indicating if it has reached its stopping criterion. When this happens, it adds the corresponding new policy to \( \Pi \) and removes all dominated policies; i.e., policies whose value vectors are no longer the best for any weight vector. To analyze this algorithm, we first consider the best-case scenario where the NewPolicy(\( w, \Pi \)) is capable of identifying optimal policies for any given weight vectors. Later, we relax this assumption and show that our algorithm’s theoretical guarantees still hold even if that is not the case. We show that it holds strong theoretical guarantees bounded only by the sub-optimality of the underlying RL algorithm chosen by the user.

**Theorem 3.3.** Let NewPolicy(\( w, \Pi \)) in Alg. 1 be any algorithm that returns an optimal policy, \( \pi_w^* \), for a given weight vector \( w \). Then, Alg. 1 is guaranteed to find a CCS in a finite number of iterations.

**Proof.** The proof follows from the fact that at each iteration, the set of corner weights, \( W_{\text{corner}} \), is finite since the number of the vertices of the polyhedron \( P \) is finite (Eq. (6)). Once the termination condition of NewPolicy(\( w, \Pi \)) is met (done = True), it returns the optimal policy for a given \( w \in W_{\text{corner}} \), \( w \) is added to \( M \), and never selected again. This is ensured because the set of candidate corner weights, \( W_{\text{corner}} \), computed by GPI-LS in any given iteration, does not (by construction) include previously- evaluated corner weights. Since \( S \) and \( A \) are finite, the number of potential corner weights is bounded by the (finite) number of vertices of the polyhedron \( P \) defined with respect to the set \( V \) containing all \( |A|^{|S|} \) possible policies. Because (i) GPI-LS selects a corner weight at each iteration; (ii) NewPolicy(\( w, \Pi \)) returns an optimal policy for \( w \), which is added to \( M \) and never selected again; and (iii) there exists a finite number of corner weights, it follows that in the worst-case, GPI-LS will test all (finite) corner weights. Thus, after a finite number of iterations, all corner weights of the CCS will be in \( M \). At this point, \( W_{\text{corner}} \) will be empty and the algorithm will return \( M \) and \( V \) (lines 5–6). We can ensure that the returned set \( V \) is a CCS due to Theorem 3.2. When there are no more corner weights to be analyzed (\( W_{\text{corner}} \) is empty), \( M \) contains all corner weights and their corresponding optimal policies. By Theorem 3.2, this means that it is not possible to make improvements to any weight vector \( w \in W \) and thus \( V \) (and its corresponding policies \( \Pi \)) is a CCS. \( \square \)

We now relax the assumption that NewPolicy finds optimal solutions and show that if NewPolicy(\( w, \Pi \)) converges to a local-minimum (an \( \varepsilon \)-optimal policy), GPI-LS returns an \( \varepsilon \)-CCS.\(^2\)

**Definition 3.4.** A set of value vectors \( V = \{ v_i \}_{i=1}^n \) (associated with policy set \( \Pi = \{ \pi_i \}_{i=1}^n \)) is an \( \varepsilon \)-CCS if, for all weight vectors \( w \in W \), the corresponding utility loss \( \Delta(w, \Pi) \) is at most \( \varepsilon \):

\[
\max_{w \in W} \Delta(w, \Pi) = \max_{w \in W} (v_w^* - v_w^\pi) \leq \varepsilon.
\]

Intuitively, an \( \varepsilon \)-CCS is a convex coverage set whose maximum utility loss is at most \( \varepsilon \); i.e., based on it, for any possible weight vectors \( w \in W \), it is possible to identify a policy whose value differs from the value of the optimal policy for \( w \) by at most \( \varepsilon \).

**Theorem 3.5.** Let NewPolicy(\( w, \Pi \)) in Alg. 1 be an algorithm that produces an \( \varepsilon \)-optimal policy, \( \pi_w \), when its termination condition is met (when it returns done = True); that is, \( v_w^* - v_w^\pi \leq \varepsilon \). Then, Alg. 1 is guaranteed to return an \( \varepsilon \)-CCS.

**Proof.** Let \( \Pi_i \) be the set of policies computed by GPI-LS at iteration \( i \) and \( V_i \) be the corresponding set of value functions. Let \( \Delta_i = \max_{w \in W} \Delta(w, \Pi_i) \) be the maximum utility loss incurred by the partial CCS computed by GPI-LS at iteration \( i \). If \( \Delta_i \leq \varepsilon \), then by definition the set of value vectors \( V_i \) computed by GPI-LS at iteration \( i \), is an \( \varepsilon \)-CCS. If not, then by Theorem 3.2 there exists a corner weight \( w' \) such that \( \Delta(w', \Pi_i) = \Delta_i \). Because GPI-LS will, in the worst-case, test all corner weights and NewPolicy returns \( \varepsilon \)-optimal policies (where \( \varepsilon < \Delta_i \)), then at some iteration \( j > i \), GPI-LS will select \( w' \), compute its optimal policy \( \pi_{w'} \) and add it to \( \Pi_j \). Because by definition \( \Delta(w', \Pi_j) \leq \varepsilon \), it follows that \( \Delta_j < \Delta_i \); that is, the maximum utility loss incurred by the partial CCS can be strictly decreased. Thus, within a finite number of iterations, \( k \), we can guarantee that \( \Delta_{i+k} \leq \varepsilon \). At this point, by definition, an \( \varepsilon \)-CCS has been identified. \( \square \)

Finally, we introduce a formal bound that characterizes the maximum utility loss (with respect to the optimal solution) incurred by any partial solutions computed by our method, prior to convergence, in case the agent follows a GPI policy based on the partially- computed CCS produced by our algorithm at an arbitrary iteration.\(^3\)

**Theorem 3.6.** Let \( V = \{ v_i \}_{i=1}^n \) be a set of value vectors with corresponding policies \( \Pi = \{ \pi_i \}_{i=1}^n \) optimizing weight vectors \( M = \{ w_i \}_{i=1}^n \). Let \( q_{w_i}^s(a, s) - q_{w_i}^\pi(s, a) \leq \delta \) for all \( (s, a) \in S \times A \), and \( i \in \{ 1, ..., n \} \). Let \( r_{\text{max}} = \max_{s, a} |r(s, a)| \). Then, for any \( w \in W \),

\[
q_{w}^s(a, s) - q_{w}^\text{GPI}(s, a) \leq \frac{2}{1 - \gamma} \left( r_{\text{max}} \min_i ||w - w_i|| + \delta \right). (8)
\]

\(^1\)A key challenge in proving convergence and bounded utility loss of our algorithm is that, unlike existing techniques, it does not require optimal (or \( \varepsilon \)-optimal) policies to be returned at every iteration. Instead, it can incrementally (and monotonically) improve its CCS’s quality even when given partially learned, possibly suboptimal policies.

\(^2\)Notice that when the CCS is infinite (e.g., there is an infinite number of optimal policies), GPI-LS still converges to a CCS asymptotically/in the limit.

\(^3\)This result is a straightforward adaptation of Theorem 2 of Barreto et al. [8] to the setting where we wish to bound the maximum utility loss of a partially-computed CCS.
This theorem guarantees that as the coverage of the weight vector set, $\mathcal{W}$, increases with respect to the simplex, the maximum utility loss of Alg. 1 is strictly bounded at any given iteration during learning.

### 3.2 Prioritizing Experiences via GPI

A complementary approach for increasing sample efficiency is to use a model-based RL algorithm to learn policies for different preferences. In this paper, we introduce an experience prioritization technique based on a Dyna-style MORL algorithm (introduced and discussed later). An important open question related to efficiently deploying Dyna algorithms is to determine which artificial, model-simulated experiences should be generated to accelerate learning.

We introduce a new, principled GPI-based prioritization technique (based on Theorem 3.7) for identifying which experiences are most relevant to rapidly learn the optimal policy, $\pi_w$, that optimizes a preference $w \in \mathcal{W}$. Importantly, unlike similar existing methods (e.g., Prioritized Experience Replay [39]), our method is specifically designed to accelerate learning in MORL problems.

We start by introducing a theorem based on which we will be able to define a principled experience prioritization scheme for accelerating learning in MORL problems.

**Theorem 3.7.** Let $\Pi$ be an arbitrary set of policies, and let $\pi_w \in \Pi$ be a deterministic policy tasked with optimizing some $w \in \mathcal{W}$. Then, $q^{\text{GPI}}_w(s, a) = q^*_w(s, a)$ for all state-action pairs in $\mathcal{S} \times \mathcal{A}$ if and only if $q^*_w(s, a) = q^*_{\Pi}(s, a)$. In other words, $\pi_w$ is guaranteed to be an optimal policy for $w$ iff the GPI policy computed over $\Pi$, for optimizing $w$, cannot improve the $q$-function of $\pi_w$ for any state-action pairs. *

As a result of this theorem, it follows that to rapidly learn an optimal policy, we may wish to bring the value of $q^*_{\Pi}(s, a)$ closer to $q^*_w(s, a)$: when they are equal in all state-action pairs, we have found $q^*_{\Pi}(s, a) = q^*_w(s, a)$. State-action pairs whose value gap, $d^*_w = q^*_w(s, a) - q^*_w(s, a)$, is maximal (or large) are pairs that, if updated, more rapidly approximate $q^*_{\Pi}(s, a)$ and $q^*_{\Pi}(s, a)$ (in terms of Max-Norm distance). Intuitively, these are promising candidate experiences to be sampled from the model and used in updates to improve policy $\pi_w$. By decreasing the above-mentioned gap, we are guaranteed to move $q^*_w(s, a)$ closer to $q^*_w(s, a)$. When this gap is zero for all state-action pairs, we are guaranteed to have identified an optimal policy for $w$. Based on these observations, which follow from Theorem 3.7, we propose (during the process of learning $\pi_w$) to prioritize experiences proportionally to the magnitude of the value improvement resulting from using a GPI policy. That is, for a given weight vector $w \in \mathcal{W}$ and state-action pair, we assign corresponding priorities $P_w(s, a)$ proportionally to the gap $q^{\text{GPI}}_w(s, a) - q^*_w(s, a)$:

$$P_w(s, a) \propto q^{\text{GPI}}_w(s, a) - q^*_w(s, a).$$

(9)

Notice that in practice we do not have direct access to the action-values of the GPI policy, $q^{\text{GPI}}_w(s, a)$, unless we evaluate it beforehand with a policy evaluation algorithm (which may be costly). We can, however, efficiently compute the value of employing GPI for a single step. Let $q^{\text{GPI}}(S_t, A_t)$ be defined as the value of executing action $A_t$ in state $S_t$, following the GPI policy for one step, and thereafter following the same policy $\pi \in \Pi$ that was used in this first step: $q^{\text{GPI}}(S_t, A_t) = \mathbb{E}[R_t \cdot \mathbf{w} + \gamma \max_{a' \in \mathcal{A}} \max_{\pi \in \Pi} q^*_w(S_{t+1}, a')]$. It is possible to show that $q^*_w(s, a) \geq q^{\text{GPI}}_w(s, a) \geq q^*_w(s, a)$. Importantly, Theorem 3.7 still holds if we replace $q^*_w(s, a)$ with $q^{\text{GPI}}_w(s, a)$. Given a transition $(S_t, A_t, R_t, S_{t+1})$ and a weight vector $w \in \mathcal{W}$, we thus compute experience priorities as follows:

$$P_w(S_t, A_t) \propto |R_t \cdot w + \gamma \max_{a' \in \mathcal{A}} \max_{\pi \in \Pi} q^*_w(S_{t+1}, a') - q^*_w(S_t, A_t)|.$$  

(10)

Notice that when $|\Pi| = 1$, that is, when we only have one policy, Eq. (10) reduces to the commonly-used Prioritized Experience Replay prioritization scheme (Eq. (5)). Hence, we can see Eq. (10) as a form of generalized TD-error-based prioritization.

### 4 GPI-PRIORITYED DYNA

We now introduce GPI Prioritized Dyna (GPI-PD), a novel model-based MORL algorithm that improves sample efficiency by simultaneously using GPI-LS (Section 3.1) to prioritize and select weight vectors to more rapidly construct a CCS, and our GPI-based experience prioritization scheme (Section 3.2) to more rapidly learn policies for a given preference. GPI-PD learns an approximate multi-objective dynamics model, $\rho$, predicting the next state and reward vector given a state-action pair. This model is used to perform Dyna updates to multi-objective action-value functions via model-generated, simulated experiences. Its pseudocode is shown in Alg. 2.

---

*The proof of this theorem can be found in the Appendix.
Notice that GPI-PD is a MORL algorithm designed to deal with high-dimensional and continuous state spaces; it models the action-value function via a single neural network ($Q_\theta$) conditioned on state and weight vector, as in [3, 52]. The multi-objective action-value function is defined as $Q_\theta(s, a, w) \approx q^{w_s}(s, a)$, where $\theta$ are the learned neural network parameters. Similarly to [11], we now rewrite the GPI policy definition (Eq. 4) so that the GPI policy can be defined with respect to an approximator:

$$
\pi_{GPI}(s; w) \in \arg \max \max_{a \in A} \max_{w' \in \mathcal{M}} Q_\theta(s, a, w') \cdot w,
$$

where $\mathcal{M}$ is the weight support set containing the corner weights selected by GPI-LS (lines 4–8 of Alg. 2). Notice that we add to $\mathcal{M}$ the top-$k$ weight vectors, according to Eq. (7) (line 8). This implies that each top-performing weight will be optimized during a corresponding episode. We found that this accelerates learning in the function approximation setting. Finally, the parameters $\theta$ of $Q_\theta$ are updated by minimizing the mean-squared error of the multi-objective TD error:

$$
\mathcal{L}(\theta; w) = \mathbb{E}_{(S, A, R, S', T, w, S, A, w') \sim \mathcal{B}} [(R + y \max_{a' \in A} Q_\theta(S', a', w) - Q_\theta(S, A, w))^2],
$$

where $a' = \arg \max_{a' \in A} Q_\theta(S', a', w)$ and $\gamma$ are the parameters of a target neural network, which are updated to match $\theta$ periodically. In line 22 of Alg. 2, we optimize $\mathcal{L}(\theta; w)$ both for the current episode’s weight vector, $w_t$, and for weight vectors $w' \sim \mathcal{M}$ sampled from the weight support set.3 Training in this manner is known to avoid catastrophic forgetting [3].

### 4.1 Dyna with a Learned MOMP Model

GPI-PD learns a model $p_\theta$ of the environment’s dynamics and its multi-objective reward function. In particular, it represents the joint distribution, $p_\theta(S_{t+1}, R_t | S_t, A_t)$, of the next state and the complete multi-objective reward vector, $R_t$, when conditioned on state and action. Notice that this model may be used to accelerate learning of policies for any $w \in \mathcal{W}$.

Before proceeding, we emphasize that GPI-PD is designed to tackle high-dimensional MORL problems with continuous state spaces. To achieve this, we extend state-of-the-art model-based algorithms used in the single-objective setting [14, 23, 25]. We achieved this by employing an ensemble of probabilistic neural networks for predicting the reward vector, $R_t$, instead of a scalar reward. The learned model $p_\theta$ is composed of an ensemble of $n$ neural networks, $\{p_{\theta_i}\}_{i=1}^n$, each of which outputs the mean and diagonal covariance matrix of a multivariate Gaussian distribution:

$$
p_{\theta_i}(S_{t+1}, R_t | S_t, A_t) = \mathcal{N}(\mu_{\theta_i}(S_t, A_t), \Sigma_{\theta_i}(S_t, A_t)).
$$

Each model in the ensemble is trained in parallel to minimize the following negative log-likelihood loss function, using different bootstraps of experiences in the buffer $\mathcal{B}$:

$$
\mathcal{L}(\varphi) = \mathbb{E}_{(S_t, A_t, R_t, S_{t+1}) \sim \mathcal{B}} [-\log p_{\theta_i}(S_{t+1}, R_t | S_t, A_t)].
$$

In lines 16–19 of Alg. 2, GPI-PD samples states according to $p_{\theta_i}(\cdot, A_t)$, where $A_t$ is the action given by the GPI policy. It then generates corresponding model-based simulated experiences for this state-action pair and stores them in the buffer $\mathcal{B}_{\text{model}}$. We combine these experiences with “real” ones, collected by the agent while interacting with the environment (and stored in the buffer $\mathcal{B}$), according to a mixing ratio $\beta$. This results in a mini-batch of experiences that are used to optimize $Q_\theta$. Combining experiences in this manner is a commonly-used approach when training model-based RL algorithms [23, 31].

### 5 EXPERIMENTS

**Figure 1:** (a) Deep Sea Treasure; (b) Minecart; (c) MO-Hopper.

We evaluate the performance of our algorithms in challenging multi-objective tasks—both with discrete and continuous state and action spaces—and compare it with state-of-the-art MORL algorithms. We consider three environments [6] with qualitatively different characteristics (see Fig. 1). We follow the performance evaluation methodology proposed in [21, 53], which focuses on utility-based metrics. These are relevant since they more directly reflect an algorithm’s performance with respect to a user’s preferences. Concretely, when evaluating a given algorithm, we compute the expected utility of its solutions: $EU(\Pi) = \mathbb{E}_{w \sim \mathcal{W}} \max_{\pi \in \Pi} \pi^w$. We also quantify the maximum utility loss of an algorithm: $MUL(\Pi) = \mathbb{E}_{w \sim \mathcal{W}} (\pi^w - \pi_*^w)$. We report the mean and its 95% confidence interval over multiple random seeds.

**Deep Sea Treasure.** This is a classic MORL domain in which a submarine has to trade-off between collecting treasures and minimizing a time penalty (Fig. 1a). When evaluating our method in this setting, we employed a simplified, tabular version of GPI-PD (see Appendix). Recall that our GPI-LS algorithm (Alg. 1), unlike GPI-PD (Alg. 2), is model-free, does not perform Dyna planning steps, and uses a standard, unprioritized experience replay buffer with uniform sampling. We compare GPI-PD with two state-of-the-art competitors: Envelope MOQ-Learning [52], and SFOLS [5] (which uses OLS [29, 37] for selecting which weight vectors to train on).

In Fig. 2, we evaluate each algorithm’s expected utility (EU) and maximum utility (MUL) both (i) if each algorithm—when optimizing a given preference—follows its best known policy (two leftmost plots); and (ii) if each algorithm—when optimizing a given preference—follows the GPI policy derived from the known policies (two rightmost plots). We consider a setting where agents are constrained and have a limited budget of 4,000 learning time steps per iteration. We now make a few important empirical observations, which hold for both above-mentioned settings (i and ii):

- The performance of the Envelope algorithm (both in terms of EU and MUL) has high variance since it trains on randomly-sampled experiences with “real” ones, collected by the agent while interacting with the environment (and stored in the buffer $\mathcal{B}$), according to a mixing ratio $\beta$. This results in a mini-batch of experiences that are used to optimize $Q_\theta$. Combining experiences in this manner is a commonly-used approach when training model-based RL algorithms [23, 31].

---

3We abuse notation and use the operator $\sim$ both to denote sampling from a given distribution, but also, when writing $x \sim Y$ (where $Y$ is a set and $x \in Y$) to refer to the process of sampling an element $x$ uniformly at random from the set $Y$.

4We approximate the expectation $\mathbb{E}_{w \sim \mathcal{W}} [\cdot]$ by averaging over 100 equidistant weight vectors covering the simplex $\mathcal{W}$.

5When following the GPI policy, EU and MUL can be written as $EU(\Pi) = \mathbb{E}_{w \sim \mathcal{W}} [\pi_*^w]$ and $MUL(\Pi) = \max_{w \sim \mathcal{W}} (\pi_*^w - \pi^w)$, respectively.

6All algorithms’ hyperparameters are presented in the Appendix.
weight vectors at each iteration. Furthermore, its asymptotic performance is always worse than our algorithms’ performances.

- SFOLS converges to a suboptimal CCS: it consistently adds suboptimal policies (the best ones it could discover under the training time constraints) to its CCS. As a result, SFOLS incorrectly prioritizes preferences.

- GPI-LS, by contrast, is consistently capable of identifying the most promising weight vectors, i.e., those with the highest guaranteed improvement according to Eq. (7). This allows it to always rapidly reach near-zero maximum utility loss. These results are consistent with Theorems 3.3 and 3.5, which formally ensure that GPI-LS always converges to optimal solutions and that it monotonically improves the quality of its solutions over time (up to approximation errors due to experimental statistics having been computed based on finite data points). When GPI-LS also employs experience prioritization (that is, when combined with GPI-PD), the resulting method produces even higher improvements at each iteration, thereby enhancing overall sample efficiency.

**Minecart.** This domain involves controlling a cart (Fig. 1b) tasked with reaching ore mines, collecting and selling ores, and minimizing fuel consumption [3]. There are three conflicting objectives, representing the agent’s preferences for collecting each of the two types of ores, and for saving fuel. This is a continuous state space problem, and so we can directly deploy GPI-PD (Alg. 2); GPI-PD uses GPI both to prioritize weight vectors and experiences (Section 4.1). When comparing it with SFOLS and Envelope, we ensure that such competitors use the same neural network architecture and hyperparameters as GPI-PD, to allow for a fair comparison.

In Fig. 3, the performance of all algorithms follows a qualitatively similar behavior as observed in the Deep Sea Treasure experiment. Our methods (GPI-LS, and GPI-LS combined with GPI-PD) consistently identify optimal solutions, reach near-zero maximum utility loss, and achieve performance metrics that strictly dominate that of competitors. The Envelope algorithm is the least sample-efficient method during the first ten iterations—although it slightly surpasses the expected utility of SFOLS at the end of the learning process. These results further emphasize the importance of our weight prioritization technique (Section 3.1), which allows GPI-LS to allocate its training time to preferences whose performances are guaranteed to be improvable. As a result, GPI-LS outperforms the competing algorithms in terms of sample efficiency, and always converges to solutions with higher EU and MUL. Similarly, GPI-PD consistently reaches near-zero maximum utility loss.

**MO-Hopper.** This domain involves controlling a one-legged agent tasked with learning to hop while balancing two objectives: maximizing forward speed and jumping height. Unlike the previously-discussed domains, MO-Hopper operates over a continuous-action space. Neither Envelope nor SFOLS can tackle such a setting. For this reason, we compared our algorithms with a state-of-the-art MORL technique capable of dealing with continuous actions: PG-MORL [50]. This is an evolutionary algorithm that assigns weight vectors to a population of agents (six, in our experiments, as in [50]). Agents are trained in parallel using PPO [40]. We extended our methods—both GPI-PD and GPI-LS—to deal with continuous actions by adapting the TD3 algorithm [18] to the MORL setting (see the Appendix). In Fig. 4, we compare the expected utility and Pareto frontiers generated by GPI-PD and PG-MORL. Here, we analyze Pareto frontiers because we cannot compute the maximum utility loss exactly, since the optimal CCS is unknown. Pareto frontiers were computed by selecting all Pareto-efficient policies among those collected during 20 runs of the algorithms, and evaluating each such policy 5 times over 100 weight vectors. Once again, GPI-LS and GPI-PD have superior performance both in terms of expected
utility and quality of the Pareto frontier. This is the case even though we allowed each of the six PGMORL agents to collect 1.5 million experiences: they could interact with their environment ten times more often than our method/agent. Even then, GPI-PD (with or without using a learned model) consistently achieved higher expected utility (Fig 4a) during learning, and converged to a final solution with superior performance. Fig 4b depicts how the Pareto frontier identified by GPI-LS covers a broader range of behaviors that trade-off between the two objectives. This is made clear by observing that the points in the blue curve of Fig 4b dominate the corresponding Pareto frontier of PGMORL. Similarly, GPI-LS, when combined with GPI-PD, also produces a Pareto frontier that dominates that of PGMORL with the exception of one value vector. Notice that because GPI-LS+GPI-PD prioritize both weight and experience selection, the resulting frontier is denser—it more thoroughly covers the space of all possible trade-offs between objectives.

6 RELATED WORK

In this section, we briefly discuss some of the most relevant related work. For a complete discussion, see the Appendix.

Model-Free MORL. Several model-free MORL algorithms have been proposed in the literature [2, 21, 34, 44]. Here, we discuss the ones that are most related to our work. In [3], Abels et al. proposed training neural networks capable of predicting multi-objective action-value functions when conditioned on state and weight vector. Yang et al. [52] extended the previously-mentioned approach to increase sample efficiency by introducing a novel operator for updating the action-value neural network parameters. Our method differs with respect to these in that (i) we introduced a novel, principled prioritization scheme to actively select weights on which the action-value network should be trained, in order to more rapidly learn a CCS (Section 3.1); and (ii) we introduced a principled prioritization method for sampling previous experiences for use in Dyna planning to accelerate learning optimal policies for particular agent preferences. The methods above, by contrast, use uniform sampling strategies that implicitly assume that all training data is equally important to rapidly identify optimal solutions. Alegre et al. [5] showed how to use GPI in MORL settings, specifically in case the OLS algorithm is used to learn a CCS. Importantly, however, the heuristic used by OLS to select which weight vectors to train is based on upper bounds that are frequently exceedingly optimistic and loose. This means that OLS often focuses its training efforts on optimizing policies that do not necessarily improve the maximum utility loss incurred by the resulting CCS. Our method, by contrast, uses a technique for selecting which preferences to train on via a mathematically principled technique for determining a lower bound on performance improvements that are formally guaranteed to be achievable. This results in a novel active-learning approach that more accurately infers the ways by which a CCS can be rapidly improved. Xu et al. [50] proposed an evolutionary algorithm that trains a population of agents specialized in different weight vectors. We used a single neural network conditioned on the weight vector to simultaneously model all policies in the CCS. Fig. 4a shows that our approach surpasses the performance of this algorithm, even when given ten times fewer environment interactions.

Model-Based MORL. Compared to model-free MORL algorithms, model-based MORL methods have been relatively less explored. Wiering et al. [48] introduced a tabular multi-objective dynamic programming algorithm. Yamaguchi et al. [51] proposed an algorithm that learns models for predicting expected multi-objective reward vectors, rather than multi-objective Q-functions. Importantly, they tackle the average reward setting, rather than the cumulative discounted return setting. [51] is limited to discrete state spaces and focuses its experiments on small MDPs with at most ten states. Similarly, [48] assumes both discrete and deterministic MOMDPs. Similarly, the model-based method in [4] can tackle tabular, discrete settings—even though it supports non-linear utility functions. The methods proposed in [22, 30, 35, 47] investigate multi-objective Monte Carlo tree search approaches, but unlike our algorithm, require prior access to known transition models of the MOMDP.

7 CONCLUSIONS

In this paper we introduced two principled prioritization methods that significantly improve sample efficiency, as well as the first model-based MORL algorithm capable of dealing with continuous state spaces. Both prioritization schemes we introduced were derived from properties of GPI and resulted in an effective, sample-efficient algorithm with important theoretical guarantees. In particular, we (i) proved strong convergence guarantees to an optimal solution in a finite number of steps, or to an $\epsilon$-optimal solution (for a bounded $\epsilon$) if the agent is limited and can only identify possibly sub-optimal policies; (ii) proved that our method is an anytime algorithm capable of monotonically improving the quality of the solution throughout the learning process; and (iii) formally defined a bound that characterizes the maximum utility loss incurred by partial solutions computed by our technique at any given iteration. Finally, we empirically showed that our algorithm outperforms state-of-the-art MORL algorithms in qualitatively different multi-objective problems, both with discrete and continuous state and
action spaces. As future work, we would like to combine our algo-
rithmic contributions with other types of model-based approaches, such as predecessor/backward models [13] and value equivalent models [20]. We are also interested in extending our method to settings with non-linear utility functions.

ACKNOWLEDGMENTS
This study was financed in part by the following Brazilian agencies: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001; CNPq (grants 140500/2021-9 and 304932/2021-3); and FAPESP/MCTI/CGI (grant number 2020/05165-1). This work is supported by the Research Foundation Flanders (FWO)[G062819N], the AI Research Program from the Flemish Government (Belgium) and the Francqui Foundation.

REFERENCES