PARETO POLICY ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a policy gradient method for Multi-Objective Reinforcement Learning under unknown, linear preferences. By enforcing Pareto stationarity, a first-order condition for Pareto optimality, we are able to design a simple policy gradient algorithm that approximates the Pareto front and infers the unknown preferences. Our method relies on a projected gradient descent solver that identifies common ascent directions for all objectives. Leveraging the solution of that solver, we introduce *Pareto Policy Adaptation* (*PPA*), a loss function that adapts the policy to be optimal with respect to any distribution over preferences. PPA uses implicit differentiation to back-propagate the loss gradient bypassing the operations of the projected gradient descent solver. Our approach is straightforward, easy to implement and can be used with all existing policy gradient and actor-critic methods. We evaluate our method in a series of reinforcement learning tasks.

1 INTRODUCTION

Deep reinforcement learning has a pivotal role in solving several control problems of practical interest that would otherwise be intractable, such as robotic locomotion (1), the Atari (2) and Go games (3) to name a few. While such approaches have focused on the scalar reward setting, there exist many real-world problems that have multiple, conflicting objectives and, therefore, cannot be addressed by the current reinforcement learning tools. In such scenario, it is challenging to find an optimal control policy as the trade-offs (*preferences*) among the objectives may not be precisely known at training time or may differ from user to user. The paradigm of Multi-Objective Reinforcement Learning (MORL) provides a generalized framework for dealing with multi-dimensional rewards signals and has been identified as one of the main challenges of real-world reinforcement learning (4). However, learning optimal policies for MORL has been proven to be quite challenging, because most strategies require either to have access to or be able to infer a quantification of the relative importance of the objectives, or they perform sophisticated searches in the value or policy space aiming to find an ensemble of policies that are non-inferior to each other. The former methods lack adaptability and generalizability, as their performance is tied to the given or inferred preferences, while the later ones suffer from scalability issues, because they learn multiple policies, and are usually complicated to implement. In this work, we aim to strike a balance between those two families of methods.

Related Work: Multi-Objective Optimization (MOO) provides the fundamental tools for MORL. In MOO, there are two common solution concepts: *scalarization* and *Pareto optimality*. The former one derives a scalar objective and uses standard single-objective optimization techniques (5; 6; 7; 8; 9). The later method is based on the concept of *Pareto dominance* and considers the set of all non-inferior solutions (10; 11). Multiple gradient methods that leverage first-order, necessary conditions for Pareto optimality have also been developed (12; 13; 14).

Similarly to MOO, existing work in MORL can be roughly divided into two main categories: *Single-policy* methods aim to maximize a single, scalarized reward. These methods essentially transform the problem into a single-objective MDP and differ mostly in the way they determine and express the preferences. Scalarization is usually performed using a weighted sum of the reward vector (15; 16; 17) or, less commonly, using linear mappings (18). Different single-policy methods are based on thresholds or lexicographical ordering (19) or different classes of preferences (20; 21). More recently, a scalarized Q-learning algorithm has been developed (22) which uses the concept of *corner weight* to infer and optimize over preferences. It was extended to handle dynamic preferences in (23) and to utilize the convex envelop of the Pareto front in (24). Finally, a scale-invariant supervised learning approach for encoding the preferences was developed in (25).

On the other hand, *multiple-policy* methods use the vector-valued reward signal and learn a set of policies that approximate the Pareto front. The Pareto optimal solutions are the subset of non-inferior and equally good alternative polices among all distributions in the policy space and multiple-policy methods mainly differ on the approximation scheme for the Pareto front. One common approach is to perform multiple runs (16; 15) of a single scalarized reward function over a set of preferences. Unfortunately, such methods lack scalability to high-dimensional rewards. Other approaches leverage convex approximations of the Pareto front (26) or linear approximations of the value function (27; 28) to learn optimal deterministic policies. Multi-objective fitted Q-iteration (29; 30) enables us to learn policies for all linear preferences by encapsulating the preference vector as an input to the Q-function approximator. A policy gradient method based on discrete approximations of the Pareto front was proposed in (31) and enhanced to utilize continuous approximations in (32). A gradient-based method that learns a manifold on the policy parameter space leveraging episodic exploration and importance sampling was developed in (33). Finally, a prediction-guided evolutionary algorithm which is able to find dense approximation to the Pareto front was proposed in (34).

Single-policy methods have the advantage of learning a single policy and being simple to implement. However, they suffer from instability issues, as a small change on the preferences may lead to performance collapse (35), and also rely on heuristics to infer the preferences when they are not known. On the other hand, multiple-policy methods enjoy the advantage of being able to adapt to changing preferences because the solution (being an approximation to the Pareto front) encapsulates all trade-offs between the objectives. However, this benefit comes at a higher computational cost as we typically have to learn and store multiple policies. In this paper, we aim to bridge the gab between single and multiple policy MORL methods: Our method learns a single policy along with the underlying preference vectors and is able to adapt the inferred preference to any preference distribution. Our preference vectors are inferred, not by using heuristics, but by approximating the Pareto front via a first-order condition (see Fig. 1 for an example). Our method is inspired by the multiple gradient descent algorithm for MOO introduced in (13) as well as its application in the field of multi-task supervised learning presented in (36).

Contributions: We propose a policy gradient method for multi-objective reinforcement learning under unknown, linear preferences. Initially, we present Pareto stationarity as a necessary, first-order condition for Pareto optimality and develop a multi-objective policy gradient algorithm which uses a projected gradient descent solver to search for and take steps in a common ascent direction for all objectives. Following that, we tackle the problem of adapting the policy gradient to any given preference distribution. We utilize our method from the first part and introduce the Pareto Policy Adaptation (PPA), a loss function that penalizes deviations between the given preference distribution and the recovered preferences. Using implicit differentiation, we are able to back-propagate the gradient of PPA bypassing the operations of the projected gradient descent solver, which makes our method applicable to real-world problems. We evaluate our method in a series reinforcement learning tasks.

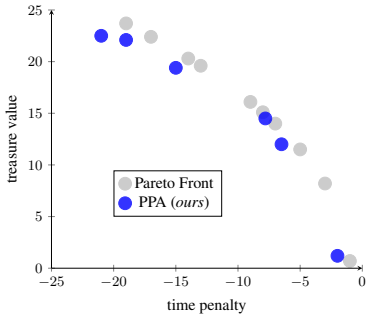


Figure 1: True and identified Pareto front for Deep Sea Treasure enviroment.

The most closely related work to ours is presented in (31). The authors discuss the existence of common ascent directions, introduce a quadratic optimization problem for their computation and give two gradient-based learning algorithms. Even though our method identifies the common ascent directions using a similar optimization problem, we leverage these directions to define the PPA loss function, which is a simple and efficient way 1) to incorporate the ascent directions into any learning algorithm and 2) to account for a given preference distribution. Leveraging implicit differentiation, PPA enables us to incorporate multiple rewards and preference distributions into any modern policy gradient algorithm by a mere modification of the loss function and by implementing a projected gradient descent optimizer.

2 PRELIMINARIES

We consider the framework of Multi-Objective Markov Decision Process (MOMDP) $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{R}}, \mathcal{P}, \gamma, \Omega, f_\Omega)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the (discrete) state and action spaces, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition kernel representing the environment dynamics, $\boldsymbol{r}(s, a) \equiv \boldsymbol{\mathcal{R}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^M$ is a function mapping state-action tuples to reward vectors, Ω is the space of preferences and $f_{\boldsymbol{\omega}} : \mathbb{R}^M \to \mathbb{R}$ a preference function which produces a scalar *utility* using a preference $\boldsymbol{\omega} \in \Omega$. Do declutter notation, we assume that there exists a given initial state. We consider linear preferences $f_{\boldsymbol{\omega}}(\boldsymbol{r}(s, a)) = \boldsymbol{\omega}^T \boldsymbol{r}(s, a)$. A fixed preference vector $\boldsymbol{\omega} \in \Omega$ allows to directly compare the vectorized reward and value function by comparing their scalarized utilities, and, therefore, the MOMDP collapses to a standard MDP. A stationary control policy $\pi : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is the Borel set of $\mathcal{A}$, is a function that assigns a probability distribution over actions for all states. The *value* $\boldsymbol{v}^\pi : \mathcal{S} \to \mathbb{R}^{M \times |\mathcal{S}|}$ of a policy π is defined as $\boldsymbol{v}^\pi(s) := \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}(s_t, a_t) \mid s_0 = s]$, where $\gamma \in (0, 1)$ is a discount factor and the expectation is taken under the distribution over trajectories $\tau = (s_0, a_0, s_1, a_, s_2, \dots)$ obtained by starting from s and following policy π thereafter. The *discounted state-action occupation measure* under policy π is defined as $d^\pi(s, a) := \sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a \mid \pi)$ for all states $s \in \mathcal{S}$ and allows us to write the value function compactly as $v^\pi(s) = \mathbb{E}_{d^\pi}[r(s, a) \mid s_0 = s]$.

Definition 1. *Considering all possible returns $\tilde{\boldsymbol{r}} := \sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}(s_t, a_t)$, we define the Reward Space Pareto Coverage Set (RS-PCS) or Pareto frontier as $\mathcal{F}_{\boldsymbol{r}}^* := \{\tilde{\boldsymbol{r}} : \tilde{\boldsymbol{r}} \in \mathbb{R}^M, \tilde{\boldsymbol{r}}' \succ_P \tilde{\boldsymbol{r}}\}$, where the symbol $\succ_P$ denotes Pareto dominance: greater or equal in all objectives and strictly greater in at least one objective. For all possible preferences in Ω and under the linear preference assumption, we define the Reward Space Convex Coverage Set (RS-CCS) as*

$$\mathcal{C}_{\boldsymbol{\mathcal{R}},\Omega}^* := \{\tilde{\boldsymbol{r}} : \tilde{\boldsymbol{r}} \in \mathbb{R}, \ \exists \boldsymbol{\omega} \in \Omega \text{ s.t. } \boldsymbol{\omega}^T \tilde{\boldsymbol{r}} \geq \boldsymbol{\omega}^T \tilde{\boldsymbol{r}}', \ \forall \tilde{\boldsymbol{r}}' \in \mathbb{R}^M\}. \tag{1}$$

The set $\mathcal{C}_{\boldsymbol{\mathcal{R}},\Omega}^*$ contains all discounted reward vectors that could be optimal for some preference vector in Ω. Anything in $\mathcal{F}_{\boldsymbol{r}}^*$ but not in $\mathcal{C}_{\boldsymbol{\mathcal{R}},\Omega}^*$ cannot be useful under linear preferences. In that case, it suffices to restrict our analysis to $\mathcal{C}_{\boldsymbol{\mathcal{R}},\Omega}^*$. Also, we assume, without loss of generality, that $\sum_m \omega_m = 1, \omega_m \geq 0, m = 1, 2 \dots M$ for all $\boldsymbol{\omega} \in \Omega$, i.e., the preference vector is a convex combination of the objectives. We extend the definition of Pareto optimality from the reward space to the policy space.

Definition 2. *A policy π dominates a policy π' if and only if $\boldsymbol{v}^\pi \succ_P \boldsymbol{v}^{\pi'}$. A policy π^* is called Pareto optimal if there exists no other policy π that dominates π^*. The set Π_P^* of all Pareto optimal policies is called Pareto policy set and its image $\mathcal{F}_\pi^* := \{\boldsymbol{v}^\pi\}_{\pi \in \Pi_P^*}$ is called Pareto policy frontier. The set*

$$\mathcal{C}_{\Pi,\Omega}^* := \{\pi : \pi \in \Pi, \ \exists \boldsymbol{\omega} \in \Omega \text{ s.t. } \boldsymbol{\omega}^T \boldsymbol{v}^\pi \geq \boldsymbol{\omega}^T \boldsymbol{v}^{\pi'}, \ \forall \pi' \in \Pi, \ \sum_m \omega_m = 1, \omega_m \geq 0\} \tag{2}$$

is called the Policy Space Convex Coverage Set (PS-CCS).

Lemma 1. *Let $\pi \in \mathcal{C}_{\Pi,\Omega}^*$ be a policy. Then, we have $\mathbb{E}_\pi[\tilde{\boldsymbol{r}}] \in \mathcal{C}_{\boldsymbol{\mathcal{R}},\Omega}^*$, i.e., the expected return vector under policy π belongs in the RS-CCS.*

Lemma 1 (proof given in the Appendix) connects the PS-CCS with the RS-CCS and allows us to use the former one in place of the later. This is very convenient because searching for Pareto optimal vectors in the reward space is not practically efficient (e.g., due to the sparsity of reward signal) or useful (e.g., because they cannot be easily mapped to values or actions). By finding a policy $\pi \in \mathcal{C}_{\Pi,\Omega}^*$, Lemma 1 guarantees that the expected return vector under π belongs in $\mathcal{C}_{\boldsymbol{\mathcal{R}},\Omega}^*$ and, therefore, maps the problem of reward space exploration to policy space exploration. Additionally, Lemma 1 imposes no restrictions on the class of policies and allows us to adopt the common practice of parameterizing the policy by high-dimensional vector $\theta \in \Theta$ (e.g., a neural network). The definition of PS-CCS given Eq. equation 3 carries over to this parametric class as is, i.e., we can define the PS-CCS in the policy parameter space as follows

$$\mathcal{C}_{\Theta,\Omega}^* := \{\theta : \theta \in \Theta, \ \exists \boldsymbol{\omega} \in \Omega \text{ s.t. } \boldsymbol{\omega}^T \boldsymbol{v}^{\pi_\theta} \geq \boldsymbol{\omega}^T \boldsymbol{v}^{\pi_{\theta'}}, \ \forall \theta' \in \Theta\}. \tag{3}$$

3 LEARNING WITH UNKNOWN PREFERENCES

In this section, we propose a new policy gradient algorithm for MORL under unknown preferences. The key idea of our approach is to infer the preference vector $\boldsymbol{\omega}$, use that $\boldsymbol{\omega}$ as scalarization coefficient for the multi-objective criterion and ascent on the gradient of the scalarized value. To search for a Pareto optimal policy, we leverage a fist-order condition known as Pareto stationarity.

Definition 3 (Pareto stationarity)**.** *A policy π_θ is called Pareto stationary if and only if $\sum_{m=1}^{M} \omega_m \nabla_\theta \mathbb{E}_{d^{\pi_\theta}}[r_m(s,a)] = 0$ for some $\omega_m \in [0,1],\ m = 1, 2 \dots M$.*

Pareto stationarity implies that there exists a vanishing convex combination of the objectives' gradients and is a necessary condition for Pareto optimality (13). The following theorem connect the PS-CCS with Pareto stationarity.

Theorem 1. *Let $\pi_\theta \in \mathcal{C}^*_{\Theta,\Omega}$ be a policy in the PS-CCS. Then, π_θ is Pareto stationary.*

Proof. Let $\pi_\theta \in \mathcal{C}^*_{\Theta,\Omega}$ be a policy and $\boldsymbol{\omega}$ the corresponding preferences, i.e., we have $\boldsymbol{\omega}^T \boldsymbol{v}^{\pi_\theta} \geq \boldsymbol{\omega}^T \boldsymbol{v}^{\pi_{\theta'}},\ \forall \theta' \in \Theta$. The last relation can be written as $\boldsymbol{\omega}^T \mathbb{E}_{\pi_\theta}[\tilde{\boldsymbol{r}}] \geq \boldsymbol{\omega}^T \mathbb{E}_{\pi_{\theta'}}[\tilde{\boldsymbol{r}}],\ \forall \theta' \in \Theta$. This implies that θ is a global maximizer of $\boldsymbol{\omega}^T \mathbb{E}_{\pi_\theta}[\tilde{\boldsymbol{r}}]$ or, equivalently, that its gradient at θ vanishes. Therefore, we have

$$\nabla_\theta \boldsymbol{\omega}^T \mathbb{E}_{d^{\pi_\theta}}[\tilde{\boldsymbol{r}}] = \sum_{m=1}^{M} \omega_m \nabla_\theta \mathbb{E}_{d^{\pi_\theta}}[r_m(s,a)] = 0, \tag{4}$$

which corresponds to the necessary condition for Pareto stationarity as per Def. 3. □

Theorem 1 essentially states that Pareto stationarity is a necessary condition for a policy to belong to PS-CCS. Additionally, as stated in the proof, the convex combination of the objectives that leads to vanishing gradients coincides with the preferences that make the underlying policy part of the PS-CCS. This presents us with a strong necessary condition for recovering the unknown preferences and for identifying an optimal policy. This is because the problem of finding a Pareto stationary is tractable via gradient methods. In fact, if a policy is not Pareto stationary then there exists a common ascent direction for all objectives. To find this descent direction, we use the following theorem.

Theorem 2 ((13), Theorem 1)**.** *Consider the following quadratic optimization problem*

$$\min_{\omega_1,\dots\omega_M} \quad \left\{ \left\| \sum_{m=1}^{M} \omega_m \nabla_\theta \mathbb{E}_{d^{\pi_\theta}}[r_m(s,a)] \right\|_2^2 \ s.t \sum_{m=1}^{M} \omega_m = 1,\ \omega_m \geq 0,\ \forall m \right\}. \tag{P_A}$$

Let $\boldsymbol{\omega}^*$ be the optimal solution of Problem P_A and μ^* the underlying minimal-norm objective. Then:

1. *either $\mu^* = 0$ and π_{θ^*} is a Pareto stationary policy*
2. *or $\sum_{m=1}^{M} \omega_m^* \nabla_\theta \mathbb{E}_{d^{\pi_\theta}}[r_m(s,a)]$ is a common ascent direction for all objectives.*

Problem P_A is essentially equivalent to the formulation of the famous *Minimum-Norm Point* problem, which arises many fields such as optimal control (37), submodular minimization (38) and portfolio optimization (39). The MNP problem has been well studied and several combinatorial and recursive algorithms have been proposed (40; 41; 42). Such methods search for the *exact* solution and typically have exponential run-time (43), which prevents their application in high-dimensional control policies. Therefore, we adopt convex optimization approach for Problem 2. We start by noting that the constrain set $\mathcal{E} = \{\boldsymbol{\omega} \in \mathbb{R}^M : \sum_{m=1}^{M} \omega_m = 1,\ \omega_m \geq 0,\ \forall m = 1, 2 \dots M\}$ is essentially the unit simplex and the projection problem, i.e., $\Pi_{\mathcal{E}}(\boldsymbol{\omega}) = \arg\min_{\boldsymbol{\delta} \in \mathcal{E}} \|\boldsymbol{\omega} - \boldsymbol{\delta}\|$, can be efficiently solved. In fact, there exists a unique dual variable $\tau \in \mathbb{R}$ such that

$$\Pi_{\mathcal{E}}(\boldsymbol{\omega}) = (\boldsymbol{\omega} - \tau)_+, \tag{5}$$

where $(x)_+ = \max(x, 0)$. To find the dual variable, we enforce the equality constraint, i.e., $\sum_m (\omega_m - \tau)_+ = 1$. This non-linear equation of a single variable can be solved very efficiently using the Newton method. This suggests a projected gradient descent solver for solving Problem P_A and recovering the unknown preferences.

Algorithm 1: Multi-Objective Policy Gradient

Inputs: Initial parameters θ_0, learning rate α
foreach $k = 0, 1, \ldots$ **do**
  Collect trajectories $\mathcal{D} = \{\tau_i\},\ \tau_i \sim \pi_{\theta_k}$
  Compute rewards-to-go $\boldsymbol{r}_t$
  $g_m \leftarrow \frac{1}{|\mathcal{D}|} \sum_{\tau,t} \boldsymbol{r}_t \nabla_\theta \log \pi_{\theta_k}(a_t|s_t)$
  $(\omega_1, \ldots \omega_M) \longleftarrow \texttt{PGDSolver}(g_1, \ldots g_M, \alpha)$
  $\theta_{k+1} \leftarrow \theta_k + \sum_{m=1}^{M} \omega_m g_m$
end

procedure $\texttt{PGDSolver}(g_1, \ldots, g_M, \alpha)$**:**
  Initialize feasible $\boldsymbol{\omega} = (\omega_1, \ldots \omega_M)$
  while *not converged* **do**
    Find τ solving $\sum_m (\omega_m - \tau)_+ = 1$
    $\omega_m \leftarrow \omega_m + \alpha g_m^T \sum_n \boldsymbol{e}_n g_n,\ m = 1 \ldots M$
    $\boldsymbol{\omega} \leftarrow (\boldsymbol{\omega} - \tau)_+$
  end
  return $(\omega_1, \ldots \omega_M)$

Our method (Algorithm 1) extends the classical REINFORCE algorithm (44) to take gradient steps in the direction given by the weighted combination of the multi-objective gradient, with the weights being the preferences recovered by the projected gradient descent solver. Lemma 1, Theorem 1 and 2 essentially state that Algorithm 1 learns arbitrary policies in the Convex Coverage Set along with their underlying preference vectors and provide theoretical justification for our method. Algorithm 1 is simple to implement, compatible with all policy gradient and actor-critic methods and provides a practical algorithm for learning Pareto optimal policies under unknown preferences. However, it suffers from a subtle drawback: we have no control over the identified preferences and they cannot be optimized to account for any given preference distribution. This is the problem of policy adaptation which we address in the next section.

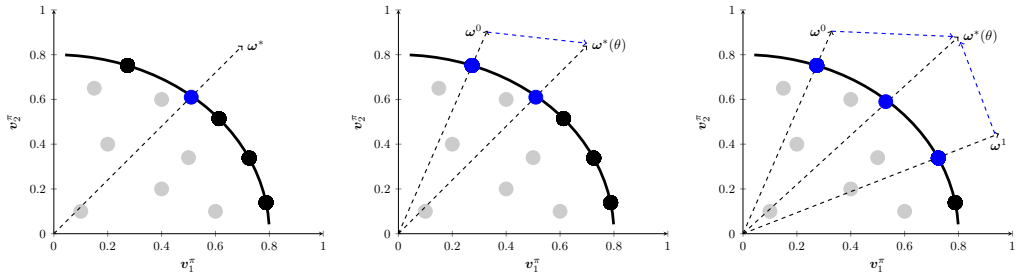
It is worth comparing our projected gradient descent solver with (36), where the authors used the Frank-Wolfe method for Problem P_A, albeit in the context of supervised learning. Both algorithms have the same convergence rate of $\mathcal{O}(1/N)$ and since the projection to the constraint set can be easily calculated, we opted for the projected gradient descent optimizer. The overhead added by solving for the dual variable τ is minimal because the Newton's method is known to enjoy quadratic convergence rate and is very efficient in practice. The trade-off of using Frank-Wolfe is that requires to obtain the feasible minimizer of the linear approximation of the objective around the current iterant, which is typically solved approximately and worsens the convergence rate by a multiplicative constant (45). Nonetheless, we expect that both methods can be used interchangeably in practice.

4 Pareto Policy Adaptation

The policy gradient algorithm presented in the previous section finds policies that are optimal (i.e., Pareto stationary) for the inferred preferences. In this section, we show how to address the problem of adapting the policy to be optimal for a given distribution $\mathcal{D}_{\boldsymbol{\omega}}$ over preferences. We start by expressing the optimal solution of Problem P_A as a function of the policy parameters $\boldsymbol{\omega}^* := \boldsymbol{\omega}^*(\theta) = (\omega_1(\theta), \omega_2(\theta), \ldots \omega_M(\theta))$. We introduce the *Pareto Policy Adaptation* loss function as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{d^{\pi_\theta}, \boldsymbol{\omega}} \left[\lambda \sum_{m=1}^{M} \omega_m^*(\theta) r_m(s,a) - (1-\lambda) \left\| \boldsymbol{\omega}^*(\theta) - \boldsymbol{\omega} \right\|_2^2 \right], \tag{6}$$

where $\lambda \in [0, 1]$ is a weight that trades off between the two losses. The first term of PPA is the scalarized reward, with the scalarization coefficient being the inferred preferences. The second term penalizes the expected deviation between the given and inferred preferences. To further motivate the PPA loss, consider the special case of a single, given preference vector $\boldsymbol{\omega}^0$. In this case, the MOMDP collapses to a standard MDP, with $\boldsymbol{\omega}^0$ being the scalarization coefficient for the reward and value functions. Additionally, assume that the policy π_θ has been *adapted* to that preference vector, i.e., it achieves $\boldsymbol{\omega}^*(\theta) = \boldsymbol{\omega}^0$ for some parameter vector θ. Then, Eq. 6 reduces to $\mathcal{L}(\theta) = \mathbb{E}_{d^{\pi_\theta}} \left[\sum_{m=1}^{M} \omega_m^0 r_m(s,a) \right]$. Maximization of this loss corresponds to taking ascent steps along $\sum_m \omega_m^0 \nabla_\theta \mathbb{E}_{d^{\pi_\theta}} [r_m(s,a)]$, which, per Theorem 2, is an ascent direction guaranteed to increase all objectives. This simplified scenario essentially states the if we *exactly* match the preferences learned from Problem P_A to the given preference vector $\boldsymbol{\omega}^0$, then not only the PPA loss function reduces to the loss function for the scalarized reward, but also the ascent steps along its



(a) (b) (c)

Figure 2: (a) The Convex Coverage Set (CCS) for a 2-dimensional value function. Algorithm 1 identifies arbitrary solutions on the Pareto front. (b) Illustration of the PPA loss function for a single preference vector $\boldsymbol{\omega}^0$: by minimizing the distance $\left\|\boldsymbol{\omega}^*(\theta) - \boldsymbol{\omega}^0\right\|$, the solution converges to the optimal solution corresponding to preference $\boldsymbol{\omega}^0$. (c) PPA loss function for two preference vectors $\boldsymbol{\omega}^0, \boldsymbol{\omega}^1$: the minimizer of $\left\|\boldsymbol{\omega}^*(\theta) - \boldsymbol{\omega}^0\right\| + \left\|\boldsymbol{\omega}^*(\theta) - \boldsymbol{\omega}^1\right\|$ (also called the *geometric median*) gives the optimal solution because it maximizes the projections onto the preference vectors.

gradient lead to concurrent improvements of all objectives. We show a geometric illustration of the PPA loss function in Fig. 2.

Despite its simplicity, one important obstacle in using the PPA loss to train agents in practice is that the optimal solution to the Problem P_A is analytically known only for $M = 2$ (solution is presented in the Appendix). For $M > 2$, it can rather only be approximated by the iterative projected gradient descent solver introduced in Algorithm 1. This makes the backward pass challenging because the gradients need to be back-propagated through the (perhaps thousands) iterations of the projected gradient descent solver. Since back-propagation is known to be a very expensive operation compared to the forward pass, this could make our approach inefficient for practical problems. The problem of back-propagating through solutions of numerical solvers arises in several deep learning architectures such as deep equilibrium models (46) and optimization-based neural network layers (47). Luckily, the implicit value theorem allows us to back-propagate the gradient bypassing the operations of the projected gradient descent solver.

Theorem 3. *Let $l := l(\boldsymbol{\omega}^*(\theta))$ be a generic loss function that depends explicitly on the the optimal solution $\boldsymbol{\omega}^*(\theta)$ of Problem P_A. Then, the gradient of l with respect to the policy parameters is*

$$\nabla_\theta l = \begin{bmatrix} H^T(\theta) & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} G(\theta) & -I_M & \boldsymbol{e} \\ -D(\boldsymbol{\mu}^*(\theta)) & -D(\boldsymbol{\omega}^*(\theta)) & \mathbf{0} \\ \boldsymbol{e}^T & \mathbf{0} & \mathbf{0} \end{bmatrix}^{-T} \begin{bmatrix} \frac{dl}{d\boldsymbol{\omega}^*} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{7}$$

where $[G(\theta)]_{m,n} = 2g_m^T(\theta)g_n(\theta)$, $[H(\theta)]_{m,\cdot} = -\sum_n \omega_n^*(\theta)\nabla_\theta^T(g_n^T(\theta)g_m(\theta))$, $g_m(\theta) = \nabla_\theta \mathbb{E}_{d^{\pi_\theta}}[r_m(s,a)]$, $\boldsymbol{\mu}^*(\theta)$ is the optimal dual variable, $\boldsymbol{e}$ and $\mathbf{0}$ are the identity and zero vectors of appropriate dimensions, I_M is the identity matrix of dimension M and $D(\cdot)$ create a diagonal matrix from a vector. When applied on a vector quantity, the operator ∇_θ denotes the Jacobian matrix and the operator ∇_θ^T denotes the transpose of the underlying gradient or Jacobian.

The proof of Theorem 3 (given in the Appendix) leverages the implicit value theorem which enables us to differentiate the KKT optimality conditions of Problem P_A with respect to the policy parameters. Observe that matrix $H(\theta)$ is essentially of dimension $M \times d_\theta$, d_θ being the dimension of θ. This dependence on d_θ makes the solution of eq:nmatrix inefficient, as we need to compute and store a matrix of size $(2M+1)d_\theta$. However, in practice, we are not interested in knowing the full gradient of $\boldsymbol{\omega}^*(\theta)$, but rather in back-propagating the gradient of the loss function; Eq. 7 has an appealing form which allows us to use vector-Jacobian products (VJPs) to evaluate the gradient $\nabla_\theta l$ one row at a time. This frees us from having to store matrix $H(\theta)$ and compute its product with another matrix. We observe that the solution in Eq. 7 requires knowledge of the optimal dual variable $\boldsymbol{\mu}^*(\theta)$, which is not directly available. However, the optimal primal solution $\boldsymbol{\omega}^*(\theta)$ given by the projected gradient descent solver (Algorithm 1) can be utilized to obtain $\boldsymbol{\mu}^*(\theta)$ by substituting it into the KKT optimality conditions of Problem P_A (details in the Appendix) and solving the resulting linear system. That system has $M+1$ variables, therefore its solution adds minimal computational overhead.

It is worth discussing similarities and differences between PPA and EQL (24). PPA retrieves and adapts to preferences during the training phase, whereas EQL first learns a preference-dependant policy and then retrieves preferences using a combination of stochastic search and policy gradient. Also, the loss functions are partially similar: the penalty term in PPA is the MSE loss between the given and inferred preferences, whereas the EQL loss function is the MSE loss of vector Q-values. Additionally, given that PPA is a policy method and EQL is a value method, they can be combined in an actor-critic algorithm. This is a very interesting direction for future work.

One limitation of our work is that it relies on Pareto stationarity, which is a necessary but not sufficient condition for Pareto optimality. i.e., there is no guarantee for Pareto optimality only stationarity. Nonetheless, Pareto optimality is hard to be achieved and we often need to resort to first-order approximations. Finally, the optimal policy obtained under a given distribution is not, in general, optimal under a different distribution and the model has to be fine-tuned. We expect that we will not need to re-train the network entirely and that we can leverage the previous optimal policy to find the new one. The technical details of how this would be achieved are a direction for future work.

5 Experimental Evaluation

In this section, we evaluate the performance of our proposed method. In our implementation, we extended the popular Proximal Policy Optimization (PPO) (48) algorithm so that it can handle our PPA loss function. We implemented all experiments on PyTorch using modified versions of OpenAI Gym environments. We give the details of our experimental setup in the Appendix.

Domains: We evaluate on 4 environments (details given in the Appendix): (a) Multi-Objective GridWorld (MOWG): a variant of the classical gridworld, (b) Deep Sea Treasure (DST): a slightly modified version of classical multi-objective reinforcement learning enviroment (49), (c) Mulit-Objective SuperMario (MOSM): modified, multi-objective variant of the popular video game that has a 5-dimensional reward signal and (d) Multi-Objective MuJoCo (MOMU): a modified version of the MuJoCo physics simulator, focusing on locomotion tasks.

Ablation Study and Baselines: To highlight the extent to which our results are driven by the PPA loss function, we perform an ablation study. In more details, we consider the following three agents: (a) Pareto Policy Adaptation (`PPA`): This is our original loss function as introduced in Sec. 4. (b) non-adaptive PPA (na-PPA): Same as `PPA` but the preferences are static, i.e., the policy is not adapted to the preference distribution. This agent is essentially Algorithm 1 without the PPA loss function. (c) Fixed Preferences: Same as `na-PPA` but the preferences are given. This corresponds to optimizing the scalarized objective. Additionally, we compare the performance of our method against a range of state-of-the-art baselines. In particular, we compare against: the Pareto Following Algorithm (PFA) (31), the Conditional Neural network with Optimistic Linear Support (CN-OLS)(22; 23), the Conditioned Network with Diverse Experience Replay (CN-DER) (23), the Envelope Q-Learning (EQL) (24) and the Prediction-Guided Multi-Objective Reinforcement Learning (PGMORL) (34).

Evaluation Metrics: To evaluate the performance of our method and compare against the baselines, we use the following metrics:

1. *HyperVolume* (`HV`): Let $\mathcal{F}$ be a reward space Pareto front approximation and $\boldsymbol{r_0}$ be a reference vector. Then, we define the *HyperVolume* as $\texttt{HV} := \int \mathbb{1}_{H(P)}(z)dz$, where $H(P) = \{z \in \mathbb{R}^M : \exists \boldsymbol{r} \in \mathcal{F} : \boldsymbol{r_0} \prec_P z \prec_P \boldsymbol{r}\}$ and $\mathbb{1}(\cdot)$ is the indicator function.
2. *Pareto Dominance* (`PD`): Let $\boldsymbol{r}_t^1$, $\boldsymbol{r}_t^2$ be the reward vectors of trajectories sampled from agent 1 and 2, respectively. We define the *Pareto Dominance* metric as $\texttt{PD}_{1,2} := \frac{1}{T}\sum_{t=0}^{T} \mathbb{1}(\boldsymbol{r}_t^1 \succ_P \boldsymbol{r}_t^2)$, where $\mathbb{1}(\cdot)$ is the indicator function. This is essentially the fraction of time when the reward vector under agent 1 Pareto dominates the reward vector under agent 2. Note that $\texttt{PD}_{1,2} + \texttt{PD}_{2,1} \leq 1$.
3. *Utility* (`UT`): For a reward vector $\boldsymbol{r}_t$ sampled from any agent, we define the *Utility* metric as $\texttt{UT} := \mathbb{E}_{\boldsymbol{\omega}}[\frac{1}{T}\sum_{t=0}^{T} \boldsymbol{\omega}^T \boldsymbol{r}_t]$, where the expected value is evaluated by sampling uniformly from the preference space Ω or from the preference distribution $\mathcal{D}_{\boldsymbol{\omega}}$.

Main Results: Figure 3 shows the results of the ablation study conducted on the MOGW and DST domains. We compare against the non-adaptive and the fixed-preferences agents. For the later case, we pick three different preference vectors: $(\omega_1, \omega_2) = (0.5, 0.5)$, $(\omega_1, \omega_2) = (0.75, 0.25)$ and $(\omega_1, \omega_2) = (0.25, 0.75)$. These plots facilitate the visual identification of Pareto dominant agents. Focusing on the MOGW (upper row), we observe that our PPA agent converges to a policy that yields a higher reward and lower time-penalty than any other agent, i..e, PPA Pareto dominates all other agents. While a similar statement cannot be made for the DST domain, we note that PPA is not Pareto dominated by any other agent and the only agent that it does not dominate is the fixed-preference one for $(\omega_1, \omega_2) = (0.5, 0.5)$. On the other hand, the non-adaptive PPA (na-PPA) seems to perform relatively poor. However, there exists some merit in using it because, in contrast to the fixed-preference agents, na-PPA does not rely on being given a preference vector. It is also worth noting that in the DST domain the preference vector $(\omega_1, \omega_2) = (0.25, 0.75)$ gives very poor results, which implies that handpicking the preferences might lead to performance collapse. Overall, these results suggest that PPA steadily achieves better performance and is superior than the other agents. It is interesting to observe that the reward and time penalty curves are quite similar in the MOWG domain. This agrees with our intuition because in the MOGW domain the reward and time penalty have an "inverse" relationship (i.e., the sooner we reach the bottom right, the higher the reward).

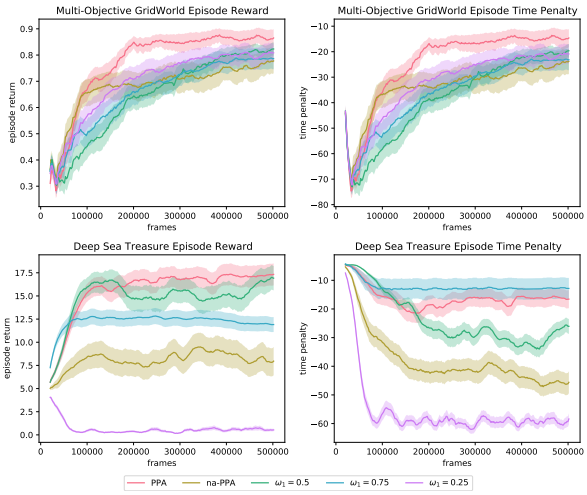


Figure 3: Results on the MOGW and DST environments.

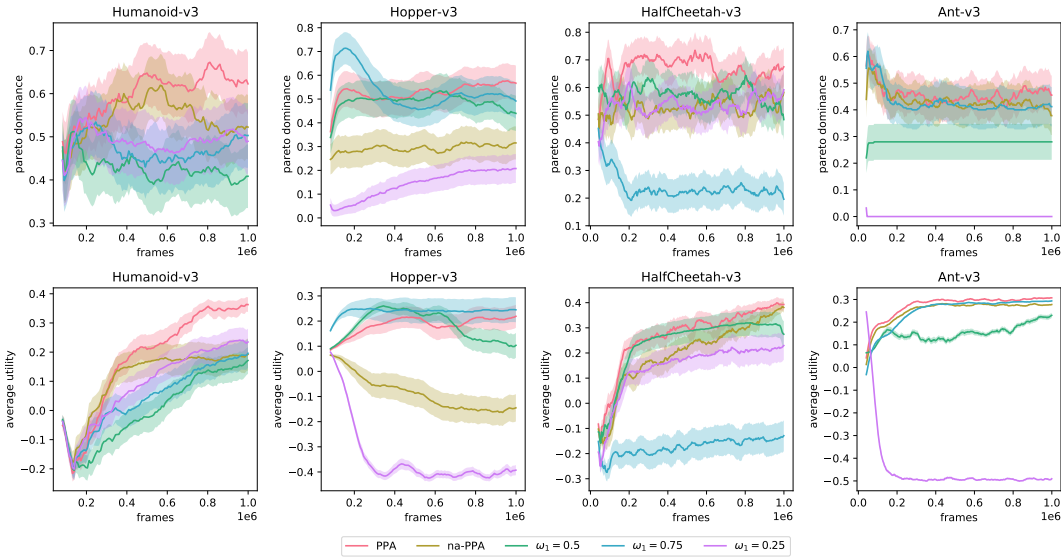


Figure 4: Learning curves on MuJoCo environments. We show the Pareto Dominance (PD) and Average Utility (UT) metrics. Experiments are run for 10 random seeds. We compare against the non-adaptive PPA and the fixed-preferences agents.

In the next line of experiments, we evaluate the performance of PPA in continuous control tasks. We modified the MuJoCo Gym interface to output 2-dimensional rewards on each step. The first one is the commonly used reward in such environments, i.e., the difference between two sequential positions of the robot. The second one is the energy consumed by the robot. We show the results on Fig. 4. We compare against the ablated agents and show the Pareto Dominance (PD) and Average

		PFA	CN-OLS	CD-DER	EQL	PGMORL	nA-PPA	PPA
MOGW	HV	2.52	2.36	2.65	3.10	3.15	2.35	**3.32**
	PD	0.52	0.51	0.46	0.53	0.55	0.33	**0.65**
	UT	0.23	0.19	-0.12	0.18	0.28	0.08	**0.28**
DST	HV	36.32	35.36	35.65	35.12	39.35	34.20	**40.12**
	PD	0.46	0.55	0.39	0.48	0.52	0.38	**0.75**
	UT	0.19	0.14	-0.11	0.20	**0.24**	0.12	0.21
MOSM	HV	-	1146.14	1364.14	1134.12	1401.14	1125.18	**1442.14**
	PD	-	0.47	0.41	0.50	0.59	0.42	**0.64**
	UT	-	0.15	-0.07	0.24	0.26	0.14	**0.27**
Humanoid-v3	HV	-	4.39	5.18	4.95	4.65	4.02	**5.15**
	PD	-	0.41	0.46	0.54	0.52	0.33	**0.55**
	UT	-	0.19	0.22	0.41	0.30	0.28	**0.48**
Hopper-v3	HV	-	3.18	2.98	3.44	**3.74**	2.23	3.73
	PD	-	0.43	0.49	0.45	0.52	0.45	**0.70**
	UT	-	-0.23	0.14	0.22	0.21	0.12	**0.25**
HalfCheetah-v3	HV	-	5.17	5.38	5.39	**5.77**	4.18	5.68
	PD	-	0.37	0.38	0.51	0.55	0.45	**0.62**
	UT	-	-0.10	0.11	0.22	0.20	0.18	**0.31**
Ant-v3	HV	-	5.95	5.47	6.30	6.35	5.25	**6.45**
	PD	-	0.26	0.30	0.44	0.41	0.32	**0.52**
	UT	-	0.36	0.31	0.34	**0.55**	0.39	**0.55**

Table 1: Comparison of MORL agents using the HyperVolume (HV), Pareto Dominance (PD), and Average Utility (UT) metrics. Possible values for each metric lie in the range $[0, \infty]$, $[0, 1]$, and $[-1, 1]$, respectively. In all cases, higher values indicate better performance. We evaluate the agents using 10 random seeds.

Utility (UT) metrics. The only case where our method does not attain the best performance is the Hopper-v3 environment under the UT metric, where the performance of the fixed-preferences agent $(\omega_1, \omega_2) = (0.75, 0.25)$ slightly surpasses the performance of PPA. However, observe that the same agent attains very poor performance in the HalfCheetah-v3 enviroment. This suggests that, given the complex inter-dependence between the two rewards, simply hand-crafting a preference vector is far from optimal and a preference vector that yields good performance in one environment may perform poorly in a different environment or judged under a different metric.

In the last line of experiments, we compare our method against state-of-the-art MORL algorithms (Table 1). Since PD is a pair-wise metric, we first compute the pair-wise $\texttt{PD}_{i,j}$ values for all agents and then average-out the j component to obtain a single value for each agent. Note that this metric is specific to the ensemble of agents that we consider and needs to be re-calculated if we compare a different set of agents. The UT metric is calculated by sampling preferences from the preference space of each domain. We assume assume a uniform preference distribution and sample 1000 points for the 2-dimensional preference case (MOGW, DST, MUMO) and 100 across each dimension for the MOSM case, where the preference space has 4 dimensions. Additionally, we scale the rewards in the range $[-1, 1]$ across each dimension. This scaling is important when calculating the average utility but has no particular influence on the Pareto dominance or hypervolume. Observe that some agents have negative utility, which is indeed possible given the aforementioned reward scaling. Our method systematically outperforms all other baselines across all domains, which is another strong empirical finding for its performance.

6 Conclusion

We introduced a policy gradient algorithm for Multi-Objective Reinforcement Learning (MORL) under linear preferences. By approximating the Pareto front via a first-order necessary condition, our method learns a *single* policy that can be adapted to *any* preference distribution. The two pillars of our method is a projected gradient descent solver that searches for common ascent direction for all objectives and a novel *Pareto Policy Adaptation* (*PPA*) loss function that leverages the solutions of that solver to adapt the policy to any preference distribution. Our method enjoys the advantages of both single and multiple policy MORL algorithms while being simple to implement and compatible with all policy gradient and actor-critic methods. We demonstrated the effectiveness of our method in several reinforcement learning tasks.

REFERENCES

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[2] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[4] Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, Apr 2021.

[5] I. Y. Kim and O. L. de Weck. Adaptive weighted sum method for multiobjective optimization: a new method for pareto front generation. *Structural and Multidisciplinary Optimization*, 31(2):105–116, Feb 2006.

[6] Timothy Ward Athan and Panos Y. Papalambros. A note on weighted criteria methods for compromise solutions in multi-objective optimization. *Engineering Optimization*, 27(2):155–176, 1996.

[7] P. L. Yu and G. Leitmann. Compromise solutions, domination structures, and salukvadze's solution. *Journal of Optimization Theory and Applications*, 13(3):362–378, Mar 1974.

[8] Ralph E. Steuer and Eng-Ung Choo. An interactive weighted tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26(3):326–344, Oct 1983.

[9] F. Waltz. An engineering approach: Hierarchical optimization criteria. *IEEE Transactions on Automatic Control*, 12(2):179–180, 1967.

[10] I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1):63–69, Aug 1997.

[11] A. Messac and A. Ismail-Yahaya. Multiobjective robust design using physical programming. *Structural and Multidisciplinary Optimization*, 23(5):357–371, Jun 2002.

[12] Jörg Fliege and Benar Fux Svaiter. Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 51(3):479–494, Aug 2000.

[13] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5):313 – 318, 2012.

[14] S. Schäffler, R. Schultz, and K. Weinzierl. Stochastic method for the solution of unconstrained vector optimization problems. *Journal of Optimization Theory and Applications*, 114(1):209–222, Jul 2002.

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

[16] Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 601–608, New York, NY, USA, 2005. Association for Computing Machinery.

[17] Andrea Castelletti, Giorgio Corani, Andrea-Emilio Rizzoli, Rodolfo Soncini-Sessa, and E. Weber. *Reinforcement learning in the operational management of a water system*, pages 325–. 01 2002.

[18] Gerald Tesauro, Rajarshi Das, Hoi Chan, Jeffrey Kephart, David Levine, Freeman Rawson, and Charles Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008.

[19] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 197–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[20] Shie Mannor and Nahum Shimkin. The steering approach for multi-criteria reinforcement learning. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002.

[21] Shie Mannor and Nahum Shimkin. A geometric approach to multi-criterion reinforcement learning. *J. Mach. Learn. Res.*, 5:325–360, December 2004.

[22] Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. Multi-objective deep reinforcement learning. *CoRR*, abs/1610.02707, 2016.

[23] Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 11–20. PMLR, 09–15 Jun 2019.

[24] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[25] Abbas Abdolmaleki, Sandy Huang, Leonard Hasenclever, Michael Neunert, Francis Song, Martina Zambelli, Murilo Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. A distributional view on multi-objective policy optimization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11–22. PMLR, 13–18 Jul 2020.

[26] Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 41–47, New York, NY, USA, 2008. Association for Computing Machinery.

[27] Daniel J. Lizotte, Michael Bowling, and Susan A. Murphy. Linear fitted-q iteration with multiple reward functions. *Journal of Machine Learning Research*, 13(105):3253–3295, 2012.

[28] Daniel J. Lizotte, Michael Bowling, and Susan A. Murphy. Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 695–702, Madison, WI, USA, 2010. Omnipress.

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

[30] Andrea Castelletti, Francesca Pianosi, and Marcello Restelli. Multi-objective fitted q-iteration: Pareto frontier approximation in one single run. In *2011 International Conference on Networking, Sensing and Control*, pages 260–265, 2011.

[31] Simone Parisi, Matteo Pirotta, Nicola Smacchia, Luca Bascetta, and Marcello Restelli. Policy gradient approaches for multi-objective sequential decision making. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2323–2330, 2014.

[32] Simone Parisi, Matteo Pirotta, and Marcello Restelli. Multi-objective reinforcement learning through continuous pareto manifold approximation. *J. Artif. Int. Res.*, 57(1):187–227, September 2016.

[33] Simone Parisi, Matteo Pirotta, and Jan Peters. Manifold-based multi-objective policy search with sample reuse. *Neurocomputing*, 263:3–14, 2017.

[34] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10607–10616. PMLR, 13–18 Jul 2020.

[35] Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Int. Res.*, 48(1):67–113, October 2013.

[36] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[37] Hongwei Jiao and Yongqiang Chen. A global optimization algorithm for generalized quadratic programming. *Journal of Applied Mathematics*, 2013:215312, Oct 2013.

[38] Satoru Iwata. Submodular function minimization. *Mathematical Programming*, 112(1):45, Jan 2007.

[39] Satoru Fujishige, Hiroko Sato, and Ping Zhan. An algorithm for finding the minimum-norm point in the intersection of a convex polyhedron and a hyperplane. *Japan Journal of Industrial and Applied Mathematics*, 11(2):245, Jun 1994.

[40] P. Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11:128–149, 1976.

[41] Satoru Fujishige and Ping Zhan. A dual algorithm for finding the minimum-norm point in a polytope. *Journal of the Operations Research Society of Japan*, 33, 01 1990.

[42] Kazuyuki Sekitani and Yoshitsugu Yamamoto. A recursive algorithm for finding the minimum norm point in a polytope and a pair of closest points in two polytopes. *Mathematical Programming*, 61(1):233–249, Aug 1993.

[43] Jesús A. De Loera, Jamie Haddock, and Luis Rademacher. The minimum euclidean-norm point in a convex polytope: Wolfe's combinatorial algorithm is exponential. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 545–553, New York, NY, USA, 2018. Association for Computing Machinery.

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

[45] Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 427–435, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[46] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 688–699, 2019.

[47] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 06–11 Aug 2017.

[48] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

A APPENDIX

A THEORETICAL FRAMEWORK

In the first part of the Appendix, we give the proofs omitted in Sections 2 and 4 as well as the analytical solution of Problem P_A for the special case of 2-dimensional reward signals. For the ease of reference, we re-state the underlying theorem.

A.1 PROOFS

Lemma 1. *Let $\pi \in \mathcal{C}^*_{\Pi,\Omega}$ be a policy. Then, we have $\mathbb{E}_\pi[\tilde{\boldsymbol{r}}] \in \mathcal{C}^*_{\boldsymbol{\mathcal{R}},\Omega}$, i.e., the expected return vector under policy π belongs in the RS-CCS.*

Proof. Consider a policy $\pi \in \mathcal{C}^*_{\Pi,\Omega}$ and a policy $\pi' \in \Pi$ such that $d^{\pi'}(s,a) = 0$ wherever $d^\pi(s,a) = 0$. Then, we can write

$$\boldsymbol{v}^{\pi'} = \sum_{s,a} d^{\pi'}(s,a)\boldsymbol{r}(s,a) = \sum_{s,a} d^\pi(s,a)\frac{d^{\pi'}(s,a)}{d^\pi(s,a)}\boldsymbol{r}(s,a) = \mathbb{E}_{d^\pi}[\boldsymbol{r}'(s,a)], \tag{8}$$

where we used the importance sampling identity and defined $\boldsymbol{r}'(s,a) := d^{\pi'}(s,a)/d^\pi(s,a)\boldsymbol{r}(s,a)$. Per Def. 2, there exists $\boldsymbol{\omega} \in \Omega$ such that $\boldsymbol{\omega}^T\boldsymbol{v}^\pi \geq \boldsymbol{\omega}^T\boldsymbol{v}^{\pi'}$. Leveraging the fact that the value function can be expressed as $\boldsymbol{v}^\pi = \mathbb{E}_\pi[\tilde{\boldsymbol{r}}]$ as well as Eq. equation 8, we have $\boldsymbol{\omega}^T\,\mathbb{E}_\pi[\tilde{\boldsymbol{r}}] \geq \boldsymbol{\omega}^T\,\mathbb{E}_\pi[\tilde{\boldsymbol{r}}']$. This implies that $\mathbb{E}_\pi[\tilde{\boldsymbol{r}}] \in \mathcal{C}^*_{\boldsymbol{\mathcal{R}},\Omega}$ and concludes the proof. □

Remark 1. *Under the additional assumption of normalized rewards, we can prove the converse of Lemma 1. However, since we are are concerned with approximating $\mathcal{C}^*_{\boldsymbol{\mathcal{R}},\Omega}$ by $\mathcal{C}^*_{\Pi,\Omega}$ (and not the other way around), we refrain from making that additional assumption to prove the converse of Lemma 1.*

Theorem 3. *Let $l := l(\boldsymbol{\omega}^*(\theta))$ be a generic loss function that depends explicitly on the the optimal solution $\boldsymbol{\omega}^*(\theta)$ of Problem P_A. Then, the gradient of l with respect to the policy parameters is*

$$\nabla_\theta l = \begin{bmatrix} H^T(\theta) & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} G(\theta) & -I_M & \boldsymbol{e} \\ -D(\boldsymbol{\mu}^*(\theta)) & -D(\boldsymbol{\omega}^*(\theta)) & \mathbf{0} \\ \boldsymbol{e}^T & \mathbf{0} & \mathbf{0} \end{bmatrix}^{-T} \begin{bmatrix} \frac{dl}{d\boldsymbol{\omega}^*} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{9}$$

where $[G(\theta)]_{m,n} = 2g_m^T(\theta)g_n(\theta)$, $[H(\theta)]_{m,\cdot} = -\sum_n \omega_n^*(\theta)\nabla_\theta^T(g_n^T(\theta)g_m(\theta))$, $\boldsymbol{e}$ and $\mathbf{0}$ are the identity and zero vectors of appropriate dimensions, I_M is the identity matrix of dimension M and $D(\cdot)$ create a diagonal matrix from a vector.

Proof. We start by writing the Lagrangian of Problem P_A

$$L(\boldsymbol{\omega}, \lambda, \boldsymbol{\mu}) = \left\| \sum_{m=1}^M \omega_m g_m(\theta) \right\|^2 + \lambda\big(\sum_{m=1}^M \omega_m - 1\big) + \sum_{m=1}^M \mu_m\omega_m, \tag{10}$$

where $g_m(\theta) = \nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}[r_m(s,a)]$. Let $(\boldsymbol{\omega}^*(\theta), \lambda^*(\theta), \boldsymbol{\mu}^*(\theta))$ be the policy-dependant optimal primal and dual variables. The KKT conditions for stationarity, primal feasibility and complementary slackness are

$$\frac{\partial L}{\partial \omega_m^*(\theta)} = 2\sum_{n=1}^M \omega_n^*(\theta)g_n^T(\theta)g_m(\theta) + \lambda^*(\theta) + \mu_m^*(\theta) = 0,\ m = 1, 2 \ldots M \tag{11a}$$

$$\frac{\partial L}{\partial \lambda^*(\theta)} = \lambda^*(\theta)\big(\sum_{m=1}^M \omega_m^*(\theta) - 1\big) = 0 \tag{11b}$$

$$\frac{\partial L}{\partial \mu_m^*(\theta)} = \mu_m^*(\theta)\omega^*(\theta) = 0,\ m = 1, 2 \ldots M, \tag{11c}$$

By taking the gradient of the above equations with respect to the policy parameters, we obtain

$$2\sum_{n=1}^{M}\nabla_\theta\omega_n^*(\theta)g_n^T(\theta)g_m(\theta)+2\sum_{n=1}^{M}\omega_n^*(\theta)\nabla_\theta\big(g_n^T(\theta)g_m(\theta)\big)+$$
$$\nabla_\theta\lambda^*(\theta)+\nabla_\theta\mu_m^*(\theta)=0,\ m=1,2\dots M, \tag{12a}$$
$$\sum_{m=1}^{M}\nabla_\theta\omega_m^*(\theta)=0, \tag{12b}$$
$$\nabla_\theta\mu_m^*(\theta)\omega^*(\theta)+\mu_m^*(\theta)\nabla_\theta\omega^*(\theta)=0,\ m=1,2\dots M. \tag{12c}$$

Using matrix notation, we can rewrite the above system of equations as follows:

$$\begin{bmatrix} G(\theta) & -I_M & \boldsymbol{e} \\ -D(\boldsymbol{\mu}^*(\theta)) & -D(\boldsymbol{\omega}^*(\theta)) & \mathbf{0} \\ \boldsymbol{e}^T & \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \nabla_\theta^T\boldsymbol{\omega}^*(\theta) \\ \nabla_\theta^T\boldsymbol{\mu}^*(\theta) \\ \nabla_\theta^T\lambda^*(\theta) \end{bmatrix}=\begin{bmatrix} H(\theta) \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{13}$$

By inverting Eq. equation 13 and using the chain rule $\nabla_\theta l(\omega(\theta))=\frac{\partial l}{\partial\omega}\nabla_\theta l$ we obtain the claimed result. □

We observe that the solution of Eq. equation 13 requires knowledge of the optimal dual variable $\boldsymbol{\mu}^*(\theta)$, which is not directly available when we use the projected gradient descent solver of Algorithm 1; that solver only gives us the optimal primal variable $\boldsymbol{\omega}^*(\theta)$. However, we can substitute that optimal $\boldsymbol{\omega}^*(\theta)$ into the KKT conditions given by Eq. (11a–11c) and solve the resulting linear system of $M+1$ variables to find the optimal dual variables. The number of variables in that system depends only on the dimension of the reward signal M and not on the dimension of θ, therefore it adds minimal computational overhead. This step would not be necessary if we chose a primal-dual method for solving Problem P_A. However, primal-dual methods may add to the complexity of the implementation, which may not be worthy given the simplicity of Problem P_A. Exploring such a trade-off is an interesting direction for future work.

A.2 Pareto Stationarity: 2-dimensional case

In general, the problem of finding common ascent directions (Problem P_A) can be solved either *exactly* using the methods from the minimum norm points literature (as discussed in Sec. 3) or *approximately* using convex optimization methods. Due to the dimensionality of the gradient vector, the former class of methods are not applicable in our case. Therefore, we resorted to the projected gradient descent solver presented in Algorithm 1. This solver is essentially an internal optimizer which we run before every parameter update. Luckily, Problem P_A is convex and the gradient descent optimizer converges rapidly in practice with little tuning. Additionally, for the special case of two objectives, we can leverage the analytical solution to achieve further performance improvements. We re-state Problem P_A for convenience bellow:

$$\min_{\epsilon_1,\dots\epsilon_M}\quad\Big\{\Big\|\sum_{m=1}^{M}\epsilon_m\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_m(s,a)\right]\Big\|_2^2\ \text{s.t}\ \sum_{m=1}^{M}\epsilon_m=1,\ \epsilon_m\geq 0,\ \forall m\Big\}. \tag{P_A}$$

For $M=2$, the problem reduces to $\min_{\epsilon\in[0,1]}\|\epsilon\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_1(s,a)\right]+(1-\epsilon)\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_2(s,a)\right]\|_2^2$, which is a one-dimensional quadratic optimization problem of a single variable and has the following analytical solution:

$$\epsilon^*=\left[\frac{(\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_2(s,a)\right]-\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_1(s,a)\right])^T\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_2(s,a)\right]}{\|\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_2(s,a)\right]-\nabla_\theta\,\mathbb{E}_{d^{\pi_\theta}}\left[r_1(s,a)\right]\|_2^2}\right]_+, \tag{14}$$

where $[\cdot]_+$ represents the clipping operator, i.e., $[x]_+=\max\left(\min\left(x,1\right),0\right)$. Geometrically, Eq. equation 14 implies that minimal-norm point lies either on the boundary on the convex hull ($\epsilon=0$ or $\epsilon=1$) and the desired descent direction is parallel to corresponding gradient or is equal to the vector perpendicular to the difference of the gradients. The former one is a corner case and occurs when the ascent direction for one of the objectives is a non-decreasing direction for the other one. While this is only applicable to $M=2$, it enables us to replace the inner projected gradient descent optimizer in Algorithm 1 with the analytical solution in Eq. equation 14. Additionally, this special case is applicable to several real-world cases since there are several problems that have the *reward-cost* structure.

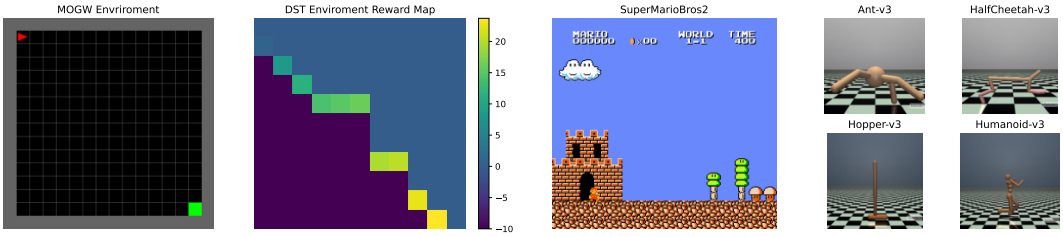


Figure 5: OpenAI Gym enviroments used in our experimental results.

B Experimental Details

B.1 Environments

Our implementation uses modified versions of 4 OpenAI Gym enviroments (Fig. 5). The first two are synthetic domains and act as a proof-of-concept for our method, while the other two are complex, real domains. The details are listed below:

Multi-Objective GridWorld (MOGW): A multi-objective variant of the classic gridworld environment. The agent starts from the top-left corner, moves in a 16×16-grid and receives a *reward* equal to 1 when it reaches the bottom-right corner. Each action results in a small *time penalty* of -0.1. Any policy that results in a trajectory that minimizes the taxicab distance between the agent's starting position and the goal is Pareto optimal. Our implementation uses the `gym-minigrid`[1] enviroment, which we modify to output the aforementioned 2-dimensional reward.

Deep Sea Treasure (DST): A classic multi-objective reinforcement learning environment. The agent is located in a 10×11-grid and controls a submarine searching for treasures. There are multiple treasures the *value* of which increases as their distance increases. The agent receives a *time penalty* of -0.1 for each action. The Pareto front of non-dominated policies can be easily extracted (49) and is also shown on Fig. 1. As in the MOGW case, we use the `gym-minigrid` package and extend it to the reward map shown in Fig. 5.

Multi-Objective SuperMario (MOSM): We use a modified, multi-objective variant of the popular video game. There are 5 scalar rewards: *x-pos*: the difference in Mario's horizontal position between current and last time point, *time*: a small time penalty of -0.1, *deaths*: a large negative penalty of -100 each time Mario dies, *coin*: rewards for collecting coins, and *enemy*: rewards for eliminating an enemy. We use the `gym-super-mario-bros`[2] package and the multi-objective extension developed in (24).

Multi-Objective MuJoCo (MOMU): We focus on locomotion tasks and modify the environments provided by the MuJoCo physics simulator to output vector rewards. To do so, we create a custom wrapper for the built-in OpenAI Gym interface. There are 2 scalar rewards: *x-pos*: the difference in the robot's current and last horizontal position, *r-pos*: a positive reward for moving in circular trajectories.

B.2 Implementation Details

We implement our algorithm in PyTorch building on top of the `torch-ac` package[3]. Our implementation extends the Proximal Policy Optimization (PPO) algorithm to handle our PPA loss function (Eq. equation 6). We use the clipped version of PPO with the clip ratio threshold set to 0.2. We set the GAE parameter to $\lambda = 0.95$ and the discount factor to $\gamma = 0.99$.

Network Architecture: We use a consistent network architecture across all our simulation for both the actor and the critic: Two hidden layers of 64 and 64 neurons each and a ReLU non-linearity. In the discreet action case (MOGW, DST, MOSM), we append the actor network with an output layer of dimension equal to the number of actions. In the MuJoCo enviroments, where the actions are

[1] https://github.com/maximecb/gym-minigrid
[2] https://github.com/Kautenja/gym-super-mario-bros
[3] https://github.com/lcswillems/torch-ac

continous, we use that output layer to parametrize a multi-dimensional Gaussian distribution, via its mean and variance, assuming uncorrelated actions. Each non-terminal layer has a ReLU non-linearity. Even though the hyperbolic tangent is known to be preferred for reinforcement learning applications, we experimentally observed that it leads to vanishing gradients and slows down learning. This may be due to the fact that the PPA loss function requires second-differentiation. In the SuperMario enviroment, we worked on the pixel domain and two convolutional layer of 32 and 64 layers as feature extractors, which were shared between the actor and the critic.

Hyperparameters and Training: To train the neural network, we use the Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.999$) with a learning rate of 0.0001. In the MOGW, DST and MOMU domains, we optimize the parameters of the actor and critic separately. In the MOSM domain, given that the actor and critic share the convolutional layers, we sum their losses (using weights 1 and 0.5, respectively) and use a common optimizer. We run all of our simulations in the Google Cloud Platorm using 48 vCores and one NVIDIA Tesla T4 GPU. For the MOGW and DST environments, we train the agent for 0.5M frames and for the MOSM and MOMU for 1M frames. For each 512 frames we perform one update of the network parameters iterating over 10 epochs of the collected data and using a mini-batch size of 64.