Journal of Artificial Intelligence Research 70 (2021) 319-349 Submitted 07/2020; published 01/2021

Efficient Multi-objective Reinforcement Learning via Multiple-Gradient Descent with Iteratively Discovered Weight-Vector Sets

Huixin Zhan HUIXIN.ZHAN@UTSA.EDU
Yongcan Cao YONGCAN.CAO@UTSA.EDU
Department of Electrical and Computer Engineering
University of Texas, San Antonio, TX 78249, USA

Abstract

Solving multi-objective optimization problems is important in various applications where users are interested in obtaining optimal policies subject to multiple (yet often conflicting) objectives. A typical approach to obtain the optimal policies is to first construct a loss function based on the scalarization of individual objectives and then derive optimal policies that minimize the scalarized loss function. Albeit simple and efficient, the typical approach provides no insights/mechanisms on the optimization of multiple objectives due to the lack of ability to quantify the inter-objective relationship. To address the issue, we propose to develop a new efficient gradient-based multi-objective reinforcement learning approach that seeks to iteratively uncover the quantitative inter-objective relationship via finding a minimum-norm point in the convex hull of the set of multiple policy gradients when the impact of one objective on others is unknown *a priori*. In particular, we first propose a new PAOLS algorithm that integrates pruning and approximate optimistic linear support algorithm to efficiently discover the weight-vector sets of multiple gradients that quantify the inter-objective relationship. Then we construct an actor and a multi-objective critic that can co-learn the policy and the multi-objective vector value function. Finally, the weight discovery process and the policy and vector value function learning process can be iteratively executed to yield stable weight-vector sets and policies. To validate the effectiveness of the proposed approach, we present a quantitative evaluation of the approach based on three case studies.

1. Introduction

In recent years, the application of reinforcement learning (RL) in tasks with high-dimensional sensory inputs has shown the potential of creating artificial agents that can learn to accomplish a number of challenging tasks, including the Atari games (Mnih, et al., 2015; Guo, et al., 2014; Schaul, et al., 2015; Wang, et al., 2016; Van Hasselt, Guez, & Silver, 2016; Oh, et al., 2015; Nair, et al., 2015), Go (Maddison, et al., 2014; Silver, et al., 2016, 2018), and self-driving cars (Pan, et al., 2017). However, the approaches developed therein mainly focus on finding a single usable strategy, without considering the trade-off among potential alternatives that can increase one objective's value at the cost of another.

In the multi-objective setting, the completion of a task requires the simultaneous satisfaction of multiple objectives such as balancing power consumption and performance in Web servers (Tesauro, et al., 2008). Such problems can be modeled as multi-objective Markov decision processes (MOMDPs) and solved by some existing multi-objective reinforcement learning (MORL) algorithms (Tesauro et al., 2008; Nguyen, 2018; Tajmajer, 2018; Abels, et al., 2018; Van Moffaert, Drugan, & Nowé, 2013; Vamplew, Dazeley, & Foale, 2017) based on the assumption that either the weighting factor for different objective functions or the ordering information is available. The solution for parameterized algorithms assumes the weighting factor for different objective functions can be obtained either directly (*i.e.*, known *a priori*) or indirectly (through learning). Hence, the linear scalarization approach can be used to transfer the multi-objective problem to a single-objective problem, which can be solved via one or several parameterized single-objective optimization problems. For example, an approach was proposed to use both linear weighted sum and nonlinear thresholded lexicographic ordering methods to develop a multi-objective deep RL framework that includes both single- and multi-policy strategies (Nguyen, 2018). An architecture that uses separated deep Q-networks (DQNs) was proposed to control the agent's behavior with respect to particular objectives (Tajmajer, 2018). Each DQN has an additional decision value output that acts as a dynamic weight that is used while summing up Q-values. Another method was proposed to generalize across weight changes and high-dimensional inputs by proposing a multi-objective Q-network in a dynamic weight setting (Abels et al., 2018). Other parameterized methods include the convex hull (Roijers, Whiteson, & Oliehoek, 2015), the varying parameters approaches (Abels et al., 2018; Liu, Xu, & Hu, 2015), the constraint method (Konak, Coit, & Smith, 2006), the sequential method (Nakayama, Yun, & Yoon, 2009), and the max-min method (Lin, 2005). When the weighting factor is assumed to be unobtainable, parameter-free multi-objective optimization techniques usually apply an ordering of the importance of the vector objective functions, *i.e.*, the use of softmax-epsilon selection based on a nonlinear action selection operator (Vamplew et al., 2017). The agents incorporate an action selection function that is defined as an ordering over the Q-values. However, this optimization process is usually augmented by an interactive procedure of linear preferences to specify the ordering. Due to the above limitations, new MORL methods are required when neither the weighting factor for different objective functions nor the ordering can be obtained *a priori*.

Recently, some interesting research was conducted to propose an upper bound for the multi-objective loss and prove that optimizing this upper bound via gradient-based multi-objective optimization yields a Pareto optimal solution (Sener & Koltun, 2018). The Frank-Wolfe solver is used to find a minimum-norm point in the convex hull of the set of input points. This paper provides a new perspective for parameter-free objective balancing when the gradients for all objectives are considered as the min-norm points in the convex hull. However, the proposed method has two major limitations. First, this paper shows success in large-scale multi-label learning tasks without addressing complex continuous space planning tasks. Second, when the number of the vector value function grows, the Frank-Wolfe solver is not applicable because the constrained convex optimization problem cannot be directly formulated by minimizing the linear approximation of multiple differentiable and convex real-valued functions.

In contrast to the previous work, we focus on discovering the weight-vector sets of multiple gradients of the loss instead of learning the weights for the vector value functions and scalarizing it using the weights. Hence, the proposed approach is a parameter-free approach because the objective functions are treated individually without employing any scalarization. Here is a brief description of the main idea behind the proposed approach.

Because some objectives are more sensitive than others, we seek to reach a designed point in the Pareto set such that it is impossible to reduce the local loss of any objective without increasing at least one other loss. In other words, we aim to discover the weight-vector sets of multiple gradients towards convergence such that when moving in the direction of this learned scalarization of multiple gradients of the loss, we can finally reach a designed point in the Pareto set. Therefore, it is essential to learn the gradient directions that quantify the relationship among these objectives and then train a decision policy that can balance these objectives based on the quantitative inter-objective relationship, when the values for multiple objectives are considered a vector and balancing them is required in the decision making process.

To address this issue, this paper focuses on proposing a new efficient gradient-based multi-objective reinforcement learning approach via multiple-gradient descent with discovered weight-vector sets. The proposed approach is a gradient-based multi-policy method that focuses on efficiently discovering an approximate convex coverage set and then designing actor-critic policy learning algorithms. The proposed approach has three main contributions. First, we propose a new learning algorithm that integrates pruning and approximate optimistic linear support algorithm, named PAOLS, to efficiently learn the weights that quantify the inter-objective relationship. This algorithm allows us to compute a minimum-norm point in the convex hull of the policy set and find the corresponding weights of multiple gradients. Different from the standard optimistic linear support, which is developed for multi-objective planning (Mossalam, et al., 2016), PAOLS is motivated by the need to address multi-objective reinforcement learning. Second, instead of using a scalarized Q-value in the reinforcement learning based action selection, the proposed method supports vector value functions. In particular, all value functions can be used sequentially in the training of the actor network to update the control policy. Third, our multi-objective optimization is applicable in high-dimensional continuous action spaces, such as robotics motion planning. Lastly, we develop an iterative framework and provide rigorous analysis on the stability of the proposed approach. To our best knowledge, this is the first time that actor critic with learned quantifiable inter-objective relationship is developed to solve MORL using vector value functions with rigorous stability analysis. To verify the effectiveness and advantages of the proposed approach, we finally provide three case studies and compare the performance of the proposed approach with some state-of-the-art approaches.

2. Previous Work

In this section, we will provide a brief review of related technical approaches and the motivation of the work.

Typical multi-objective optimization (MOO) studies the problem of optimizing a set of possibly conflicting objectives. One main strategy to solve this problem is the scalarization approach (Ward & Lee, 2001; Nguyen, 2018; Tajmajer, 2018; Vamplew et al., 2017; Van Moffaert et al., 2013), where one or several single-objective optimization problems are solved. Two major disadvantages of these approaches are (1) the choice of the weighting factors is needed, leading to the burden of choosing them in the model, and (2) scalarization only results in a properly efficient solution (Ward & Lee, 2001). Another typical approach is the multi-criteria optimization. This approach usually uses an ordering of different criteria, *i.e.,* an ordering of importance of different objectives (Miettinen & Mäkelä, 1995; Vamplew et al., 2017). In this case, the ordering needs to be specified, leaving the decision maker with the burden of deciding priorities among alternatives.

When the weighting factors for different objectives are unavailable, some important approaches have been developed. For example, a Pareto Q-learning approach was proposed to learn the entire Pareto front when each state-action pair is sufficiently sampled, without the knowledge of the weighting factors (Van Moffaert & Nowé, 2014). A multi-objective version of the Bellman optimality operator was proposed to learn a single parametric representation of all optimal policies over the space of preferences (Yang, Sun, & Narasimhan, 2019). A deep optimistic linear support learning algorithm was proposed to compute the convex coverage set containing all potential optimal solutions of the convex combinations of the objectives via using features from the high-dimensional inputs (Mossalam et al., 2016). These studies provided very interesting and important approaches for multi-objective optimization without knowing the weighting factor. However, an explicit quantitative analysis of the inter-objective relationship has not been conducted.

Another class of relevant approaches that have been proposed is the gradient-based MOO. Gradient-based methods play an important role in multi-objective optimization. For example, the gradient-based multi-objective optimization methods (Fliege & Svaiter, 2000; Désidéri, 2012) use multi-objective Karush-Kuhn-Tucker (KKT) conditions (Gordon & Tibshirani, 2012) to find a descent direction that decreases all objectives. This approach was then extended to the case of a stochastic gradient descent (Peitz & Dellnitz, 2016; Poirion, Mercier, & Désidéri, 2017). One key disadvantage of these approaches is that they scale poorly with the dimensionality of the gradients. An importance sampling approach was proposed for multi-objective reinforcement learning that can achieve good control performance in partially observable Markov decision processes with few data (Shelton, 2001). A new constrained policy gradient reinforcement learning approach that integrates policy gradient reinforcement learning algorithms and techniques used in nonlinear programming was proposed to maximize the long-term average intrinsic reward under the inequality constraints induced by the extrinsic rewards (Uchibe & Doya, 2007). Two new manifold-based algorithms that combine episodic exploration and importance sampling were proposed to efficiently learn a manifold in the policy parameter space such that its image in the objective space accurately approximates the Pareto frontier (Parisi, Pirotta, & Peters, 2017). There are also numerous other gradient-based methods to solve multi-objective optimization (Pirotta, Parisi, & Restelli, 2015; Parisi, Pirotta, & Restelli, 2016; Parisi, et al., 2014a, 2014b; Pinder, 2016). For example, policy gradient techniques were developed to approximate the Pareto frontier in multi-objective Markov decision processes (Pirotta et al., 2015; Parisi et al., 2016, 2014a, 2014b). Note that an explicit quantitative study of the inter-objective relationship is also lacking in these studies.

A recent study proposed the optimization of an upper bound for the multi-objective loss via a gradient-based method and show success in large-scale multi-label learning tasks (Sener & Koltun, 2018). However, whether the method can be extended to MORL (MOO on reinforcement learning tasks) remains unknown. In complex continuous space planning tasks, it is typical that both the large number of the gradients and the high dimensionality of the gradients need to be considered. When a large number of gradients are employed, the Frank-Wolfe solver is not directly applicable. Different from this approach, we focus on solving the problem of optimizing the upper bound of the multiple value functions of the learned policy set via an efficient PAOLS algorithm whose running time is bounded by some polynomial function of the number of gradients. This algorithm is more computationally efficient than the optimistic linear support method (Mossalam et al., 2016) to compute all potential optimal solutions of the convex combinations of the objective. In particular, this proposed algorithm can gradually improve the approximation of the convex coverage set via maintaining a lexicographically maximum order of the gradients (Ehrgott, 1995). In addition, PAOLS is scalable to both the number of gradients and the dimension of gradients.

As the proposed method is a multi-policy MORL approach, we will also briefly review some relevant work on multi-policy optimization. The existing multi-policy algorithms mainly focus on learning multiple policies that form an approximation of the Pareto front (Vamplew, et al., 2011). For example, some earlier studies focused on combining the basic RL algorithms with an online estimate of the minimally approachable target set (Mannor & Shimkin, 2002, 2004). This method does not hold when the vector value function is monotonically increasing in all objectives without constraints because the approachability in the presence of a finite set of steering gradient directions cannot be proved. Another approach was proposed to compute the set of the multiple criteria on the convex hull for multiple policies (Barrett & Narayanan, 2008). Note that the policies are still built via computing the set's maximum weighted sum value. In contrast, the scalarization of the original vector value functions is not needed in the proposed method. Instead, it focuses on learning the weights of gradients that can quantify inter-objective relationship via computing a discrete approximation of the set of gradients for high-dimensional continuous action spaces.

3. Preliminaries

In this section, we will provide some background information, including the multi-objective Markov decision processes (MOMDP), the multi-objective decision making setting, and the MORL setting.

3.1 MOMDP

The readers are referred to the survey (Roijers, et al., 2013) for a thorough introduction of algorithmic solutions for MOMDP. By following the typical MOMDP settings (Roijers et al., 2013), we here adopt a finite MOMDP that is a tuple $< X, \mathcal{A}, T, R, \mu, \gamma >$, where

- X is a finite set of states,
- $\mathcal{A}$ is a finite set of actions,
- $T : X \times \mathcal{A} \times X \rightarrow [0, 1]$ is a state transition function specifying, for each state, action, and next state, the probability that the next state occurs,
- $R : X \times \mathcal{A} \times X \rightarrow \mathbb{R}^I$ describes a vector of I rewards, one for each objective,
- $\mu : X \rightarrow [0, 1]$ is a probability distribution over initial states, and
- $\gamma \in [0, 1)$ is a discount factor specifying the relative importance of immediate rewards.

A control policy π is a map that specifies the probability of taking a specific action at any given state. If the policy π is stationary, *i.e.*, it conditions only on the current state, it can be formalized as $\pi : X \times \mathcal{A} \to [0, 1]$. The vector value function $\mathbf{V}^\pi = [\mathbf{V}_1^\pi, \cdots, \mathbf{V}_I^\pi]' \to \mathbb{R}^I$ specifies the expected cumulative discounted reward vector $\mathbf{r}$

$$\mathbf{V}^\pi(x) = E\left[\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{t+k+1} | \pi, x_t = x\right], \tag{1}$$

where $E[\cdot]$ is the expectation operator and $\mathbf{r}$ is the immediate vector reward for all objectives under the policy π.

3.2 Multi-objective Decision Making

In this paper, we consider the problem when multiple objectives $\mathcal{T}_i,\ i = 1, \cdots, I$, need to be optimized for a given mission, where I denotes the number of objectives. For example, in robotic locomotion, maximizing forward velocity but minimizing joint torque and impact with the ground, may result in numerous options to consider. We use $\mathbf{V}^\pi = [\mathbf{V}_1^\pi, \cdots, \mathbf{V}_I^\pi]' \in \mathbb{R}^I$, to represent the vector value function for $\mathcal{T}_i,\ i = 1, \cdots, I$, subject to the control policy π. An approach to optimize objectives $\mathcal{T}_i,\ i = 1, \cdots, I$, is to construct a scalarized value function of the form $V_w^\pi \in \mathbb{R}^1 = \mathcal{W}\mathbf{V}^\pi$, where $\mathcal{W} = [w_1, \cdots, w_I]$ satisfies $w_i \geq 0,\ i = 1, \cdots, I$, and $\sum_{i=1}^I w_i = 1$, where the weight w_i specifies how much each objective contributes to the scalarized objective. Hence, a multi-objective optimization problem can be converted to a single-objective optimization problem. This scalarization approach provides no quantitative insights/mechanisms on the optimization of multiple objectives due to the lack of ability to quantify the inter-objective relationship.

Instead of employing a scalarized method, this paper focuses on developing a multi-policy method. The main idea of the paper is motivated by the AOLS method (Roijers, et al., 2014b). The existing AOLS method is an approximate MDP solver to produce control policies that are very close to the optimal one, which can be solved by OLS (Roijers, et al., 2014a). Because OLS may be computationally infeasible, AOLS is appealing since it is computationally feasible although still inefficient. One main objective of the paper is to address the inefficiency issue of the existing AOLS method. In particular, our goal is to obtain the ϵ-approximate solution more efficiently than AOLS via eliminating unnecessary searches in AOLS. Because the one-time discovered weight-vector sets associated with the ϵ-approximate solution and the obtained control policy are interdependent, our second objective is to develop a new actor-critic network to learn the optimal control policy associated with the one-time discovered weight-vector sets. Finally, we show that an iterative process can yield stable solutions.

3.3 Multi-objective Reinforcement Learning Setup

The typical training in a single-objective reinforcement learning setting, given by $\mathcal{T} = \{\mathcal{L}(x_1, a_1, \cdots, x_H, a_H), \mu(x_1), T(x_{t+1}|, x_t, a_t), H\}$, consists of a loss function $\mathcal{L}$ defined on the trajectory $(x_1, a_1, \cdots, x_H, a_H)$ of length H, a distribution over initial observations $\mu(x_1)$, a transition distribution $T(x_{t+1}|x_t, a_t)$, and an episode length H. In the presence of multiple objectives, say, I objectives, we adopt the K-shot classification setting that uses

K input/output pairs from each objective, for a total of IK data points for I objectives (please refer to Yoo, et al., 2018 for more details). In other words, we adopt K samples, namely, trajectories, for each of the I objectives in the training of reinforcement learning algorithms. We consider a uniform distribution over trajectories $p(\mathcal{T})$ that we want our model $f_\theta : X \to \mathbf{V}$ to fit. The model is trained to fit a new trajectory $\mathcal{T}_i$ drawn from $p(\mathcal{T})$ from only K samples drawn from μ_i and the loss $\mathcal{L}_{\mathcal{T}_i}$ that is generated by $\mathcal{T}_i$. Formally, the model's parameters θ will be updated via the single objective gradient update as

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta), \tag{2}$$

where α is the step size.

At the end of each IK-shot training, the model parameters are trained by optimizing the performance of $f_{\theta_i'}$ with respect to θ across all objectives. Formally,

$$\theta = \theta - \alpha \nabla_\theta \sum_{i=1}^{I} w_i \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}), \tag{3}$$

where w_i is some static or dynamically computed weights for the objective i. The K-shot learning procedure is shown in Algorithm 5.

Recall that the objectives are possibly conflicting. Consider a parametric hypothesis per objective as $f^i(x; \theta_{IO}, \theta_{IS}^i) : X \to \mathbf{V}_i^\pi$, such that some parameters (θ_{IO}) are shared between objectives and some (θ_{IS}^i) are objective-specific. The network architecture detail for this parametric hypothesis setting can be found in Subsection 6.1.

A basic intuition behind the weighted summation formulation (3) is that it is impossible to define global optimality in the MORL setting. For instance, for objective i_1, solution θ is better when $\hat{\mathcal{L}}^{i_1}\left(\theta_{IO}, \theta_{IS}^{i_1}\right) < \hat{\mathcal{L}}^{i_1}\left(\overline{\theta}_{IO}, \overline{\theta}_{IS}^{i_1}\right)$, while for objective i_2, solution $\overline{\theta}$ may be better when $\hat{\mathcal{L}}^{i_2}\left(\theta_{IO}, \theta_{IS}^{i_2}\right) > \hat{\mathcal{L}}^{i_2}\left(\overline{\theta}_{IO}, \overline{\theta}_{IS}^{i_2}\right)$, where $\hat{\mathcal{L}}$ denotes the approximated loss using samples. Hence, it is impossible to compare the two solutions without a pairwise importance of objectives, which is typically unavailable.

We here formulate the multi-objective optimization (MOO) of MORL using a vector value loss $\mathbb{L}$:

$$\min_{\theta_{IO}, \theta_{IS}^1, \cdots, \theta_{IS}^I} \mathbb{L}(\theta_{IO}, \theta_{IS}^1, \cdots, \theta_{IS}^I) = \min_{\theta_{IO}, \theta_{IS}^1, \cdots, \theta_{IS}^I} [\hat{\mathcal{L}}^1\left(\theta_{IO}, \theta_{IS}^1\right), \cdots, \hat{\mathcal{L}}^I\left(\theta_{IO}, \theta_{IS}^I\right)]', \tag{4}$$

where $\hat{\mathcal{L}}^i$, $i = 1, \cdots, I$, is a smooth loss function. The goal of multi-objective optimization is to achieve Pareto optimality, defined below.

Definition 1 (Pareto optimality for MORL)**.**

- *A solution θ dominates another solution $\overline{\theta}$ if $\hat{\mathcal{L}}^i\left(\theta_{IO}, \theta_{IS}^i\right) \leq \hat{\mathcal{L}}^i\left(\overline{\theta}_{IO}, \overline{\theta}_{IS}^i\right)$ for every objective i and $\mathbb{L}(\theta_{IO}, \theta_{IS}^1, \cdots, \theta_{IS}^I) \neq \mathbb{L}(\overline{\theta}_{IO}, \overline{\theta}_{IS}^1, \cdots, \overline{\theta}_{IS}^I)$.*
- *A solution $\theta^\star$ is called* Pareto optimal *if there exists no solution θ that dominates $\theta^\star$.*

It is not difficult to observe that different gradient weights $w_i,\ i = 1, \cdots, I$, will have different impacts on the optimization of multiple objectives based on (3). In fact, the weights $w_i,\ i = 1, \cdots, I$, should be selected based on how important the optimization of one objective will impact others. Hence, our objective is to not only obtain the Pareto optimal solution but also learn the gradient weights $w_i,\ i = 1, \cdots, I$, via, e.g., (8), that are used to explain the inter-objective relationship quantitatively.

Let $\Omega_i = \nabla_{\theta_{IO}} \hat{\mathcal{L}}^i(\theta_{IO}, \theta^i_{IS}),\ i = 1, \cdots, I$, be the local gradients. According to the Karush-Kuhn-Tucke (KKT) conditions (Gordon & Tibshirani, 2012), a solution θ is Pareto stationary if

- There exist $w_1, \cdots, w_I \geq 0$ such that $\sum_{i=1}^I w_i = 1$ and $\sum_{i=1}^I w_i \Omega_i = \mathbf{0}$,
- For all objectives i, $\nabla_{\theta^i_{IS}} \hat{\mathcal{L}}^i(\theta_{IO}, \theta^i_{IS}) = \mathbf{0}$.

In addition, if the objective functions $\hat{\mathcal{L}}^i,\ i = 1, \cdots, I$, are convex and continuously differentiable, a Pareto stationary solution θ is also a Pareto optimal solution (Hosseinzade & Hassanpour, 2011). Hence, if $\hat{\mathcal{L}}^i,\ i = 1, \cdots, I$ are convex and continuously differentiable, solving the optimization in (4) is the same as finding the Pareto stationary solution based on the two KKT conditions.

Typically, the second KKT condition holds when freezing the objective-specific parameters θ^i_{IS}. Any solution that satisfies these conditions is called a Pareto stationary point. Although every Pareto optimal point is Pareto stationary, the reverse may not be true. For the first KKT condition, namely, $\sum_{i=1}^I w_i \Omega_i = \mathbf{0}$, one needs to learn both the weights $w_i,\ i = 1, \cdots, I$ and the policy θ such that $\left\| \sum_{i=1}^I w_i \Omega_i \right\|_2^2$ is minimized (to zero). This optimization problem admits a general solution $\overline{v}$ in the convex hull of the family of vectors $\Omega_i,\ i = 1, \cdots, I$,

$$\overline{v} = \left\{ \sum_{i=1}^I w_i \Omega_i \middle| w_i \geq 0\ , \sum_{i=1}^I w_i = 1 \right\}. \tag{5}$$

Equivalently, the minimization of $\left\| \sum_{i=1}^I w_i \Omega_i \right\|_2^2$ can be written as learning the weights $\mathcal{W}$ corresponding to solving the minimum-norm element in $\overline{v}$. Hence, the weights can be learned via

$$\mathcal{W} = \arg\min_{\mathcal{W}} \left\{ \left\| \sum_{i=1}^I w_i \Omega_i \right\|_2^2 \Big| \sum_{i=1}^I w_i = 1, w_i \geq 0 \right\}, \tag{6}$$

where $\mathcal{W} = [w_1, \cdots, w_I]$.

Because the dimension of θ_{IO}, typically described by neural networks, can be very large (in thousands or more), it is very computationally expensive to compute the minimum-norm point in the convex set. To reduce the dimension, by following the idea in (Sener & Koltun, 2018), define a new representation $\mathbf{z} = [z_1, \cdots, z_m] \in \mathbb{R}^M$, where $z_i = g_i(x; \theta_{IO})$. Ω_i can be rewritten using the chain rule as

$$\Omega_i = \frac{\partial \hat{\mathcal{L}}(\theta_{IO}, \theta^i_{IS})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \theta_{IO}}. \tag{7}$$

Because each vector Ω_i includes the term $\frac{\partial \mathbf{z}}{\partial \theta_{IO}}$, which is independent of $\mathcal{W}$, one can obtain

$$\sum_{i=1}^{I} w_i \Omega_i = \left(\sum_{i=1}^{I} w_i \frac{\partial \hat{\mathcal{L}}(\theta_{IO}, \theta_{IS}^i)}{\partial \mathbf{z}} \right) \frac{\partial \mathbf{z}}{\partial \theta_{IO}}.$$

When $\frac{\partial \mathbf{z}}{\partial \theta_{IO}}$ has linearly independent rows, which is typically true when the dimension of θ_{IO} is large, the optimization problem in (6) can be equivalently written as

$$\mathcal{W} = \arg\min_{\mathcal{W}} \left\{ \left\| \sum_{i=1}^{I} w_i \nabla_{\mathbf{z}} \hat{\mathcal{L}}^i(\theta_{IO}, \theta_{IS}^i) \right\|_2^2 \Big| \sum_{i=1}^{I} w_i = 1, w_i \geq 0 \right\}. \tag{8}$$

Because the dimension of $\mathbf{z}$ can be much smaller than that of θ_{IO}, the computation complexity can be reduced significantly.

Since the only requirement for the selection of the loss function $\mathbb{L}$ is that each $\hat{\mathcal{L}}^i$ is convex, one can select $\mathbb{L}$ as $-\mathbf{V}^\pi$ because each function in $-\mathbf{V}^\pi$, namely, $-\mathbf{V}_i^\pi$, is a convex combination of the negative immediate vector reward $-\mathbf{r}_{t+k+1}$, defined in (1).

4. The Proposed Approach

In this section, we will provide a detailed description of the proposed approach, including the development of the PAOLS method to efficiently learn the weights of multiple gradients and the design of an actor and a multi-objective critic that can be used to update the objective values and the action policy. Different from the standard optimistic linear support, which is developed for multi-objective planning (Mossalam et al., 2016), PAOLS is motivated by the need to address multi-objective reinforcement learning.

Before presenting the approach to compute the minimum-norm point in the convex hull, we first prove that the minimum-norm admits at least one realization of a minimum.

Theorem 1. *The minimum-norm of $\overline{v}$ admits at least one realization of a minimum in $\overline{v}$. Moreover, $\overline{v}$ has a unique minimum-norm point.*

Proof. Because $\overline{v}$ is a convex set, its norm must have a minimum, which corresponds to at least one realization. Hence, the first statement is true. We next prove the second statement by contradiction.

Assume that there are two realizations of the minimum-norm in $\overline{v}$ given by $\|u_1\|_2 = \|u_2\|_2$. Since $\overline{v}$ is convex, $v = (1-\epsilon)u_1 + \epsilon u_2 = u_1 + \epsilon u_2 1 \in \overline{v}$ for all $\epsilon \in [0,1]$, where $u_{21} = u_2 - u_1$. Because u_1 and u_2 are two realizations of the minimum-norm, it follows that $\|v\|_2 \geq \|u_1\|_2$, namely,

$$\|u_1 + \epsilon u_{21}\|_2^2 = \langle u_1 + \epsilon u_{21}, u_1 + \epsilon u_{21} \rangle \geq \langle u_1, u_1 \rangle = \|u_1\|_2^2, \tag{9}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. Because $\langle u_1 + \epsilon u_{21}, u_1 + \epsilon u_{21} \rangle = \langle u_1, u_1 \rangle + 2\epsilon \langle u_1, u_{21} \rangle + \epsilon^2 \langle u_{21}, u_{21} \rangle$, (9) holds if and only if $2\epsilon(u_1, u_{21}) + \epsilon^2(u_{21}, u_{21}) \geq 0$ holds. Then $(u_1, u_{21}) \geq 0$ must hold because otherwise $2\epsilon(u_1, u_{21}) + \epsilon^2(u_{21}, u_{21}) < 0$ for a sufficiently small ϵ. Because (9) holds for all $\epsilon \in [0,1]$, when $\epsilon = 1$,

$$\langle u_1 + \epsilon u_{21}, u_1 + \epsilon u_{21} \rangle > \langle u_1, u_1 \rangle$$

unless $u_{21} = \mathbf{0}$, namely $u_1 = u_2$. Hence, $\overline{v}$ has a unique minimum-norm point. □

The intuition to find the minimum-norm point and the associated marginal weight in the convex hull defined in (5) is that we can find model parameters that are sensitive to changes in training each objective, such that small changes in the parameters will produce large improvements on the loss function of any objective drawn from $p(\mathcal{T})$. When altered in the direction of the gradients, these vectors are often associated with the criteria that have already achieved a fair degree of convergence.

Remark 1. *The direction of the minimum-norm vector $\overline{V}$, the weighted sum of vectors Ω_i with a minimum norm, is mostly influenced by the gradients of small norms in the family of vectors, as illustrated in Figure 1 when $I = 2$ and $\mathbf{z} \in \mathbb{R}^2$. In the course of the K-shot optimization, these vectors are often varied towards convergence. In the visualiza-*

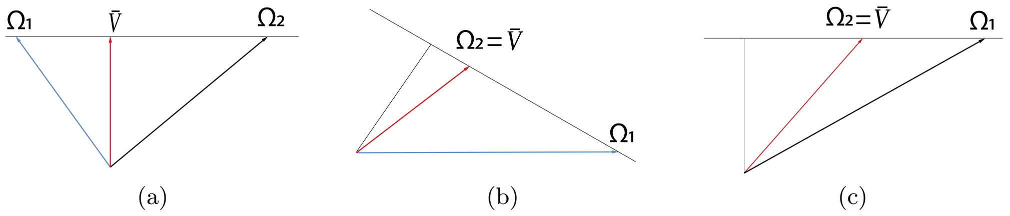


Figure 1: Possible directions of $\overline{V}$ with respect to the two gradients Ω_1 and Ω_2 when $I = 2$ and $\mathbf{z} \in \mathbb{R}^2$.

tion of the mininum-norm point in the convex hull of two dimensional vectors Ω_1 and Ω_2 ($\min_{w_1 \in [0,1]} \|w_1\Omega_1 + (1 - w_1)\Omega_2\|_2$), we can see that the solution is either a vector itself or a perpendicular vector. As the computational geometry suggests, one of the following cases occurs: (1) the solution to this optimization problem is 0 and the resulting point is Pareto stationary; and (2) the solution defines a descent direction common to all the criteria (Désidéri, 2012).

We next discuss how to compute the minimum-norm point. We define the weight $\mathcal{W}$ at the minimum-norm point as the marginal weight. A typical approach is the approximate optimistic linear support (AOLS) approach (Roijers et al., 2014b). AOLS is an ε-approximate MDP solver that can compute a set of policies for which the scalarized value for each possible weight is at most a factor ε away from the optimal value for the MDP. AOLS is developed to deal with the computational infeasibility issue of optimistic linear support (OLS) although OLS can theoretically obtain solution to an MOMDP with linear scalarizations. Before discussing the AOLS approach, it is necessary to define the undominated set (US).

Definition 2. *The undominated set $U_S(\mathcal{V})$ is the subset of all possible vectors $\mathcal{V}$ that are optimal for some weight vector $\mathcal{W}$ with a scalarized comparison:*

$$U_S(\mathcal{V}) = \{\mathcal{V} | \forall \mathcal{V}' \in \mathbf{V}^\pi : \mathcal{W}\mathcal{V}' \leq \mathcal{W}\mathcal{V}, \exists \mathcal{W}\}. \tag{10}$$

where $\mathcal{W}\mathcal{V}'$ and $\mathcal{W}\mathcal{V}$ are the scalarized values of $\mathcal{V}'$ and $\mathcal{V}$ with respect to a given weight vector $\mathcal{W}$.

We now describe AOLS in model detail. The AOLS is a method that can gradually improve the approximation of US. Given a maximum improvement threshold $\varepsilon > 0$, the

AOLS algorithm can compute an approximated ε-optimal US, denoted as $\overline{U_S}$, which may diverge from the optimal US by at most ε. Before a complete US is obtained, a partial US can be obtained by evaluating the largest improvement for weights $\mathcal{W}$ via the priority queue of the marginal weight set in this step. An element in the vector value function over a partial US is defined by $V_S^*(\mathcal{W}) = \max_{\mathcal{V} \in S} \mathcal{W}\mathcal{V}$, where S is the partial undominated set.

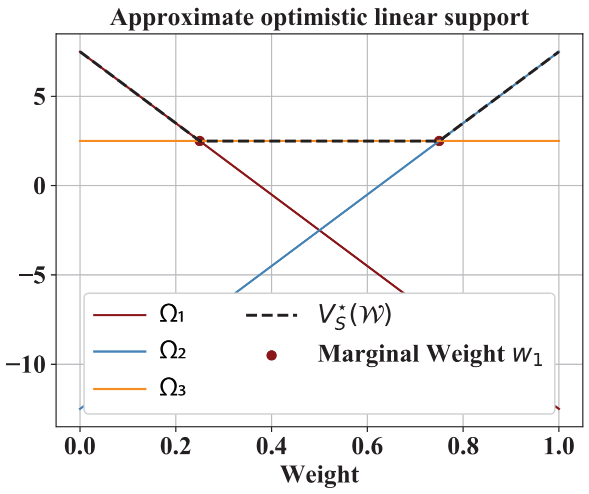


Figure 2: An example of $V_S^*(\mathcal{W})$ for an S with $\mathbf{z} \in \mathbb{R}^2$.

As described in Section 3.3, the dimension of the vectors Ω_i has been shrinked to the dimension of $\mathbf{z}$ by rewriting the optimization problem (6) into (8). An example of $V_S^*(\mathcal{W})$ for an S containing three value vectors and $\mathbf{z} \in \mathbb{R}^2$ is given in Figure 2. The vectors Ω_1, Ω_2, and Ω_3 are represented in the $2D$ space. Each line Ω_i denoted $\beta \frac{\partial \hat{\mathcal{L}}(\theta_{io}, \theta_{is}^i)}{\partial \mathbf{z_1}} + (1-\beta) \frac{\partial \hat{\mathcal{L}}(\theta_{io}, \theta_{is}^i)}{\partial \mathbf{z_2}}$. $V_S^*(\mathcal{W})$ is a piecewise linear and convex function that consists of line segments, each of which is the upper surface among all scalarized gradient vectors. The marginal weight is the weight corresponding to the minimum-norm point marked with red 'o'. When $\mathbf{z} \in \mathbb{R}^3$, each of the element in $V_S^*(\mathcal{W})$ associated with a policy is a plane instead of a line. When there are more than three objectives, each element in $V_S^*(\mathcal{W})$ can be represented as a hyperplane. AOLS always selects the minimum-norm point and $\mathcal{W}$ that reflects the largest difference between $V_{\overline{U_S}}(\mathcal{W})$ and $V_S^*(\mathcal{W})$ on an optimistic upper bound, *i.e.*, $V_{\overline{U_S}}(\mathcal{W}) - V_S^*(\mathcal{W})$, which can be updated iteratively to obtain a more accurate $\mathcal{W} = \arg\max V_{\overline{U_S}}(\mathcal{W})$ in the convex hull. The pseudocode for AOLS is shown in the Algorithm 1, where the minimum-norm point $\mathcal{W}V_S^*(\mathcal{W})$ with respect to $\mathcal{W}$ is appended in the minimum point queue P.

4.1 PAOLS

In this subsection, we will propose a new approach that is more efficient than the existing AOLS method. Before describing the new approach, let us discuss the time complexity associated with the AOLS algorithm. Let the number of the marginal weights in the AOLS algorithm be given by $|\mathcal{W}|$, where $|\cdot|$ is the cardinality of a set. The while loop in Algorithm 1 will run for $|\mathcal{W}|$ times until Q is empty. In each run, $|\mathcal{V}|$ linear equations (also referred to as primitive operations) need to be solved for up to $|\mathcal{W}|$ times. Hence, the total number of primitive operations in AOLS can be up to $|\mathcal{W}|\,|\mathcal{V}|\,(1+|\mathcal{W}|)\,|\mathcal{W}|$, namely, in an order of $O(|\mathcal{W}|^3\,|\mathcal{V}|)$. We next present a new algorithm that can reduce the time complexity to the order of $O(|\mathcal{W}|^2\,|\mathcal{V}|)$.

The proposed new method is the introduction of pruning in the AOLS algorithm, named PAOLS. While still employing the standard AOLS procedure, PAOLS focuses on identifying the elements in the undominated set that are not optimal and hence can be removed from the list to be visited by AOLS procedure. In other words, the PAOLS algorithm seeks to reduce the number of the weighted sum $\mathcal{W}\mathcal{V}_{old}$ to be visited in the $\overline{U_S}$ set, hence improving the efficiency. In particular, PAOLS replaces the original OLS subroutine (Algorithm 2) by a new PAOLS subroutine (Algorithm 3). In the new PAOLS subroutine, we introduce a new function $lexgt(\cdot)$ that uses lexicographic order, which is a dictionary order of vectors, to obtain the lexicographically maximized element. More specifically, $lexgt(\cdot)$ in Algorithm 3 is defined as

$$lexgt(W[i]\mathcal{V}[j], W[i]g(\mathcal{V}[j]))\leftarrow\text{true}, \quad \begin{array}{l}\text{if } \exists W[i], W[i]\mathcal{V}[j] > W[i]g(\mathcal{V}[j]) \\ \text{and } \forall i' < i, W[i']\mathcal{V}[j] < W[i']g(\mathcal{V}[j]).\end{array} \tag{11}$$

Because the lexicographically maximum vector $g(\mathcal{V}[j])$ is chosen so that $W\,[i]\,g(\mathcal{V}[j])$ is lexicographically maximized for all $\mathcal{V}[j]$, no other element can be used to construct a larger $W[i]g(\mathcal{V}[j]) - V_S^*\,(W\,[i])$ while still satisfying the marginal weight $\mathcal{W}_{max} = W[i]$. In other words, the lexicographic order can further eliminate the non-optimal elements in the undominated set so that these non-optimal elements will not be visited for obtaining the corresponding policies. Meanwhile, such a $\mathcal{W}_{max}$ is guaranteed to be a marginal weight because it dominates all previous vectors due to the lexicographic order operation in (11). One can also observe that the eliminated non-optimal elements are not marginal weights since they are dominated by the lexicographically maximum vector.

We next show that the new algorithm can reduce the complexity to the order of $O(|\mathcal{W}|^2\,|\mathcal{V}|)$.

Theorem 2. *The new PAOLS algorithm described in Alg. 1 runs in polynomial time in an order of* $O(|\mathcal{W}|^2\,|\mathcal{V}|)$.

Proof. In computing $\mathcal{W}$, the total number of $\mathcal{W}$ added to the unchecked set depends on the size of $\mathcal{W}$, namely $|\mathcal{W}|$. In the new subroutine PAOLS, a number of $|\mathcal{W}|\,|\mathcal{V}|$ primitive operations is needed. Each path in the while loop of Algorithm 1 consumes an element from a total number of $|\mathcal{W}|$ unchecked weights. Hence, the total number of primitive operations can be up to $|\mathcal{W}|\,|\mathcal{W}|\,|\mathcal{V}|$, namely, in an order of $O(|\mathcal{W}|^2\,|\mathcal{V}|)$. □

Because the proposed PAOLS algorithm can reduce the time complexity from $|\mathcal{W}|^3\,|\mathcal{V}|$ to $|\mathcal{W}|^2\,|\mathcal{V}|$, it is more efficient than the AOLS algorithm.

4.2 Value-Function and Policy Update

In this subsection, we describe how to update the policy network and the value-function network given the discovered marginal weight set. Both the PAOLS and the value-function and policy update will be run iteratively to yield a stable solution. The stability analysis of the entire framework will be provided in Section 5.

```
Data: # of steps in each MORLSubroutine: $k$, improvement threshold: $\varepsilon$
Result: $\overline{U_S}$, $\Delta_{max}$, $\mathcal{W}$, $P$
S: partial US, W: list of explored marginal weight, Q: priority queue of the initial
  marginal weight, $\Delta_{max}$: improvement, P: minimum-norm point queue
$S \leftarrow \varnothing$; $\mathcal{WV}_{old} \leftarrow \varnothing$
forall extreme weights of infinite priority $\mathcal{W}_{\max} = e_1$ do
    Q.add $(\mathcal{W}_{\max}, \infty)$
end
while $\neg$ Q.isEmpty() $\wedge \neg$ timeOut do
    $\mathcal{W}_{max}, \Delta_{max}, \mathcal{V} \leftarrow$Q.pop()
    $\Pi, \hat{\mathcal{L}}(\theta_{IO}, \theta^i_{IS}) \leftarrow$ MORLSubroutine$(k, \mathcal{W}_{\max})$
    $\mathcal{V} \leftarrow$ compute $\mathcal{V}$ via (7)
    $\mathcal{WV}_{old} = \mathcal{WV}_{old} \cup \{(\mathcal{W}_{\max}, \mathcal{W}_{\max}\mathcal{V})\}$
    if $\mathcal{V} \notin S$ then
        $S \leftarrow S \cup \{\mathcal{V}\}$
        $W \leftarrow$ recompute marginal weight $V^*_S(\mathcal{W})$
        for $K \in 1, ..., len(W)$ do
            if $e_K \neq \mathcal{W}_{\max}$ then
                return$(e_K, \infty)$
            end
        end
        if Using aols algorithm then
            $W[K], V_{US}[K] \leftarrow$olsSubroutine$(\mathcal{WV}_{old}, W[\cdot], S)$
            if $V_{\overline{U_S}}[K] - V^*_S(W[K]) > \varepsilon$ then
                $Q$.add$(W[K], V_{\overline{U_S}}[K] - V^*_S(W[K]), V^*_S(W[K]))$
                $P$.add$(W[K]V^*_S(W[K]))$
            end
        end
        if Using paols algorithm then
            $W[i], W[i]g(\mathcal{V}[j]) - V^*_S(W[i]), V^*_S(W[i]) \leftarrow$ paolsSubroutine$(W, S)$
            $Q$.add$(W[i], V_{\overline{U_S}}[i] - V^*_S(W[i]), V^*_S(W[i]))$
            $P$.add$(W[i]V^*_S(W[i]))$
        end
    end
    $W \leftarrow W \cup \{W[i]\}$
end
```

Algorithm 1: function aols(k, ε)/ paols(k, ε)

```
Data: $\mathcal{WV}_{old}, W[\cdot], S$
Result: $W[K], V_{US}[K] - V_S^*(W[K])$
for weights in $W[\cdot]$ do
    max $\mathcal{WV}$, s.t.: $\forall (\mathcal{W}, u) \in \mathcal{WV}_{old} : \mathcal{WV} \leq u$
    $K \leftarrow \arg\max_K V_{US}[K] - V_S^*(W[K])$
    return $W[K], V_{US}[K] - V_S^*(W[K])$
end
```

Algorithm 2: function olsSubroutine($\mathcal{WV}_{old}, W[\cdot], S$)

```
Data: $W, S$
Result: $W[i], W[i]g(\mathcal{V}[j]) - V_S^*(W[i]), V_S^*(W[i])$
for $W[i] \in W$ do
    maxval $= -\infty$
    for $\mathcal{V}[j] \in S$ do
        if $W[i]\mathcal{V}[j] \geq maxval$ and lexgt($W[i]\mathcal{V}[j]$,$W[i]g(\mathcal{V}[j])$) then
            maxval$\leftarrow W[i]\mathcal{V}[j]$
            $g(\mathcal{V}[j]) \leftarrow \mathcal{V}[j]$
        end
    end
    return $W[i], W[i]g(\mathcal{V}[j]) - V_S^*(W[i]), V_S^*(W[i])$
end
```

Algorithm 3: function paolsSubroutine(W, S)

We here propose an asynchronous advantage actor-critic (A3C) network (Babaeizadeh, et al., 2016) with *an actor* and an *I-objective critic*, where the actor network is used to generate policy that can maximize the I objective values and the I-objective critic network is used to map from the state space X to $\mathbf{V}^{\pi}(x)$. In a continuous action space, we assume that an actor network with weights θ_π is used to generate control actions via $a = \pi(x; \theta_\pi)$. The weights θ_π can be updated using policy gradient (Vamvoudakis & Lewis, 2010) given by $\Delta\theta_\pi \sim \sum_k \bigtriangledown_{\theta_\pi} \log \pi_\theta(x_k, a_k)\, \delta_{k,t}^i$, where $\delta_{k,t}^i$ is the expected value of the ith objective, also known as the temporal difference (TD) residual of $\hat{V}_i^\pi$ with discount γ (Sutton & Barto, 2018), given by

$$\delta_{k,t}^i = r_i(x_{k,t}, a_{k,t}) + \gamma \hat{\mathbf{V}}_i^\pi(x_{k,t+1}; \phi^-) - \hat{\mathbf{V}}_i^{\pi^-}(x_{k,t}; \phi^-), \tag{12}$$

where $r_i(x_{k,t}, a_{k,t})$ is the immediate reward at the tth time step on the kth experience, $\hat{\mathbf{V}}_i^{\pi^-}(x_{k,t}; \phi^-)$ is the approximation of the value function $\mathbf{V}_i$ based on the old policy π^- for the actor network and the old weights ϕ^- for the critic network, and $\hat{\mathbf{V}}_i^\pi(x_{k,t+1}; \phi^-)$ is the approximation of the value function $\mathbf{V}_i$ based on the updated policy π for the actor network and the old weights ϕ^- for the critic network.

In the standard TD-residual method, the value of one action evaluated via (12) is an incremental form of value iteration. The key drawback of the standard TD-residual method includes the need for a large number of samples and large variance of policy gradient estimate. To address these issues, an existing approach, called generalized advantage estimator (GAE) (Schulman, et al., 2015b), can be used to substantially reduce the variance of policy gradient estimates at the cost of some bias (Schulman, et al., 2017, 2015a; Rockafellar & Wets, 1991).

The GAE is defined by

$$\hat{A}_t^i(x_t, a_t) = \lim_{H\to\infty}(1-\lambda)\sum_{j=1}^{H}\lambda^{j-1}\sum_{k=1}^{j}\gamma^{j-1}\delta_{k,t+j-1}^i = \sum_{l=0}^{H}(\gamma\lambda)^l\,\delta_{k,t+l}^i, \tag{13}$$

where $\lambda \in [0,1]$ and $\gamma \in [0,1]$ adjusts the bias-variance tradeoff of GAE. To further improve the performance as well as decrease the complexity of implementation and computation, we also use proximal policy optimization (PPO) with clip to constrain the objective in (13) as

$$\tilde{A}_t^i(\theta) = E_{D^k}\Big[\min\Big(\frac{\pi_\theta(a_t|x_t)}{\pi_{\theta^-}(a_t|x_t)}\hat{A}_t^i(x_t,a_t), g\left(\epsilon, \hat{A}_t^i(x_t,a_t)\right)\Big)\Big],$$

where $D^k = \{[x_1, a_1, \cdots, x_H]\}$ is a set of trajectories generated by running policy $\pi = \pi(\theta^-)$, and

$$g\left(\epsilon, \hat{A}_t^i(x_t,a_t)\right) = \begin{cases}(1+\epsilon)\hat{A}_t^i(x_t,a_t), & \hat{A}_t^i(x_t,a_t) \geq 0,\\ (1-\epsilon)\hat{A}_t^i(x_t,a_t), & \hat{A}_t^i(x_t,a_t) < 0.\end{cases}$$

Note that θ includes inter-objective weights θ_{IO} and objective specific weights θ_{IS}^i.

For the I-objective critic network, its ith output value with hyperparameters ϕ is used to approximate each element in the vector value function $\mathbf{V}^\pi(s)$. The weights can be updated via $\Delta_\phi \sim -\nabla_\phi\sum_k\|\delta_{k,t}\|_2^2$, where $\delta_{k,t} = [\delta_{k,t}^1, \cdots, \delta_{k,t}^n]'$.

After new weights of the A3C network models are obtained, $\mathbf{V}^\pi$ can be obtained via new samples using the updated policy. Afterwards, the procedure in Subsection 4.1 can be implemented to obtain the updated $\mathcal{W}_{max}$. The entire process will iterate until $V_{\overline{U_S}}(\mathcal{W}) - V_S^*(\mathcal{W}) < \epsilon$. The pseudocode is given in the Algorithm 4.

5. Convergence Analysis

In this section, we will prove the convergence property of the policy in the proposed algorithm. To do this, we first need to define a few additional notations.

Let $\mu_a(x,\varphi)$, $\varphi \in \mathbb{R}^n$, represent the probability of selecting action a at state x subject to a policy $\pi(\varphi)$, where φ is the parameter for the policy. We define one stage reward as $g_{x_k}(\varphi)$, where x_k is the state at the time step k (one-stage state). Based on the value function defined in (1), a typical performance metric to compare different policies is the average discounted reward given by $\lambda(\varphi) = \lim_{t\to\infty} E\left[\sum_{k=0}^t \gamma^k g_{x_k}(\varphi)\right]$, where E is the expectation operator. For any φ and x, the differential reward $v_x(\varphi)$ of observation x is defined as

$$v_x(\varphi) = E\left[\sum_{k=1}^{T-1}\left(g_{x_k}(\varphi) - \lambda(\varphi)\right)|x_0 = x\right],$$

where $T = \min\{k > 0 | x_k \in \{x_i, i = 0, \cdots, k-1\}\}$.

Before deriving an explicit form of the gradient of $\lambda(\varphi)$, we make the following two assumptions.

```
Data: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$, $\mathcal{W}_{max}$
Result: $\Pi$, $\hat{\mathcal{L}}(\theta_{IO}, \theta^i_{IS})$
$\Pi$: empty queue of policy $\pi$; $K$: # of time steps in one episode; $\theta_{i,k}$: param. of the
  ith objective at the kth time step; $\theta^k_{IO}, \theta^i k_{IS}$: $\theta_{IO}, \theta^i_{IS}$ at the kth time step
for $k = 1, \cdots, K$ do
    Collect a set of trajectories $D^{i,k}$ by running policy $\pi = \pi(\theta_{i,k})$
    Compute rewards-to-go
      $\hat{\mathbf{R}}_t = [r_1(x,a) + \gamma \hat{\mathbf{V}}^\pi_1(x;\phi_k), \cdots, r_I(x,a) + \gamma \hat{\mathbf{V}}^\pi_I(x;\phi_k)]'$
    Compute advantage estimates $\hat{A}^i_t$ using the GAE method based on $\hat{\mathbf{V}}_i(x;\phi_k)$
    Obtain $\hat{\mathbf{V}}(x;\phi_k) = [\hat{\mathbf{V}}_1(x;\phi_k), \cdots, \hat{\mathbf{V}}_I(x;\phi_k)]$
    Evaluate $\hat{\mathcal{L}}(\theta_{i,k}) = \hat{\mathcal{L}}(\theta^k_{IO}, \theta^{ik}_{IS})$
    Update policy by maximizing PPO-Clip objective:
      $$\theta_{i,k+1} = \arg\max_\theta \frac{1}{|D^{i,k}|T} \sum_{D^{i,k}} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|x_t)}{\pi_{\theta_{i,k}}(a_t|x_t)} \hat{A}^i(x_t,a_t), g\left(\epsilon, \hat{A}^i(x_t,a_t)\right)\right)$$
      via stochastic gradient ascent (e.g., Adam (Kingma & Ba, 2014))
    Fit value function by regression on the mean-squared error:
      $$\phi_{k+1} = \arg\min_{\phi_k} \frac{1}{|D^{i,k}|T} \sum_{\tau^i_n \in D^{i,k}} \sum_{t=0}^{T} \left\| \mathbf{V}(x_t;\phi_k) - \hat{\mathbf{R}}_t \right\|_2^2$$
      via stochastic gradient descent.
end
$\Pi$.add($\pi(\theta_{i,K}), \hat{\mathcal{L}}(\theta_{i,K})$)
```

Algorithm 4: function MORLSubroutine($k, \mathcal{W}_{max}$)

Assumption 1. *For each φ, the Markov chains $\{X_n\}$ and $\{X_n, \mathcal{A}_n\}$, denoting the sequence of states and state-action pairs, are irreducible and aperiodic under the stationary probabilities $\pi_x(\varphi)$.*

Assumption 2. *For every $x, x' \in X$, the transition probability $p_{xx'}(\phi)$ and $g_x(\phi)$ are bounded, twice differentiable, and have bounded first and second derivatives. In addition, there exists a valued function $\psi_a(x,\varphi)$ such that, for every observation x and action a, there exists a bounded function satisfying*

$$\psi_a(x,\varphi) = \frac{\nabla \mu_a(x,\varphi)}{\mu_a(x,\varphi)}, \tag{14}$$

where the mapping $\psi_a(x,\varphi)$ has first bounded derivatives for any fixed x and a.

When Assumption 1 holds true, the MOMDP admits a unique invariant probability distribution for each objective (Tsitsiklis & Van Roy, 1999). Hence, the policy $\pi(\varphi)$ is composed of a steady-state probability of state x. When Assumption 2 holds true, $\mu_a(x,\varphi)$ is a smooth function on φ, hence $\lambda(\varphi)$ has a bounded first derivative. Let Assumptions 1 and 2 hold, $\lambda(\varphi)$ and $\pi_x(\varphi)$ are twice differentiable, and have bounded first and second derivatives. This property is needed in the convergence analysis. Furthermore, (14) holds whenever $\mu_a(x,\varphi)$ is nonzero. In order to project $\bigtriangledown\lambda(\varphi)$ to a subspace and derive a general from that can show the proposed algorithm is applicable to the case of infinite space, we first need to rewrite $\bigtriangledown\lambda(\varphi)$.

Lemma 1. *Let Assumptions 1 and 2 hold. The gradient of $\lambda(\varphi)$ can be represented by* [1]

$$\bigtriangledown\lambda(\varphi) = \sum_{x\in X}\sum_{a\in\mathcal{A}} \eta_a(x,\varphi) q_{x,a}(\varphi)\psi_a^i(x,\varphi), \tag{15}$$

where $\eta_a(x,\varphi) = \pi_x(\varphi)\mu_a(x,\varphi)$, $q_{x,a}(\varphi) = (g_{x,a} - \lambda(\varphi)) + \sum_{x'\in X} p_{xx'}(a)v_{x'}(\varphi)$, and $\psi_a^i(x,\varphi)$ is the ith component of $\psi_a(x,\varphi)$.

Proof. Recall that the gradient of $\lambda(\varphi)$ is stated by (Marbach & Tsitsiklis, 2001)

$$\bigtriangledown\lambda(\varphi) = \sum_{x\in X}\pi_x(\varphi)\left(\bigtriangledown g_x(\varphi) + \sum_{x'\in X}\bigtriangledown p_{xx'}(\varphi)v_{x'}(\varphi)\right). \tag{16}$$

The expected reward per stage $g_x(\varphi)$ is given by $g_x(\varphi) = \sum_{a\in\mathcal{A}}\mu_a(x,\varphi)g_{x,a}$, where $g_{x,a}$ denotes the one stage reward when taking action a at state x. Then the gradient of $g_x(\varphi)$ can be written as

$$\bigtriangledown g_x(\varphi) = \sum_{a\in\mathcal{A}}\bigtriangledown\mu_a(x,\varphi)g_{x,a} = \sum_{a\in\mathcal{A}}\bigtriangledown\mu_a(x,\varphi)g_{x,a} - \bigtriangledown\sum_{a\in\mathcal{A}}\mu_a(x,\varphi)\lambda(\varphi)$$

because $\sum_{a\in\mathcal{A}}\mu_a(x,\varphi) = 1$ and hence $\bigtriangledown\sum_{a\in\mathcal{A}}\mu_a(x,\varphi) = 0$. Then we can further obtain

$$\bigtriangledown g_x(\varphi) = \sum_{a\in\mathcal{A}}\bigtriangledown\mu_a(x,\varphi)g_{x,a} - \sum_{a\in\mathcal{A}}\bigtriangledown\mu_a(x,\varphi)\lambda(\varphi) = \sum_{a\in\mathcal{A}}\bigtriangledown\mu_a(x,\varphi)(g_{x,a} - \lambda(\varphi)) \tag{17}$$

by moving the gradient inside the summation. Meanwhile, the transition probability is given by:

$$p_{xx'}(\varphi) = \sum_{a\in\mathcal{A}}\mu_a(x,\varphi)p_{xx'}(a). \tag{18}$$

By following a similar analysis as that for $\bigtriangledown g_x(\varphi)$, we can obtain:

$$\sum_{x'\in X}\bigtriangledown p_{xx'}(\varphi)v_{x'}(\varphi) = \sum_{x'\in X}\sum_{a\in\mathcal{A}}\bigtriangledown\mu_a(x,\varphi)p_{xx'}(a)v_{x'}(\varphi). \tag{19}$$

By inserting (17) and (19) into (16) and making a few rearrangements, we can obtain (15). □

Based on Lemma 1, we now show that the gradient $\bigtriangledown\lambda(\varphi)$ can be written in the form of inner products given in the following lemma. Before moving on, let us define q_φ and $\psi(\varphi)$ as the vectors of, respectively, $q_{x,a}(\varphi)$ and $\psi_a(x,\varphi)$ on $X\times\mathcal{A}$. Define the inner product of two real value functions q_φ and $\psi(\varphi)$ as

$$\langle q_\varphi, \psi(\varphi)\rangle_\varphi = \sum_{x\in X}\sum_{a\in\mathcal{A}}\eta_a(x,\varphi)q_{x,a}(\varphi)\psi_a(x,\varphi). \tag{20}$$

1. $q_{x,a}(\varphi)$ in (15) corresponds to the TD residual in (12).

Lemma 2. *Let Assumptions 1 and 2 hold. The gradient of $\lambda(\varphi)$ can be computed by the inner product of two real value functions given by*

$$\triangledown\lambda(\varphi) = \langle q_\varphi, \psi(\varphi)\rangle_\varphi = \left\langle \prod_\varphi q_\varphi, \psi(\varphi) \right\rangle_\varphi , \tag{21}$$

where

$$\prod_\varphi q = \arg\min_{\widehat{q}\in\zeta_\varphi} \|q - \widehat{q}\|_\varphi \tag{22}$$

with ζ_φ denoting the span of the vectors $\left\{\psi_a^i(x,\varphi); i = 1,\cdots,n\right\}$ in $\mathbb{R}^{|X|\times|\mathcal{A}|}$.

Proof. Based on the definition in (20) and Lemma 1, we can obtain that $\triangledown\lambda(\varphi) = \langle q_\varphi, \psi(\varphi)\rangle_\varphi$. We next show that the second equality in (21) holds.

We can rewrite (15) as

$$\frac{\partial}{\partial\varphi_i}\lambda(\varphi) = \left\langle q(\varphi), \psi^i(\varphi)\right\rangle_\varphi, \quad i = 1,\cdots,n, \tag{23}$$

where n is the dimension of φ. For a high (or even infinite) dimensional space, computing the gradient of $\lambda(\varphi)$ depends on $q_{x,a}(\varphi)$, (equivalently q_φ in (23)), and is typically difficult. An alternative approach is to use the project of q_φ based on (22) in the computation of inner product. Based on (20), the inner product $\langle q_\varphi, \psi(\varphi)\rangle_\varphi$ is equivalent to the inner product of $\psi(\varphi)$ and the projection of q_φ on ζ_φ. Hence, $\langle q_\varphi, \psi(\varphi)\rangle_\varphi = \left\langle \prod_\varphi q_\varphi, \psi(\varphi)\right\rangle_\varphi$ always holds. In other words, the projection of q_φ onto ζ_φ is sufficient to learn $\triangledown\lambda(\varphi)$ since $\langle q_\varphi, \psi(\varphi)\rangle_\varphi = \left\langle \prod_\varphi q_\varphi, \psi(\varphi)\right\rangle_\varphi$. □

With Lemma 2, we are ready to present the proof of convergence in Theorem 3. Before moving on, the following two assumptions are needed.

Assumption 3. *The value update stepsize sequence for the critic $\left\{\gamma_k^i\right\}$ and the actor $\{\beta_k\}$ are positive and nonincreasing, and satisfies $\sum_{k=0}^\infty \vartheta_k = \infty$ and $\sum_{k=0}^\infty \vartheta_k^2 < \infty$, $\vartheta \in \{\gamma,\beta\}$. In addition, the actor updates much slower that the critic, i.e., $\frac{\beta_k}{\gamma_k^i} \to 0$.*

Assumption 4. *For every $\varphi_k \in \mathbb{R}^n$, $\Phi\varphi_k \neq e$, where e is equal to all-one vector and Φ is a $m\times n$ matrix whose kth row is equal to φ_k. In addition, the column vectors of Φ are linearly independent.*

Theorem 3. *Let Assumptions 1, 2, 3, and 4 hold. The proposed policy will converge with probability* 1.

Proof. Under Assumption 3, the size of actor updates is negligible compared with the size of the critic updates. If the critic network is stable, the actor network is stationary. We next show the convergence of critic network.

Under Assumptions 1 and 2 hold, the gradient $\triangledown\lambda(\varphi)$ can be written in the form of inner products, as shown in Lemma 2. When Assumptions 3 and 4 hold, the convergence analysis in (Tsitsiklis & Van Roy, 1999) can be used to prove that any critic in the proposed policy will converge with probability 1. Once the critic networks converge (*i.e.,* are stationary), all weights in the critic networks are stationary. Because $\lambda(\varphi)$ has a second bounded derivative when Assumptions 1 and 2 hold, the update of the actor network can be proved stationary as well by the stochastic approximation algorithm (Spall, 1992).

This completes the proof of Theorem 3. $\square$

6. Experiments

In this section, we evaluate the performance of the proposed approach. We first describe the experiment setup and demonstrate how the gradient weights among various objectives can be quantified via computing a min-norm point in the convex optimization. After that, we will show how the maximal relative improvement $\Delta_r(\mathcal{W}) = \frac{V_{\overline{U_S}}(\mathcal{W}) - V_S^*(\mathcal{W})}{V_{\overline{U_S}}(\mathcal{W})}$ changes with respect to the number of training episodes. Finally, we show the testing results and the solution stability.

6.1 Setup

In our experiment setting, the parametric hypothesis $f^i(x; \theta_{IO}, \theta_{IS}^i) : X \to V_i^\pi$, where θ_{IS}^i is the objective specific weight and θ_{IO} is the inter-objective weight, is a CNN shown in Figure 3. The architecture specification is given in Table 1. The $FC - n$ and $FC - I$ correspond to the n-action policy $\pi(\cdot|x_t)$ and the value function $\mathbf{V}(x_t) \in \mathbb{R}^{I \times 1}$, respectively. The screen is resized to $84 \times 84 \times 3$ RGB image as the network input. In the proposed K-shot RL setting, a network starts with a single row: a CNN having the layer l with feature maps X_{l-1}^1, and weights W_l^1 trained when the objective number $i = 1$ in Algorithm 4. When switching to the training of the second objective, the parameters W_l^1 are frozen and a new row with weights W_l^2 is instantiated, where feature maps X_l^2 receive input from both X_{l-1}^1 and X_{l-1}^2 via lateral connections with weights FW_l^1 and W_l^2. The generalization to I objectives is given by

$$X_l^i = f(W_l^i * X_{l-1}^i + \sum_{j<i} FW_l^{i:j} * X_{l-1}^j), \tag{24}$$

where W_l^i is the weight matrix of layer l of row i ,$FW_l^{i:j}$ are the lateral connections from layer $l-1$ of row i, $*$ is the convolution operation. In summary, for the ith objective, θ_{IO} represents the weights for the $(i-1)$th objective, and θ_{IS}^i is $\theta \setminus \theta_{IO}$. The proposed approach is shown in Algorithm 5.

Table 1: CNN architecture for objective i

Layer #	1	2	3	4
Parameters	$C4 \times 4 - 32 \times i, S2$	$C4 \times 4 - 32 \times i, S2$	$FC - n$	$FC - I$

We focus on 3 environments: (1) Ant-v2, (2) Humanoid-v2, and (3) HumanoidStandup-v2, on the MuJoCo physics engine (Todorov, Erez, & Tassa, 2012). For Ant-v2, we select four objectives: Reward Control (Rctrl), Reward Contact (Rcont), Reward Survive (Rsurv),

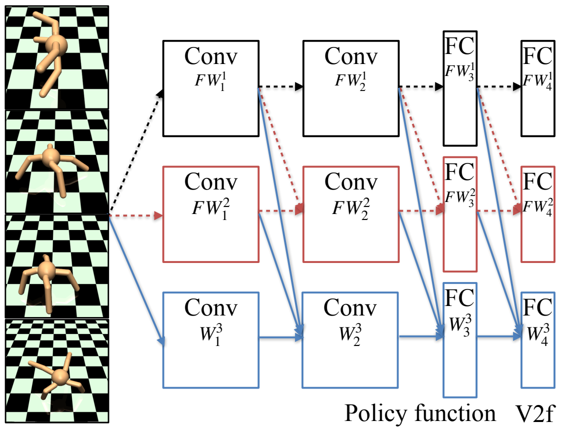


Figure 3: The architecture of the shared parameter network.

```
Data: # of steps in each MORLSubroutine: $k$, improvement threshold: $\varepsilon$
Result: $\Pi$, $\overline{U_S}$
while not done do
    $Q \leftarrow$ function AOLS$(k, \epsilon)$
    $\mathcal{W}_{max}, \Delta_{max} \leftarrow Q$.pop()
    for $i = 1, \cdots, I$ do
        Sample trajectories $D_i = \{x_1, a_1, \cdots, x_H\}$ using $\Pi[i]$ in
          MORLSubroutine$(k, \mathcal{W}_{max})$
        $\theta'_i \leftarrow \theta - \alpha \nabla_{\mathbf{z}} \mathcal{W}_{max} \cdot \mathbb{L}(f_\theta)$, where $\mathbf{z} = [g_1(x; \theta_{IO}), \cdots, g_m(x; \theta_{IO})]$
    end
    $\theta \leftarrow \theta - \alpha \nabla_{\mathbf{z}} \sum_{i=1}^{I} w_i \hat{\mathcal{L}}^i(f_{\theta'_i})$, where $\mathbf{z} = [g_1(x; \theta'_{IO;i}), \cdots, g_m(x; \theta'_{IO;i})]$
end
```

Algorithm 5: The proposed approach.

and Reward Forward (Rfor). For Humanoid-v2, we select five objectives: Mean Episode Length (Mel), Mean Episode Reward (Mer), Linear Velocity (Lvel), Quadratic Control (Qctrl), and Quadratic Impact Cost (Qim). For HumanoidStandup-v2, three objectives, namely, Standup Cost (Stc), Quadratic Control (Qctrl), and Quadratic Impact Cost (Qim), are selected.

We use the proximal policy optimization clipping algorithm with $\epsilon = 0.2$ as the optimizer. The discounting factor is selected as $\gamma = 0.99$. One episode, characterizing the number of time steps of the vectorized environment per update, is chosen as 10240. For the purpose of stabilization, we execute parallel episodes in each batch. The batch size is chosen as the product of the episode size and the number of environment copies simulated in parallel. The number of environment copies is selected as 8. The results are an average of 6 runs. The parameters are optimized using the Adam algorithm (Kingma & Ba, 2014) and a learning rate of 3×10^{-4}. The dimension of $\mathbf{z}$, namely, M, is selected as the number of actions in each scenario. All experiments were conducted using TensorFlow, which allows for automatic differentiation through the gradient updates (Abadi, et al., 2016), on a NVIDIA GeForce RTX 2070 GPU.

6.2 Computation of the Weights

As stated in Algorithm 4 and Algorithm 5, $\mathcal{W}$ can be estimated dynamically by computing the marginal weight via I loops, where I is the number of objectives. For example, for Humanoid-v2, we consider 5 objectives, *i.e.,* $I = 5$. According to Algorithm 4, we separate the batch size, *i.e.,* the number of steps of the vectorized environment per update, into five loops equally. These five loops correspond to five objectives: Mel, Mer, Lvel, Qctrl, and Qim. In each loop, only the ith objective value $\mathbf{V}_i^\pi$ is fitted via regression based on the mean-squared error. The parametric hypothesis per objective is considered in the form of $f^i(x; \theta_{IO}, \theta_{IS}^i) : X \rightarrow V_i^\pi$, in which θ_{IS}^i is the objective specific parameter while θ_{IO} is the inter-objective parameter defined in (3). In other words, $w_{ii} \in \mathcal{W}$ characterizes the self dependency of objective i while $w_{ij} \in \mathcal{W}$ characterizes the impact of objective i on objective j. After Ik time steps, by arranging weights according to $\mathcal{W} = \begin{bmatrix} w_{11} & \cdots & w_{1I} \\ \vdots & \ddots & \vdots \\ w_{I1} & \cdots & w_{I1} \end{bmatrix}$, we can obtain a correlation matrix at each batch with a size of $I \times I$. For Humanoid-v2, the correlation matrix is a 5×5 matrix. One example of a stablized correlation matrix after 2088000 time steps is given in Figure 4.

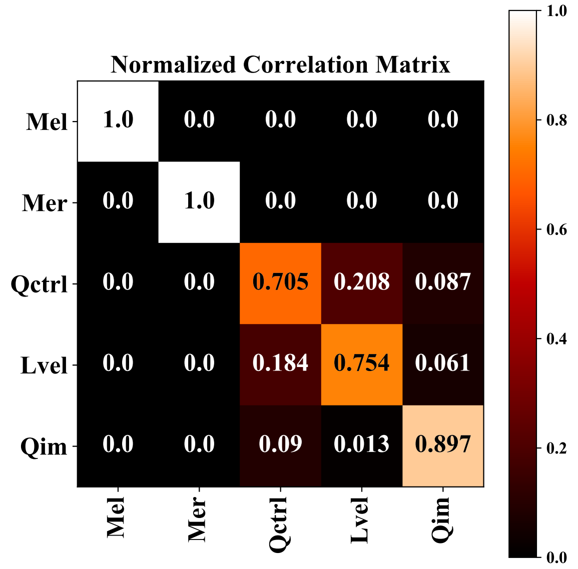
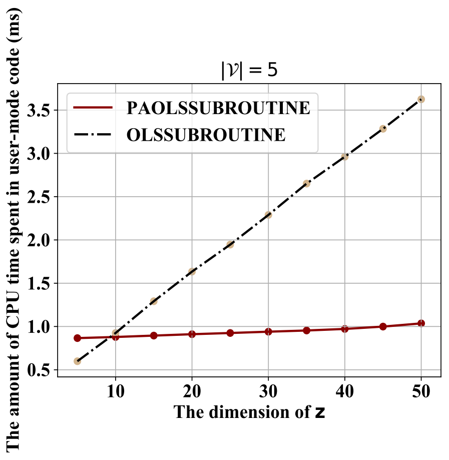


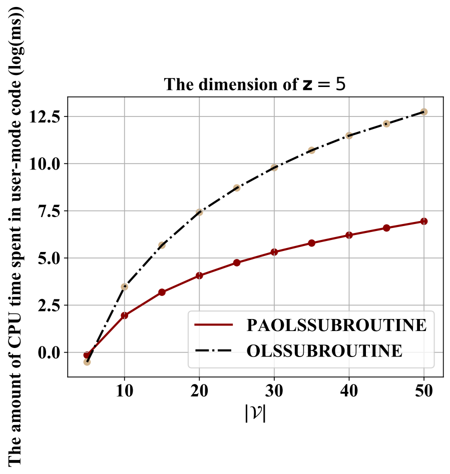
Figure 4: Graph representation of $\mathcal{W}$ using correlation matrix.

From the matrix, we can observe that the first two objectives Mel and Mer, corresponding to the first two rows, are independent. In other words, Mel and Mer will be optimized without affecting other objectives, corresponding to Figure 1b and and Fig 1c in the computing of the min-norm point. It can also be observed that the last three objectives, namely Qctrl, Lvel and Qim, are conflicting with Mel and Mer. In particular, the third row of W indicates that objective 3 has direct impact on objectives 4 and 5. Similarly, the fourth and fifth rows of W indicate that objectives 4 and 5 have direct impact on other objectives except the first two objectives, corresponding to Figure 1a in the computing of the min-norm point. In other words, Qctrl, Lvel, and Qim are dependent. Moreover, Qctrl has larger impact on Lvel than that on Qim because $w_{34} = 0.208 > 0.087 = w_{35}$. Hence, the quantitative relationship among these objectives is explicitly described by the matrix. The gradients taking a vector-valued form reflects the weighted sum of the gradients of all objectives based on the impact of each objective on other (possibly conflicting) objectives. Each element of the vector value function will gradually converge to a value with a very small variance.

Figure 5a and Figure 5b show the time complexity of paolsSubroutine and aolsSubroutine. In order to show that the proposed PAOLS subroutine provides much better scalability with respect to the dimension and the number of gradients, we report the amount of CPU time spent in a user-mode code (outside the kernel). In Figure 5a, when the number of the gradients, namely, $|\mathcal{V}|$, is 5, the CPU time of the proposed PAOLS subroutine grows much slower than the AOLS subroutine as the dimension of $\mathbf{z}$ increases. Similarly, in Figure 5b, when the dimension of $\mathbf{z}$ is 5, the logarithm of the CPU time of the proposed PAOLS subroutine also grows much slower than the AOLS subroutine. All results are based on an average of 100 runs.



(a)



(b)

Figure 5: Time complexity comparison between PAOLS and AOLS.

6.3 Accuracy vs Episodes

We further investigated the effects of the number of time steps on the maximal relative improvement of the $\overline{U_S}$. Figure 6 shows how the maximal relative improvement $\Delta_r(\mathcal{W})$ of the $\overline{U_S}$ evolves with respect to the number of time steps. It can be seen from Figure 6 that the error is highly affected by the number of time steps. Although the aols method is unable to provide a sufficient accuracy to build the $\overline{U_S}$ initially, the deviation will gradually decrease to 0. The proposed paols method can provide better accuracy in building the $\overline{U_S}$ initially and the deviation will decrease to 0 as well.

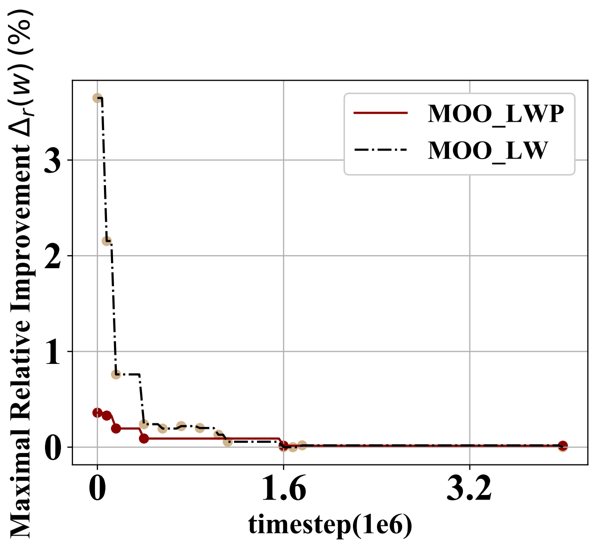


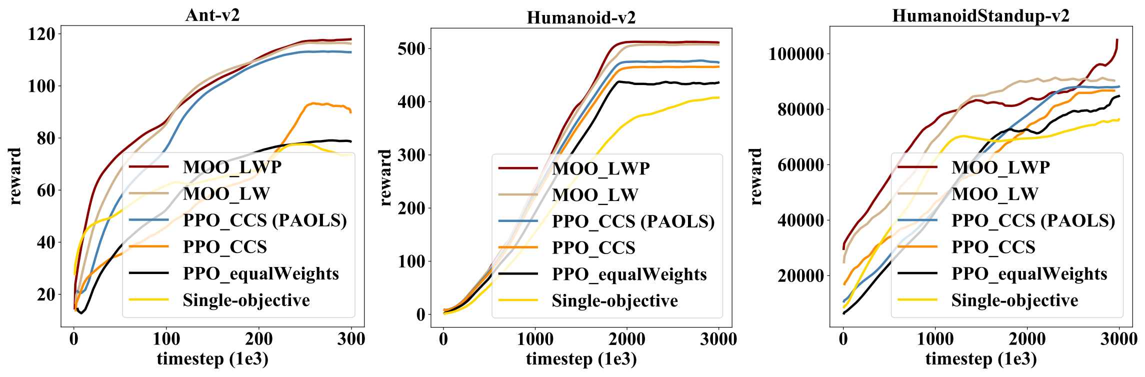
Figure 6: Evolution of $\Delta_r(\mathcal{W})$ with respect to time steps in percentage. MOO_LWP: proposed multi-objective optimization approach with discovered weights and PAOLS. MOO_LW: proposed multi-objective optimization approach with discovered weights.

6.4 Testing Results and Discussion

We now present the main testing results based on the proposed method. We focus on testing the proposed multi-objective optimization approach with discovered weights and PAOLS (MOO_LWP) and the proposed multi-objective optimization approach with discovered weights (MOO_LW). To show the benefit of the proposed MOO_LWP and MOO_LW methods, we will also show the results when (1) MOO is solved via one single-objective optimization, (2) the discovered weight in our method is replaced by the corner points of the convex coverage set (CCS) (Roijers et al., 2015), named PPO_CCS, and (3) MOO is solved with equally distributed weights. All these results are based on the MuJoCo simulator (Todorov et al., 2012).

In the first scenario, the goal is to make a three-dimensional bipedal robot walk forward as fast as possible while saving cost simultaneously in the Humanoid-v2 environment. More specifically, our goal is to maximize the Mean Episode Length (Mel), the Mean Episode Reward (Mer), and the Linear Velocity (Lvel), while minimizing the Quadratic Control (Qctrl) and the Quadratic Impact Cost (Qim). In the second test scenario, the goal is to make a three-dimensional bipedal robot stand up as fast as possible while saving cost simultaneously in the HumanoidStandup-v2 environment. More specifically, our goal is to maximize the standup cost (Stc) while minimizing the quadratic control (Qctrl) and the quadratic impact cost (Qim). In the third test scenario, the goal is to make a four-legged Ant-v2 walk forward as fast as possible while saving cost simultaneously. More specifically, our goal is to maximize the reward forward (Rfor) and the reward survive (Rsurv) while minimizing the reward control (Rctrl) and the reward contact (Rcont).

We take the current reward function in the OpenAI Gym environments as a baseline, use the cumulative reward trained by the single objective PPO as a benchmark (Brockman, et al., 2016; Dhariwal, et al., 2017), and compare it with the proposed method. It is worthwhile to emphasize that HumanoidStandup-v2 does not have a specified reward threshold beyond which "stand up" is considered successful. Figure 7 shows the performance of various algorithms on the three scenarios. It can be observed that the proposed MOO_LWP and MOO_LW methods outperform other methods in yielding higher rewards. In addition, the use of the proposed PAOLS algorithm in PPO can improve the performance of the standard PPO method using the AOLS algorithm.



(a) Ant-v2 (b) Humanoid-v2 (c) HumanoidStandup-v2

Figure 7: Comparison among MOO_LW, MOO_LWP, PPO_CCS, PPO_CCS (PAOLS), PPO_equalWeights, and PPO with single objective on Ant-v2, Humanoid-v2, and HumanoidStandup-v2.

To provide a comprehensive comparison of the performance of various methods on different objectives of the three test scenarios, Table 2, Table 3, and Table 4 show all objective values under different methods. It can be observed from Table 2, Table 3, and Table 4 that the proposed MOO_LWP method can generate higher rewards in most cases because it can provide a higher accuracy to build the $\overline{U_S}$ by pruning the visiting weighted sum $\mathcal{WV}_{old}$ in the $\overline{U_S}$ set. Meanwhile, because the marginal weight used in the MOO_LWP and MOO_LW methods can provide more stabilized weights to be pruned than CCS, MOO_LW can outperform PPO_CCS even if PPO_CCS also uses vector value functions. All tables shows that PAOLS outperforms AOLS, without pruning, with less time steps.

Table 2, Table 3, and Table 4 also show that the proposed MOO_LWP method can optimize more objectives than other methods in all three testing scenarios. In Table 2, the goal is to maximize Rsurv and Rfor while minimize Rctrl and Rcont. In Table 3, the goal is to maximize Mel, Mer, and Lvel while minimize Qctrl and Qim. In Table 4, the goal is to maximize Stc while minimize Qctrl and Qim. The bold entries show the optimal solutions, evaluated first by mean values and then by variances, among different single-objective and multi-objective RL algorithms for the corresponding objectives. This shows the advantages and values of the proposed PAOLS method and the proposed new optimization method with discovered weights in solving multi-objective optimization problems with an unknown inter-objective relationship.

Table 2: Multi-objective Value for Ant-v2

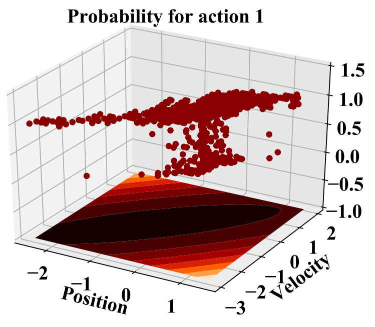
	Single-objective		Multi-objective (AOLS)	
	Rfor	Rsurv	PPO_CCS	MOO_LW
Time steps to hit threshold	270000	270183	250041	249992
Rsurv	76.341 ± 18.566	77.445 ± 14.542	92.546 ± 18.446	117.351 ± 20.587
Rfor	0.541 ± 0.026	0.740 ± 0.057	0.818 ± 0.147	1.012 ± 0.093
Rctrl	−4.011 ± 0.730	−4.019 ± 0.825	−5.025 ± 0.011	−5.791 ± 0.051
Rcont	−3.442 ± 0.250	−3.596 ± 0.011	−4. ± 0.023	−4.092 ± 0.104
	Single-objective	MOO (Equal weights)	Multi-objective (PAOLS)	
	Rctrl	PPO_equalWeights	PPO_CCS	MOO_LWP
Time steps to hit threshold	263134	262591	249000	249092
Rsurv	41.079 ± 10.011	78.981 ± 10.062	117.881 ± 12.244	**120.440 ± 8.899**
Rfor	0.229 ± 0.132	0.747 ± 0.0.094	0.819 ± 0.13	**1.1 ± 0.091**
Rctrl	**−10.978 ± 1.613**	−4.917 ± 0.499	−5.391 ± 0.01	−5.903 ± 0.03
Rcont	−3.09 ± 0.009	−4. ± 0.025	−4. ± 0.019	**−4.359 ± 0.1**

Table 3: Multi-objective Value for Humanoid-v2

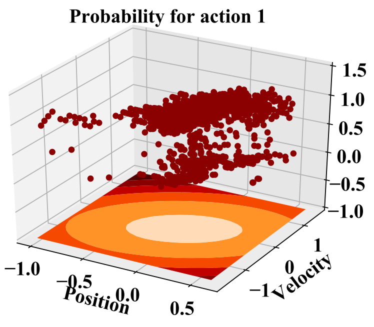
	Single-objective		Multi-objective (AOLS)	
	Alive Bonus	Velocity	PPO_CCS	MOO_LW
Time steps to hit threshold	1759910	1756200	1700310	1693670
Mel	60.162 ± 15.073	47.670 ± 11.943	62.013 ± 15.581	63.384 ± 15.966
Mer	405.643 ± 87.661	45.53 ± 11.383	463.186 ± 91.253	506.832 ± 92.401
Qctrl	−0.235 ± 0.070	−0.232 ± 0.012	−0.231 ± 0.059	−0.23 ± 0.041
Lvel	0.310 ± 0.262	**1.013 ± 0.481**	0.385 ± 0.253	0.490 ± 0.197
Qim	−0.43 ± 0.14	−0.48 ± 0.09	−0.64 ± 0.04	−0.70 ± 0.05
	Single-objective	MOO (Equal weights)	Multi-objective (PAOLS)	
	Quadratic Impact Cost	PPO_equalWeights	PPO_CCS	MOO_LWP
Time steps to hit threshold	1730000	1719000	1690000	1689440
Mel	45.802 ± 9.025	61.232 ± 19.451	63.004 ± 10.079	**64.951 ± 7.344**
Mer	40.067 ± 10.802	434.882 ± 90.231	471.187 ± 74.048	**510.794 ± 75.237**
Qctrl	−0.232 ± 0.018	−0.232 ± 0.009	**−0.235 ± 0.041**	−0.233 ± 0.039
Lvel	0.494 ± 0.204	0.201 ± 0.194	0.812 ± 0.233	0.901 ± 0.199
Qim	−0.8 ± 0.072	−0.6 ± 0.017	−0.74 ± 0.032	**−0.80 ± 0.04**

Table 4: Multi-objective Value for HumanoidStandup-v2

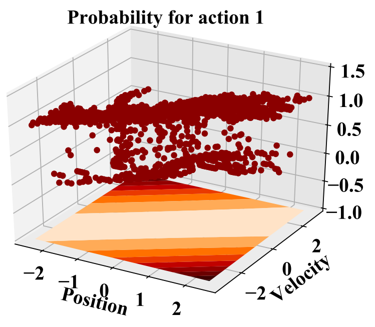
	Single-objective		Multi-objective (AOLS)	
	Standup Cost	Quadratic Control	PPO_CCS	MOO_LW
Time steps to hit threshold	2705910	2692440	2651260	2619820
Qctrl	−0.215 ± 0.027	−0.216 ± 0.039	−0.215 ± 0.026	−0.216 ± 0.027
Stc	74417.1 ± 16135.020	789.094 ± 212.467	85793.17 ± 18951.362	88899.88 ± 18934.457
Qim	−0.210 ± 0.012	−0.207 ± 0.024	−0.227 ± 0.016	−0.230 ± 0.015
	Single-objective	MOO (Equal weights)	Multi-objective (PAOLS)	
	Quadratic Impact Cost	PPO_equalWeights	PPO_CCS	MOO_LWP
Time steps to hit threshold	2704610	2667830	2619130	2610810
Qctrl	−0.215 ± 0.031	−0.216 ± 0.02	−0.216 ± 0.02	**−0.217 ± 0.019**
Stc	813.236 ± 181.248	80853.871 ± 17821.993	89024.824 ± 18000.346	**89997.921 ± 17900.821**
Qim	−0.230 ± 0.01	−0.227 ± 0.023	−0.228 ± 0.011	**−0.230 ± 0.007**



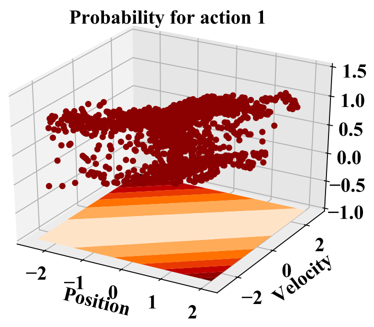
(a) Episode 43



(b) Episode 89



(c) Episode 153



(d) Episode 157

Figure 8: The actor $\pi(x; \theta)$ trained on the Humanoid-v2. The surfaces represent the functions. The blue dots show the trajectories in the state-action using the current policy. The red and blue contours show the forces that shape the surfaces.

6.5 Solution Stability

We finally show that the proposed method can provide stable solutions. Figure 8 shows the learned policy at the 43th, 89th, 153th, and 157th episodes for the first scenario. One episode includes 10240 time steps. The red contours show the forces, $\sum_k \nabla_{\theta_\pi} \log \pi_\theta(x_k, a_k)\, \delta^i_{k,t}$, that shape the surfaces. The lines show the probabilities for one particular action. After the 157th episode, the contours show that the learned policy becomes more and more stable because the policy becomes more consistent across episodes, indicating that the policy is "settled" after an appropriate number of episodes.

It is worth noting that the stability of the proposed method is determined by both the stability of the actor-critic network and the stability in learning $\mathcal{W}$. Moreover, the stability of the actor-critic network and the stability in learning $\mathcal{W}$ are interdependent. On the one hand, if the actor-critic network is stable, $\mathcal{W}$ can often be stabilized since $\mathcal{W}$ becomes the main one to be learned after the stabilization of the actor-critic network. On the other hand, if $\mathcal{W}$ is stable, the actor-critic network becomes the main one to be trained after the value of $\mathcal{W}$ is stabilized. Notice that $\mathcal{W}$ described in Subsection 6.2 becomes stable as the training proceeds. Hence, the actor-critic network will also become stable, which is consistent with the obtained stable action policy shown in Figure 8.

7. Conclusions

In multi-objective optimization problems, the possibly conflicting objectives necessitates a trade-off when multiple objectives need to optimize simultaneously. A typical approach is to minimize a loss of a weighted linear summation of all objective functions. However, this approach may be effective for limited cases (e.g., when the objectives do not compete). To address the potential competing nature among these objectives, we proposed a new efficient gradient-based multi-objective deep reinforcement learning to solve high-dimensional multi-objective decision making problems in continuous control environments. The proposed method optimizes vectorized proxy objectives sequentially based on proximal policy optimization, actor-critical network, and the derivation of optimal weights via marginal weight using a new efficient PAOLS method.

By explicitly quantifying inter-objective relationship via solving the min-norm point in the convex optimization, the relative importance of the objectives unknown *a prior* can be obtained via reinforcement learning. Each entry in the discovered weights $\mathcal{W}$ specifies and explains the relative impact of one objective on another objective in the optimization step.

Acknowledgment

The work was supported by the Office of Naval Research under Grants N000141712613 and N000141912278.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: a system for large-scale machine learning.. In *USENIX Symposium on Operating Systems Design and Implementation*, Vol. 16, pp. 265–283.

Abels, A., Roijers, D. M., Lenaerts, T., Nowé, A., & Steckelmacher, D. (2018). Dynamic weights in multi-objective deep reinforcement learning. *arXiv preprint arXiv:1809.07803*.

Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., & Kautz, J. (2016). Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*.

Barrett, L., & Narayanan, S. (2008). Learning all optimal policies with multiple criteria. In *International Conference on Machine Learning*, pp. 41–47.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Désidéri, J.-A. (2012). Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus Mathematique*, *350*(5-6), 313–318.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., & Wu, Y. (2017). Openai baselines. *GitHub, GitHub Repository.*

Ehrgott, M. (1995). Lexicographic max-ordering-a solution concept for multicriteria combinatorial optimization..

Fliege, J., & Svaiter, B. F. (2000). Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, *51*(3), 479–494.

Gordon, G., & Tibshirani, R. (2012). Karush-kuhn-tucker conditions. *Optimization*, *10*(725/36), 725.

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems*, pp. 3338–3346.

Hosseinzade, E., & Hassanpour, H. (2011). The Karush-Kuhn-Tucker optimality conditions in interval-valued multiobjective programming problems. *Journal of Applied Mathematics & Informatics*, *29*(5_6), 1157–1165.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, *91*(9), 992–1007.

Lin, J. G. (2005). On min-norm and min-max methods of multi-objective optimization. *Mathematical Programming*, *103*(1), 1–33.

Liu, C., Xu, X., & Hu, D. (2015). Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *45*(3), 385–398.

Maddison, C. J., Huang, A., Sutskever, I., & Silver, D. (2014). Move evaluation in Go using deep convolutional neural networks. *arXiv preprint arXiv:1412.6564.*

Mannor, S., & Shimkin, N. (2002). The steering approach for multi-criteria reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1563–1570.

Mannor, S., & Shimkin, N. (2004). A geometric approach to multi-criterion reinforcement learning. *Journal of Machine Learning Research*, *5*, 325–360.

Marbach, P., & Tsitsiklis, J. N. (2001). Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, *46*(2), 191–209.

Miettinen, K., & Mäkelä, M. (1995). Interactive bundle-based method for nondifferentiable multiobjeective optimization: Nimbus. *Optimization*, *34*(3), 231–246.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Mossalam, H., Assael, Y. M., Roijers, D. M., & Whiteson, S. (2016). Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707.*

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. (2015). Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*.

Nakayama, H., Yun, Y., & Yoon, M. (2009). *Sequential approximate multiobjective optimization using computational intelligence*. Springer Science & Business Media.

Nguyen, T. T. (2018). A multi-objective deep reinforcement learning framework. *arXiv preprint arXiv:1803.02965*.

Oh, J., Guo, X., Lee, H., Lewis, R. L., & Singh, S. (2015). Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems*, pp. 2863–2871.

Pan, X., You, Y., Wang, Z., & Lu, C. (2017). Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*.

Parisi, S., Pirotta, M., & Peters, J. (2017). Manifold-based multi-objective policy search with sample reuse. *Neurocomputing*, *263*, 3–14.

Parisi, S., Pirotta, M., & Restelli, M. (2016). Multi-objective reinforcement learning through continuous pareto manifold approximation. *Journal of Artificial Intelligence Research*, *57*, 187–227.

Parisi, S., Pirotta, M., Smacchia, N., Bascetta, L., & Restelli, M. (2014a). Policy gradient approaches for multi-objective sequential decision making. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 2323–2330.

Parisi, S., Pirotta, M., Smacchia, N., Bascetta, L., & Restelli, M. (2014b). Policy gradient approaches for multi-objective sequential decision making: A comparison. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 1–8.

Peitz, S., & Dellnitz, M. (2016). Gradient-based multiobjective optimization with uncertainties. *arXiv preprint arXiv:1612.03815*.

Pinder, J. (2016). *Multi-objective reinforcement learning framework for unknown stochastic & uncertain environments*. Ph.D. thesis, University of Salford.

Pirotta, M., Parisi, S., & Restelli, M. (2015). Multi-objective reinforcement learning with continuous pareto frontier approximation. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Poirion, F., Mercier, Q., & Désidéri, J.-A. (2017). Descent algorithm for nonsmooth stochastic multiobjective optimization. *Computational Optimization and Applications*, *68*(2), 317–331.

Rockafellar, R. T., & Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, *16*(1), 119–147.

Roijers, D. M., Vamplew, P., Whiteson, S., & Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, *48*, 67–113.

Roijers, D. M., Whiteson, S., Oliehoek, F. A., et al. (2014a). Linear support for multi-objective coordination graphs. In *The International Conference on Autonomous Agents & Multiagent Systems*, pp. 1297–1304.

Roijers, D. M., Scharpff, J., Spaan, M. T., Oliehoek, F. A., De Weerdt, M., & Whiteson, S. (2014b). Bounded approximations for linear multi-objective planning under uncertainty. In *International Conference on Automated Planning and Scheduling*, pp. 262–270.

Roijers, D. M., Whiteson, S., & Oliehoek, F. A. (2015). Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, *52*, 399–443.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sener, O., & Koltun, V. (2018). Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pp. 527–538.

Shelton, C. R. (2001). Importance sampling for reinforcement learning with multiple objectives..

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140–1144.

Spall, J. C. (1992). Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, *37*(3), 332–341.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tajmajer, T. (2018). Modular multi-objective deep reinforcement learning with decision values. In *2018 Federated Conference on Computer Science and Information Systems*, pp. 85–93.

Tesauro, G., Das, R., Chan, H., Kephart, J., Levine, D., Rawson, F., & Lefurgy, C. (2008). Managing power consumption and performance of computing systems using reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1497–1504.

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pp. 5026–5033.

Tsitsiklis, J. N., & Van Roy, B. (1999). Average cost temporal-difference learning. *Automatica*, *35*(11), 1799–1808.

Uchibe, E., & Doya, K. (2007). Constrained reinforcement learning from intrinsic and extrinsic rewards. In *6th IEEE International Conference on Development and Learning*, pp. 163–168.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., & Dekker, E. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, *84*(1-2), 51–80.

Vamplew, P., Dazeley, R., & Foale, C. (2017). Softmax exploration strategies for multiobjective reinforcement learning. *Neurocomputing*, *263*, 74–86.

Vamvoudakis, K. G., & Lewis, F. L. (2010). Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem. *Automatica*, *46*(5), 878–888.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning.. In *Thirtieth AAAI conference on Artificial Intelligence*, Vol. 2, p. 5.

Van Moffaert, K., Drugan, M. M., & Nowé, A. (2013). Scalarized multi-objective reinforcement learning: Novel design techniques.. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 191–199.

Van Moffaert, K., & Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, *15*(1), 3483–3512.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1995–2003.

Ward, D., & Lee, G. (2001). Generalized properly efficient solutions of vector optimization problems. *Mathematical Methods of Operations Research*, *53*(2), 215–232.

Yang, R., Sun, X., & Narasimhan, K. (2019). A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *Advances in Neural Information Processing Systems*, pp. 14636–14647.