ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200202

A Two-Stage Multi-Objective Deep Reinforcement Learning Framework

Diqi Chen[1] and **Yizhou Wang**[2] and **Wen Gao**[3]

Abstract. In multi-objective decision making problems, multi-objective reinforcement learning (MORL) algorithms aim to approximate the Pareto frontier uniformly. A naive approach is to learn multiple policies by repeatedly running a single-objective reinforcement learning (RL) algorithm on scalarized rewards. The scalarization methods denote the preferences of objectives, which are different in each run. However, in this way, the model representation and computation are redundant. Furthermore, uniform preferences can not guarantee a uniformly approximated Pareto frontier. To address these problems and leverage the expressive power of deep neural networks, we propose a two-stage MORL framework integrating a multi-policy deep RL algorithm and an evolution strategy algorithm. Firstly, a multi-policy soft actor-critic algorithm is proposed to collaboratively learn multiple policies which are assigned with different scalarization weights. The lower layers of all policy networks are shared. The first-stage learning can be regarded as representation learning. Secondly, the multi-objective covariance matrix adaptation evolution strategy (MO-CMA-ES) is applied to fine-tune policy-independent parameters to approach a dense and uniform estimation of the Pareto frontier. Experimental results on two benchmarks (Deep Sea Treasure and Adaptive Streaming) show the superiority of the proposed method.

1 Introduction

Deep reinforcement learning (RL) algorithms have been applied in many challenging decision making problems, such as video games [21, 44, 20], the game of Go [34, 35] and robotics [15, 9, 8]. In these scenarios, only one objective is optimized. Nevertheless, many real-world decision making problems consider more than one objective. Network routing takes energy, latency and channel capacity into account [23]. Medical treatment needs to release symptoms and minimize side effects [17, 16]. Economic systems are analyzed from both economic and ecological perspectives [41].

A naive approach is to run a single-objective RL algorithm independently and repeatedly. In each iteration, a specific scalarization method is adopted to transfer multiple objectives into a single one. The scalarization method denotes a specific preference for the objectives. For example, in a linear weighted scalarization method, the higher weight reflects that the related objective is more preferred. This kind of algorithms learns only one policy in a single run, so it is named a single-policy method. However, the naive single-policy method is inefficient due to the redundancy in both computation and model representation. Furthermore, uniform preferences can not guarantee a uniform approximation of the Pareto frontier.

To address these problems and leverage the expressive power of deep neural networks, we propose a two-stage MORL framework integrating a deep multi-policy RL algorithm and a multi-objective evolution strategy algorithm. At the first stage, the soft actor-critic (SAC) algorithm [9, 8] is extended to a multi-policy soft actor-critic (MPSAC) algorithm. The extension is based on the assumption that exploration efficiency can be improved by collaboratively learning multiple policies with different targets. The model collaboratively learns a group of policy networks. Each policy targets on a specific scalarized objective. In this work, the linear weighted scalarization method is adopted. During training, the model maintains a multi-channel replay buffer. Each channel is related to a group of scalarization weights. The collaboration is conducted through replay buffer sharing. Furthermore, to reduce the redundancy of model representation and computation, all policy networks share the same low-level parameters.

At the second stage, the multi-objective covariance matrix adaptation evolution strategy (MO-CMA-ES) is applied to fine-tune the policy-independent parameters. The policy-independent parameters are vectorized as the chromosomes of individuals. The individuals of the first generation are initialized from the policies learned at the first stage. In each generation, every individual generates one offspring through its covariance matrix. The new offsprings are evaluated and marked by a fitness score vector. Then all individuals are ranked and half elitist individuals are selected as the parents of the next generation. The ranks are determined by the non-dominated ranks at first. Then, in each non-dominated set, the crowding distance is utilized as the second ranking criterion to approximate the Pareto frontier uniformly.

The proposed framework can be applied in both Markovian and non-Markovian environments. Specifically, in the non-Markovian situation, recurrent neural networks (RNN) are adopted as the policy networks. The recurrent connections are learned at the first learning stage and fixed at the second stage. Previous works [33, 37] adopt evolution strategy and neuroevolution to train deep neural networks, but training RNNs are not considered in their frameworks. Safe Mutations [14] is proposed to learn RNN policies, while this method needs to compute gradients during the neuroevolution algorithm.

The principal contribution of this paper is as follows: (1) A multi-policy gradient-based deep reinforcement learning algorithm with high sampling efficiency. (2) A multi-objective reinforcement learning framework to approach a well-distributed Pareto frontier. (3) This

[1] Key Lab of Intelligent Information Processing, Institute of Computing Technology and University of Chinese Academy of Sciences, China, E-mail: cdq@pku.edu.cn
[2] National Engineering Lab for Video Technology, Institute of Digital Media, Key Lab of Machine Perception (MoE), School of Electronics Engineering and Computer Science (EECS), Peking University, and Deepwise AI Lab, China, E-mail: Yizhou.Wang@pku.edu.cn
[3] National Engineering Lab for Video Technology, Institute of Digital Media, Key Lab of Machine Perception (MoE), School of Electronics Engineering and Computer Science (EECS), Peking University, China. E-mail: wgao@pku.edu.cn

framework learns compact model representation with lower computational complexity. (4) This framework can be applied in the non-Markovian environment to train deep recurrent neural networks.

Experimental results on two benchmarks (Deep Sea Treasure and Adaptive Streaming) show the superiority of our proposed method.

2 Preliminaries

Markov Decision Process (MDP). In a single-objective RL problem [39], an agent learns from interacting with the environment to achieve a goal. The problem can be modeled as a Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$. At each time step t, the agent receives the current state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ according to a stochastic policy $\pi(a_t|s_t)$. Then the agent observes a reward $r_t = R(s_t, a_t) \in \mathcal{R}$ and transitions to a new state s_{t+1}. The transition is based on the transition probability $p(s_{t+1}|s_t, a_t)$. The goal is to maximize an expectation over the discounted accumulated reward $\sum_{t=0}^{T} \gamma^t r_t$, where $\gamma \in [0, 1]$ is a discounted factor. $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ denote the state space, the action space and the reward space, respectively. Following a specific policy π, $\rho_\pi(s, a) = \mathbb{E}_\pi[\frac{1}{T}\sum_{t=1}^{T} \mathbb{1}(s_t = s, a_t = a)]$ denotes the state-action marginal distribution, which measures the probability of the state-action pairs being visited in a finite-length episode. In this work, we consider the continuous state space and the discrete action space.

Multi-Objective Markov Decision Process (MOMDP). In an MOMDP, at each time step, the agent receives a reward vector $\boldsymbol{r}_t = \boldsymbol{R}(s_t, a_t) \in \mathcal{R}^M$, where M is the number of the objectives. The ultimate goal is to approximate the Pareto frontier uniformly. The Pareto frontier represents a set of non-dominated solutions.

Soft Actor-Critic (SAC) algorithm. SAC algorithm [9, 8] is utilized as our baseline single-policy RL algorithm. To make this paper as self-contained as possible, SAC is described elaborately. SAC is a policy iteration method designed on a maximum entropy RL framework. The objective function is:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{1}$$

where $\mathcal{H}$ denotes the entropy of the policy and α denotes the temperature parameter. During learning, SAC updates the soft Q-function and the soft policy iteratively. The soft Q-function is defined as $Q(s_t, a_t)$, which is approximated by a neural network, called Q-network. Under this framework, the Bellman update of Q-network is processed by minimizing the square error of the estimated Q-value and the target value $\bar{Q}(s_t, a_t)$:

$$\begin{aligned} J_Q &= [Q(s_t, a_t) - \bar{Q}(s_t, a_t)]^2, \\ \bar{Q}(s_t, a_t) &= [r(s_t, a_t) + \gamma\mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}[Q(s_{t+1}, a_{t+1}) - \\ &\qquad \alpha \log \pi(a_{t+1}|s_{t+1})]]. \end{aligned} \tag{2}$$

Then the policy network is updated towards the exponential of the new soft Q-function:

$$\begin{aligned} \pi_{\text{new}} &= \arg\min_{\pi\in\Pi} J_\pi \\ &= \arg\min_{\pi\in\Pi} \mathrm{D}_{\mathrm{KL}}\Big(\pi(\cdot|s_t)\Big\|\frac{\exp(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t, \cdot))}{Z^{\pi_{\text{old}}}(s_t)}\Big), \end{aligned} \tag{3}$$

where J_π denotes the objective function of the policy network, D_{KL} denotes the Kullback-Leibler divergence and $Z^{\pi_{\text{old}}}(s_t)$ represents the partition function.

SAC is an off-policy algorithm which maintains a replay buffer during learning. The model learning is an iterative process and each iteration consists of a sampling stage and a learning stage. At the sampling stage, the agent interacts with the environment following the current policy and the experience is stored in the replay buffer. At the learning stage, the method updates the Q-network and the policy network. In the practical implementation, SAC mainly adopts two techniques to make learning more stable and efficient. Firstly, two soft Q-functions are adopted to mitigate positive bias, which is suggested in double Q-learning [44]. Specifically, two Q-networks are learned simultaneously, and the minimum value of two estimates are used to compute the target value for the Bellman update. Secondly, target networks are utilized to calculate the target value, which is suggested in [22]. Since SAC maintains two Q-networks, SAC also maintains two target networks, which are fixed during the Bellman update and updated slowly in the end of each iteration.

Multi-objective evolution strategy. CMA-ES [10] is a real-valued optimization method relying on covariance matrix adaptation. It is known to work well for solution spaces of up to a few thousand parameters [7, 6, 33, 37]. MO-CMA-ES [13, 11, 40, 47] is a multi-objective extention of CMA-ES. As claimed in [33], the advantages using evolution strategy for RL are: 1) little influence from the sparsely distributed rewards and long time horizons; 2) no gradients computation; 3) avoidance of approximating the value function.

3 Method

The proposed method learns a set of policies $\Pi^\star = \{\pi_i^\star\}_{i=1}^N$ to approximate the Pareto frontier uniformly.

3.1 Multi-Policy Soft Actor-Critic Algorithm

Algorithm. The single-policy SAC is extended to a multi-policy algorithm, which learns more than one policy in the same time. Each policy is related to a specific preference of objectives. The preference is represented by specific weights. During learning, A multi-channel replay buffer is maintained. Each channel is associated with a specific policy. Collaborative learning is conducted through buffer sharing to improve sampling efficiency.

Specifically, Algorithm 1 shows the pseudocode, in which N_0 denotes the number of learned policies. The algorithm maintains N_0 groups of Q-networks and policy networks. The double Q-learning technique and target network technique are also adopted in this algorithm to stabilize learning. Therefore, in the i-th group, there are two Q-networks $\theta_{i,1}, \theta_{i,2}$, two target Q-networks $\bar{\theta}_{i,1}, \bar{\theta}_{i,2}$ and one policy network ϕ_i. The target networks are initialized randomly. And each group is associated with a specific weight vector $\boldsymbol{w}_i$. In each epoch, the algorithm includes two key procedures. Firstly, the agents sample data by interacting with the environment. The experience tuple $(s_t, a_t, \boldsymbol{r}(s_t, a_t), s_{t+1}, i)$ of agent i is stored in the i-th channel of the replay buffer. Secondly, training data is sampled from the replay buffer to update Q-networks and policy networks. Each group of Q-networks and policy networks is optimized by the single-policy SAC on the rewards scalarized by linear weights $\boldsymbol{w}_i$. The objective functions J_Q and J_π are defined in Preliminaries. Thus the target networks are updated consequently. As suggested in [5], the A2C scheme is adopted to learn the proposed algorithm. Specifically, the agents interact with the environment asynchronously, while the networks are optimized synchronously in a central agent. In this setting, the central agent needs to broadcast actions and receive rewards and

Algorithm 1 Multi-Policy Soft Actor-Critic Algorithm

Input: Initialize N_0 Q-networks with $\{\theta_{i,1}, \theta_{i,2}\}_{i=1}^{N_0}$, N_0 target Q-networks with $\{\bar{\theta}_{i,1}, \bar{\theta}_{i,2}\}_{i=1}^{N_0}$ and N_0 policy networks with $\{\phi_i\}_{i=1}^{N_0}$.
Initialize an empty multi-channel replay buffer: $\mathcal{D} = \{\mathcal{D}_i\}$, where $\mathcal{D}_i \leftarrow \emptyset$ for $i = 1, 2, \cdots, N_0$.
Initialize the step sizes λ_Q, λ_π, τ for learning Q-networks, policy networks and target Q-networks, respectively.
Set the scalarization weights: $\{\boldsymbol{w}_i\}_{i=1}^{N_0}$.
for each epoch **do**
  for each environment step **do**
    for $i = 1, 2, \cdots, N_0$ **do** asynchronously
      Sample action from the policy: $a_t \sim \pi_{\phi_i}(a_t|s_t)$
      Get transition from the environment: $s_{t+1} \sim p(\cdot|s_t, a_t)$
      Store the experience in the replay buffer: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{(s_t, a_t, \boldsymbol{r}(s_t, a_t), s_{t+1}, i)\}$
    end for
  end for
  for each gradient step **do**
    Sample experience from the replay buffer $\mathcal{X}_i \subseteq \mathcal{D}_i$ for $i = 1, 2, \cdots, N_0$.
    for $i = 1, 2, \cdots, N_0$ **do** synchronously
      Construct training data: $\widetilde{\mathcal{X}}_i = \bigcup_{j=1}^{N_0} \mathcal{X}_j$ or $\widetilde{\mathcal{X}}_i = \mathcal{X}_i$ or $\widetilde{\mathcal{X}}_i \leftarrow \mathcal{X}_i \bigcup_{j \in \Omega_i} \mathcal{X}_j$. Ω_i denotes the set of adjacent policies.
      Weighted scalarization: $\widetilde{\mathcal{X}}_i \leftarrow \{(s_t, a_t, \boldsymbol{w}_i^T \boldsymbol{r}(s_t, a_t), s_{t+1})\}$, where the channel index is neglected.
      Update the Q-function parameters: $\theta_{i,j} \leftarrow \theta_{i,j} - \lambda_Q \hat{\nabla}_{\theta_{i,j}} J_Q(\theta_{i,j})$ for $j \in \{1, 2\}$
      Update policy weights: $\phi_i \leftarrow \phi_i - \lambda_\pi \hat{\nabla}_{\phi_i} J_\pi(\phi_i)$
      Update target network weights: $\bar{\theta}_{i,j} \leftarrow \tau\theta_{i,j} + (1-\tau)\bar{\theta}_{i,j}$ for $j \in \{1, 2\}$
    end for
  end for
end for
Output: Parameters of learned policy networks: $\{\phi_i\}_{i=1}^{N_0}$

the next states. See Algorithm 1 for a more formal description of the algorithm.

SAC is not the only alternative RL algorithm for our framework. Other single-objective off-policy RL algorithms such as Deep Q-Learning [21] and DDPG [15] can be extended in the same way.

Linear scalarization method. Considering a specific policy, the linear scalarization method transfers the reward vector $\boldsymbol{r}_t$ to a scalar: $r_t = \boldsymbol{w}^T \boldsymbol{r}_t$, where $\boldsymbol{w}$ denotes the weights. Since the number of objectives is M, $\boldsymbol{w}$ is a M-dim vector. Each element of the weight vector should be equal or larger than zero. And at least one element is larger than zero. How to assign the weights reveals the preference/importance of the related objectives. Furthermore, the strategy of assigning weights is different from one problem to another. Thus, considering two objectives, the method empirically fixes the weight of the first objective as 1 and varifies the weight of the second objective in a predefined range uniformly. Even though there are limitations in the linear scalarization method when the Pareto frontier is concave [43]. Specifically, the methods with linear scalarization method can not achieve the Pareto optimal solutions in the concave region. This problem is handled by the evolution strategy at the second stage.

Shared low-level layers. To reduce computational complexity, the policy networks share low-level network parameters. From the perspective of representation learning, joint optimization learns a more compact and robust representation, thus the performance is improved. The learned representation guarantees the performance at the second stage, in which the low-level parameters are fixed. In the non-Markovian environment, recurrent neural networks (RNN) are adopted as policy networks. The recurrent connections are utilized to gather the information of previous time steps. At the second stage, the recurrent part of RNN is fixed. Note that we adopt asynchronous exploration and synchronous learning, so there is no communication bottleneck for sharing low-level layers. Since the shared low-level structures are not compulsive for MPSAC, specified symbols of shared parameters and policy-independent parameters are not demonstrated in Algorithm 1.

Replay buffer sharing. The collaborative learning is implemented by sharing replay buffer. We belive that each policy learns a specific exploration mode since each policy aims to optimize a specific weighted goal. Hence closer scalarization weights may lead to closer modes. The collaboration is two folds: firstly, policies with close modes follow similar/overlap exploration pattern to cooperate on data sampling; secondly, if a policy trained by the sampled data from another policy working in different mode, the value function is learned more critically, thus its exploration efficiency is improved. The multi-channel replay buffer consists of experience tuples. Each tuple is represented by $\{(s_t, a_t, \boldsymbol{r}(s_t, a_t), s_{t+1}, i)\}$, which includes the current state s_t, action a_t, reward vector $\boldsymbol{r}(s_t, a_t)$, the next state s_{t+1} and the index of associated policy i. The index can also be explained as the channel index. By storing the reward vectors rather than the scalarized rewards, each policy can utilize data sampled from other policies. According to which part of the replay buffer a policy can access, we investigate three buffer sharing strategies. In the global sharing strategy, a policy accesses the whole buffer. In the no sharing strategy, each policy only accesses its buffer. And in the neighbor sharing strategy, each policy can utilize its buffer and the buffer of the adjacent policies. The adjacent set is defined as the set of policies with closest scalarization weights. Through ablation experiments, the global sharing strategy is regarded as the default strategy since this strategy achieves the best result.

3.2 Multi-Objective Covariance Matrix Adaptation Evolution Strategy (MO-CMA-ES)

The implementation of MO-CMA-ES is adapted from the Shark machine learning library [12], which is mainly based on [13, 47]. The

Table 1. Experimental results of Deep Sea Treasure. HI denotes hypervolume indicator. The number in the bracket of deterministic results indicates the number of approached Pareto optimal policies. The number of iterations of our algorithm includes two parts since the method has two stages, thus "+" is used to separate two numbers.

	HI (Deterministic)	HI (Stochastic)	Iterations	Wall-Clock Time	Parameters
HB	$1364.00 \pm 0.00(1.00 \pm 0.00)$	-	$4,000$	$6.68 \times 10^4 s$	8.97×10^4
RA	$1406.33 \pm 54.41(2.17 \pm 0.37)$	1857.75 ± 156.19	$4,000$	$7 \times 10^4 s$	8.45×10^4
PFA	$1346.47 \pm 125.58(2.03 \pm 0.18)$	1713.71 ± 274.73	$1,400$	$2.45 \times 10^4 s$	8.45×10^4
Ours	$\mathbf{1756.13 \pm 28.20(9.53 \pm 0.62)}$	-	$400 + 100$	$3 \times 10^3 s$	1.31×10^4
Ours	-	$\mathbf{2134.81 \pm 88.80}$	$400 + 1500$	$1.93 \times 10^4 s$	1.31×10^4

Algorithm 2 Multi-Objective Convariance Matrix Adaptation Evolution Strategy

1: **Input:** Initialize the first generation with N individuals. Each individual a_i consists of a search point $\bar{\phi}_i$, a global step size σ_i and a covariance matrix C_i.
2: Evaluate the first generation, each individual a_i is labeled with a fitness vector $\boldsymbol{y}_i$.
3: **for** each generation **do**
4: Generate one offspring for each individual: $\bar{\phi}_i^{(g+1)} \sim \mathcal{N}(\bar{\phi}_i^{(g)}, \sigma_i^{(g)} C_i^{(g)})$.
5: Evaluate N offsprings by simulation, each individual $a_i^{(g)}$ is labeled with a fitness vector $\boldsymbol{y}_i^{(g)}$.
6: Sort $2N$ individuals (parents and offsprings), then select first N individuals.
7: Update step size and covariance matrix for each individual.
8: **end for**
9: **Output:** Optimized parameters: $\{\phi_i\}_{i=1}^N$

MO-CMA-ES aims to refine the rough approximation of the Pareto frontier from the first stage.

Algorithm 2 shows the general procedures (four main steps) of MO-CMA-ES. The method maintains N individuals. Note that it is not compulsive to constrain that N must be equal to N_0. Each individual a_i consists of a chromosome (search point) $\bar{\phi}_i$, a global step size σ_i and a covariance matrix C_i. The chromosome $\bar{\phi}_i$ is a n-dimensional vector initialized by vectorizing the policy-independent parameters of a randomly selected learned policy at the first stage. σ_i is initialized as 0.1 and C_i is initialized as a n-dimensional identity matrix. The initialized individual a_i is evaluated by simulation and labeled by a fitness vector $\boldsymbol{y}_i$. Each element of $\boldsymbol{y}_i$ represents the accumulated reward of the related objective. In every generation, each individual generates one offspring by sampling from the Gaussian distribution: $\bar{\phi}_i \sim \mathcal{N}(\bar{\phi}_i, \sigma_i C_i)$, in which the mean and the covariance matrix are $\bar{\phi}_i$ and $\sigma_i C_i$. New offsprings are evaluated by simulation and labeled with fitness vectors. Then all individuals are ranked and the N elitists are selected as the parents of the next generation. At last, the covariance matrix of each selected individual is updated consequently. Please refer to [13, 47] for more details. The algorithm repeats this procedure until the stopping criterion is met.

Selection step. Two ranking criteria are considered: the non-dominance level and the crowding distance, as suggested by [4]. The non-dominance level is determined by recursively exclude non-dominant individuals from the solution set. Formally, let A be a population and a, a' be two individuals. The non-dominant set of A is defined as:

$$\mathrm{ndom}(A) = \{a \in A | \nexists a' \in A : a \prec a'\}, \tag{4}$$

where $\prec$ denotes the Pareto-dominance relation. Let $A_0 = A$, $A_l = A_{l-1} \backslash \mathrm{ndom}(A_{l-1})$, for $l \geq 1$, where "\" denotes a removing operation. Thus $\mathrm{rank}(a, A) = i + 1$, iff $a \in \mathrm{ndom}(A_i)$. When the Pareto frontier is concave, the optimal solution in the concave region still has a high non-dominance level, then the limitation of linear scalarization is handled.

Through the first criterion, the individuals are ranked to many non-dominant sets. The crowding distance is used as the second ranking criterion to rank the points in a non-dominant set. For M objectives, the crowding distance of point a in the non-dominant set A' is defined by:

$$c(a, A') = \sum_{m=1}^{M} c_m(a, A') / (f_m^{\max} - f_m^{\min}), \tag{5}$$

where $f_m^{\max}$ and $f_m^{\min}$ are maximum and minimum values of the m-th objective respectively. $c_m(a, A')$ is positive infinite when a is a boundary point. Otherwise, $c_m(a, A')$ is the distance between two closest points of a w.r.t. the m-th objective. Therefore, the boundary points and the less crowded points are preferred, so that the approximated Pareto frontier is more spread and uniform.

4 Experiments

In this section, we introduce the comparative methods, the evaluation protocol, the experimental results on two benchmark problems and ablation studies. Furthermore, some illustrative tables and figures are demonstrated for a detailed comparison.

4.1 Comparative Methods

The proposed methods are compared with three methods, the hypervolume-based (HB) [45] algorithm, the radial algorithm (RA) [30] and the Pareto following algorithm (PFA) [30]. HB is an extension of Deep Q-learning [21]. This method maintains value function approximation for each objective and takes actions according to the largest hypervolume indicator. Hence, no scalarization method is applied in HB. In the other two comparative methods, the linear scalarization method is adopted. RA optimizes more than one policy simultaneously. Each policy is assigned with a specific weight vector on objectives. In each iteration, through applying SAC, every individual policy is optimized independently by the gradients related to its assigned preference. From another idea, PFA is optimized directedly on the Pareto frontier. Specifically, at first, an initial model is learned by SAC with initialized preference weights. Then the weights are tuned gradually and uniformly, and the model is fine-tuned for each preference iteratively. Naive stochastic gradient descent methods are utilized in RA and PFA, considering the computational complexity. For a fair comparison, the same network structures are adopted for the proposed methods and the comparative methods.

4.2 Evaluation Protocol

The comparative experiments are conducted on two benchmark problems: Deep Sea Treasure and Adaptive Streaming. The hypervolume indicator [50] is utilized as our evaluation protocol, which is suggested by [42] because of its sensitivity on the improvements in any frontal characteristics (accuracy, extent, diversity). The experimental results are represented by the mean and standard deviation to show the overall performance and stability. The results are calculated from repeated independent runs. The number of the repeats is 30 and 10 for two benchmark problems, respectively. Different random seeds are used for training and testing. We also demonstrate the wall-clock time and the number of parameters. The codes run on PyTorch 1.0 [31], Ubuntu 16.04 system with 44 Intel(R) Xeon(R) CPUs.

4.3 Problem 1: Deep Sea Treasure

Problem Description. This is an episodic task. A submarine searches for treasures in a 10×11 grid. The treasures with different values are in different locations. At each episode, the submarine starts from the top left corner, searches by four actions- left, right, up and down. The end condition is reaching a treasure or moving $1,000$ steps. The reward is a 2-dim vector. The first element is negative time and the second element is the reached treasure value. Therefore, the goal is to maximize the treasure value and minimize the time penalty. This task is created to highlight the limitations of linear weighted scalarization [43], since the Pareto frontier of this task is globally concave.

This task is tested in two different ways. Firstly, the learned policies are tested deterministically. The agent always takes action with maximum probability. In this setting, there are ten Pareto optimal policies. Each optimal policy leads to one treasure location. Secondly, the policies are tested stochastically. During testing, the actions are sampled according to the learned policy. The results are averaged from 20 simulations.

Consequently, the proposed algorithm adopts different fitness evaluation mechanisms at the evolution strategy stage to handle these two test settings. For testing deterministic policies, the evaluation is conducted in only one episode. During learning stochastic policy, the fitness score vector is averaged from the results of 20 episodes.

Implementation Details. The input state is a 61-dim binary vector. Each element of the vector is related to a specific position. All elements are zero except the element representing the current position of the submarine. Both the policy network and the Q-network consist of two linear layers with ReLU activation functions. The hidden layer has 128 hidden units. A softmax layer is on the top of the policy network. At the first stage, the model simultaneously learns five policies by 25 agents. The linear weights of learned policies are $(1.0, 0.1)$, $(1.0, 0.12)$, $(1.0, 0.14)$, $(1.0, 0.16)$ and $(1.0, 0.18)$, respectively. The model learns 400 epochs, each of which includes 100 sampling iterations and 100 training iterations. Each mini batch includes 100 samples from each agent. The learning rate is 0.001. At the second stage, the number of individuals in each generation is 10. To calculate the hypervolume indicator, the reference point is $(-30, 0)$.

Experimental Results of Deterministic Policies. HB only maximizes hypervolume indicator for a single policy, which causes poorly diversified solutions and low hypervolume indicator on all learned policies. In this problem, HB converges to the policy approaching the highest treasure value. As shown in Table 1, when testing deterministic policies, policy gradient methods with linear scalarization (RA and PFA) are not able to approach all Pareto frontier. Both RA and PFA achieve near two out of ten Pareto optimal solutions, which causes low hypervolume indicator. Our method with evolution strategy can approach all optimal policies and higher hypervolume indicator.

Experimental Results of Stochastic Policies. HB is not tested in this scenario because it is not trivial to transfer HB into a stochastic policy. As shown in Table 1, compared with PFA, RA achieves higher average hypervolume indicator and lower standard deviation, which shows that RA is more effective and more stable than PFA in this task. Nevertheless, PFA needs less training time since it operates on the Pareto frontier with model reusing. Our method outperforms these comparative methods with a considerable margin. Furthermore, to consider the smoothness of the approximated Pareto frontier, the solutions are ranked so that one objective is monotonically increasing and the other objective is monotonically decreasing. Then the Euclidean distance between all pairs of neighbor solutions is calculated. By doing this, a distance array is achieved. The standard deviation of this distance array is utilized to measure the smoothness. The smoothness of RA, PFA and our method are 21.72, 19.54 and 11.65 respectively, which indicates that our method approaches better distributed frontier.

Work Loads. As shown in Table 1, considering the number of iterations, the wall-clock time and the number of parameters, our algorithm has lower time and space complexity. Especially, to store optimal solutions of deterministic policies, our algorithm uses much fewer parameters to represent more optimal solutions than comparative methods.

4.4 Problem 2: Adaptive Streaming

Problem Description. This problem is derived from [1], which proposed a rate-distortion optimization framework for Adaptive Streaming [38, 19, 36, 49]. In this problem, the server segments each video into a set of 4-second consecutive video chunks. Each chunk is encoded into different representations. Each representation is characterized by a perceptual quality and a bitrate. The perceptual quality is estimated from the encoded video segment and the assumed viewing device. During downloading, the controller determines which representation to download for the next chunk. Then the environment downloads the chunk, updates the buffer occupancy, tracks the throughput and returns the new states to the agent. Except for the perceptual quality, the stalling effect and adaptation effect are also considered in a unified Quality-of-Experience (QoE) score. In the rate-distortion optimization framework, the first objective is to maximize the overall QoE and the second one is to minimize the average bitrate consumption. Since the environment in this problem is non-Markovian, RNNs are used to handle temporal information. Consequently, during learning MPSAC models, a complete experience sequence is treated as the storing/retrieving unit.

Implementation Details. The dataset includes 5 source videos and 300 network traces. The videos are encoded by H.264 video encoder as 13 representations. We only consider Phone as the viewing device. The same videos are used during training, validation and testing. Network traces are equally split into a training set, a validation set and a testing set. Both the policy network and Q-network are constructed by two GRU layers and one linear layer. All hidden layers consist of 128 neurons. The activation function is ReLU. A softmax layer is on the top of the policy network. At the first stage, the model learns $1,000$ epochs, each of which includes 100 sampling iterations and 20 training iterations. The model simultaneously learns five policies by 25 agents. The scalarization weights are $(1.0, 0.0)$, $(1.0, 0.0025)$, $(1.0, 0.0050)$, $(1.0, 0.0075)$ and $(1.0, 0.01)$. Each mini batch includes 100 samples from each agent. The learning rate is 0.0003.

Table 2. Experimental results of Adaptive Streaming. HI denotes hypervolume indicator. The number of iterations of our algorithm includes two parts since the method has two stages, thus "+" is used to separate two numbers.

	HI	Iterations	Wall-Clock Time	Parameters
HB	61.252 ± 0.134	$50,000$	$3.35 \times 10^5 s$	9.06×10^6
RA	73.140 ± 0.058	$50,000$	$1.25 \times 10^6 s$	8.97×10^6
PFA	73.115 ± 0.073	$16,000$	$4 \times 10^5 s$	8.97×10^6
Ours	$\mathbf{73.460 \pm 0.048}$	$1,000 + 2,00$	$2.66 \times 10^5 s$	2.62×10^5

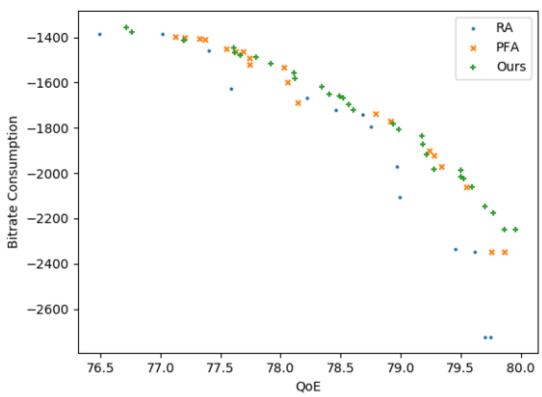

Figure 1. Approximation of the Pareto frontier in Adaptive Streaming problem.

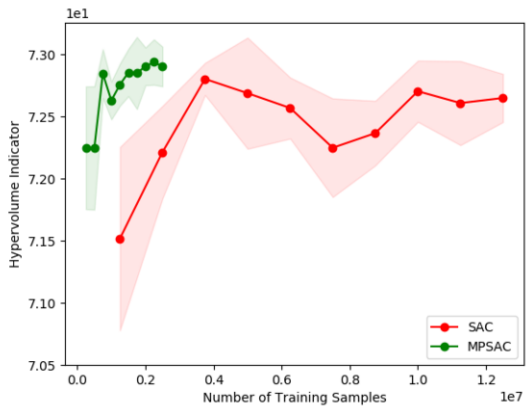

Figure 2. Training curves showing sampling efficiency of MPSAC.

At the second stage, the number of individuals is 50. Since the lowest QoE score is 0 and the highest bitrate is 16800, the reference point $(0, -16800)$ and normalizing factor $(10, 1680)$ are utilized in the objective space for calculating hypervolume indicator.

Experimental Results. Table 2 shows the experimental results of Adaptive Streaming. Similar to the results of Deep Sea Treasure, HB achieves lowest hypervolume indicator due to the poor diversity. RA outperforms PFA and our proposed algorithm achieved the highest hypervolume indicator and stability. Our algorithm has lower time and space complexity comparing with the comparative methods. Specifically, RA method needs the longest training time. PFA method is more efficient on time complexity while it is still redundant for model representation. Therefore, our algorithm is efficient in both ways.

Furthermore, the estimated Pareto frontiers of RA, PFA and our method are shown in Figure 1. Our method achieves more smooth frontier than RA and PFA. As for the quantitive evidence, the smooth measure of RA, PFA and our method is 0.042, 0.035 and 0.012 respectively. Considering the region of the highest QoE, our method uses fewer bitrates than the other methods.

4.5 Ablation Study

The ablation studies are conducted on Adaptive Streaming.

Which data sharing strategy is better? To reveal the sampling efficiency of MPSAC, we investigate three different data sharing strategies: global sharing, no sharing and neighbor sharing. These three methods are named as MPSAC.A, MPSAC.O and MPSAC.N respectively, which means that each policy attains all samples, own samples and neighbor samples. For comparison, the result of SAC is achieved from five independent repetitions. Each run is related to a specific preference. Table 3 shows the results of different data sharing strategies. All three strategies outperform the baseline method with less computational redundancy. MPSAC.O outperforms SAC, which reveals the advantage from sharing the low-level parameters. MPSAC.A and MPSAC.N exceed MPSAC.O, which shows the advantage of data sharing. MPSAC.A outperforms MPSAC.N, which reveals that all sharing is a better data sharing strategy. Furthermore, Figure 2 shows the training curves of SAC and MPSAC.A. Our method approaches higher performance and higher stability. And our method needs fewer training samples to achieve the same hypervolume indicator, which reveals the exploration efficiency introduced by collaborative learning.

Is representation learning important? To answer this question, the model learning at the first stage is modulated by investigating the number of learned policies. We fix the number of learning epoches but change the number of learned policies to 1, 5, 10, 15, 20, and 25, respectively. The comparative results in Table 4 show that the final performance improves through adding the number of learned policies at the first stage. This ablation study shows the importance of the representation learning at the first stage.

Is the second-stage learning important? Further more, as shown in Table 4, MPSAC(50) denotes the MPSAC algorithm learning 50 policies with 50 agents in the same time. By adopting the MO-CMA-ES algorithm at the second learning stage, the performance improves from 73.286 to 73.663, which shows a considerable performance gain of the second-stage learning.

5 Discussion

The proposed MPSAC algorithm is similar to the bootstrapped DQN [25], which is a deep version of the bootstrapped Thompson

Table 3. Ablation study on three exploration strategies. Single-policy SAC is executed repeatedly for five times with different preferences on objectives. MPSAC maintains five policies at the same time.

	HI	Iterations	Wall-Clock Time
SAC	72.50 ± 0.18	5,000	$1.25 \times 10^5 s$
MPSAC.A	$\mathbf{72.97 \pm 0.11}$	1,000	$2.4 \times 10^4 s$
MPSAC.O	72.93 ± 0.16	1,000	$2.35 \times 10^4 s$
MPSAC.N	72.85 ± 0.10	1,000	$2.6 \times 10^4 s$

Table 4. Ablation study on the number of learned policies at the first stage. MPSAC(m,n) denotes a first-stage model which learns m policies using n agents by MPSAC. ES(m) denotes a second-stage model learning m policies by MO-CMA-ES.

Models	HI
MPSAC(1,25)+ES(50)	73.269
MPSAC(5,25)+ES(50)	73.460
MPSAC(10,25)+ES(50)	73.453
MPSAC(15,25)+ES(50)	73.463
MPSAC(20,25)+ES(50)	73.435
MPSAC(25,25)+ES(50)	**73.489**
MPSAC(50,50)	73.286
MPSAC(50,50)+ES(50)	**73.663**

sampling [27, 32]. Inspired from [28], the bootstrapped DQN utilizes randomized value functions to improve sampling efficiency, rather than models an intractable exact posterior estimate. The value functions are represented by deep neural networks. This posterior sampling method is adopted in [25, 26, 28] as well. Both bootstrapped DQN and our method maintain more than one deep networks for Q-value estimation. These networks share the same network structure. The difference is that DQNs in [25] are optimized for the same objective, thus DQNs utilize randomized initialization and independent training samples to impel the randomness of model learning to improve exploration efficiency. In our method, the policy networks pursue differently scalarized objectives. Therefore, our method naturally sustains the model randomness to improve sampling efficiency, since different policies pursue different goals.

6 Related Work

Multi-objective reinforcement learning methods can be divided into single-policy methods and multi-policy methods based on the number of the learned policies in a single run.

Single-policy methods learn only one policy at each time. It needs to be repeated many times to approximate the Pareto frontier. Commonly, scalarization methods are adopted to transfer multiple objectives into a single one. Then classical RL algorithms can be applied without any modification. The simplest scalarization method is linear weighted scalarization [41, 42, 24]. The limitation of linear scalarization is demonstrated in [43] when the Pareto front is concave. Chebyshev scalarization method [46] and thresholded scalarization method [42, 24] are proposed to alleviate this limitation. Van *et. al.* [45] extended Q-learning to a multi-policy method using hypervolume metric as an action selection strategy. Instead of simple repetition, Parisi *et. al.* [30] proposed Pareto path following algorithm optimizing directly on the Pareto frontier.

Multi-policy methods learn more than one policy in a single run. Beyond discrete solutions, some methods approach continuous approximation. Barrett *et. al.* [2, 17, 16, 3] introduced an approach which learns the set of optimal policies for all weights through involving linear weights to represent value functions. Parisi *et. al.* [29] proposed a method, which utilizes a function to define a manifold in the policy parameters space, then approximates the Pareto frontier continuously by performing a single gradient ascent on the parameters of the function.

Furthermore, some studies considered MORL problems from other perspectives. Mannor and Shimkin [18] consider MORL problems as a constrained optimization problem. In the constrained problem, rather than optimize a single objective, the method drives the agent to approach a target set. Wiering *et. al.* [48] proposed a model-based MORL method through learning a multi-objective model of the environment.

7 Conclusion

In this paper, we propose a two-stage algorithm to address the multi-objective reinforcement learning problem. The first stage is a multi-policy soft actor-critic algorithm learning multiple policies with different preferences on objectives collaboratively. The second stage is a multi-objective evolution strategy algorithm to achieve a smooth approximation of the Pareto frontier. Our method is efficient on data exploration and model representation.

ACKNOWLEDGEMENTS

The authors would like to express our thanks for the support from the following research grants: 2018AAA0102004, NSFC-61625201, NSFC-61527804.

REFERENCES

[1] Anonymous authors, 'Adaptive streaming: From bitrate maximization to rate-distortion optimization'.
[2] Leon Barrett and Srini Narayanan, 'Learning all optimal policies with multiple criteria', in *International Conference on Machine Learning*, pp. 41–47. ACM, (2008).
[3] Andrea Castelletti, Francesca Pianosi, and Marcello Restelli, 'Tree-based fitted Q-iteration for multi-objective Markov decision problems', in *International Joint Conference on Neural Networks*, pp. 1–8. IEEE, (2012).
[4] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan, 'A fast and elitist multiobjective genetic algorithm: NSGA-II', *IEEE Transactions on Evolutionary Computation*, **6**(2), 182–197, (2002).
[5] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.
[6] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra, 'Pathnet: Evolution channels gradient descent in super neural networks', *arXiv preprint arXiv:1701.08734*, (2017).
[7] David Ha and Jürgen Schmidhuber, 'Recurrent world models facilitate policy evolution', in *Advances in Neural Information Processing Systems*, pp. 2450–2462, (2018).
[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine, 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor', *arXiv preprint arXiv:1801.01290*, (2018).

[9] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al., 'Soft actor-critic algorithms and applications', *arXiv preprint arXiv:1812.05905*, (2018).

[10] Nikolaus Hansen, 'The CMA evolution strategy: A tutorial', *arXiv preprint arXiv:1604.00772*, (2016).

[11] Christian Igel, Nikolaus Hansen, and Stefan Roth, 'Covariance matrix adaptation for multi-objective optimization', *Evolutionary Computation*, **15**(1), 1–28, (2007).

[12] Christian Igel, Verena Heidrich-Meisner, and Tobias Glasmachers, 'Shark', *Journal of Machine Learning Research*, **9**(Jun), 993–996, (2008).

[13] Christian Igel, Thorsten Suttorp, and Nikolaus Hansen, 'Steady-state selection and efficient covariance matrix update in the multi-objective CMA-ES', in *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 171–185. Springer, (2007).

[14] Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O Stanley, 'Safe mutations for deep and recurrent neural networks through output gradients', in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 117–124. ACM, (2018).

[15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, 'Continuous control with deep reinforcement learning', *arXiv preprint arXiv:1509.02971*, (2015).

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

[17] Daniel J Lizotte, Michael H Bowling, and Susan A Murphy, 'Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis', in *International Conference on Machine Learning*, pp. 695–702. Citeseer, (2010).

[18] Shie Mannor and Nahum Shimkin, 'A geometric approach to multi-criterion reinforcement learning', *Journal of Machine Learning Research*, **5**(Apr), 325–360, (2004).

[19] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh, 'Neural adaptive video streaming with pensieve', in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 197–210. ACM, (2017).

[20] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, 'Asynchronous methods for deep reinforcement learning', in *International Conference on Machine Learning*, pp. 1928–1937, (2016).

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, 'Playing atari with deep reinforcement learning', *arXiv preprint arXiv:1312.5602*, (2013).

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

[23] Sriraam Natarajan and Prasad Tadepalli, 'Dynamic preferences in multi-criteria reinforcement learning', in *International Conference on Machine Learning*, pp. 601–608. ACM, (2005).

[24] Thanh Thi Nguyen, 'A multi-objective deep reinforcement learning framework', *arXiv preprint arXiv:1803.02965*, (2018).

[25] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy, 'Deep exploration via bootstrapped DQN', in *Advances in Neural Information Processing Systems*, pp. 4026–4034, (2016).

[26] Ian Osband, Daniel Russo, and Benjamin Van Roy, '(more) efficient reinforcement learning via posterior sampling', in *Advances in Neural Information Processing Systems*, pp. 3003–3011, (2013).

[27] Ian Osband and Benjamin Van Roy, 'Bootstrapped thompson sampling and deep exploration', *arXiv preprint arXiv:1507.00300*, (2015).

[28] Ian Osband, Benjamin Van Roy, and Zheng Wen, 'Generalization and exploration via randomized value functions', *arXiv preprint arXiv:1402.0635*, (2014).

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

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

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., 'Pytorch: An imperative style, high-performance deep learning library', in *Advances in Neural Information Processing Systems*, pp. 8024–8035, (2019).

[32] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al., 'A tutorial on thompson sampling', *Foundations and Trends® in Machine Learning*, **11**(1), 1–96, (2018).

[33] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever, 'Evolution strategies as a scalable alternative to reinforcement learning', *arXiv preprint arXiv:1703.03864*, (2017).

[34] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., 'Mastering the game of Go with deep neural networks and tree search', *Nature*, **529**(7587), 484, (2016).

[35] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al., 'Mastering the game of Go without human knowledge', *Nature*, **550**(7676), 354, (2017).

[36] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman, 'BOLA: Near-optimal bitrate adaptation for online videos', in *IEEE International Conference on Computer Communications*, pp. 1–9. IEEE, (2016).

[37] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune, 'Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning', *arXiv preprint arXiv:1712.06567*, (2017).

[38] Gary J Sullivan, Thomas Wiegand, et al., 'Rate-distortion optimization for video compression', *IEEE Signal Processing Magazine*, **15**(6), 74–90, (1998).

[39] Richard S Sutton, Andrew G Barto, et al., *Introduction to reinforcement learning*, volume 135, MIT press Cambridge, 1998.

[40] Thorsten Suttorp, Nikolaus Hansen, and Christian Igel, 'Efficient covariance matrix update for variable metric evolution strategies', *Machine Learning*, **75**(2), 167–197, (2009).

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

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

[43] Peter Vamplew, John Yearwood, Richard Dazeley, and Adam Berry, 'On the limitations of scalarisation for multi-objective reinforcement learning of Pareto fronts', in *Australasian Joint Conference on Artificial Intelligence*, pp. 372–378. Springer, (2008).

[44] Hado Van Hasselt, Arthur Guez, and David Silver, 'Deep reinforcement learning with double Q-learning', in *AAAI Conference on Artificial Intelligence*, (2016).

[45] Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé, 'Hypervolume-based multi-objective reinforcement learning', in *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 352–366. Springer, (2013).

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

[47] Thomas Voß, Nikolaus Hansen, and Christian Igel, 'Improved step size adaptation for the MO-CMA-ES', in *Annual Conference on Genetic and Evolutionary Computation*, pp. 487–494. ACM, (2010).

[48] Marco A Wiering, Maikel Withagen, and Mădălina M Drugan, 'Model-based multi-objective reinforcement learning', in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 1–6. IEEE, (2014).

[49] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli, 'A control-theoretic approach for dynamic adaptive video streaming over HTTP', in *ACM SIGCOMM Computer Communication Review*, volume 45, pp. 325–338. ACM, (2015).

[50] Eckart Zitzler and Lothar Thiele, 'Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach', *IEEE Transactions on Evolutionary Computation*, **3**(4), 257–271, (1999).