Project Report: Multi-Objective Reinforcement Learning

Peiru Li
Department of Electrical and Computer Engineering
New York University
New York, USA
pl2825@nyu.edu

***Abstract*—This report is related to the course of *System Optimization Method* at Tandon School of Engineering. In this report, I will introduce the definition of multi-objective optimization problems and explain how reinforcement learning can be used to solve them. Then, I will implement an efficient multi-objective reinforcement learning algorithm, Pareto Q-learning, and compare it to outer-loop weighted Q-learning on the classic test bench for multi-objective reinforcement learning: the Deep Sea Treasure problem. Related code is available at github repository.**

I. Introduction

Multi-Objective Optimization(MOO) refers to the optimization problems that have two or more objectives [2]. This type of problems are very common in real-world scenarios such as engineering, economics and agriculture. MOO has attracted increasing attention from researchers.

Reinformence Learning(RL) is the problem faced by an agent to learn behavior through trial-and-error interactions with unknown environment [4]. In modern setting, an agent is often replaced by a policy that returns an action given the observed state. By iteratively trying different actions under different states, policy can be gradually optimized. Q-learning is a well-known model-free reinforcement learning algorithm where an agent learns how to better react to observed state using immediate reward and value of future state [9].

This project focuses on addressing multi-objective optimization problems using reinforcement learning (RL) techniques. Unlike traditional RL tasks that aim to maximize a single scalar reward, multi-objective RL deals with environments where multiple, often conflicting, objectives must be considered simultaneously. The goal is to find a set of optimal trade-off solutions, known as the Pareto front, rather than a single optimal policy.

In this work, I implement and evaluate a well-known algorithm in this field—Pareto Q-learning. This algorithm extends classical Q-learning by maintaining a set of non-dominated Q-values to capture the trade-offs among objectives. To demonstrate its effectiveness, I apply it to the Deep Sea Treasure problem, a widely used benchmark in multi-objective reinforcement learning. The results show that Pareto Q-learning is capable of learning diverse policies that approximate the Pareto front, highlighting its potential for solving real-world multi-objective decision-making tasks.

The following chapters are organized as follows: Section II introduces the background of this project, including the definition of multi-objective optimization problems, the concept of multi-objective reinforcement learning, and the main categories of multi-objective reinforcement learning algorithms. Section III focuses on the Pareto Q-learning algorithm, which is the main method adopted in this project. Section IV presents the experimental setup and results. Section V provides a summary of the project and discusses directions for future work.

II. Background

In this section, I will present the background of Multi-Objective Reinforcement Learning.

A. Multi-Objective Optimization

Single objective optimization problem is defined as follows:

$$\begin{aligned} \text{Minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \ldots, m \\ & h_j(\mathbf{x}) = 0, \quad j = 1, 2, \ldots, p \\ & \mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n \end{aligned}$$

Here:

- $\mathbf{x} \in \mathbb{R}^n$ is the decision variable vector.
- $\mathbf{f}(\mathbf{x})$ is objective function to be minimized.
- $g_i(\mathbf{x})$ are inequality constraints.
- $h_j(\mathbf{x})$ are equality constraints.
- $\mathcal{X}$ is the feasible decision space.

Multi-Objective Optimization problem is an extension of single objective optimization problem with a vector of objective functions to be optimized [6]:

$$\text{Minimize} \quad \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]^T$$

In most real-world scenarios, these objectives are conflicting with each other so the general purpose of multi-objective optimization algorithm is to find trade-off solutions among different objectives and meanwhile optimize each objective functions.

Most state-of-the-art multi-objective algorithms aim to find a Pareto Set (PS) that closely approximates the true Pareto

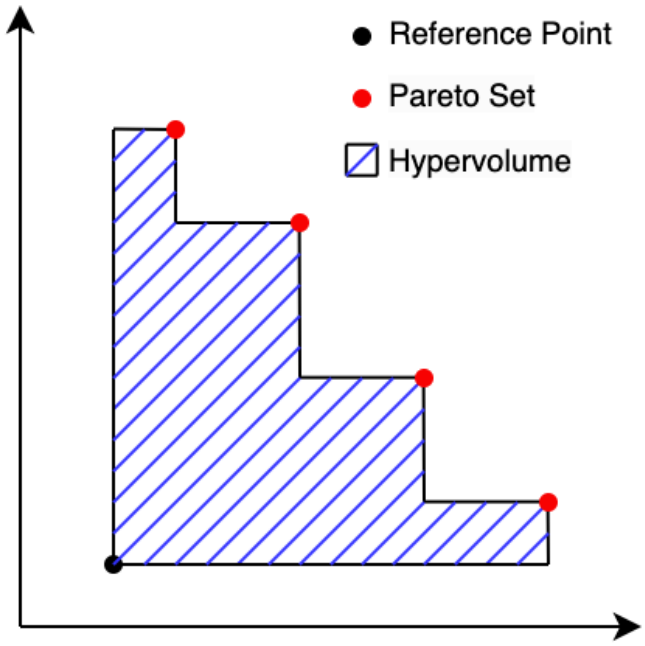

Fig. 1. Illustration of Hypervolume.

Front (PF), which represents the set of optimal trade-off solutions where no objective can be improved without degrading at least one other. Achieving a good approximation of the Pareto Front involves two main challenges: **convergence**—how close the solutions are to the true PF, and **diversity**—how well the solutions are distributed along the PF. A well-designed algorithm must balance both aspects to provide decision-makers with a wide range of high-quality trade-off solutions.

In MOO, hypvervolume(HV) is a widely used performance metric to evaluate the quality of a Pareto front approximation by measuring the volume of the objective space dominated by the solution set relative to a reference point. An illustration of hypervolume in two-objective maximization problem is presented in Fig. 1. Larger HV indicates better performance.

B. Multi-Objective Reinforcement Learning

Most Multi-Objective Reinforcement Learning(MORL) algorithms are trying to solve the multi-objective sequential decision problem which can be modeled as Multi-Objective Markov Decision Process(MOMDP). A MOMDP is represented by the tuple $\langle S, A, T, \gamma, \mu, \mathbf{R} \rangle$ [3],where:

- S is the state space
- A is the action space
- $T : S \times A \times S \to [0, 1]$ is a probabilistic transition function
- $\gamma \in [0, 1)$ is a discount factor
- $\mu : S \to [0, 1]$ is a probability distribution over initial states
- $\mathbf{R} : S \times A \times S \to \mathbb{R}^d$ is a vector-valued reward function, specifying the immediate reward for each of the considered $d \geq 2$ objectives.

The key distinction between a single-objective MDP and a MOMDP lies in the reward function $\mathbf{R}$. While traditional MDPs use a scalar reward signal, MOMDPs use a vector of rewards, providing separate feedback for each objective. This allows the agent to consider multiple, potentially conflicting, objectives simultaneously. The dimension of the reward vector corresponds directly to the number of objectives in the problem.

C. Multi-Objective Reinforcement Learning Algorithms

In the context of MOMDP, environment provides a vector of rewards rather than a scalar reward given any state-action pair.

$$\mathbf{R}(s, a) = [R_1(s, a), R_2(s, a), ..., R_k(s, a)]$$

Using MORL to solve MOMDP, the value function of given state is vectorized as:

$$\mathbf{V}^{\pi}(s) = E_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{t+k+1} \mid s_t = s \right\}.$$

where π is optimal policy. Pareto dominance relation is also adopted here to evaluate the relative quality of different policies. A policy π_1 is said to pareto dominate policy π_2 if no objective values of V_1^{π} is worse than V_2^{π} and at least one objective value of V_1^{π} is better than V_2^{π}. Two policies are non-dominated if neither of them dominates the other. A policy π is Pareto-optimal if it either dominates or non-dominated with other policies. The set of all Pareto-optimal policies is referred to as Pareto-Front(PF).

1) Single Policy Algorithm: Single-policy algorithms are one of the most straightforward and widely adopted approaches in MORL. These algorithms extend traditional single-objective, model-free, value-based methods (such as Q-learning) to handle environments with multiple objectives. The core idea is to adapt single-objective methods to multi-objective settings by making two key modifications:

(1) storing Q-values as vectors instead of scalars, and

(2) using a scalarisation function to map multi-objective Q-values to a single scalar value, which is then used to select actions.

This approach naturally leads to a single-policy solution, as the underlying single-objective methods are designed to converge to a single optimal policy.

2) Multi-Policy Approaches: Multi-policy approaches are designed to generate a set of policies that cover the PF.Unlike single-policy approaches, which converge to a single optimal policy based on a scalarisation function, multi-policy approaches aim to provide a diverse set of solutions that can accommodate varying user preferences or dynamic environments. These approaches are particularly valuable in scenarios where the user's utility function is unknown, complex, or subject to change.

One type of multi-policy approach is *Out-loop* method which operates by solving a series of single-objective problems iteratively to construct an approximate covering set(CS) of policies [5]. The other kind of multi-policy approach is *Inner-loop* method which directly explore the multi-objective space

and produce a set of policies that approximate the Pareto front without requiring iterative scalarization, such as Pareto-Q-Learning [8].

III. Adopted Methods

In this project, I implement Pareto-Q-Leaning(PQL) [8] from scratch to solve the Deep Sea Treasure (DST) problem, without referring to any existing code. The complete implementation can be found on GitHub via the following link: [link].

The key idea behind PQL is learning immediate and future reward separately which can preserve the correspondence between Q vector of current state-action pair and next state value vector.

PQL uses $\overline{\mathfrak{R}}(s,a)$ to store the average observed immediate reward vector of (s, a) and $ND_t(s,a)$ to store the set of non-dominated Q vectors in the next state s which is reached through action a at time step t. By storing $\overline{\mathfrak{R}}(s,a)$ and $ND_t(s,a)$, they can converge separately. And $\hat{Q}_{set}$ of (s,a) can be calculated at runtime by doing a vector-sum over the average immediate reward and the set of Pareto non-dominated future rewards:

$$\hat{Q}_{set}(s,a) \leftarrow \overline{\mathfrak{R}}(s,a) \oplus \gamma ND_t(s,a)$$

where $\oplus$ operator performs a vector-sum between a vector v and a set of vectors V. In this way, user can traverse the tree of $\hat{Q}_{set}(s,a)$ by applying a preference function.

Pseudo codes of PQL is presented in Alg.1:

Algorithm 1 Pareto Q-Learning Algorithm

1: Initialize empty $\hat{Q}_{set}(s,a)$
2: **for** each episode t **do**
3: Initialize state s
4: **repeat**
5: Choose action a from s using a policy derived from the $\hat{Q}_{set}$'s
6: Take action a and observe state $s' \in S$ and reward vector $r \in \mathbb{R}^m$
7: $ND_t(s,a) \leftarrow ND(\cup_{a'} \hat{Q}_{set}(s',a'))$
8: $\overline{\mathfrak{R}}(s,a) \leftarrow \overline{\mathfrak{R}}(s,a) + \frac{r - \overline{\mathfrak{R}}(s,a)}{n(s,a)}$
9: $s \leftarrow s'$
10: **until** s is terminal
11: **end for**

Firstly, Q set is initialized to be empty set for each state-action pair. Then for each episode, algorithm starts from the initial state. At each time slot, choose some action according to current policy based on $\hat{Q}_{set}$. After action selected, we take the action and observe the feedback from environment(line 6). Non-dominated vector set of current state-action pair is updated by non-dominated set of vectors at next state s' doing all the actions. In line 8, average immediate reward is iteratively updated using difference between current reward and average immediate reward divide sample time of state-action pair $n(s,a)$.

For action selection policy, I adopt HV-based ϵ-Greedy algorithm which is elaborated in pseudo Alg 2:

Algorithm 2 HV-based ϵ-Greedy Action Selection

Require: State s, Exploration rate $\epsilon \in [0,1]$
Ensure: Selected action a^*
1: Retrieve current state s
2: Generate random $p \sim \text{Uniform}(0,1)$
3: **if** $p < \epsilon$ **then**
4: $a^* \leftarrow$ Randomly select a from $\mathcal{A}$
5: **else**
6: evaluations $\leftarrow \{\}$
7: **for** each action a in action space $\mathcal{A}$ **do**
8: $hv_a \leftarrow HV(\hat{Q}_{set}(s,a))$
9: Append (a, hv_a) to evaluations
10: **end for**
11: $a^* \leftarrow \text{argmax}_a$ evaluations$[a].hv_a$
12: **end if**
13: **return** a^*

The proposed HV-based ϵ-greedy action selection scheme dynamically balances exploration and exploitation for multi-objective decision-making. With probability ϵ, a random action is chosen to explore the environment (line 3–4). Otherwise, the algorithm evaluates all actions by computing the hypervolume (HV) of their associated Pareto front approximations $\hat{Q}_{set}(s,a)$ (line 6–10), then selects the action maximizing HV to exploit the current knowledge (line 11). This approach ensures Pareto-compliant optimization while maintaining stochastic exploration.

IV. Experiments and Results

A. Test Bench: Deep Sea Treasure Environment

The Deep Sea Treasure(DST) is proposed by [7] and is a well-known MORL test instance. In this project, I use MO-Gymnasium code package [1] to do the simulation of DST environment.

In DST environment, an agent controls a submarine navigating a grid-world ocean to collect treasures scattered at varying depths. There are two conflicting objectives:

1. Time Penalty: Each step incurs a small negative reward.
2. Treasure Value: Deeper treasures offer higher rewards but require longer paths.

The agent must balance quick, low-value treasures versus costly, high-value ones to approximate the Pareto front. The DST's simple visual representation and interpretable trade-offs make it ideal for visualizing multi-objective decision-making.

In this project, we use DST with version name "deep-sea-treasure-v0". The true PF is visualized in Fig 2.

Setting the reference point to be (-25, 0), the HV of PF is 401.8.

B. Training Process of PQL

The hyper-parameters are set up as follows: A total of 2000 training episodes (EPISODES) ensure sufficient exploration and convergence. The learning rate (α=0.1) balances update

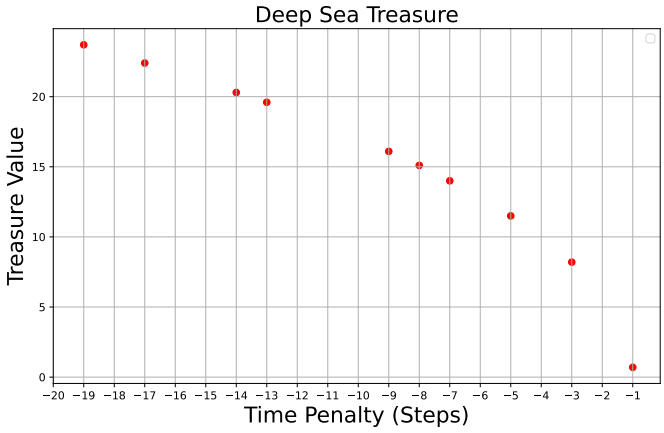

Fig. 2. Pareto Front of Deep Sea Treasure.

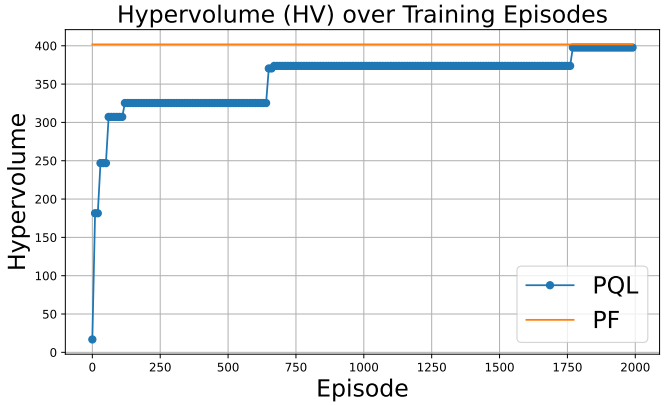

Fig. 3. Training Process of PQL

magnitudes to stabilize Q-value convergence. An elevated discount factor (γ=0.999) emphasizes long-term rewards, suitable for sparse-reward environments like Deep Sea Treasure. The epsilon-greedy strategy decays exponentially from 1.0 (ϵ_{start}) to 0.01 (ϵ_{end}) with a decay rate of 0.9998 (ϵ_{decay}), achieving a smooth transition from exploration to exploitation. These values collectively prioritize thorough early-stage exploration while progressively focusing on optimal policy refinement.

The training process is visualized in Fig. 3.

where blue line is the HV variation to episodes and orange line is the HV of true PF. The training process of the Pareto Q-Learning (PQL) algorithm demonstrates a clear and steady improvement in the quality of the learned policy over time. At the beginning of training, the HV is near zero, indicating that the learned policies are either not yet reaching the Pareto front or only covering a small, suboptimal region of the objective space. As the number of episodes increases, the HV rises steadily, showing that the algorithm is progressively identifying more diverse and higher-quality solutions that better approximate the true Pareto front.

This increasing trend in HV reflects the effectiveness of the multi-objective reinforcement learning strategy in capturing the trade-offs between the competing objectives, such as time and treasure value in the Deep Sea Treasure environment. The gradual decay of epsilon during training ensures that the agent explores sufficiently in the early stages and then exploits its learned knowledge to refine the policy, which is evident in the stabilization of HV growth towards the later episodes. Overall, the plot confirms that PQL is successfully learning to approximate the Pareto front, and the improvement in HV suggests that the algorithm is becoming increasingly capable of identifying non-dominated solutions over the course of training.

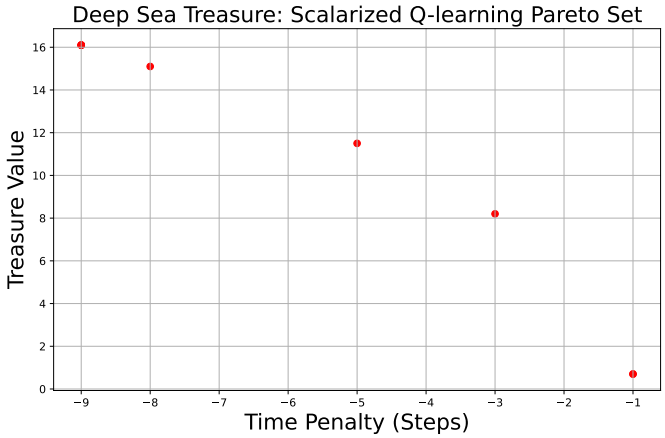

Fig. 4. Final Result of Scalarized Q-Learning.

C. Comparison with Scalarized Q-Learning

In this part, I compare PQL with scalarized Q-learning(SQL) to prove the effectiveness and efficiency of PQL. 9 sets of weights are adopted from 0.1/0.9 to 0.9/0.1 with equal gap. As a result, we can get 9 solutions on the PF. However, some of them are duplicated so the final results are only part of PF. The final results are visualized in Fig. 4.

Under identical parameter settings, Pareto Q-learning (PQL) demonstrates superior performance to scalarized Q-learning (SQL) in hypervolume (HV) metrics, as SQL can only approximate partial Pareto fronts due to its reliance on predefined weight vectors. This limitation prevents SQL from fully capturing the complete trade-off space between objectives. Additionally, SQL exhibits significantly higher computational costs, requiring 10× more iterations than PQL to evaluate multiple weight combinations. The runtime disadvantage stems from SQL's need to solve multiple scalarized subproblems independently, whereas PQL's native multi-objective optimization efficiently explores the entire PF in a single training process. These results confirm that PQL provides better solution quality and computational efficiency for true multi-objective problems.

The limitations of Scalarized Q-learning (SQL) stem from several key factors. Most notably, SQL suffers from computational inefficiency as it must exhaustively explore the entire solution space while retaining only the immediate next-state values. In contrast, Pareto Q-learning (PQL) employs a non-dominated set to store all Pareto-optimal vectors at subsequent

states for each action. This approach not only expands the agent's decision-making perspective but also enables more informed policy selections by maintaining a comprehensive view of potential trade-offs.

V. Conclusion and Future Work

In this work, I implemented Pareto Q-learning (PQL) from scratch and evaluated its performance against scalarized Q-learning (SQL) on the Deep Sea Treasure environment. Our experimental results demonstrate that PQL achieves superior performance in both solution quality and computational efficiency. By maintaining a non-dominated set of Q-vectors, PQL effectively captures the complete Pareto front, whereas SQL is limited to partial solutions due to its reliance on predefined scalarization weights. Furthermore, PQL's integrated multi-objective optimization framework eliminates the need for repetitive weight tuning, significantly reducing runtime compared to SQL's exhaustive search approach. These findings validate PQL as a more robust and scalable method for multi-objective reinforcement learning tasks with explicit trade-offs.

A promising direction for future research involves extending this work to deep reinforcement learning (DRL) settings, where high-dimensional state spaces and complex reward structures pose significant challenges. Adapting PQL to function approximation with neural networks (e.g., Pareto Deep Q-Networks) could enhance its applicability to real-world problems, such as robotics or resource management. Additionally, investigating hybrid approaches that combine the strengths of PQL and SQL—such as adaptive weight sampling or curriculum-based scalarization—may further improve convergence and scalability. Finally, benchmarking these methods on more diverse multi-objective environments would provide deeper insights into their generalizability and limitations.

References

[1] Felten, F., Alegre, L. N., Nowé, A., Bazzan, A. L. C., Talbi, E. G., Danoy, G., and Silva, B. C. da. A toolkit for reliable benchmarking and research in multi-objective reinforcement learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)* (2023).

[2] Gunantara, N. A review of multi-objective optimization: Methods and its applications. *Cogent Engineering 5*, 1 (2018), 1502242.

[3] Hayes, C. F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L. M., Dazeley, R., Heintz, F., et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems 36*, 1 (2022), 26.

[4] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research 4* (1996), 237–285.

[5] Roijers, D. M., Whiteson, S., Brachman, R., and Stone, P. *Multi-objective decision making*. Springer, 2017.

[6] Sharma, S., and Kumar, V. A comprehensive review on multi-objective optimization techniques: Past, present and future. *Archives of Computational Methods in Engineering 29*, 7 (2022), 5605–5633.

[7] Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning 84* (2011), 51–80.

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

[9] Watkins, C. J., and Dayan, P. Q-learning. *Machine learning 8* (1992), 279–292.