Pareto Q-Learning with Reward Machines

Arnaud Lequen[1]*, Clément Legrand-Lixon[2]*, Léo Saulières[3]*

[1]Linköping University, Sweden
[2]Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
[3]Univ. Toulouse, INRAE-MIAT, Toulouse, France
arnaud.lequen@liu.se, clement.legrand-lixon@univ-lille.fr, leo.saulieres@inrae.fr

Abstract

We present Pareto Q-Learning with Reward Machines (PQLRM), a multi-objective reinforcement learning algorithm for tasks whose reward structure is specified by a set of reward machines (RMs). PQLRM combines Pareto Q-Learning (PQL), which maintains sets of vector-valued Q-estimates to approximate the Pareto front, with enhancements from Q-Learning with Reward Machines (QRM), which exploits the factored automaton structure of the reward signal. This yields a multi-policy algorithm that remains sample-efficient under non-Markovian, RM-encoded rewards. Experimental trials show that PQLRM converges faster than a naive PQL baseline applied to the cross-product MDP and can synthesize Pareto-optimal policies that QRM cannot.

1 Introduction

Reinforcement Learning (RL) is a paradigm for sequential decision-making in which an agent learns to act through interaction with an environment, with the aim of maximizing a cumulative reward signal. It has driven significant breakthroughs across a broad range of domains, including video games (Badia et al. 2020), traffic control (Kusic et al. 2021), computer networks (Omoniwa, Galkin, and Dusparic 2022), and robotics (Gürtler et al. 2023). Central to RL is the design of a reward function that encodes the desired behaviour of the agent. Yet, real-world tasks rarely reduce to a single scalar objective: they typically demand balancing several, often conflicting criteria such as performance, safety, or energy consumption.

This observation has motivated the development of Multi-Objective Reinforcement Learning (MORL), which extends the classical RL framework to vector-valued rewards, each component corresponding to a distinct objective. In such cases, no single policy is universally optimal. Instead, MORL seeks to approximate the *Pareto front*, i.e., the set of policies that are non-dominated across objectives, so that a user can later select an appropriate trade-off. Efficiently learning such a set, however, remains challenging, especially when the reward structure of the task is itself complex.

Reward Machines (RMs) (Toro Icarte et al. 2018) have recently emerged as a powerful abstraction for representing non-Markovian reward functions. A reward machine is a finite-state automaton that encodes the reward structure of a task in a modular and structured way. This representation offers several advantages. First, it provides an interpretable description of the reward function, making task specifications easier to design, understand and verify. Second, the explicit state structure of reward machines can be exploited to accelerate learning, for example by decomposing value functions across the states of the automaton.

In this work, we bring together the strengths of reward machines and multi-objective reinforcement learning. We propose a novel multi-objective RL algorithm that integrates reward machines with Pareto Q-Learning (Moffaert and Nowé 2014) to learn a set of non-dominated policies. Our approach leverages the structural decomposition induced by reward machines to guide exploration and value propagation across objectives, while maintaining a Pareto-based representation of value functions. As a result, the algorithm is inherently multi-policy, as it synthesizes a set of trade-off solutions after a single run.

2 Background

2.1 Multi-Objective Reinforcement Learning

Multi-Objective Markov Decision Problem A Multi-Objective Markov Decision Problem (MOMDP) (Wiering and de Jong 2007) is a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathbf{R}, p, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote, respectively, the state space and action space, and $\gamma \in (0, 1]$ is the discount factor. $\mathbf{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^d$ is the *reward function*, which is a function that returns a vector composed of d scalars, each representing the reward associated with an objective. $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition function, where, for all $s \in \mathcal{S}, a \in \mathcal{A}$, $p(s, a, \cdot)$ is a probability distribution over $s' \in \mathcal{S}$. $p(s, a, s')$ is the probability of reaching the state s' by performing action a from state s. In this paper, however, we focus on deterministic MDPs, where $p(s, a, s') \in \{0, 1\}$. We will write $\mathsf{succ} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the function such that $\mathsf{succ}(s, a) = s'$, where s' is the only state such that $p(s, a, s') = 1$.

A solution to an MOMDP is a *policy*, which is a mapping $\pi : \mathcal{S}^+ \to \mathcal{A}$ (where $\mathcal{S}^+$ are tuples of states of size at least 1) that associates the history of visited states with the action that the agent chooses. For a given state sequence $s_0, \ldots, s_t$, $\pi(s_0, \ldots, s_t) = a_t$ is the action chosen by the agent at step t, leading to state $\mathsf{succ}(s_t, a_t) = s_{t+1}$.

*These authors contributed equally.

Pareto-Dominance Let Π be the set of all policies for an MOMDP M, and let $\pi, \pi' \in \Pi$. Policies are compared using their *value functions* $\mathbf{V}^\pi$, defined as:

$$\mathbf{V}^\pi = \sum_{t=0}^{\infty} \gamma^t \mathbf{r}_{t+1}^\pi$$

where $\mathbf{r}_t^\pi$ is the reward obtained at time step t after unfolding the policy π.

Let $\mathbf{V}_i^\pi$ be the i-th component of the value function of π. Policy π then dominates π', noted $\mathbf{V}^\pi \succ_P \mathbf{V}^{\pi'}$, if we have that, for all $i \leq d$, $\mathbf{V}_i^\pi \geq \mathbf{V}_i^{\pi'}$, and for at least one $i \leq d$, $\mathbf{V}_i^\pi > \mathbf{V}_i^{\pi'}$.

The Pareto front of M is then the set of non-dominated policies $PF(\Pi)$:

$$PF(\Pi) = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \mathbf{V}^{\pi'} \succ_P \mathbf{V}^\pi\}$$

The Pareto front contains the set of *Pareto-optimal* policies that are not Pareto dominated in the sense of $\succ_P$, i.e., policies for which there is no other policy with equal or greater value for all objectives.

Since policy quality in the multi-objective setting cannot be scalarized, the convergence of the Pareto front is often evaluated through alternative metrics. A widely used one is the hypervolume (Bader and Zitzler 2011), which measures the volume of the region dominated by the front relative to a fixed reference point.

2.2 MORL with Reward Machines

Reward Machines A Reward Machine (RM) is a finite state machine that represents a global structured reward signal, where each transition is associated with a reward function (Toro Icarte et al. 2018). Given a set of propositional symbols $\mathcal{P}$, a state space $\mathcal{S}$ and action space $\mathcal{A}$, an RM is defined as a tuple $\mathcal{R} = \langle U, u_o, \delta_{st}, \delta_{re} \rangle$. U and $u_o \in U$ denote, respectively, a finite set of states and an initial state. $\delta_{st} : U \times 2^{\mathcal{P}} \to U$ denotes the state-transition function. $\delta_{st}(u, \sigma)$ is the state reached after receiving truth assignment $\sigma \in 2^{\mathcal{P}}$ while being in state u. $\delta_{re} : U \times U \to [\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}]$ denotes the reward-transition function; $\delta_{re}(u, u')$ is the reward function to use when transitioning from state u to u' of the RM.

MOMDP with RMs An MOMDP with RMs (MOMDPRM) over the set of propositions $\mathcal{P}$ is a tuple $M = \langle \mathcal{S}, \mathcal{A}, \boldsymbol{\mathcal{R}}, p, \gamma, L \rangle$ where $\mathcal{S}$, $\mathcal{A}$, p and γ are defined as in an MOMDP. $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}^1, \ldots, \mathcal{R}^d \rangle$ is a vector of d reward machines $\mathcal{R}^i = \langle U^i, u_o^i, \delta_{st}^i, \delta_{re}^i \rangle$, for all $i \leq d$. $L : \mathcal{S} \to 2^{\mathcal{P}}$ is a *labelling function* that maps every state of the MOMDPRM with a truth assignment of the set of propositions it is built upon.

2.3 Algorithms

Multi-Objective RL algorithms can be split into two families (Hayes et al. 2022), depending on whether the user's preferences over objectives are known a priori. When such preferences are available, *single-policy* algorithms can scalarize the vector reward into a single signal and learn one policy aligned with these preferences, reducing the problem to a standard single-objective RL problem. When preferences are unknown, *multi-policy* algorithms instead synthesize a set of policies realising different trade-offs between the objectives, with the aim of finding policies on the Pareto front.

Pareto Q-Learning Pareto Q-learning (PQL) (Moffaert and Nowé 2014) extends Q-learning (Watkins and Dayan 1992) to the multi-objective setting by maintaining, for each state-action pair (s, a), a set $\hat{Q}_{set}(s, a)$ of vector-valued Q-values corresponding to Pareto-dominating policies.

One of the main mechanisms behind PQL is that the algorithm learns the immediate and future components separately: $\overline{\mathcal{R}}(s, a)$ stores the running average of the observed immediate reward vector after taking action a in state s. $ND_t(s, a)$ stores the set of non-dominated vectors reachable from s with action a at time t:

$$ND_t(s, a) = ND\left(\bigcup_{a'} \hat{Q}_{set}(s', a')\right)$$

where the ND operator only preserves the non-dominated vectors of the set it is provided. The $\hat{Q}_{set}$ is then reconstructed at run time via:

$$\hat{Q}_{set}(s, a) \leftarrow \overline{\mathcal{R}}(s, a) \oplus \gamma\, ND_t(s, a) \tag{1}$$

where $\oplus$ denotes the vector-set sum, defined for a vector $\mathbf{v}$ and a set of vectors V as:

$$\mathbf{v} \oplus V = \left\{\mathbf{v} + \mathbf{v}' \mid \mathbf{v}' \in V\right\}$$

This decoupling removes the need for explicit vector correspondence, but also lets each component converge independently.

A specific Pareto-optimal policy is reconstructed at execution time by committing to a target vector in $\bigcup_{a_o \in \mathcal{A}} \hat{Q}_{set}(s_0, a_0)$ and, at each subsequent step, selecting the action whose set contains the vector consistent with the chosen one, using the immediate reward to take into account the difference induced by the action applied.

Q-Learning with Reward Machines Q-learning with Reward Machines (QRM) (Toro Icarte et al. 2018) exploits the structure of an RM to decompose a (possibly non-Markovian) task into a set of Markovian sub-tasks, one for each RM state. For each state $u \in U$ of the RM, QRM maintains a separate Q-function $q^u(s, a)$ that predicts the expected return obtained from the environment state s assuming the RM is currently in state u.

During an episode, the current agent's RM state u_p is tracked. The decision rule selects actions using the Q-function associated with u_p, i.e. q^{u_p}.

Having multiple Q-functions may make the learning process significantly slower. The key idea is to update *all* Q-functions of an RM in parallel from a single transition (s, a, s'). For every RM state u_j, QRM uses the RM to synthesise the reward $r_j = \delta_{re}(u_j, u_k)(s, a, s')$ that would have been received from u_j (where $u_k = \delta_{st}(u_j, L(s'))$ is the corresponding next RM state) and applies the standard update:

$$q^{u_j}(s, a) \xleftarrow{\alpha} r_j + \gamma \max_{a'} q^{u_k}(s', a')$$

As the RM acts as a deterministic model of the reward structure, every transition observed in the environment yields a valid training sample for every sub-task simultaneously, which considerably improves sample efficiency. QRM is shown to converge to an optimal policy of the underlying MDPRM in the tabular setting.

3 Pareto Q-Learning with Reward Machines

Our approach to solving MOMDPRMs essentially consists in enhancing PQL with the core ideas of QRM, allowing the algorithm to leverage the factored decomposition of the state space induced by the reward machines.

Algorithm 1: PQLRM

Input : MOMDPRM M, number of episodes T
Output: A set of Q-sets $Q^{\vec{\mathsf{v}}}_{set}(s,a)$

1 Initialize $Q^{\vec{\mathsf{v}}}_{set}(s,a)$ as empty sets
2 **for** T *episodes* **do**
3 Initialize state s of the environment
4 $\vec{\mathsf{u}} := \langle \mathsf{u}_1, \dots, \mathsf{u}_d \rangle \leftarrow \langle u^1_0, \dots, u^d_0 \rangle$
5 **while** *s is not terminal* **do**
6 Choose action a using s and $\vec{\mathsf{u}}$
7 $s' \leftarrow \mathsf{succ}(s,a)$
8 **for** *each* $\vec{\mathsf{v}} := \langle \mathsf{v}_1, \dots, \mathsf{v}_d \rangle$ *of RM states* **do**
9 $\sigma \leftarrow L(s')$
10 $\vec{\mathsf{v}}' \leftarrow \langle \delta^1_{st}(\mathsf{v}_1, \sigma), \dots, \delta^d_{st}(\mathsf{v}_d, \sigma) \rangle$
11 **for** $i \le d$ **do**
12 $r_i \leftarrow \delta^i_{re}(\mathsf{v}_i, \mathsf{v}'_i)$
13 $\mathbf{r} \leftarrow (r_1(s,a), \dots, r_d(s,a))$
14 $ND^{\vec{\mathsf{v}}}(s,a) \leftarrow ND(\bigcup_{a'} Q^{\vec{\mathsf{v}}'}_{set}(s',a'))$
15 $\overline{\mathscr{R}}^{\vec{\mathsf{v}}}(s,a) \leftarrow \overline{\mathscr{R}}^{\vec{\mathsf{v}}}(s,a) + \frac{\mathbf{r} - \overline{\mathscr{R}}^{\vec{\mathsf{v}}}(s,a)}{n(s,a)}$
16 $s \leftarrow s'; \vec{\mathsf{u}} \leftarrow \langle \delta^1_{st}(\mathsf{u}_1, \sigma), \dots, \delta^d_{st}(\mathsf{u}_d, \sigma) \rangle$

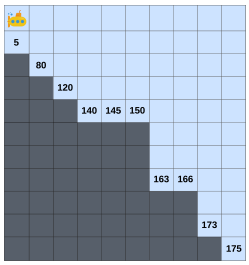
The algorithm we propose is described in Algorithm 1. The procedure starts by initializing one set of Q-vectors $Q^{\vec{\mathsf{v}}}_{set}(s,a)$ per pair (s,a) and per joint RM state $\vec{\mathsf{v}} = \langle \mathsf{v}_1, \dots, \mathsf{v}_d \rangle$ (line 1). This is the multi-objective analogue of the family of Q-functions $\{q^u\}_{u \in U}$ maintained by QRM: where QRM stores one scalar Q-value per RM state, PQLRM stores, for each configuration of RM states, a whole set of Pareto-dominating value vectors as in PQL.

At the start of every episode (line 2), the states of both the environment and of the RMs are reset (lines 3 and 4). The agent then interacts with the environment until a terminal state is reached (line 5), or until a fixed number of steps have been executed. At each step, an action a is selected (line 6) following PQL's set-evaluation rule applied to the Q-sets $Q^{\vec{\mathsf{u}}}_{set}(s,\cdot)$ (e.g. a hypervolume-based or Pareto-cardinality-based ϵ-greedy strategy).

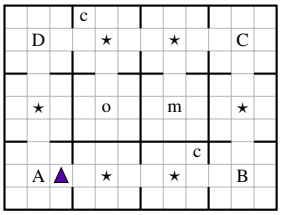
The key ingredient borrowed from QRM is the update of all RM states, done at lines 8–15: instead of updating only the entry corresponding to the currently visited RM tuple $\vec{\mathsf{u}}$, the algorithm iterates over *every* joint RM state $\vec{\mathsf{v}}$ and updates $Q^{\vec{\mathsf{v}}}_{set}(s,a)$ as if the agent had been in $\vec{\mathsf{v}}$ when taking (s,a). For each such $\vec{\mathsf{v}}$, the corresponding next RM tuple $\vec{\mathsf{v}}'$ is computed from the label $\sigma = L(s')$ of the observed successor state (lines 9–10), and the vector reward $\mathbf{r}$ that *would* have been emitted from $\vec{\mathsf{v}}$ is synthesized by querying each RM's reward function $\delta^i_{re}(\mathsf{v}_i, \mathsf{v}'_i)$ (lines 11–13). Because the dynamics of the environment are shared across all RM tuples, a single transition (s,a,s') thus produces $|U^1| \cdots |U^d|$ simultaneous updates, drastically improving sample efficiency compared to a naive PQL applied to the cross-product MDP.

The two updates that follow implement the idea of PQL of decoupling the learning of the immediate reward and the future reward. Line 14 updates the set $ND^{\vec{\mathsf{v}}}(s,a)$ of non-dominated future return vectors reachable from (s,a) under joint RM state $\vec{\mathsf{v}}$, by collecting the $Q^{\vec{\mathsf{v}}'}_{set}(s',a')$ over all potential future actions a' and preserving only the non-dominated vectors. Line 15 updates the average immediate reward vectors with the synthesized $\mathbf{r}$ and $n(s,a)$, tracking the number of times action a was taken from state s during training. As in PQL, $Q^{\vec{\mathsf{v}}}_{set}(s,a)$ is then reconstructed on the fly via Equation 1.

Finally, line 16 advances both the environment state and the *actually visited* joint RM state $\vec{\mathsf{u}}$ according to the labelling of s', which is required for action selection.



(a) PBST

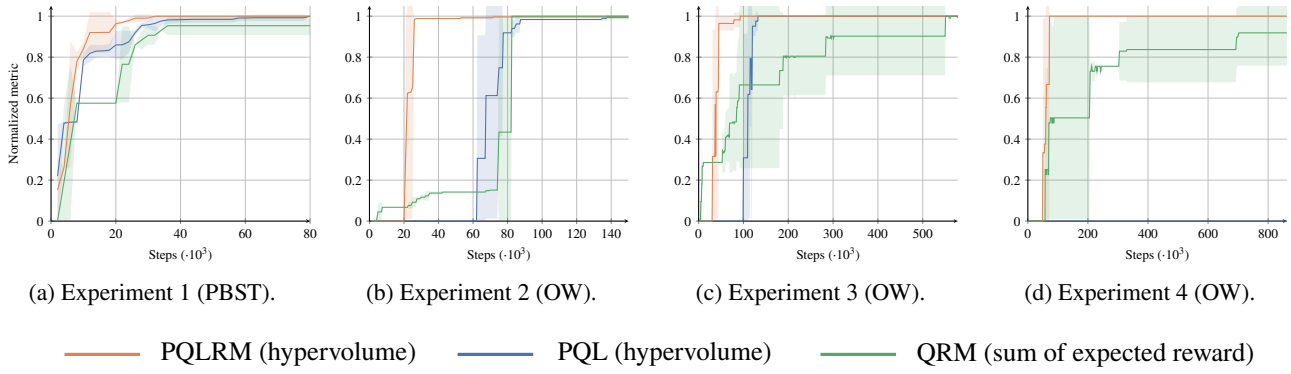


(b) Office World

Figure 1: The two environments used in our experiments. (a) In PBST, the submarine starts at the top-left corner and must choose among treasures of increasing value located deeper in the grid; dark cells are inaccessible. (b) In Office World (Toro Icarte et al. 2018), the agent (triangle) navigates an office-like grid containing several points of interest.

Policy reconstruction A concrete policy is extracted by first committing to a target Q-vector $\mathbf{v}^\star \in Q^{\vec{\mathsf{u}}_0}_{set}(s_0, a_0)$.

Then, at each step, the joint RM state $\vec{\mathsf{u}}$ is advanced alongside s. The residual target is updated via $\mathbf{v}^\star \leftarrow (\mathbf{v}^\star - \mathbf{r})/\gamma$, and we then select the next action whose Q-set contains a vector closest to it. Indexing the Q-set by $\vec{\mathsf{u}}$ (rather than s alone) is essential here, as each $Q^{\vec{\mathsf{v}}}_{set}(s,a)$ is only meaningful under the assumption that the RMs are in state $\vec{\mathsf{v}}$ in the current environment in which the agent is evolving.



(a) Experiment 1 (PBST). (b) Experiment 2 (OW). (c) Experiment 3 (OW). (d) Experiment 4 (OW).

Figure 2: Progress of training over the number of training steps. For multi-policy algorithms (i.e. PQLRM and PQL), we report the normalized hypervolume. For QRM, we report the normalized sum of the expected reward of each policy from the initial state. Policies are evaluated every 2,000 steps. The first experiment reports results on PBST and the others on Office World.

4 Experimental Evaluation

4.1 Experimental Setup

Setup All experiments were conducted on an AMD Ryzen 7 PRO 5850U processor, with memory usage capped at 8 GB and a wall-clock limit of 10 minutes per run. Our implementation is in Python 3.10 and builds on the Gymnasium environment interface (Towers et al. 2024). Our code is available at: https://github.com/arnaudlequen/PQLRM.

Baselines We compare against PQL, naively adapted to support RMs, by running it on the cross-product state space $S \times U^1 \times \cdots \times U^d$, thus folding the joint RM state into the environment. We also compare against QRM. The algorithm, however, only learns policies on the extremities of the Pareto-front, since it only searches simultaneously policies that are single-objective.

4.2 Environments

Pressurized Bountiful Sea Treasure In the Pressurized Bountiful Sea Treasure (PBST) environment (Moffaert and Nowé 2014), illustrated in Figure 1a, is a grid-world multi-objective reinforcement learning problem in which an agent controls a submarine exploring the seabed to collect treasures. The agent must simultaneously optimize three conflicting objectives: minimizing the time to reach a treasure, maximizing the value of the collected treasure, and minimizing the pressure incurred by diving deep underwater. Higher-value treasures are located deeper and farther away, forcing the agent to trade off between speed, safety (low pressure), and reward.

The PBST environment is modeled as a MOMDP: $\mathcal{M} = \langle S, \mathcal{A}, \mathbf{R}, p \rangle$. Episodes terminate when the agent reaches a treasure state. A state corresponds to the position of the submarine on a two-dimensional grid: $s = (x, y) \in S$ where x denotes the depth (row index) and y the horizontal position (column index). The grid is of finite size 10×11. The initial state is located at the surface, at the top-left corner. In this environment, the black positions are not part of the state space; the positions marked with a number are terminal states containing a treasure whose value corresponds to the number; and the blue positions are part of the state space (see Figure 1a). The agent can take four deterministic actions: $\mathcal{A} = \{up, down, left, right\}$. Each action moves the agent to an adjacent cell in the corresponding direction. Actions that would move the agent outside the grid leave the state unchanged. The transition function is deterministic and defined as:

$$P(s' \mid s, a) = \begin{cases} 1 & \text{if } s' = s + \delta(a) \\ 0 & \text{otherwise} \end{cases}$$

where $\delta(a)$ is the move associated with action a. The resulting state is clipped to remain within grid boundaries and reachable positions. The reward function is vector-valued with three components: $\mathbf{R}(s, a, s') = \left(r_{\text{time}}, r_{\text{treasure}}, r_{\text{pressure}}\right)$. At each time step, the agent incurs a constant penalty: $r_{\text{time}} = -1$. A positive reward is obtained only when reaching a treasure state:

$$r_{\text{treasure}} = \begin{cases} tr_i & \text{if } s' \text{ is a treasure location } i \\ 0 & \text{otherwise} \end{cases}$$

where tr_i is the value of treasure located at position i; it increases with depth and distance. The pressure penalty depends on the number of consecutive down actions performed by the agent. Let $n_{\downarrow}$ denote the number of consecutive *down* actions executed up to and including the current action since the last non-*down* action. Then

$$r_{\text{pressure}} = \begin{cases} -1 & \text{if } n_{\downarrow} = 1, \\ -3 & \text{if } n_{\downarrow} = 2, \\ -5 & \text{if } n_{\downarrow} \geq 3. \end{cases}$$

This reward component is non-Markovian because its value depends on the history of previously executed actions. In particular, the tuple (s, a, s') is not sufficient to determine r_{pressure}, since the same transition may receive different pressure penalties depending on how many consecutive *down* actions preceded it. Consequently, this objective cannot be represented by a standard state-transition reward function and

instead requires an RM. The RM tracks the number of consecutive downward movements through its internal state and relies on a propositional symbol emitted whenever a *down* action is performed. Note that in our experiments, r_{time} and r_{treasure} are also encoded by RMs, although this is not a requirement, unlike r_{pressure}.

Office World In the Office World environment (Toro Icarte et al. 2018), an agent navigates an office-like map to complete temporally extended tasks such as delivering coffee or mail, patrolling rooms, or avoiding decorations (Figure 1b).

The Office World environment is modeled as an MOMDP: $\mathcal{M} = \langle S, \mathcal{A}, \mathbf{R}, P \rangle$. Episodes terminate when the task specification is completed or when the agent violates a forbidden condition (e.g., stepping onto a decoration tile). A state corresponds to the position of the agent on a two-dimensional grid: $s = (x, y) \in S$, where x and y denote the row and column indices. The grid contains several special locations associated with propositional events: coffee locations c, mail locations m, the office o, decorations $\star$, and marked locations A, B, C, D used for patrol tasks. The agent can execute four deterministic actions: $\mathcal{A} = \{up, down, left, right\}$. Each action moves the agent to the adjacent grid cell in the corresponding direction. Actions that would move the agent into a wall or outside the map leave the state unchanged. The transition dynamics are deterministic and defined by:

$$P(s' \mid s, a) = \begin{cases} 1 & \text{if } s' = s + \delta(a), \\ 0 & \text{otherwise,} \end{cases}$$

where $\delta(a)$ denotes the move induced by action a. The resulting position is constrained to valid cells of the office map. Tasks are encoded by RMs. For the *get mail* and *get coffee* tasks, the agent receives a reward of 1 when it reaches the object's location and 1 when it reaches the office. For the *patrol* task involving going to positions A, B, C, and D (in this order), and the *no hit decoration* task involving not reaching the positions of the decorations, the agent receives a reward only when it has completed its task by reaching the office.

4.3 Results

We report the convergence speed with regard to the number of iterations of each algorithm in Figure 2. For multi-policy algorithms (i.e. PQLRM and PQL), we report the normalized hypervolume, while we report the normalized sum of the values of each policy for QRM. Each value is reported every 2,000 steps. Experiment 1 (Fig. 2a) is performed on the PBST environment, with the three objectives described. Both PQL and PQLRM reach all 15 Pareto optimal policies. Note that for some treasure there exist two or three policies leading due to the conflicting objectives. QRM is able to solve the three tasks, but it struggles to reach the farthest treasures. QRM converges a bit slower than the multi-policy algorithms, PQLRM converges slightly faster than PQL.

Experiments 2-4 are performed on the office world environment. In experiment 2 (Fig. 2b) the agent learns three tasks to reach the office, to get a coffee and to get the mail in a minimum number of steps. The results show that PQLRM converges faster towards the six Pareto optimal vectors than PQL. QRM is able to solve each task independently. Experiment 3 (Fig. 2c) is performed with the following three objectives: reaching the office without hitting a decoration, reaching the office with a coffee cup and reaching the office with the mail. The results highlight that PQLRM is able to reach all four Pareto optimal policies, while PQL is not able to solve the tasks without hitting decorations and only returns three non-dominated policies that are not Pareto optimal. Moreover, the results show that QRM is not robust on solving all tasks independently, and sometimes it fails to solve the mail task in the number of steps allowed. Finally, experiment 4 (Fig. 2d) is performed with two tasks: reaching the office after patrolling all checkpoints (in the right order A, B, C, D) and reaching the office without hitting any decoration. In this case, PQLRM is able to reach the two Pareto optimal policies quite quickly, although the policies are harder to find than for the other experiments. PQL is not able to solve any of the tasks given (i.e. the hypervolume is 0 for it). QRM is able to solve both tasks independently if enough steps are provided.

5 Conclusion

We introduced PQLRM, a multi-objective reinforcement learning algorithm that lifts Pareto Q-Learning to tasks whose objectives are specified by reward machines, by reusing QRM's update method across the joint RM state space. Experimental results show that PQLRM is more sample-efficient than PQL, while being more flexible than QRM as it synthesizes more Pareto-optimal policies. Natural next steps include scaling the approach to stochastic environments.

References

Bader, J.; and Zitzler, E. 2011. HypE: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary computation*, 19(1): 45–76.

Badia, A. P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, Z. D.; and Blundell, C. 2020. Agent57: Outperforming the Atari Human Benchmark. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Gürtler, N.; Widmaier, F.; Sancaktar, C.; Blaes, S.; Kolev, P.; Bauer, S.; Wüthrich, M.; Wulfmeier, M.; Riedmiller, M.; Allshire, A.; et al. 2023. Real Robot Challenge 2022: Learning Dexterous Manipulation from Offline Data in the Real World. In *NeurIPS 2022 Competition Track*, 133–150. PMLR.

Hayes, C. F.; Rădulescu, R.; Bargiacchi, E.; Källström, J.; Macfarlane, M.; Reymond, M.; Verstraeten, T.; Zintgraf, L. M.; Dazeley, R.; Heintz, F.; et al. 2022. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1): 1–59.

Kusic, K.; Ivanjko, E.; Vrbanic, F.; Greguric, M.; and Dusparic, I. 2021. Spatial-Temporal Traffic Flow Control on Motorways Using Distributed Multi-Agent Reinforcement Learning. *Mathematics*, 9(23).

Moffaert, K. V.; and Nowé, A. 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. *J. Mach. Learn. Res.*, 15(1): 3483–3512.

Omoniwa, B.; Galkin, B.; and Dusparic, I. 2022. Energy-Aware Optimization of UAV Base Stations Placements via Decentralized Multi-Agent Q-Learning. In *IEEE Consumer Communications and Networking Conference (CCNC)*.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2112–2121.

Towers, M.; Kwiatkowski, A.; Terry, J. K.; Balis, J. U.; Cola, G. D.; Deleu, T.; Goulão, M.; Kallinteris, A.; Krimmel, M.; KG, A.; Perez-Vicente, R.; Pierré, A.; Schulhoff, S.; Tai, J. J.; Tan, H.; and Younis, O. G. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *CoRR*, abs/2407.17032.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8(3): 279–292.

Wiering, M. A.; and de Jong, E. D. 2007. Computing Optimal Stationary Policies for Multi-Objective Markov Decision Processes. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 158–165.