Pareto Q-Learning with Reward Machines

Arnaud Lequen Clément Legrand-Lixon Léo Saulières

Linköping University Univ. Lille (CRIStAL) Univ. Toulouse (INRAE-MIAT)

Outline

One agent, several conflicting objectives

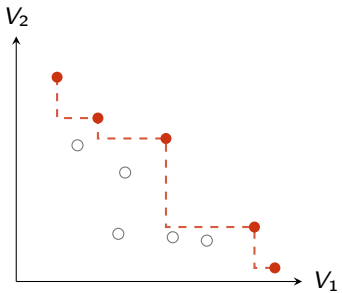
Reinforcement Learning optimises a *single* scalar reward.

But real tasks balance **several conflicting criteria**:

- performance vs. safety vs. energy...

Multi-Objective RL (MORL)

Rewards become **vectors** $\mathrm{R} \in \mathbb{R}^d$.
No single best policy $\Rightarrow$ seek the **Pareto front** of non-dominated trade-offs.

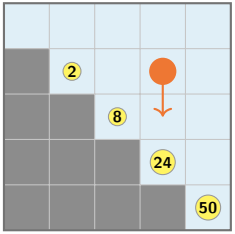


. . . and a reward that is far from Markovian

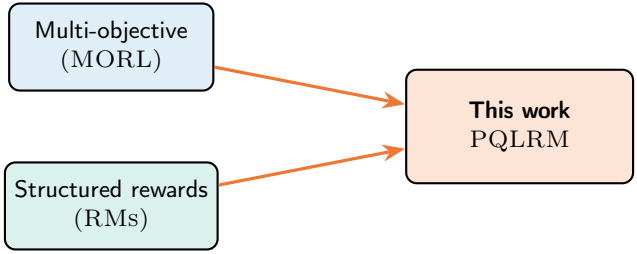
Reward can depend on the **history**, not just the current state.

Example (submarine): a deep-diving *pressure* penalty

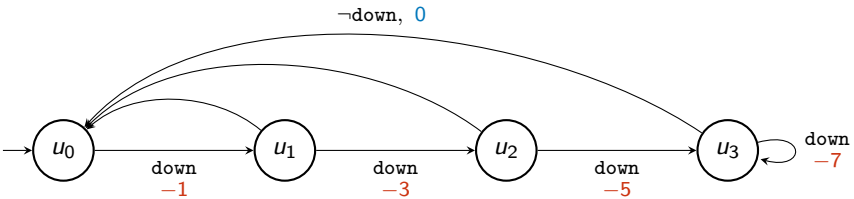
The penalty grows with the number of *consecutive* `down` moves $\Rightarrow$ the same transition $(s, \texttt{down}, s')$ can yield different rewards.



Reward Machines (RMs) encode non-Markovian rewards as finite automata



A **Reward Machine** $\mathcal{R} = \langle U, u_0, \delta_{st}, \delta_{re} \rangle$ is a finite automaton: transitions read truth assignments $\sigma \in 2^{\mathcal{P}}$ and emit a reward.



- States u_i track the **history** (here: # of consecutive `down` moves).
- A vector of RMs $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}^1, \ldots, \mathcal{R}^d \rangle$ specifies a task with d objectives.

Two specialised Q-learning-based methods

PQL — Pareto Q-Learning

Multi-objective, **multi-policy**.

[Van Moffaert & Nowé, 2014]

- Keeps a *set* of vector Q-values per (s, a).
- Decouples reward & future:

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

$\Rightarrow$ approximates the **whole Pareto front**.

$\times$ blind to the RM structure.

QRM — Q-Learning with RMs

Single objective, exploits the RM.

[Toro Icarte et al., 2018]

- One Q-function q^u *per RM state* u.
- **One transition updates *all* RM states** at once:

$$q^u(s,a) \overset{\alpha}{\leftarrow} r_j + \gamma \max_{a'} q^{u'}(s',a')$$

$\Rightarrow$ very **sample-efficient**.

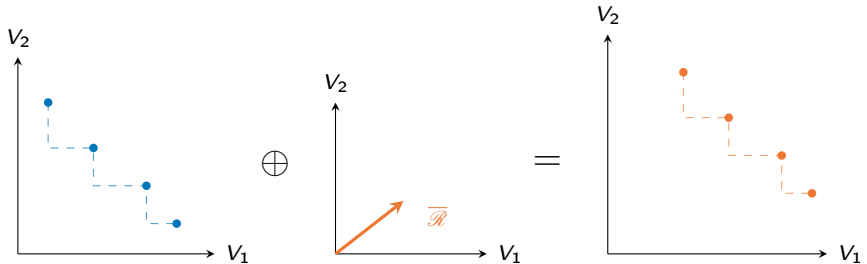
$\times$ a single scalar policy only.

Idea: lift PQL onto the factored RM state space

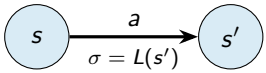
- Maintain one Q-*set* $Q^{\vec{v}}_{set}(s, a)$ per (s, a) **and per joint RM state** $\vec{v} = \langle v_1, \ldots, v_d \rangle$.
- Each environment step (s, a, s') triggers a **counterfactual update** of *every* joint RM state $\vec{v}$

Rebuilding each Pareto set (PQL decoupling)

$$\gamma \quad \underbrace{ND^{\vec{\mathbf{v}}}(s,a)}_{\text{non-dominated futures}} \quad \oplus \quad \underbrace{\overline{\mathscr{R}}^{\vec{\mathbf{v}}}(s,a)}_{\text{avg. immediate reward}} \quad = \quad Q^{\vec{\mathbf{v}}}_{set}(s,a)$$



One real step ⇒ *every* RM configuration learns

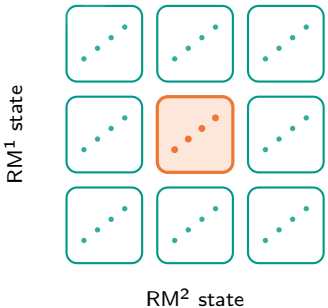


One transition is observed in the environment.

⇓

all configurations updated at once.

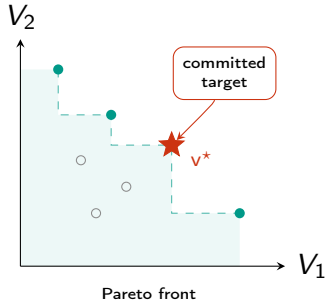
For a fixed (s, a):



- For a given (s, a), each cell $\vec{v}$ stores a set of **non-dominated vectors** $ND^{\vec{v}}(s, a)$
- = configuration actually visited
  = updated counterfactually

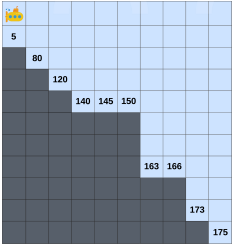
Reading out one policy: commit to a target on the front

Pareto-optimal vectors are concrete policies:



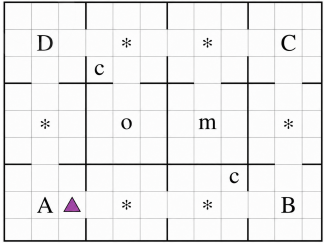
1. **Commit** to a target $\mathbf{v}^\star \in Q_{set}^{\vec{u}_0}(s_0, a_0)$.
2. **Advance** the joint RM state $\vec{u}$ alongside s.
3. **Discount** the residual: $\mathbf{v}^\star \leftarrow (\mathbf{v}^\star - \mathbf{r})/\gamma$.
4. **Act** towards the closest vector in the current Q-set.

Two multi-objective, RM-encoded environments



Pressurized Bountiful Sea Treasure

3 objectives: time / treasure value / pressure



Office World

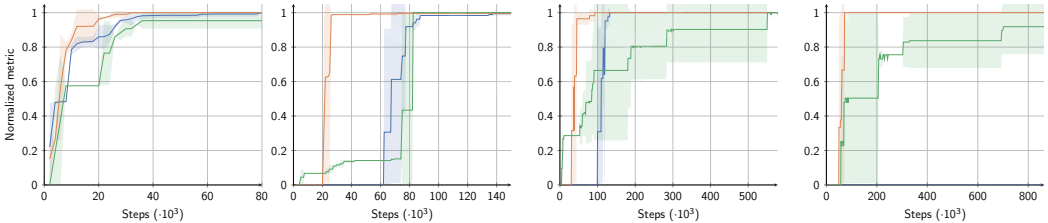
tasks: get `coffee`, get `mail`, `patrol` A B C D, avoid ⋆`decorations`

Baselines:

PQL on the cross-product MDP

QRM (single-objective)

Results: faster than PQL, richer than QRM



Exp. 1 — PBST | Exp. 2 — OW | Exp. 3 — OW | Exp. 4 — OW

PQLRM | PQL | QRM

- **PQLRM converges faster than PQL**
- It recovers **Pareto-optimal policies QRM misses** (Exp. 3–4: PQL even fails to solve some tasks).

Take-aways

PQLRM: Pareto Q-Learning with Reward Machines

- Lifts **Pareto Q-Learning** to tasks whose objectives are specified by **reward machines**.
- Reuses QRM's counterfactual updates across the **joint RM state space** $\Rightarrow$ one transition updates many Pareto sets.
- **More sample-efficient** than naive PQL, **more flexible** than QRM (a whole set of policies in a single run).

Take-aways

PQLRM: Pareto Q-Learning with Reward Machines

- Lifts **Pareto Q-Learning** to tasks whose objectives are specified by **reward machines**.
- Reuses QRM's counterfactual updates across the **joint RM state space** $\Rightarrow$ one transition updates many Pareto sets.
- **More sample-efficient** than naive PQL, **more flexible** than QRM (a whole set of policies in a single run).

Thank you! Questions?