Leveraging Reward Machines for Efficient Multi-Objective Reinforcement Learning

Panos Aronis
Utrecht University
Utrecht, the Netherlands
p.aronis@uu.nl

Mehdi Dastani
Utrecht University
Utrecht, the Netherlands
m.m.dastani@uu.nl

Roxana Rădulescu
Utrecht University
Utrecht, the Netherlands
r.t.radulescu@uu.nl

Giovanni Varricchione
Utrecht University
Utrecht, the Netherlands
g.varricchione@uu.nl

ABSTRACT

Reinforcement Learning (RL) provides a powerful framework for sequential decision-making, typically assuming a single scalar reward function and a Markovian reward structure. However, many real-world problems involve multiple, potentially conflicting objectives and require temporally extended behaviours that fundamentally violate the Markov property. These complexities have separately motivated the study of Multi-Objective Reinforcement Learning (MORL) and Reward Machines (RMs), both of which demand new representations and learning strategies to ensure effective, sample-efficient, and interpretable solutions. To address these challenges, we propose a novel framework, Multi-Objective Reward Machines (MORMs), to handle non-Markovian reward structure and multiple objectives in a unified setting, enabling more principled and sample-efficient learning in complex sequential decision-making tasks.

KEYWORDS

Multi-Objective Reinforcement Learning, Reward Machines, Reinforcement Learning

1 INTRODUCTION

Classical reinforcement learning (RL) [12] provides a powerful framework for sequential decision-making, enabling an agent to learn how to act in an environment so as to maximise long-term reward. However, it assumes that the agent interacts with an environment defined by a single scalar reward function and that this reward is Markovian, depending only on the current state and action. Moreover, the reward function is typically treated as a black box, that is, the agent observes numerical rewards but has no access to the structure, logic, or intent behind them. While these assumptions simplify the mathematical framework, they also limit the kinds of problems that can be expressed and the efficiency with which they can be solved, especially when reward signals are sparse, temporally dependent, or procedurally structured [11].

Many real-world decision-making problems require balancing multiple, often conflicting objectives rather than optimizing a single scalar quantity [4]. Examples include robotic systems trading off speed, safety, and energy consumption, traffic controllers balancing throughput and congestion, or medical treatments weighing effectiveness against side effects. Multi-objective reinforcement learning (MORL) [3] models such problems via vector-valued rewards, with solutions consisting of sets of policies that represent different trade-offs. The motivation for MORL extends beyond the observation that "real problems have many metrics". It challenges the reward hypothesis [10], which claims that all goals can be reduced to a scalar signal. In practice, scalarisation may be impossible, infeasible, or undesirable (e.g., when preferences are a-priori unknown, when they cannot be meaningfully quantified or agreed upon) [8]. By preserving the natural multi-dimensional structure of preferences, MORL provides a principled alternative. However, like single-objective RL, MORL still treats reward signals as black boxes and assumes Markovian dependence.

A central limitation of both RL and MORL is thus the assumption that reward functions are Markovian and opaque. Yet many tasks inherently involve non-Markovian reward structure, where rewards depend on temporally extended behaviour rather than the current state alone. Reward Machines (RMs) [6] address this issue by providing a finite-state representation capable of encoding non-Markovian reward functions compactly. An RM tracks relevant historical information through its automaton state, allowing sequences of subtasks, persistent conditions, loops, or conditional dependencies to be represented in a principled way. This yields an equivalent Markovian formulation over an augmented state space, reconciling temporally extended logic with standard RL algorithms. Beyond enabling expressive reward specification, RMs also justify a shift in how reward functions are provided to an agent. While the environment's transition dynamics are rightly assumed unknown, reward functions need not be hidden. Since the agent should receive external reward information for every transition, exposing the underlying reward structure of a task allows it to exploit prior knowledge for more efficient learning.

While RMs have proven effective in single-objective settings, they have not yet been explored in multi-objective domains. This creates a fundamental gap: complex real-world tasks often involve multiple reward components, each with its own temporal structure, overlapping sub-goals, and structured dependencies that cannot be expressed in existing MORL or RM formalisms. For example, consider an assembly line agent that needs to efficiently move between different work stations, in the correct sequence, and perform

Proc. of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/. 2026.

the necessary actions, in a possibly repeated fashion. This work[1] addresses how to represent such vector-valued, non-Markovian reward functions in a structured, compact, and learnable form, and how to leverage this structure to achieve sample-efficient multi-objective learning. We summarise our contributions as follows: (i) we introduce *Multi-Objective Reward Machines* (MORMs), a framework that composes single-objective RMs into a product automaton encoding the full temporal structure of all objectives; (ii) we examine how this product automaton can be leveraged by existing MORL algorithms through an augmented MOMDP, and how RM-based learning techniques must be adapted to multi-objective rewards to maintain efficiency; (iii) we empirically validate our proposed framework on a collection of simple yet novel non-Markovian multi-objective tasks.

2 PRELIMINARIES

In this section, we introduce the necessary background on multi-objective reinforcement learning, and reward machines.

2.1 Multi-Objective Reinforcement Learning

In multi-objective reinforcement learning (MORL) [4], the environment is modelled via a *multi-objective Markov decision process* (MOMDP).

Definition 1 (Multi-Objective Markov Decision Process)**.** A MOMDP is a tuple $\mathcal{M} = \langle S, A, T, \boldsymbol{R}, \mu, \gamma \rangle$, where S is a non-empty set of states, A is a non-empty set of actions, $T : S \times A \times S \to [0, 1]$ is the transition function mapping each state-action pair to a probability over the set of MOMDP states, $\boldsymbol{R} : S \times A \times S \to \mathbb{R}^d$ is the vector reward function, $\mu : S \to [0, 1]$ is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor.

At each timestep t, the MOMDP is in some state $s_t \in S$: as the agent takes an action $a_t \in A$, the MOMDP transitions to the new state s_{t+1} sampled from $T(s_t, a_t)$, i.e., $s_{t+1} \sim T(s_t, a_t)$, and returns the vector-valued reward $\boldsymbol{r}_{t+1} = \boldsymbol{R}(s_t, a_t, s_{t+1})$.

The agent's behaviour is represented via a *policy*, i.e., a function $\pi : S \times A \to [0, 1]$ mapping each state of the MOMDP to a probability distribution over the set of actions. The goal of the agent is to maximise the discounted sum of vectorial rewards $\sum_{k=0}^{\infty} \gamma^k \boldsymbol{r}_{k+1}$, from any MOMDP state.

The value function of a policy π in a MOMDP is defined as:

$$\boldsymbol{V}^{\pi} = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \boldsymbol{r}_{k+1} \,\middle|\, \pi, \mu \right], \tag{1}$$

where $\boldsymbol{r}_{k+1} = \boldsymbol{R}(s_k, a_k, s_{k+1})$ is the reward received at timestep $k + 1$ and μ is the initial state distribution. Vectorizing the value function, $\boldsymbol{V}^{\pi} \in \mathbb{R}^d$, only provides a partial ordering of the policy space. To identify the optimal policy, more information is required about how to prioritise the different objectives. This trade-off can be represented by a *utility function*, $u : \mathbb{R}^d \to \mathbb{R}$, which maps the vector to a single scalar value. When the utility function is unknown, approaches can adopt the multi-policy paradigm and return a complete set of possible solutions. In these cases, *non-stationary* policies, which condition on the full history of interactions, should also be considered for optimality [16].

Solution sets. In the most general case, the *Pareto set* is defined as the optimal solution set, under the minimal assumption that u is monotonically increasing and it is based on the Pareto dominance relationship. Informally, Pareto dominance ($\succ_P$) introduces a partial ordering over vectors, where one vector is preferred over another when it is at least equal on all objectives and strictly better on at least one. Let Π be a set of policies. The Pareto set $\mathcal{P}(\Pi)$ then contains all pairwise undominated policies, i.e.,

$$\mathcal{P}(\Pi) = \{\pi \in \Pi \mid \nexists\, \pi' \in \Pi : \boldsymbol{V}^{\pi'} \succ_P \boldsymbol{V}^{\pi}\} \tag{2}$$

The *Pareto front* (PF), denoted as $\mathcal{F}(\mathcal{P})$, contains the value vectors corresponding to all Pareto optimal policies $\pi \in \mathcal{P}(\Pi)$. If u is a positively-weighted linear sum, then the solution set will be the *convex hull* (CH) of value functions $\boldsymbol{V}^{\pi}$.

2.2 Solving MOMDPs without scalarising

When the MOMDP model is given, *Pareto Value Iteration* (PVI) [15] extends dynamic programming to the multi-objective setting, where the Bellman equation takes the form:

$$\boldsymbol{Q}_{set}(s, a) = \boldsymbol{R}(s, a) \oplus \gamma \sum_{s'} T(s, a, s') \mathcal{ND}(\cup_{a'} \boldsymbol{Q}_{set}(s', a')) \tag{3}$$

Here, the Pareto front at state-action $\langle s, a \rangle$ is approximated by element-wise adding ($\oplus$) the immediate vector reward $\boldsymbol{R}(s, a)$ to every discounted and transition probability weighted vectorial value from the successor state s', after the $\mathcal{ND}$ operator prunes Pareto dominated values. The iteration process converges to the Pareto-optimal value vectors attainable by deterministic but generally non-stationary policies.

Building on PVI, Van Moffaert and Nowé [14] introduced *Pareto Q-Learning* (PQL), a model-free, value-based MORL algorithm that learns Pareto-optimal value sets directly from sampled experience. A key contribution of PQL is its separation of immediate and future rewards, which stabilises learning by allowing these components to converge independently. This avoids the need for brittle pairwise correspondences between vectors from $\boldsymbol{Q}_{set}$, which would otherwise break as sets evolve through continual updates and Pareto pruning. PQL learns in an *off-policy* manner, incorporating any MOMDP experience tuple $\langle s, a, \boldsymbol{r}, s' \rangle$. The average immediate reward vector $\bar{\boldsymbol{R}}(s, a)$ is updated toward $\boldsymbol{r}$, and the set of non-dominated future rewards $\boldsymbol{ND}(s, a)$ is updated from $\mathcal{ND}(\cup_{a'} \boldsymbol{Q}_{set}(s', a'))$. The Pareto-optimal value set for each state-action pair can be computed on demand as: $\boldsymbol{Q}_{set}(s, a) \leftarrow \bar{\boldsymbol{R}}(s, a) \oplus \gamma \boldsymbol{ND}(s, a)$. *Set evaluation mechanisms*, such as the *hypervolume metric*, assign scalar scores to $\boldsymbol{Q}_{set}$, enabling standard action-selection strategies (ε-greedy) without discarding the underlying multi-objective structure. For tabular domains, PQL recovers the entire Pareto front for arbitrary monotonic preferences.

2.3 Reward Machines

Reward machines were introduced in [6, 13] to encode non-Markovian reward functions, i.e., reward functions whose output does not depend solely on the current state-action-state triple, $\langle s, a, s' \rangle$, but also on previous ones.

[1] This work is based on contributions developed in the first author's master thesis [1].

Definition 2 (Reward Machine)**.** A reward machine (RM) is a tuple $\mathcal{R} = \langle U, u_0, F, \delta, \lambda \rangle$, where U is a non-empty finite set of states, $u_0 \in U$ is the initial state, $F \subset U$ is the set of final states, $\Sigma = 2^P$ is the alphabet of input symbols, with P a finite set of propositional symbols, $\delta : U \times 2^P \to U$ is the state transition function, and $\lambda : U \to (S \times A \times S \to \mathbb{R})$ is the reward function.

A variant of RMs, termed *simple reward machines* [6], modifies reward functions to be of the form $\lambda : U \times 2^P \to \mathbb{R}$. A simple reward machine can be converted into an equivalent reward machine, producing the same non-Markovian rewards. In the following, we always assume reward machines are defined in their general form. However, we use simple rewards machines in our experiments for ease of exposition.

A reward machine is used in conjunction with an MDP[2] and a *labelling function* $L : S \times A \times S \to 2^P$, a function which labels each state-action-state transition with events, given as input to the reward machine. At each timestep, the RM is in some state u: if the agent performs action a and the MDP moves from the current state s to the state s', the RM's state is updated to $u' = \delta(u, L(s, a, s'))$ and it outputs the reward $r = \lambda(u)(s, a, s')$. If the RM reaches a final state $u \in F$ the process terminates.

When using RMs, agents learn policies that are defined in a new environment obtained by taking the *product* between the MDP and the RM. This is called an *MDPRM*.

Definition 3 (MDPRM)**.** An MDPRM $\mathcal{M}_\mathcal{R} = \langle \mathcal{M}, P, L, \mathcal{R} \rangle$ is a tuple where $\mathcal{M}$ is an MDP, P is a finite set of propositional symbols, $L : S \times A \times S \to 2^P$ is a labelling function for P, and $\mathcal{R}$ is a reward machine with input alphabet 2^P producing rewards for $\mathcal{M}$.

An MDPRM $\mathcal{M}_\mathcal{R}$ induces a *product MDP* $\mathcal{M}' = \langle S', A', T', R', \gamma' \rangle$ obtained from the product between the MDPRM's MDP and RM, where: $S' := S \times U$; $A' := A$; $T'(\langle s, u \rangle, a, \langle s', u' \rangle) := T(s, a, s')$ if $u' = \delta(u, L(s, a, s'))$ and is 0 otherwise; $R'(\langle s, u \rangle, a, \langle s', u' \rangle) := \lambda(u)(s, a, s')$; $\gamma' := \gamma$.

By having an agent learning a policy in the product MDP $\mathcal{M}'$, we can train it by using an RM. The policy will thus be defined over the product between the state space of the MDP and of the RM, i.e., it will be of the form $\pi : (S \times U) \times A \to [0, 1]$.

Exploiting Counterfactual Experiences. Aside from being introduced to encode non-Markovian reward functions, another goal of RMs was to improve sample efficiency in training RL agents. This was achieved in [6, 13] by proposing off-policy approaches (QRM and CRM) which exploit *counterfactual experiences*. It should be noted that these approaches are not learning algorithms, but should be used in conjunction with one as they only provide a way to generate more training samples. Given any transition in the product MDP $\langle s, u, a, s', u' \rangle$ and associated reward r, these off-policy counterfactual approaches generate a set of synthetic experiences, one per non-terminal state u_c of the RM. This is done by changing the starting RM state u with u_c to obtain the new counterfactual state u'_c and reward r_c, where r_c is the reward given by the RM assuming that the starting state is u_c and not u. Then, all of these counterfactual experiences are used to perform policy updates using the off-policy algorithm of choice.

[2]We note that when using a reward machine, the MDP's reward function is usually ignored.

3 MULTI-OBJECTIVE REWARD MACHINES

We introduce *Multi-Objective Reward Machines* (MORMs), a general automaton-based framework for specifying vector-valued non-Markovian reward functions. MORMs preserve the dimensionality of multi-objective rewards and are therefore compatible with the entire spectrum of MORL algorithms. Moreover, MORMs leverage existing research on RMs [6] by allowing independently designed single-objective RMs to be composed into a unified multi-objective structure. Our framework offers a principled methodology for constructing, minimising, and integrating MORMs into the RL problem formulation, while also enabling extensions of RM-based sample-efficiency techniques to the multi-objective setting.

Related Work. To our knowledge, there is a notable absence of research addressing non-Markovian rewards in a fully multi-objective setting. Some recent work by [7] explores learning normative behaviour where multiple norms are treated as separate objectives. Although the authors frame the problem using a *lexicographic* MORL setting, the resulting MOMDP is ultimately collapsed into a single-objective RL problem via a scalarisation function.

3.1 Composing Single-Objective RMs

The MORM structure unifies a collection of single-objective RMs.

Definition 4 (MORM)**.** A d-tuple of RMs $\langle \mathcal{R}_i \rangle_{1 \le i \le d}$, where each $\mathcal{R}_i = \langle U_i, u_{i,0}, F_i, \delta_i, \lambda_i \rangle$ is defined over a set of propositional symbols P_i (possibly overlapping), can be composed into a *Multi-Objective Reward Machine* (MORM) $\mathcal{R} = \langle U, \boldsymbol{u}_0, F, \boldsymbol{\delta}, \boldsymbol{\lambda} \rangle$ defined over propositional symbols $P = \cup_i P_i$, using a *product automaton* construction [5] as follows:

- $U := \times_i U_i = U_1 \times \cdots \times U_d$
- $\boldsymbol{u}_0 := \langle u_{1,0}, \ldots, u_{d,0} \rangle$
- $F := \{\langle u_1, \ldots, u_d \rangle \in U \mid \exists i : u_i \in F_i\}$
- $\boldsymbol{\delta}(\langle u_1, \ldots, u_d \rangle, \sigma) := \langle u'_1, \ldots, u'_d \rangle$ where $\forall i : u'_i = \delta_i(u_i, \sigma \cap P_i)$
- $\boldsymbol{\lambda}(\langle u_1, \ldots, u_d \rangle)(s, a, s') := \langle r_1, \ldots, r_d \rangle$ where $\forall i : r_i = \lambda_i(u_i)(s, a, s')$

Each MORM state $\boldsymbol{u} = \langle u_1, \ldots, u_d \rangle \in U$ is a tuple consisting of single-objective RM states, thereby capturing exactly the reward-relevant history required for every dimension of the multi-objective reward function. The MORM initial state $\boldsymbol{u}_0 \in U$ consists of the initial states of all objective RMs, while the MORM terminal state set $F \subset U$ contains any MORM state where at least one objective RM terminates. The MORM transition function $\boldsymbol{\delta} : U \times 2^P \to U$ updates the MORM state given a truth assignment $\sigma \subseteq P$ by applying each objective RM's transition function on the subset of propositional symbols relevant to that objective $\sigma \cap P_i$. Finally, the MORM reward function $\boldsymbol{\lambda} : U \to (S \times A \times S \to \mathbb{R}^d)$ specifies a vector-valued Markovian reward function for each MORM state by combining the corresponding scalar reward functions for each objective RM.

3.2 MORM State Minimisation

The MORM state space U is exponential when combining several complex single-objective RMs. Reducing its size by removing MORM states that do not affect the represented multi-objective non-Markovian rewards yields a more compact automaton, which can substantially lower the computational cost of the corresponding MORL problem. To achieve this, we apply two complementary

techniques from standard automata theory [5] to eliminate both *equivalent* and *unreachable* MORM states, thereby maintaining a *minimal* representation whenever possible.

First, *equivalent* RM states are those that yield identical future reward behaviour for all possible agent-environment trajectories. Such states can be detected and merged recursively since they must define the same output reward function and transition to equivalent states for every possible input truth assignment. Because a MORM is constructed as a product automaton, it preserves the equivalence classes of states of its constituent single-objective RMs. Consequently, combining single-objective RMs that are already *minimal* guarantees that the resulting MORM is *minimal* with respect to state equivalence.

Second, the product construction may include large regions of *unreachable* MORM states. When two single-objective RMs $\mathcal{R}_i$ and $\mathcal{R}_j$ share a propositional symbol $p \in P_i \cap P_j$, certain combinations of their states may be impossible to reach because their temporal requirements for p are incompatible. For example, if $\mathcal{R}_i$ transitions to state $u_i \in U_i$ only when p is observed ($p \in \sigma$), while $\mathcal{R}_j$ transitions to state $u_j \in U_j$ only when p is *not* observed ($p \notin \sigma$), then any MORM state of the form $\langle u_i, u_j \rangle$ can never occur. Similar situations arise when two objectives share some temporal subgoal and must progress together rather than independently. To eliminate such unreachable states, we construct the MORM incrementally by expanding states via the transition function $\boldsymbol{\delta}$ only after confirming their reachability from the initial MORM state $\boldsymbol{u}_0$.

Together, these reductions yield a compact MORM that preserves the full reward semantics. In the worst case, when single-objective RMs do not interact and the full product automaton must be explored, the computational complexity of constructing the minimal MORM is $O(|U| \cdot 2^{|P|})$.

3.3 Solving MORL tasks with MORMs

We reformulate the MORL problem by defining a structure that combines a *Markov Decision Process with a Multi-Objective Reward Machine* (MOMDPRM) using a *single* labelling function to connect them. Formally, an MOMDPRM is a tuple $\mathcal{M}_{\mathcal{R}} = \langle \mathcal{M}, P, L, \mathcal{R} \rangle$, where $\mathcal{M} = \langle S, A, T, \mu, \gamma \rangle$ is the *no-objective* MDP (MDP without reward function R) specifying the environment dynamics, $L : S \times A \times S \to 2^P$ is the labelling function over the set of propositional symbols P, and $\mathcal{R} = \langle U, \boldsymbol{u}_0, F, \boldsymbol{\delta}, \boldsymbol{\lambda} \rangle$ is the MORM specifying the multi-objective reward function for the MDP. In an MOMDPRM, at each timestep, the environment transition $\langle s, a, s' \rangle$ is mapped by L to a truth assignment σ over P, updating the current MORM state $\boldsymbol{u}$ to $\boldsymbol{u}' = \boldsymbol{\delta}(\boldsymbol{u}, \sigma)$, while the agent receives vectorial reward $\boldsymbol{r} = \boldsymbol{\lambda}(\boldsymbol{u})(s, a, s')$.

Just like in single-objective MDPRMs, the vector-valued reward produced by an MOMDPRM are generally non-Markovian with respect to the environment state space S, but they become Markovian when considered over the *augmented state space* $S \times U$, since each MORM state encodes exactly the necessary history to determine future rewards. However, in the multi-objective setting, conditioning only on the current augmented state $\langle s, \boldsymbol{u} \rangle$ is *not* generally sufficient for optimal decision-making. As discussed in Section 2.1, Pareto-optimal policies may require *non-stationary* behaviour, meaning that an optimal action can depend on the full history of augmented states $\langle s_t, \boldsymbol{u}_t \rangle$.

Definition 5 (Cross-Product MOMDP)**.** Given a MOMDPRM $\mathcal{M}_{\mathcal{R}} = \langle \mathcal{M}, P, L, \mathcal{R} \rangle$, where $\mathcal{M} = \langle S, A, T, \mu, \gamma \rangle$ and $\mathcal{R} = \langle U, \boldsymbol{u}_0, F, \boldsymbol{\delta}, \boldsymbol{\lambda} \rangle$, we can construct the *Cross-Product* MOMDP $\mathcal{M}' = \langle S', A', T', \boldsymbol{R}', \mu', \gamma' \rangle$ as follows:

- $S' := S \times U$
- $A' := A$
- $T'(\langle s, \boldsymbol{u} \rangle, a, \langle s', \boldsymbol{u}' \rangle) := T(s, a, s')$ if $\boldsymbol{u}' = \boldsymbol{\delta}(\boldsymbol{u}, L(s, a, s'))$ (and otherwise zero)
- $\boldsymbol{R}'(\langle s, \boldsymbol{u} \rangle, a, \langle s', \boldsymbol{u}' \rangle) := \boldsymbol{\lambda}(\boldsymbol{u})(s, a, s')$
- $\mu'(\langle s, \boldsymbol{u} \rangle) := \mu(s)$ if $\boldsymbol{u} = \boldsymbol{u}_0$ (and otherwise zero)
- $\gamma' := \gamma$

Any (possibly non-stationary) policy $\pi : (S \times U)^* \times A \to [0, 1]$ will attain the same expected sum of discounted vector-valued rewards in an MOMDPRM as it does in the corresponding cross-product MOMDP.

This construction of a reward-equivalent MOMDP over the augmented state space $S \times U$ enables the direct application of standard MORL algorithms to solve MDPRMs. The only modification required to learn policies over $S \times U$ is to track the current MORM state $\boldsymbol{u} \in U$ during environment interaction and condition all maintained q-values on this. As a result, each augmented experience tuple becomes $\langle s, \boldsymbol{u}, a, \boldsymbol{r}, s', \boldsymbol{u}' \rangle$, where the next MORM state is given by $\boldsymbol{u}' = \boldsymbol{\delta}(\boldsymbol{u}, L(s, a, s'))$ and the vector-valued reward is $\boldsymbol{r} = \boldsymbol{\lambda}(\boldsymbol{u})(s, a, s')$. Analogous to the single-objective case, the *cross-product baseline* has broad applicability and preserves any theoretical guarantees of the MORL methods used but can be extremely sample-inefficient because it learns each q-value independently. This limitation becomes even more pronounced in the multi-objective setting, where multiple complex objectives increase the size of the MORM state space $|U|$.

3.4 Exploiting MORM Structure

Fortunately, a larger MORM also encodes more structured temporal information across all reward dimensions, that can be leveraged to improve sample efficiency, similar to the single-objective case.

Counterfactual Experiences for Reward Machines (CRM) [6] carries over naturally to the multi-objective setting. Because the MORM transition function $\boldsymbol{\delta}$ and reward function $\boldsymbol{\lambda}$ are fully deterministic given an environment transition $\langle s, a, s' \rangle$, counterfactual reasoning over MORM states becomes straightforward. Learning again over the cross-product space $S \times U$, for any observed augmented transition $\langle s, \boldsymbol{u}, a, \boldsymbol{r}, s', \boldsymbol{u}' \rangle$, we can determine how *every* (non-terminal) MORM state $\boldsymbol{u}_c \in U$ would have evolved under that same transition. As in the single-objective case, this yields the following set of *synthetic* experiences:

$$\{\langle s, \boldsymbol{u}_c, a, \boldsymbol{\lambda}(\boldsymbol{u}_c)(s, a, s'), s', \boldsymbol{\delta}(\boldsymbol{u}_c, L(s, a, s')) \rangle \mid \forall \boldsymbol{u}_c \in U \setminus F\} \quad (4)$$

These experiences can then be directly supplied to and utilised by any *off-policy* MORL algorithm. Finally, we expect CRM in the multi-objective setting to offer sample-efficiency improvements analogous to those observed in the single-objective case, since it generates counterfactual experiences across the entire MORM state space U at every environment interaction, thereby greatly improving coverage of the augmented state space $S \times U$.

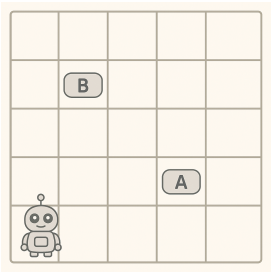


Figure 1: The *ButtonWorld* environment.

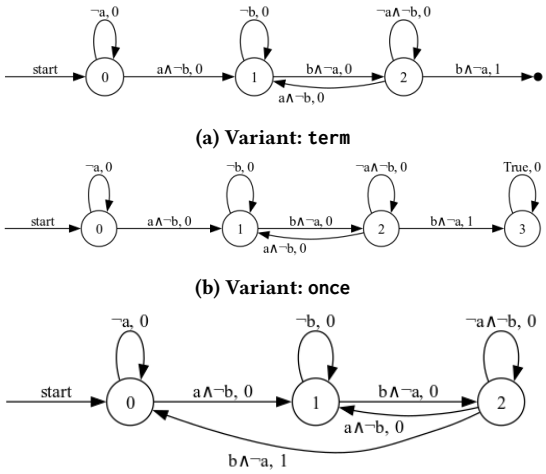


Figure 2: Reward Machines for all variants of the first objective, rewarding the event sequence *abb*.

4 EXPERIMENTS

We empirically evaluate the proposed Multi-Objective Reward Machine (MORM) framework. We first describe the experimental setup, and then present results on a series of multi-objective tasks obtained by composing single-objective components, analysing how MORM-based learning compares to the cross-product baseline and how CRM improves sample efficiency. The experiments highlight both the strengths and the limitations of MORM-based MORL.

4.1 Experimental Setup

Based on the assembly line scenario, we introduce *ButtonWorld*, a deterministic grid-world with two buttons, with no built-in rewards but with labelled transitions (Figure 1). A button is considered pressed when the agent steps onto its cell ($s_A, s_B \in S$), but it does not remain pressed once the agent leaves. The labelling function L maps transitions $\langle s, a, s' \rangle$ to truth assignments over propositional symbols $P = \{a, b\}$, where $a \in L(s, a, s')$ iff $s' = s_A$ and $b \in L(s, a, s')$ iff $s' = s_B$. This simple labelled environment supports non-Markovian task specifications of varying complexity, enabling experiments where rewards are triggered by temporal patterns of button presses.

Next, we define three single-objective non-Markovian tasks using RMs. Consider the top RM in Figure 2. The initial state `0` waits for pressing button `A` before moving to `1`, which then waits for pressing button `B` to proceed to `2`. From state `2`, pressing `A` resets progress to `1`, while pressing `B` yields a reward and terminates the environment (i.e., variant `term`). Thus, this RM rewards the labelled event sequence *abb* exactly once before halting. The next two RMs are variants that differ only in what happens after producing the reward. The `once` variant replaces the terminal state with a non-terminal *zero-sink* state, preserving the same rewards for this objective but allowing the agent-environment interaction to continue. The `cycle` variant additionally resets the RM to its initial state, enabling repeated completions of the objective sequence *abb*. We also construct a similar set of RMs rewarding the sequence *baa*. Experiments combine one variant of the first (*abb*) objective with one variant of the second (*baa*) objective. The resulting MORMs for all experiments are provided in Appendix B.

We apply two algorithms in all multi-objective experiments, Pareto Value Iteration (PVI) and Pareto Q-Learning (PQL). PVI operates on a full model of the cross-product MOMDP and serves as a planning-based upper bound on achievable performance, allowing us to approximate the true infinite-horizon Pareto front for each multi-objective task. In contrast, PQL is used to learn the Pareto front in a model-free RL setting, and we evaluate its cross-product baseline (PQL) against its MORM-exploiting variant (PQL-CRM). Beyond being fully compatible with CRM, PQL provides formal convergence guarantees in tabular environments, enabling a clean measurement of sample efficiency gains.

All approaches are trained for 50,000 timesteps. ε is decayed linearly from 1.0 to 0.1, and $\gamma = 0.99$. Episodes are truncated at 200 steps to prevent excessively long trajectories. Full implementation and hyperparameter details are presented in Appendix A.2.

Every 1000 timesteps, we evaluate the current approximation of the infinite-horizon Pareto front by reconstructing a *tracking policy* for each discovered value vector. We assess the quality of the resulting set using three metrics [4]. *Cardinality* counts the number of non-dominated policies and reflects exploration progress. *Hypervolume* measures the volume in value space dominated by the discovered front relative to the reference point $\boldsymbol{V}_{ref} = \boldsymbol{-0.5}$. *Expected Utility* (EUM) averages user utility across 50 uniformly spaced linear scalarisations, a utility-based measure of front quality. Results are averaged over 30 independent random seeds.

Full code implementation is available on the following GitHub repository: https://github.com/panaro32/mo_reward_machines.

4.2 Results

This section presents and discusses our experimental results. Each multi-objective task is denoted `X-Y`, where `X`, `Y` $\in$ {`term`, `once`, `cycle`}, with `X` referring to the variant of the first (*abb*) objective and `Y` to the variant of the second (*baa*) objective.

4.2.1 Finite-horizon Sequences (`term-term`, `once-once`). We first performed the `term-term` and `once-once` experiments to validate the testing process for the more complex scenarios that follow. In the former, both objectives are terminating upon the first sequence completion so we get two optimal policies, each completing one objective while ignoring the other. In the latter, both objectives

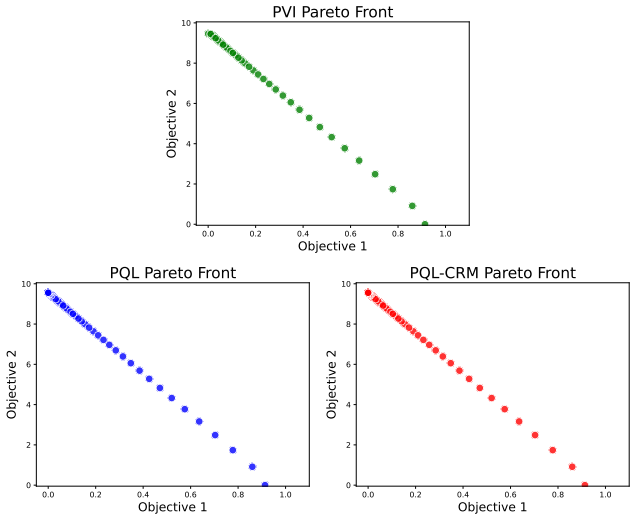


Figure 3: The Pareto fronts for bi-objective `term-cycle`.

are now non-terminating, allowing for (one) completion per objective sequence. In this scenario there are two optimal policies, each completing both objective sequences, but in different priority order: *abbaa* and *baabb*. Notice that overlapping the objective sequences is allowed and optimal policies take advantage of this when possible. The expected results for these scenarios are provided in Appendix B.1 and B.2.

4.2.2 Exiting Cycle (`term-cycle`). This experiment introduces a *cyclic* objective, i.e., the event sequence can be completed repeatedly to yield discounted rewards. Here, the *abb* objective is terminal, while *baa* can be repeated indefinitely. Thus, optimal policies must reason about how many times to execute the *baa* cycle before finally completing *abb*. Each additional repetition of *baa* increases the second objective's reward but further delays the reward for the first. Because γ is shared across objectives, this induces a *linear trade-off* between them. Consequently, the Pareto front consists of a family of *non-stationary* policies that complete $(baa)^n bb$ for some integer $n > 0$. The special case ($n = 0$) where the first objective is immediately triggered, *abb*, yields a point that lies slightly off the linear front due to the lack of overlap between the objectives in this case. As $n \to \infty$, the optimal policy approaches repeating the cycle forever, $(baa)^\omega$.

PVI recovers this entire set of Pareto-optimal values. Both PQL and PQL-CRM converge to the same value vectors matching PVI (Figure 3). However, their evaluated fronts (Figure 4) include only those tracking policies feasible under the episode length limit (200 steps). Therefore, only value vectors where $n < 20$ are trackable, including the policy that repeats *baa* until truncation. The learning curves for all three quality metrics show a clear advantage for PQL-CRM over baseline PQL, both in sample efficiency and in learning stability. This improvement arises because PQL-CRM systematically provides broad and uniform coverage of the augmented state space, reducing dependence on stochastic exploration.

4.2.3 Interrupting Cycle (`once-cycle`). This setting closely resembles `term-cycle`, the second objective *baa* is again cyclic, but the first objective *abb* is now non-terminating, although it can still

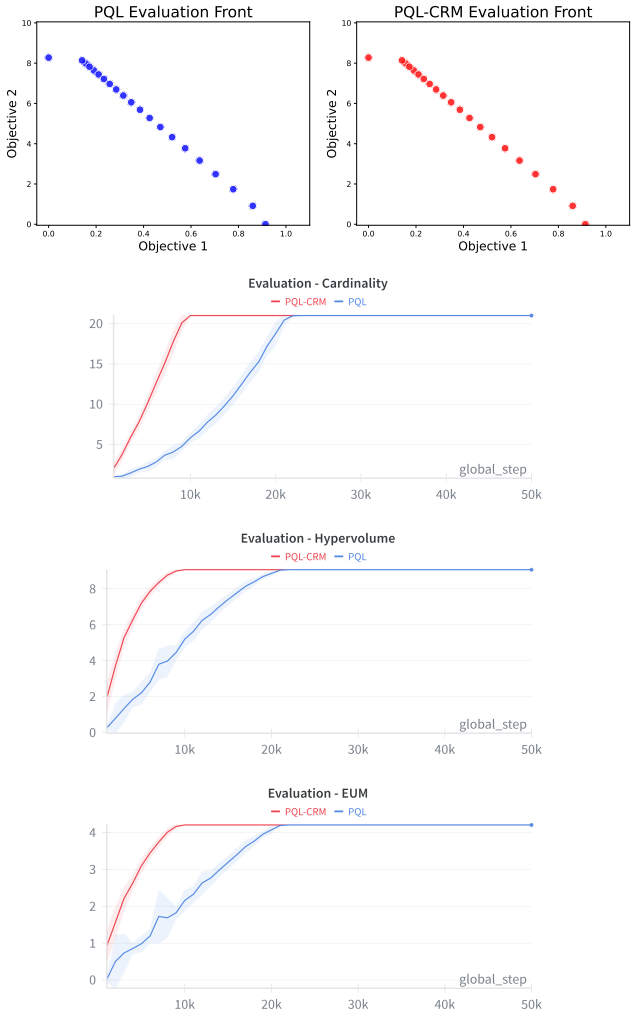


Figure 4: Evaluated fronts and quality metrics for bi-objective `term-cycle`.

be completed only once. As before, optimal policies decide how many times to execute the *baa* cycle before interrupting to complete *abb* once, after which they keep repeating *baa* indefinitely. The task becomes fully infinite-horizon and all ideal value vectors correspond to limits of infinite trajectories. As such, the Pareto front again consists of a family of *non-stationary* policies triggering $(baa)^n b(baa)^\omega$ for integers $n > 0$. The trade-off remains linear, with case $n = 0$ and the limiting case $n \to \infty$ being special like `term-cycle`.

PVI successfully recovers the entire front. PQL-CRM also converges to the same set of value vectors, but baseline PQL identifies only a couple of Pareto-optimal points (Figure 5). This failure stems from insufficient exploration in the infinite-horizon setting. Under naive ε-greedy exploration, PQL eventually finds the policy that maximises the *abb* reward, and the corresponding value for *baa* continues improving over time. Alternative policies that perform additional *baa* cycles before completing *abb* are explored but

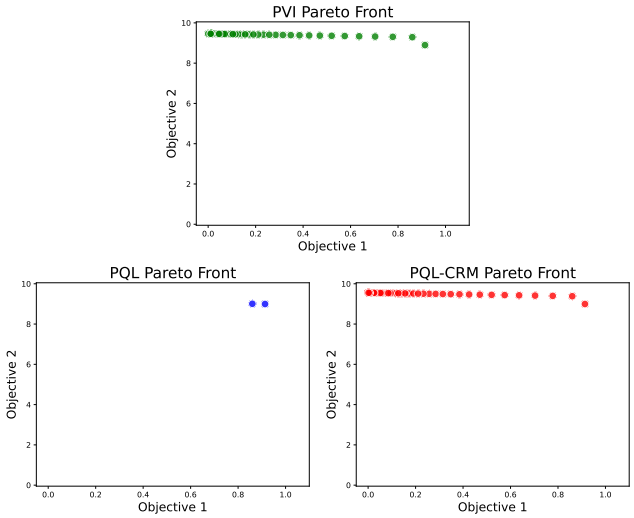


Figure 5: The Pareto fronts for bi-objective `once-cycle`.

always lag behind in discounted return, making them appear Pareto-dominated by the initially discovered point. In contrast, the guided uniform exploration of PQL-CRM resolves this issue. PQL-CRM's evaluated front mirrors the structure seen in `term-cycle` ($n < 20$), whereas PQL tracks only the single discovered value vector. Although the final quality metrics appear close due to the narrow shape of the true front, baseline PQL entirely fails to uncover its structure (Figure 6).

4.2.4 Interleaving Cycles (`cycle-cycle`). This final and most complex experiment places both objectives in their cyclic form, making the task fully infinite-horizon once again. Here, the agent may complete either reward sequence, *abb* or *baa*, in any order indefinitely, that is, $(abb \mid baa)^{\omega}$. The interesting feature of this scenario is that the ordering of completions does not merely trade discounted return between objectives. Once both objectives have a nonzero number of completions, policies that interleave the two sequences, as early and as frequently as possible, achieve *higher returns for both objectives*. This occurs because switching between objectives always allows for a skipped button press (an overlap), whereas repeating the same sequence does not. As a result, the trade-off between objectives becomes *nonlinear* because balancing completions across objectives yields additional returns for both. Consequently, optimal policies begin by interleaving the two sequences, following initial patterns such as $a(bbaa)^n$ or $b(aabb)^n$ for integers $n \geq 0$.

PVI recovers the true symmetric, *piecewise-linear* Pareto front predicted by this structure. Baseline PQL approximates parts of the front by exploring different sequence orderings but fails to discover regions requiring more deliberate sequencing, specifically, the extremes (where one objective is ignored) and the exact centre (perfect objective alternation), since these patterns are more unlikely to arise under random exploration. In contrast, PQL-CRM's structured, uniform exploration of the augmented state space once again uncovers the full true Pareto front (Figure 7).

As for the evaluated fronts, PQL is able to construct tracking policies (within the episode length limits) for a substantial subset of its discovered front. PQL-CRM, however, yields evaluated fronts

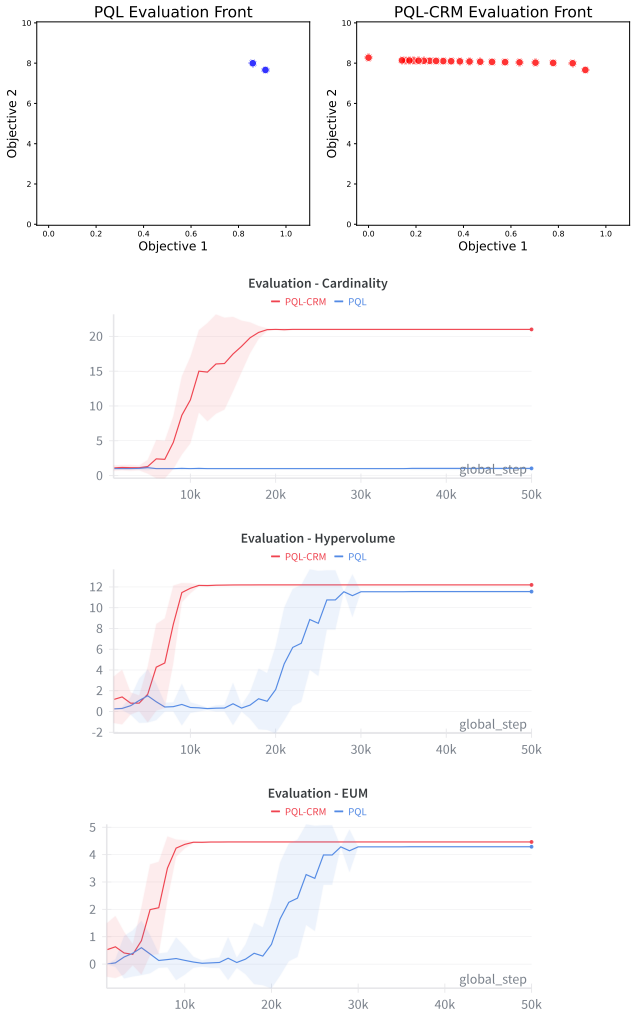


Figure 6: Evaluated fronts and quality metrics for bi-objective `once-cycle`.

that consist mostly of policies reaching either extreme points or collapsing toward the centre. This is because PQL-CRM's policy tracking over-optimises for the long-horizon limit, typically starting by balancing completions across objectives, but episode truncation prevents these policies from fully expressing the intended structure, causing many to map to the centre region (Figure 8).

The learning curves reflect this dynamic. All quality metrics improve sooner under PQL-CRM due to its superior sample efficiency. However, the final cardinality for PQL-CRM is lower, as policy tracking biases the evaluated policies toward the centre. As a result, PQL achieves a higher final hypervolume, since extreme points have a lesser contribution under the chosen reference point. Nevertheless, PQL-CRM still attains a higher expected utility, since the extreme points do make a significant contribution to EUM (Figure 8).

4.2.5 Runtime Comparison. We also provide a runtime comparison between PQL and PQL-CRM across all multi-objective tasks (Appendix Table 1). PQL-CRM requires substantially more computation time than baseline PQL. This overhead is expected because at

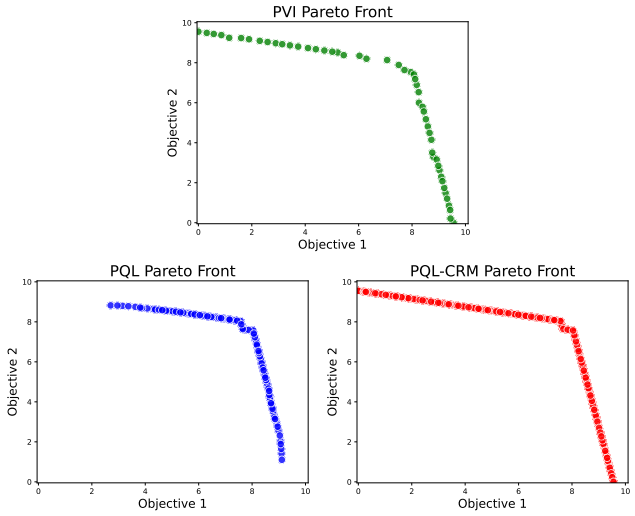


Figure 7: The Pareto fronts for bi-objective `cycle-cycle`.

every environment interaction PQL-CRM generates counterfactual experiences for all MORM states and performs many more q-set updates per timestep. In practice, this additional cost can be mitigated through parallelisation, a consideration we plan to explore in future work. By providing structured and systematic coverage of the augmented state space, PQL-CRM can succeed in settings where unguided exploration fails entirely, as demonstrated above. Thus, although PQL-CRM is computationally heavier, its ability to reduce reward sparsity and ensure reliable exploration can make it essential rather than merely beneficial.

5 CONCLUSION

We introduced Multi-Objective Reward Machines, a novel principled automaton-based framework for representing vector-valued non-Markovian reward functions. MORMs extend classical Reward Machines to the multi-objective setting while preserving full compatibility with existing MORL algorithms. They enable transparent modelling of temporally structured rewards across multiple objectives, provide a compositional method for combining independently designed single-objective specifications, and yield a minimal augmented MOMDP on which standard planning and learning algorithms can operate. By exposing the reward structure, MORMs also make it possible to generalize RM-based sample-efficiency techniques, such as Counterfactual Experiences for Reward Machines, to the multi-objective domain.

Our experimental analysis validated the theoretical results established in the methodology. When integrated with Pareto Q-Learning, CRM consistently accelerated learning across all tasks, often by large margins. In one case, PQL without CRM failed to recover the true Pareto structure entirely, whereas PQL-CRM succeeded, demonstrating that exploiting the MORM structure can be essential rather than merely beneficial. Across all experiments, PQL-CRM improved coverage, stability, and sample efficiency, confirming the practical value of the proposed framework.

In future work we plan to design new multi-objective non-Markovian benchmarks for continuous state and action domains by extending established MORL environments in *MO-Gymnasium* [2]

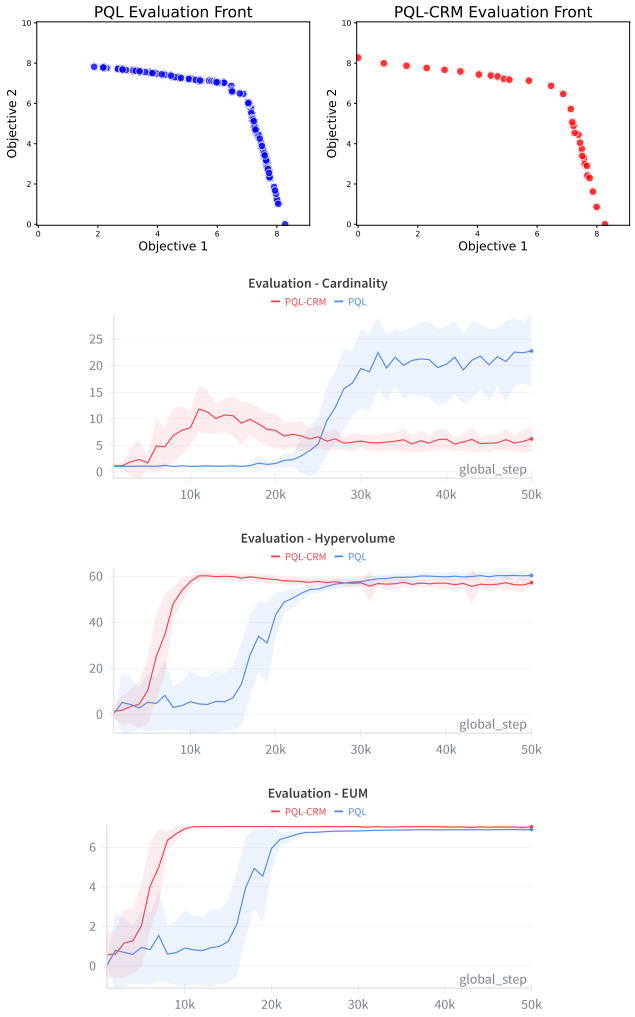


Figure 8: Evaluated fronts and quality metrics for bi-objective `cycle-cycle`.

with temporally dependent reward specifications. This would allow for testing MORM scalability and CRM performance by applying MORM-based learning to modern MORL algorithms, such as outer-loop approaches using policy-gradient subroutines [9]. These methods are required for non-tabular settings, and evaluating whether CRM still yields significant sample-efficiency gains is an important step toward practical deployment. Finally, although the hierarchical RL and automated reward shaping RM-exploiting techniques [6] do not generalise naturally to the multi-objective setting, ideas inspired by them may still be adapted or redesigned for structured exploration, option discovery, or reward guidance within MORMs.

REFERENCES

[1] Panos Aronis. 2025. *Leveraging Reward Machines for Efficient Multi-Objective Reinforcement Learning.* Master's thesis. Utrecht University, Utrecht, the Netherlands. Available at https://studenttheses.uu.nl/handle/20.500.12932/50791.

[2] Florian Felten, Lucas N Alegre, Ann Nowe, Ana Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno C da Silva. 2023. A toolkit for reliable benchmarking and research in multi-objective reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 23671–23700.

[3] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. 1998. Multi-criteria reinforcement learning.. In *ICML*, Vol. 98. 197–205.

[4] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. 2022. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 26.

[5] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News* 32, 1 (2001), 60–65.

[6] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research* 73 (2022), 173–208.

[7] Emery A Neufeld, Agata Ciabattoni, and Radu Florin Tulcan. 2025. Combining MORL with restraining bolts to learn normative behaviour. In *IJCAI 2025 Workshop on User-Aligned Assessment of Adaptive AI Systems*.

[8] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* 48 (2013), 67–113.

[9] Willem Röpke, Mathieu Reymond, Patrick Mannion, Diederik M Roijers, Ann Nowé, and Roxana Rădulescu. 2024. Divide and conquer: Provably unveiling the pareto front with multi-objective reinforcement learning. *arXiv preprint arXiv:2402.07182* (2024).

[10] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. 2021. Reward is enough. *Artificial intelligence* 299 (2021), 103535.

[11] Rohan Subramani, Marcus Williams, Max Heitmann, Halfdan Holm, Charlie Griffin, and Joar Max Viktor Skalse. 2024. On the Expressivity of Objective-Specification Formalisms in Reinforcement Learning. In *ICLR*.

[12] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[13] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. 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 2018)*. PMLR, 2112–2121.

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

[15] C Ch White and KIM KW. 1980. Solution procedures for vector criterion Markov decision processes.

[16] DJ White. 1982. Multi-objective infinite-horizon discounted Markov decision processes. *Journal of mathematical analysis and applications* 89, 2 (1982), 639–647.

A EXPERIMENTAL DETAILS

A.1 ButtonWorld

ButtonWorld is a deterministic grid-world with no built-in rewards but with labelled transitions. The agent can move in the four cardinal directions ($|A| = 4$) within a 5×5 grid ($|S| = 25$). The agent moves exactly one grid cell in the chosen direction unless this would leave the grid, in which case it remains in place. At the start of each episode, the agent is placed in the bottom-left corner of the grid. Two buttons are located at fixed grid cells. Button A lies at state $s_A \in S$, positioned three steps to the right and one step up from the agent's start. Button B lies at state $s_B \in S$, symmetric to s_A across the diagonal passing through the starting cell. A visualisation of *ButtonWorld* at the beginning of an episode is shown in Figure 9. A button is

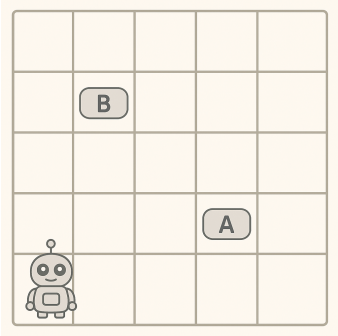
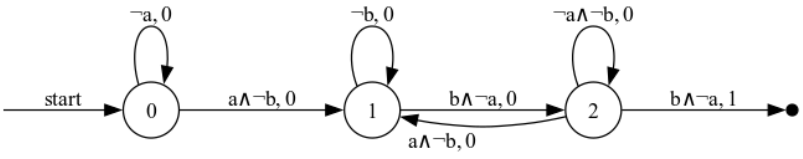
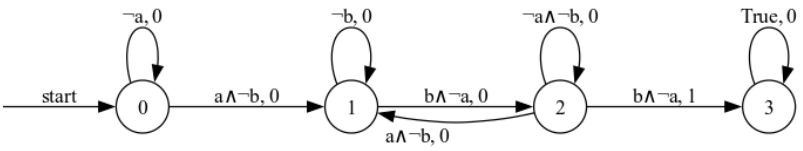


Figure 9: The *ButtonWorld* environment.

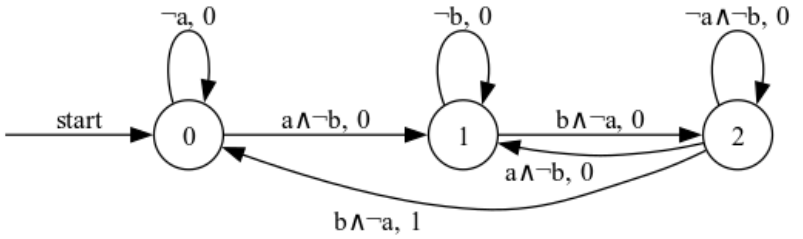
considered pressed when the agent steps onto its cell, but it does not remain pressed once the agent leaves. Formally, the labelling function L maps transitions $\langle s, a, s' \rangle$ to truth assignments over propositional symbols $P = \{a, b\}$, where $a \in L(s, a, s')$ iff $s' = s_A$ and $b \in L(s, a, s')$ iff $s' = s_B$. This simple labelled environment supports non-Markovian task specifications of varying complexity, enabling experiments where rewards are triggered by temporal patterns of button presses.



(a) Variant: `term`



(b) Variant: `once`



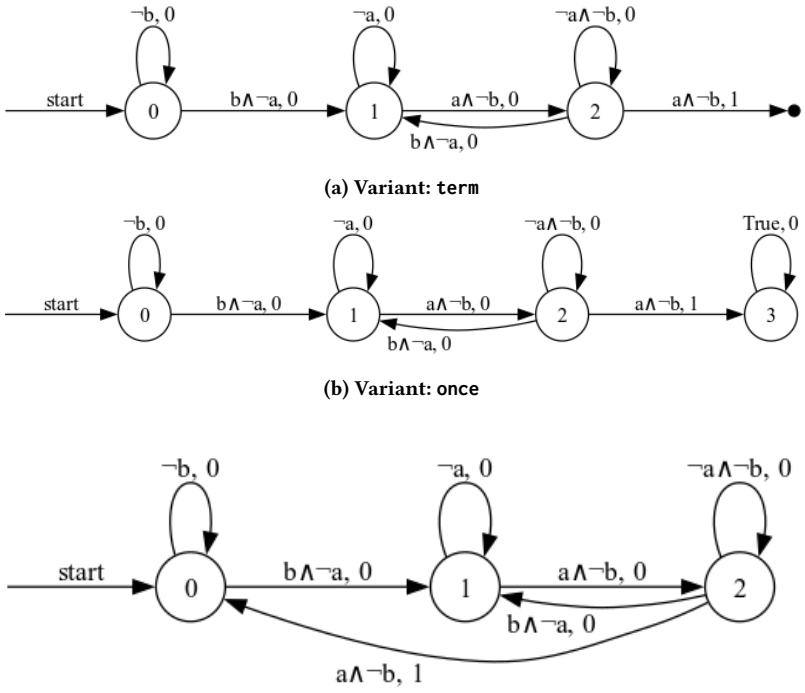
(c) Variant: `cycle`

Figure 10: Reward Machines for all variants of the first objective, rewarding the event sequence *abb*.

Next, we define a collection of single-objective non-Markovian tasks using RMs. These RMs specify scalar rewards in *ButtonWorld* based solely on the button presses produced at each timestep. Figure 10 shows graphical representations of such RMs. Nodes represent the RM state space U, the edge labeled `start` identifies the initial state u_0, and black circles denote terminal states F. Each directed edge from u to u' is labelled with a propositional logic formula φ over P and a real number r, meaning that whenever the truth assignment $\sigma \in 2^P$ satisfies φ ($\sigma \models \varphi$), the RM transitions to $u' = \delta(u, \sigma)$ and yields reward $r = \lambda(u, \sigma)$ (using the simple RM convention). Some nodes may omit transitions for certain σ, in which case the RM would halt and output no reward, though in our setting this occurs only for the impossible assignment $\sigma = P$ since the buttons occupy distinct grid cells.

Consider the top RM in Figure 10. The initial state `0` waits for button `A` before moving to `1`, which then waits for `B` to proceed to `2`. From state `2`, pressing `A` resets progress to `1`, while pressing `B` yields a reward and terminates the environment. Thus, this RM rewards the labelled event sequence *abb* exactly once before halting. The next two RMs are variants that differ only in what happens after producing the reward. The former replaces the terminal state with a non-terminal *zero-sink* state, thus preserving the same rewards for this objective but allowing any other objectives to progress. The latter also removes the terminal state but resets the RM to its initial state, thus enabling repeated completion of the sequence. We refer to these three variants as `term`, `once`, and `cycle`, respectively.

We also construct a parallel set of three RMs rewarding the sequence *baa*, again with `term`, `once`, and `cycle` variants, shown in Figure 11. All experiments combine one variant of the first (*abb*) objective with one variant of the second (*baa*) objective.



(a) Variant: `term`

(b) Variant: `once`

(c) Variant: `cycle`

Figure 11: Reward Machines for all variants (`term`, `once`, `cycle`) of the second objective, rewarding the agent for triggering the event sequence *baa*.

A.2 Hyperparameters

We now summarise the hyperparameters and implementation choices used across all multi-objective experiments. In every setting, we maintain at most 50 non-dominated value vectors, pruning the closest pair under the crowding-distance metric whenever the frontier exceeds this size. For PVI, we declare convergence once all values in the Pareto front change by less than 0.01 between iterations. For PQL and PQL-CRM, each learning session runs for exactly 50,000 timesteps. Exploration

follows an ε-greedy strategy using the hypervolume metric for q-set evaluation, with the exploration rate ε decaying linearly from 1.0 to 0.1 over the entire training duration. The discount factor is fixed at $\gamma = 0.99$ to reflect the infinite-horizon nature of the tasks and the sparse/terminal reward structure. Episodes are truncated at 200 steps to prevent excessively long trajectories.

Every 1000 timesteps, we evaluate the current approximation of the infinite-horizon Pareto front by reconstructing a *tracking policy* for each discovered value vector. At each evaluation, we assess the quality of the resulting set of feasible policies using three metrics [4]. *Cardinality* counts the number of non-dominated policies and reflects exploration progress, though it can be misleading as a quality metric since a single policy may dominate several already obtained. *Hypervolume* measures the volume in value space dominated by the discovered front relative to the reference point $\boldsymbol{V}_{ref} = \boldsymbol{-0.5}$, capturing solution spread, though it is not easy to interpret. *Expected Utility* (EUM) averages user utility across 50 uniformly spaced linear scalarisations, providing a more meaningful utility-based measure of front quality. All experiments are repeated with 30 independent random seeds $(42 - 71)$ to quantify variability due to stochastic exploration.

B EXPERIMENTAL RESULTS

This section presents and discusses our experimental results. Each multi-objective task is denoted `X-Y`, where `X`, `Y` $\in$ {`term`, `once`, `cycle`}, with `X` referring to the variant of the first (*abb*) objective (Figure 10) and `Y` to the variant of the second (*baa*) objective (Figure 11).

For every experiment, results are organised into two complementary figures. The first figure displays the Multi-Objective Reward Machine (MORM) obtained by composing the selected single-objective RMs and plots the *Pareto fronts* of multi-objective values computed by PVI (green), PQL (blue) and PQL-CRM (red). This provides a direct comparison between the true model-based front and the fronts learned by the two RL variants. The second figure reports the corresponding (final) *evaluated fronts* for the set of tracking policies recovered from PQL (blue) and PQL-CRM (red), together with their learning curves. These curves, aggregated over five independent runs, show the evolution of three quality metrics for the evaluated fronts, namely, the *cardinality*, the *hypervolume*, and the *expected utility* metrics, over training time. This dual presentation allows us to examine not only the asymptotic quality of the learned policy sets but also the stability and sample-efficiency of the learning process across methods.

B.1 Terminating Sequences (`term-term`)

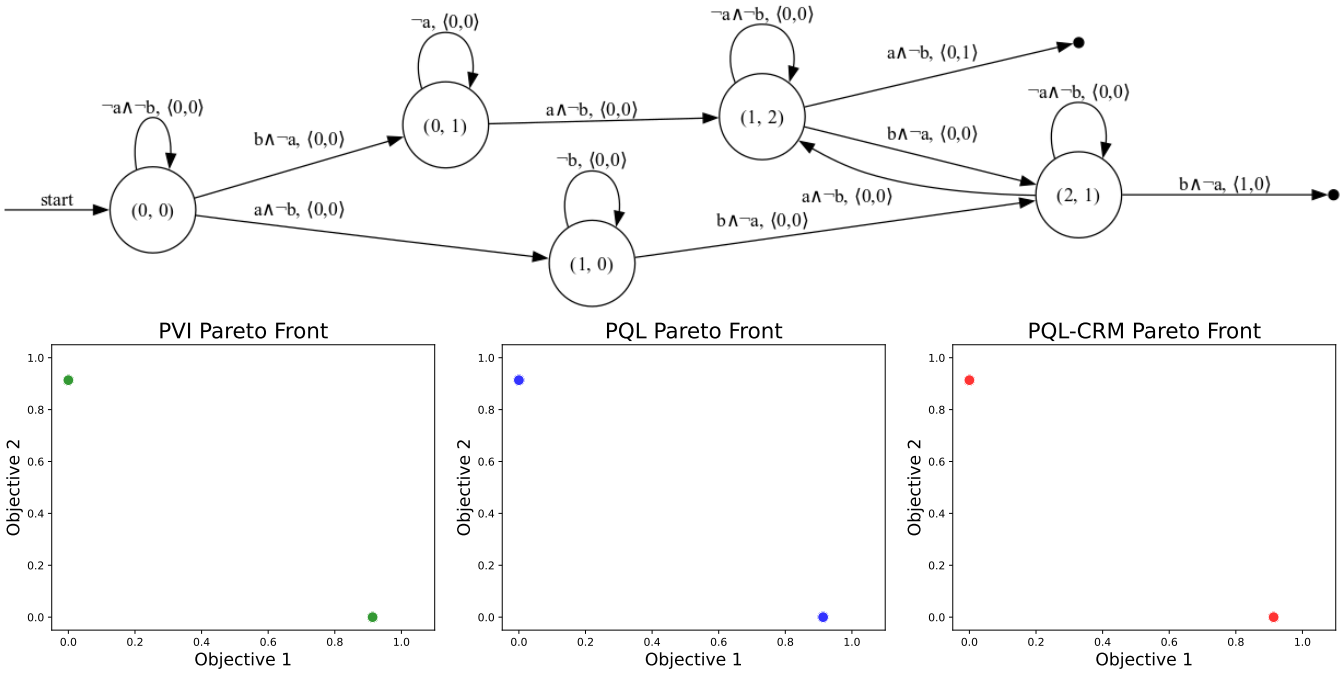


Figure 12: The MORM and Pareto fronts for bi-objective `term-term`.

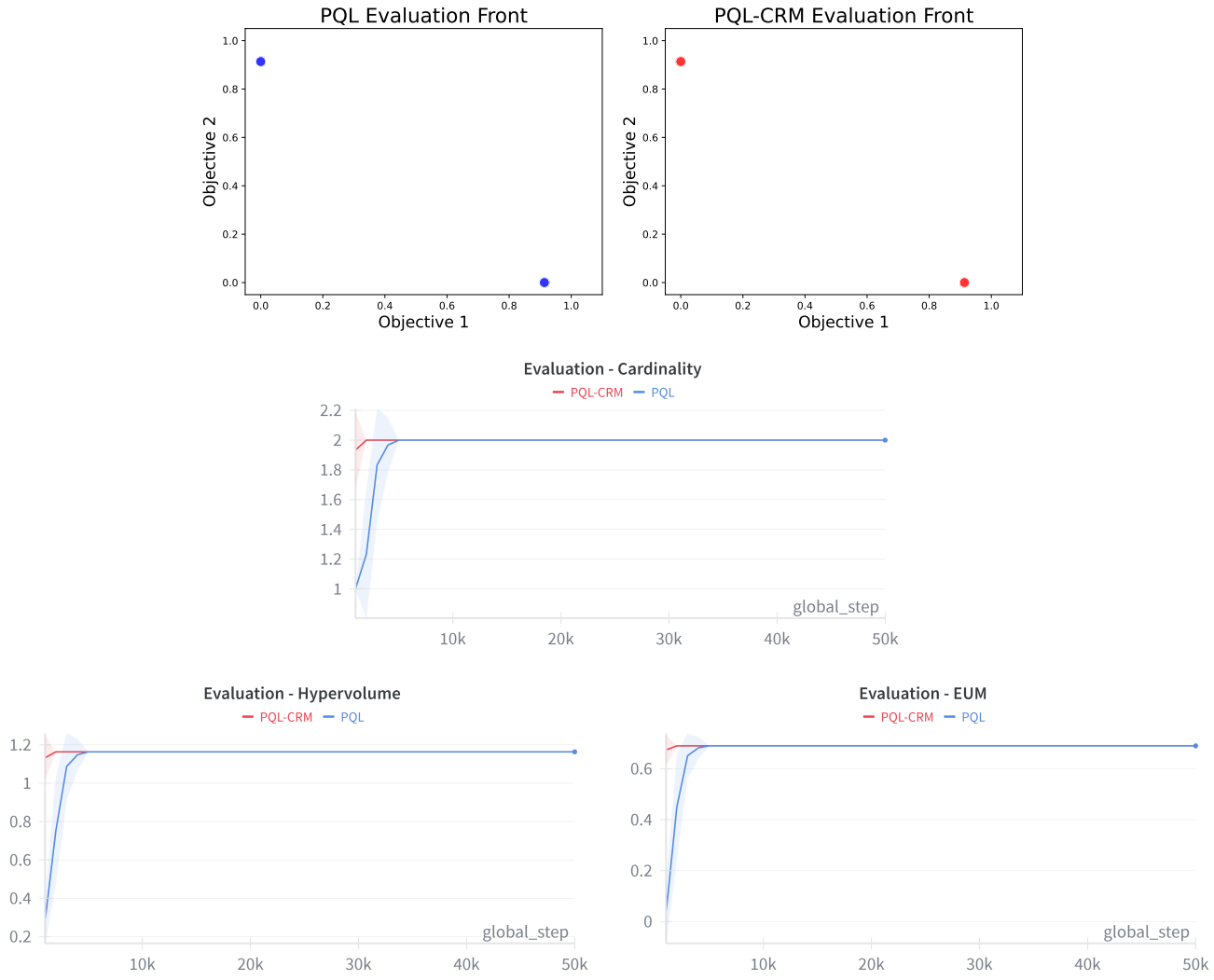


Figure 13: Evaluated fronts and quality metrics for bi-objective `term-term`.

The results for the bi-objective `term-term` task are shown in Figures 12 and 13. In this setting, both objectives terminate upon completing their respective event sequences, leaving no possibility of achieving more than one objective within a single run. Consequently, the Pareto front consists of exactly two policies, one that completes *abb* and one that completes *baa*, each maximizing its corresponding objective while yielding zero on the other.

The MORM captures this structure directly. Both non-zero rewards occur only when transitioning into terminal states. The MORM state space is also notably reduced, containing only five non-terminal states instead of the nine in the full Cartesian product. For example, the state (1, 1) is unreachable because it encodes incompatible event histories, where the first objective expects the last observed event to be *a*, whereas the second expects it to be *b*, and these cannot co-occur in a single transition.

PVI recovers the two Pareto-optimal value vectors, each assigning the optimal discounted return to one objective and zero to the other. Both PQL and PQL-CRM successfully learn these values and construct tracking policies for each point, leading the evaluated fronts to match the true front. Although the learning problem requires limited exploration and both methods converge quickly, PQL takes a few evaluation cycles (each spanning 1000 timesteps), whereas PQL-CRM identifies both points within the first evaluation. The hypervolume metric remains zero throughout because both optimal points share a zero coordinate with the chosen reference point, thus contributing zero area. The expected utility measure does not have the same problem.

B.2 Non-Terminating Sequences (once-once)

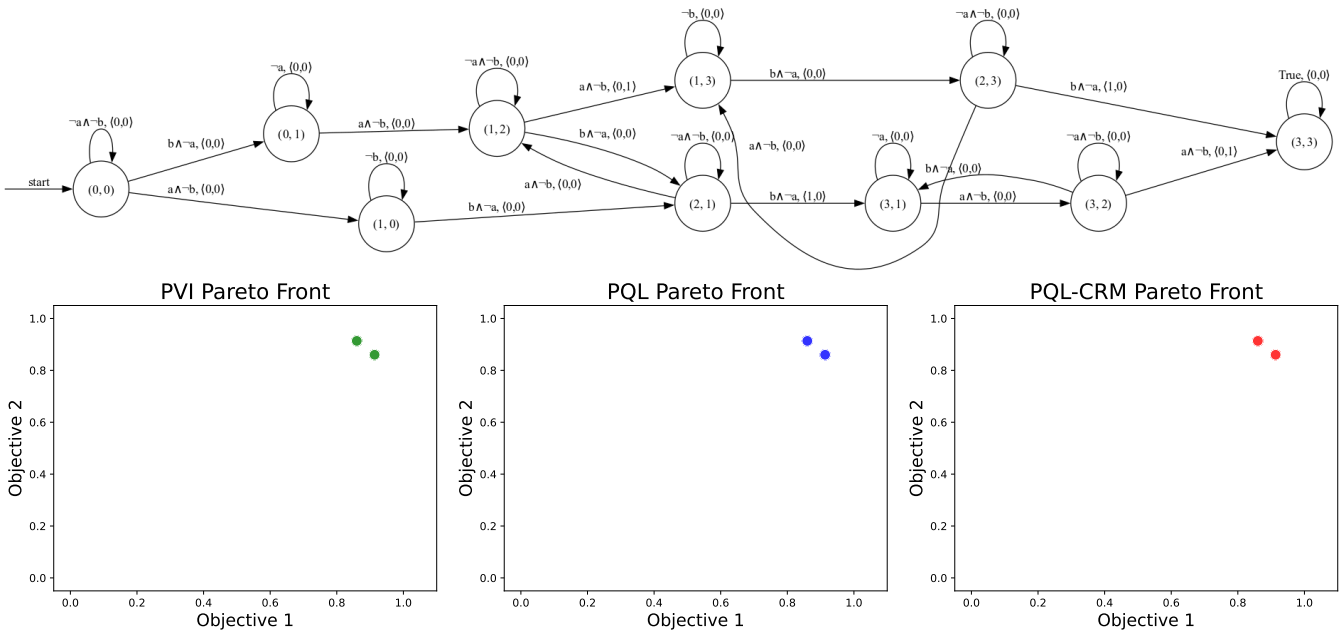


Figure 14: The MORM and Pareto fronts for bi-objective once-once.

The results for the bi-objective once-once task are shown in Figures 14, 15 and 16. This setting is similar to term-term since each objective can be completed only once. However, unlike before, neither objective terminates the interaction, allowing the agent to complete both sequences within a single episode. As a result, policies that achieve both objectives Pareto-dominate those that complete only one. Because the discount factor makes earlier rewards more valuable, the order of completion matters, that is, completing *abb* first yields a different return than completing *baa* first. The Pareto front therefore contains exactly two policies, one that achieves *abb* then *baa*, and one that achieves *baa* then *abb*. Since progress toward one sequence is not reset when the other sequence completes, these policies correspond to completing the overlapping sequences *abbaa* and *baabb*, each maximizing the second reward collected.

The resulting MORM includes additional states capturing partial progress on one objective after the other has been completed. The zero-sink state (3, 3) represents the completion of both objectives and functions effectively as a terminal state. PVI again identifies the two optimal value vectors, and both PQL and PQL-CRM recover them, as reflected in the Pareto and evaluated fronts. The increased MORM size requires more exploration, making the difference in convergence speed between PQL and

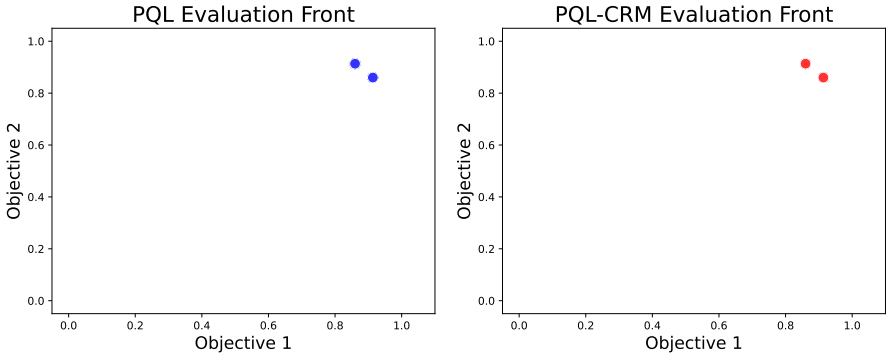


Figure 15: Evaluated fronts for bi-objective once-once.

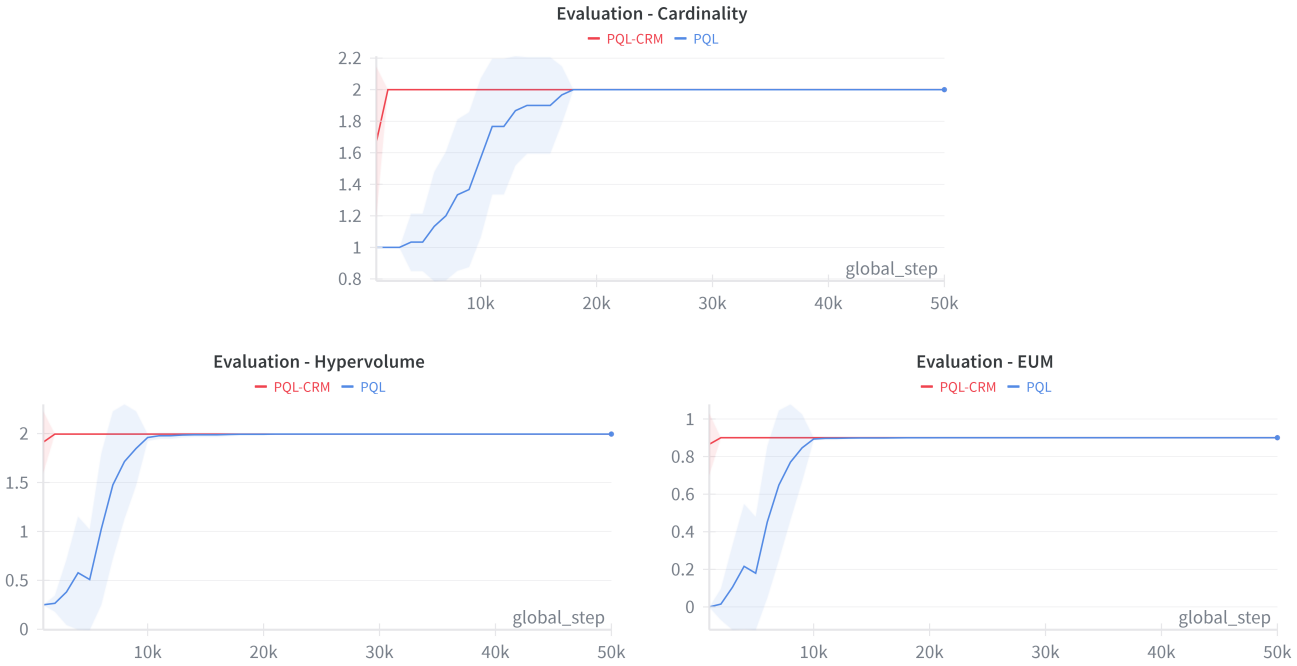


Figure 16: Quality metrics for bi-objective once-once.

PQL-CRM more pronounced. Because the two optimal points are close together, the hypervolume and expected utility metrics show limited improvement after the first solution is discovered, which can give the impression of faster convergence than in practice.

B.3 Exiting Cycle (`term-cycle`)

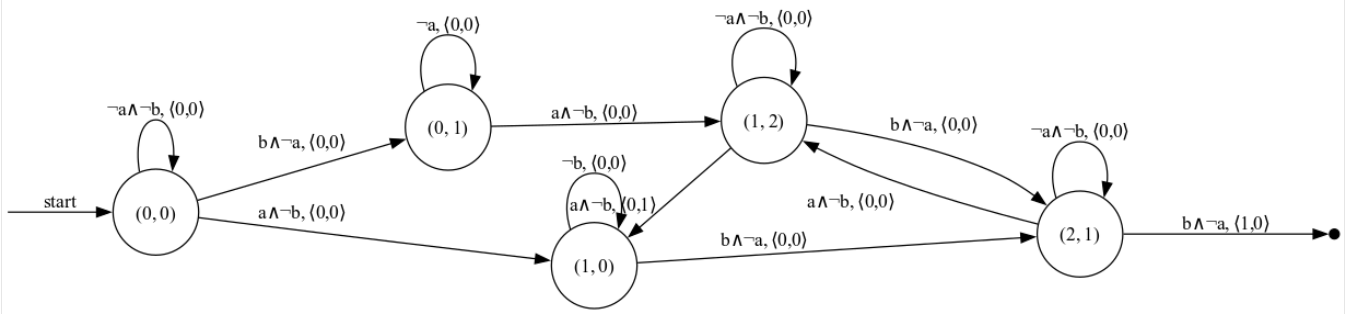


Figure 17: The MORM for bi-objective `term-cycle`.

B.4 Interrupting Cycle (once-cycle)

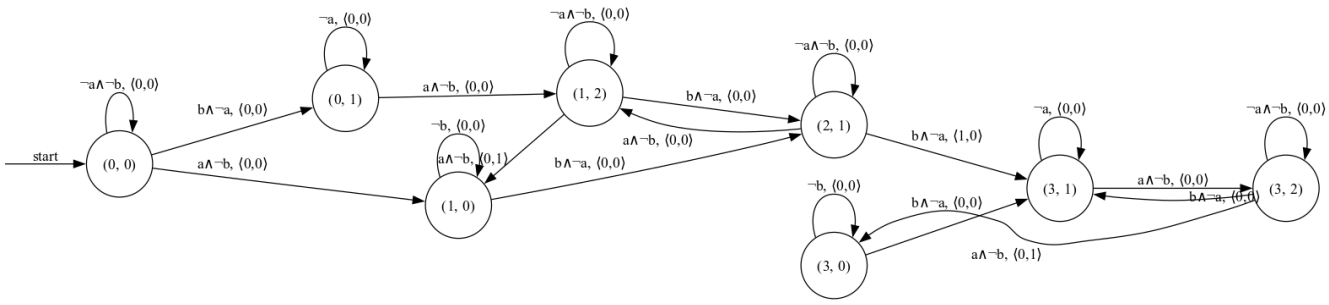


Figure 18: The MORM for bi-objective once-cycle.

B.5 Interleaving Cycles (cycle-cycle)

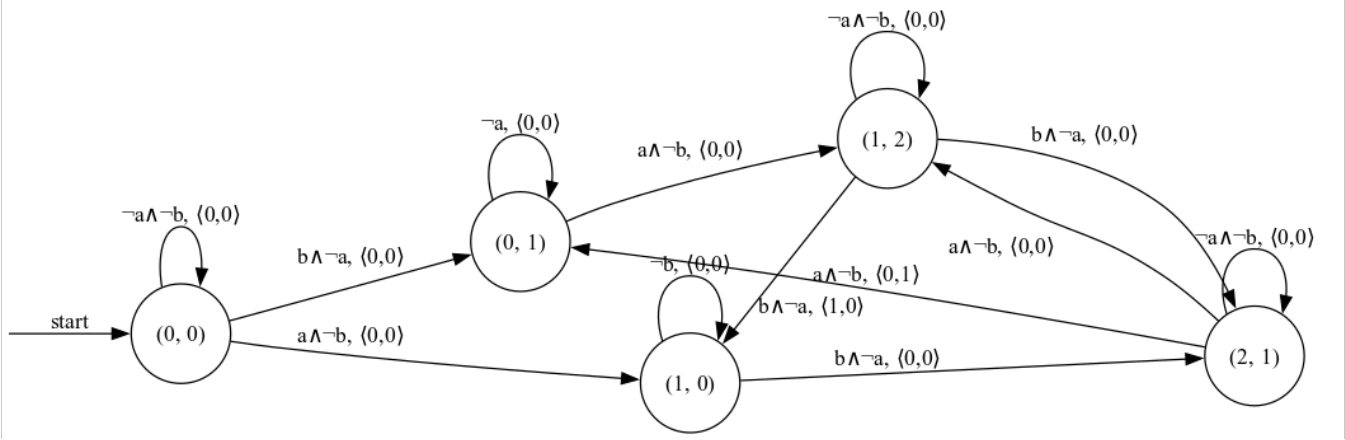


Figure 19: The MORM for bi-objective cycle-cycle.

B.6 Runtime Comparison

Runtime comparison between PQL and PQL-CRM across all multi-objective tasks is shown in Table 1. Experiments were run using a single core on an Apple M1 Pro-chip (14-inch, 2021), 16 GB memory. PQL-CRM requires substantially more computation time than baseline PQL, and this gap would be even larger if we considered only training time, since evaluation does not involve generating counterfactual experiences. This overhead is expected because at every environment interaction PQL-CRM generates counterfactual experiences for all MORM states and performs many more q-set updates per timestep, whereas baseline PQL performs only one. In practice, this additional cost can be mitigated through parallelisation, as counterfactual experiences and their updates are independent and well suited for concurrent execution. More importantly, the advantages of PQL-CRM extend beyond improved sample efficiency. By providing structured and systematic coverage of the augmented state space, PQL-CRM can succeed in settings where unguided exploration fails entirely, as demonstrated in one of our experiments. Thus, although PQL-CRM is computationally heavier, its ability to reduce reward sparsity and ensure reliable exploration can make it essential rather than merely beneficial.

	term-term	once-once	term-cycle	once-cycle	cycle-cycle
PQL	0.5 min	0.6 min	2.0 min	1.0 min	3.0 min
PQL-CRM	0.8 min	1.2 min	4.3 min	4.6 min	7.5 min

Table 1: Average runtime comparison between PQL and PQL-CRM across independent runs per bi-objective task.