Design Principles for Tabular Multi-Policy MORL in Infinite Horizons

Marcelo d'Almeida, Daniel Mossé

Keywords: multi-policy, multi objective reinforcement learning, infinite-horizon.

Summary

Multi-objective reinforcement learning (MORL) addresses problems with multiple, often conflicting goals by seeking a set of trade-off policies rather than a single solution. Existing tabular solutions are limited to episodic problems and are incompatible with infinite-horizon settings. In this work, we address this gap by deriving a set of design principles for tabular multi-policy MORL, where the proposed framework supports both stationary and non-stationary policies, prevents spurious domination, and incorporates cycle detection for robust return guarantees. Through ablation studies, we show how each principle contributes to discovering diverse and reliable policies that map the entire Pareto front.

Contribution(s)

1. A novel tabular trajectory-based framework to track and execute policies in the infinite-horizon setting, supporting robust policy selection and trade-off analysis with undiscounted returns.
   Context: Existing tabular multi-policy MORL methods are restricted to episodic settings and fail to converge in the presence of positive cycles or unbounded returns. Furthermore, standard RL techniques like discounting introduce a recency bias that results in a different set of distorted trade-offs.
2. Identification and resolution of spurious domination from differences in reward horizon, partial estimate convergence, and incomplete structural information.
   Context: In environments with cycles or varying trajectory lengths, current tabular methods are vulnerable to spurious dominations where valid policies are prematurely discarded. This occurs because existing methods fail to account for differing reward horizons, do not decouple reward estimation from return convergence, and underestimate the infinite cumulative return potential of cyclic paths due to undetected cycles.
3. Effective mechanisms to manage stationary policies and cycle detection, supporting reliable learning and retrieval of policies.
   Context: Stationary policies are critical for domains requiring predictable behavior, but learning them requires the explicit formation and tracking of cycles. Naive trajectory tracking mechanisms fail, given the ambiguity of whether a transition closes a cycle or merely extends a longer path.

Design Principles for Tabular Multi-Policy MORL in Infinite Horizons

Marcelo d'Almeida[1], **Daniel Mossé**[1]
{marcelo,mosse}@pitt.edu

[1]**Department of Computer Science, University of Pittsburgh, USA**

Abstract

Multi-objective reinforcement learning (MORL) addresses problems with multiple, often conflicting goals by seeking a set of trade-off policies rather than a single solution. Existing tabular solutions are limited to episodic problems and are incompatible with infinite-horizon settings. In this work, we address this gap by deriving a set of design principles for tabular multi-policy MORL, where the proposed framework supports both stationary and non-stationary policies, prevents spurious domination, and incorporates cycle detection for robust return guarantees. Through ablation studies, we show how each principle contributes to discovering diverse and reliable policies that map the entire Pareto front.

1 Introduction

Autonomous agents must often balance multiple, conflicting objectives over long horizons, where the optimal policy depends on the user's preference among the available trade-offs. In the simplest case, where the preference is known a priori, the problem can be reduced to a single-objective formulation, e.g., by representing user preference as a weighted combination of rewards (Roijers et al., 2013). However, when preferences are unknown beforehand or defining such weights is challenging, the agent can instead learn (or approximate) the set of *non-dominated solutions* — optimal policies for which no objective can be improved without degrading another — also known as the Pareto front (Hayes et al., 2021).

Existing multi-objective reinforcement learning (MORL) approaches for learning multiple policies fall into two categories: outer-loop methods, which iteratively derive new policies from previously learned ones (Roijers et al., 2015; Parisi et al., 2017; Röpke et al., 2024), and inner-loop methods, which learn policies simultaneously (Van Moffaert & Nowé, 2014; Ruiz-Montiel et al., 2017; Reymond & Nowé, 2019; Reymond et al., 2022). By sharing experience across all policies in a single run, inner-loop methods offer significant advantages in sample efficiency. However, existing tabular implementations of such methods (Van Moffaert & Nowé, 2014; Ruiz-Montiel et al., 2017) are restricted to acyclic episodic settings and are incompatible with the infinite-horizon setting. Learning the tabular approach can serve as a principled avenue for novel deep learning extensions.

In this paper, we address this gap by introducing a series of design principles: **(i) enabling multi-policy tracking** within a shared representation (Section 3.2); **(ii) supporting both stationary and non-stationary policies** (to ensure predictability and maximize solution coverage, respectively) (Section 3.3); **(iii) avoiding spurious dominations** to prevent policies from being prematurely discarded (Section 3.4); **(iv) computing undiscounted returns** to facilitate informed policy selection by the user (Section 3.5); **(v) identifying stationary trajectories** to ensure their complete representation (Section 3.6); and **(vi) detecting cycles** to ensure stationarity and guarantee well-defined

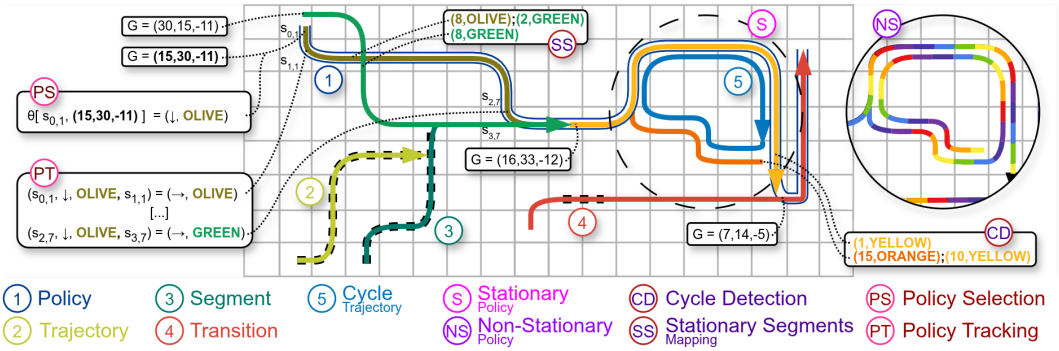


Figure 1: **Overview of components supporting our analysis.** A uniquely-colored segment forest: a policy (1) unfolds as a sequence of trajectories (2), each decomposed into segments (3) and transitions (4). Trajectories may form cycles (5), and policies can be stationary (S) or non-stationary (NS). We highlight cycle detection (CD), Stationary Segments Mapping (SS), policy selection (PS), and policy tracking (PT).

returns over infinite horizons (Section 3.7). Together, these elements provide the basis for a systematic analysis of tabular multi-policy MORL in infinite-horizon settings (Section 4), establishing a principled foundation for future extensions in deep RL (Section 5).

Our contributions include: **(i)** A novel tabular trajectory-based framework to track and execute policies in the infinite-horizon setting, supporting robust policy selection and trade-off analysis with undiscounted returns. **(ii)** Identification and resolution of spurious domination from differences in reward horizon, partial estimate convergence, and incomplete structural information. **(iii)** Effective mechanisms to manage stationary policies and cycle detection, supporting reliable learning and retrieval of policies.

2 Related Work on Tabular MORL

MORL seeks to optimize several, typically conflicting, objectives, with preferences often unknown beforehand. The goal is to find a set of non-dominated policies, where no objective can be improved without sacrificing another. Such policies represent trade-offs subject to preference rather than an objectively best solution, forming what is known as the Pareto front (PF) (Hayes et al., 2021).

Multi-objective problems can be converted into single-objective ones through *scalarization*; however, choosing weights is difficult, and even exhaustive searches often miss parts of the PF where non-isomorphic weight–objective mappings cause similar weights to yield very different trade-offs (Das & Dennis, 1997; Van Moffaert & Nowé, 2014).

Multi-policy methods approximate the PF directly, allowing users to analyze trade-offs without predefined preference weights. Outer-loop methods iteratively derive new policies from those learned in previous runs (Roijers et al., 2015; Parisi et al., 2017; Röpke et al., 2024) while inner-loop methods learn the entire policy set simultaneously in a single run (Van Moffaert & Nowé, 2014; Ruiz-Montiel et al., 2017; Mandow & de-la Cruz, 2018; Reymond & Nowé, 2019; Reymond et al., 2022; 2024).

In tabular approaches, to keep track of policies, Pareto Q-learning (PQL) (Van Moffaert & Nowé, 2014) decomposes the return into immediate and future reward components, deducting the observed reward from the target return to find the matching next target return and corresponding action. Multi-Pareto Q-learning (MPQ-learning) (Ruiz-Montiel et al., 2017), on the other hand, maps trajectories by linking state-action return estimates through indices. These methods are limited to episodic settings without positive cycles, being incompatible with the infinite-horizon setting. In this paper, we extend inner-loop tabular methods to infinite-horizon settings, laying a principled foundation for novel deep RL extensions. For a broader MORL survey, see Hayes et al. (2021).

3 Methodology

Existing inner-loop multi-policy tabular methods (Van Moffaert & Nowé, 2014; Ruiz-Montiel et al., 2017) are incompatible with infinite-horizon settings because they fail to converge in the presence of positive cycles or unbounded returns. While episodic settings are length-bounded, the existence of positive cycles can still produce divergent cumulative returns, creating conditions that mirror the infinite-horizon case. In practice, removing state terminations from such scenarios effectively transforms them into infinite-horizon tasks.

To address the incompatibility with infinite horizons, we introduce a trajectory-based multi-policy MORL framework based on TD updates. Figure 1 shows a summary of components, including tracking policies ①, trajectories ② with *colored* transitions ④ (Section 3.2) and structures them into stationary Ⓢ or non-stationary ⓃⓈ trajectories (Section 3.3). Our method detects what we call *spurious dominations* (Section 3.4) and yields policies with well-defined undiscounted returns for robust policy selection ⓅⓈ (Section 3.5). It further provides explicit tracking of stationary segments ③ ⓈⓈ (Section 3.6) and cycle detection ⑤ ⓒⓓ (Section 3.7), ensuring stationarity. These ideas lay the groundwork for our subsequent discussion of multi-policy deep RL methods.

3.1 Preliminaries: MOMDP and Pareto Optimality

We consider a Multi-Objective Markov Decision Process (MOMDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, \mathbf{R})$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P(s'|s,a) \in [0,1]$ represents the transition probability distribution, and $\mathbf{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$ is a vector-valued reward function representing m objectives.

Unlike single-objective RL, where the goal is to find a unique optimal policy, multi-policy MORL seeks to approximate the Pareto Front (PF), the set of policies representing optimal trade-offs between objectives. To formally define optimality, we use the Pareto dominance relation (Van Moffaert & Nowé, 2014). A return vector $\mathbf{u}$ is said to dominate another vector $\mathbf{v}$, denoted as $\mathbf{u} \succ \mathbf{v}$, if:

$$\forall i \in \{1, \dots, m\}, u_i \geq v_i \text{ and } \exists j \in \{1, \dots, m\}, u_j > v_j.$$

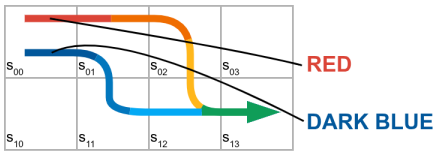
That is, $\mathbf{u}$ dominates $\mathbf{v}$ if $\mathbf{u}$ is at least as good as $\mathbf{v}$ in all objectives and strictly better in at least one. A policy π is non-dominated if there exists no policy π' such that the expected return $\mathbf{G}^{\pi'} \succ \mathbf{G}^{\pi}$.

3.2 Keeping Track of Multiple Policies

In MORL problems with unknown preferences, the optimal solution comprises a set of non-dominated policies that approximate the Pareto front. To successfully implement the user's preference over the trade-offs of the different objectives, the agent must consistently follow the corresponding policy over time. The challenge lies in keeping track of multiple policies during learning. That is, without an explicit preference signal or conditioning mechanism, state information alone is frequently inadequate to differentiate optimal policies that yield overlapping trajectories.

For example, in Figure 2, the state-action pair $(s_{00}, \rightarrow)$ initiates trajectories for two distinct non-dominated returns, $(4,1)$ and $(1,4)$. If the agent targets $(4,1)$, reaching s_{01} creates a conflict: the agent cannot distinguish which subsequent action ($\rightarrow$ or $\downarrow$) preserves the policy associated with that return. As a result, the agent may inadvertently deviate from the intended policy, potentially resulting in arbitrarily sub-optimal outcomes (Van Moffaert & Nowé, 2014).

To address this ambiguity, prior methods introduce mechanisms to track the multiple policies, like PQL (Van Moffaert & Nowé, 2014), which adopts a return-to-go approach by deducting observed rewards from target returns, and MPQ-learning (Ruiz-Montiel et al., 2017), which represents trajectories through linked indices. However, these methods are restricted to episodic settings and fail to converge in the presence of cycles or unbounded returns. We generalize the MPQ-learning formulation into a framework that robustly handles cycles and admits both stationary and non-stationary



(s,a,c)	↦ G
(s_{00},→,RED)	↦ (4, 1)
(s_{00},→,DARK BLUE)	↦ (1, 4)

extended Q-table

(s,a,c,s')	↦ (a',c')
(s_{00},→,RED,s_{01})	↦ (→,ORANGE)
(s_{01},→,ORANGE,s_{02})	↦ (↓,YELLOW)
(s_{01},↓,YELLOW,s_{02})	↦ (→,GREEN)
(s_{00},→,DARK BLUE,s_{01})	↦ (↓,BLUE)
(s_{01},↓,BLUE,s_{11})	↦ (→,LIGHT BLUE)
(s_{01},→,LIGHT BLUE,s_{11})	↦ (→,GREEN)

policy tracking table

Figure 2: **Color-based Trajectory Disambiguation.** The extended Q-table maps colored state-actions to returns, while the policy tracking table (PTT) resolves next action (a', c') given (s, a, c, s').

policies, providing a principled foundation for infinite-horizon MORL. While we build upon the index-linking logic of MPQ-learning, our framework remains conceptually compatible with the return-to-go formulation of PQL. To enforce consistent tracking, we augment the Q-table with a discrete identifier — referred to as *color* — and introduce a *policy tracking table* (PTT) to systematically map local transitions $\delta = (s, a, s')$ to their corresponding trajectories τ.

The primary function of the color is to disambiguate trajectories that *overlap* (i.e., share the same state-action pair (s, a)). For example, in Figure 2, $(s_{00}, \rightarrow)$ is differentiated by the colors *RED* and *DARK BLUE*. With the color, we extend the Q-table, with Q-function $Q(s, a, c)$ mapping state s, action a, and color c to return G (e.g., $(s_{00}, \rightarrow, \textit{RED}) \mapsto (4, 1)$). To record trajectory continuity, we use the PTT to map the *colored transition* $\delta^c = (s, a, c, s')$ to the next *colored action* (a', c'). In our example, $(s_{00}, \rightarrow, \textit{RED}, s_{01})$ leads to $(\rightarrow, \textit{ORANGE})$, and $(s_{01}, \rightarrow, \textit{ORANGE}, s_{02})$ points to $(\downarrow, \textit{YELLOW})$, continuing the sequence.

During training, value estimates are constructed via TD updates, propagating information backwards from future states. For example, in Figure 2, the agent learns the return $(4, 1)$ by transitioning $s_{00} \rightarrow s_{01}$ and updating $Q(s_{00}, \rightarrow, \textit{RED})$ using the reward $r(s_{00}, \rightarrow)$ and the subsequent return from $(s_{01}, \rightarrow, \textit{ORANGE})$, with the PTT updated accordingly. If an update renders a trajectory dominated (e.g., τ_1 dominates τ_2 at (s, a)), the corresponding Q-table and PTT entries for τ_2 are deleted, splitting the trajectory. This pruning process effectively fragments the path, preserving the downstream segment as a valid trajectory while leaving the upstream segment to be re-evaluated. As learning proceeds, trajectories branch and extend across the state space, where every entry in the Q-table marks the start of a trajectory, capturing the return from that point forward, based on the trajectories tracked by PTT. The complete algorithm is provided in Appendix A.

Post-training, policy execution begins with *policy selection*, where the user specifies a preferred trade-off, expressed as undiscounted returns (see Section 3.5). The initial action a and color c are determined via Equation 1:

$$(a_0, c_0) \sim \Theta\left(\text{ND}\left(\bigcup_{\substack{a \in A \\ c \mid \delta_\tau \in \mathcal{T}_0}} Q(s_0, a, c)\right)\right), \tag{1}$$

where the *selection operator* Θ represents the user's choice applied to the non-dominated set (identified by the *non-dominance operator* ND) from the extended Q-table; A is the action set, and $\mathcal{T}_0$ refers to the set of trajectories $\mathcal{T}(s_0, a)$. To *execute the selected policy*, the agent uses the PTT to

govern the decision process: upon transitioning to next state s' (Equation 2), the agent queries the PTT with (s, a, c, s') to retrieve the subsequent action a' and color c' (Equation 3), which are applied to reach the subsequent state s''. This recursive interaction between environment dynamics and PTT lookups ensures the agent strictly adheres to the intended trajectory τ.

$$s' \sim P(s' \mid s, a) \tag{2}$$

$$(a', c') \sim \pi(a', c' \mid s, a, c, s') \tag{3}$$

3.3 Learning Stationary and Non-Stationary Policies

Most works focus on non-stationary policies, as they can recover the entire Pareto front rather than only its convex part (White, 1982; Roijers et al., 2013; Hayes et al., 2021). Still, stationary policies solve problems requiring predictable behavior, that is, always taking the same action in a given state. In this section, we examine the construction of both stationary and non-stationary policies and how they relate to each other.

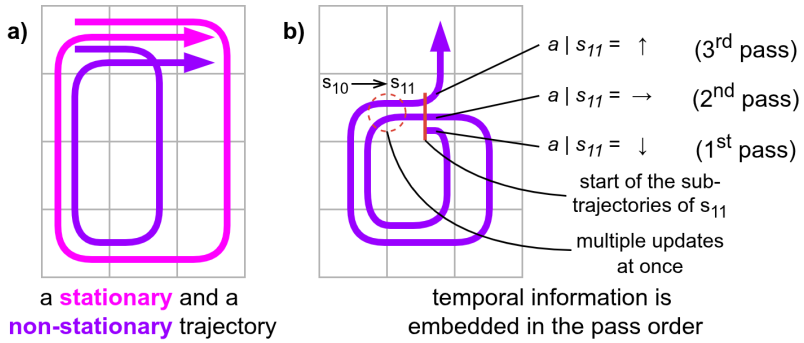


Figure 3: **Trajectory types and temporal encoding: (a)** Stationary trajectory (outer path) vs. non-stationary trajectory (inner path); **(b)** Time is encoded by visitation order.

Stationary policies depend solely on the current state (i.e., $\pi(a|s)$), while non-stationary policies also condition on time (i.e., $\pi(a|s,t)$). Consequently, when re-visiting a state, stationary policies inherently form cycles, producing closed, repeating behavior, whereas non-stationary policies have no such restriction. To emulate cycles, non-stationary policies produce a "spiral" pattern — i.e., explicitly repeating transitions over time, but only for as long as it is advantageous. Thus, topologically, the distinction depends on whether we allow trajectories to form cycles. Allowing cycle formation ensures strict stationarity (outer, pink trajectories), while preventing it admits the broader set of non-stationary policies (inner, purple trajectories), as depicted in Figure 3a.

Starting with the simplest case, tracking non-stationary policies requires only resolving local coloring conflicts. When an overlap in (s, a, c) occurs, a new color is assigned to the extending trajectory so each transition from (s, a) is uniquely identified. See Equation 4, refined from Equation 3. For simplicity, the notation implies color updates at every step (as in Figure 2), though in practice, changes are only needed when a color conflict occurs (i.e., colors may repeat across different state-actions).

$$\pi(a', c' \mid s, a, c, s'), \quad c \neq c', \text{for non-stationary} \tag{4}$$

Crucially, our trajectory-based formulation removes the need to track *time* t or *state history* h; temporal context is encoded in the trajectory structure. For instance, in Figure 3b, state s_{11} is visited three times (t=0, 6, 14), with each occurrence dictating the correct action (down, right, up). Furthermore, this implicit encoding provides a significant advantage in sample efficiency by simultaneously updating return estimates for all trajectories sharing a transition (e.g., from s_{10} to s_{11}).

For stationary policies, we need to enable the formation of cycles. When the trajectory overlaps with itself and is non-dominated, it should form a cycle. A naive approach to track whether different transitions belong to the same trajectory is to assign the same color c to all transitions in a trajectory (e.g., coloring all transitions green in Figure 2). While such an assignment correctly tracks overlaps when one trajectory dominates, it fails when both the forming cycle and the existing trajectory are non-dominated, as the single color cannot disambiguate them. In Sections 3.6 and 3.7, we introduce a more robust approach with a refined coloring scheme and cycle detection.

3.4 Addressing Spurious Dominations

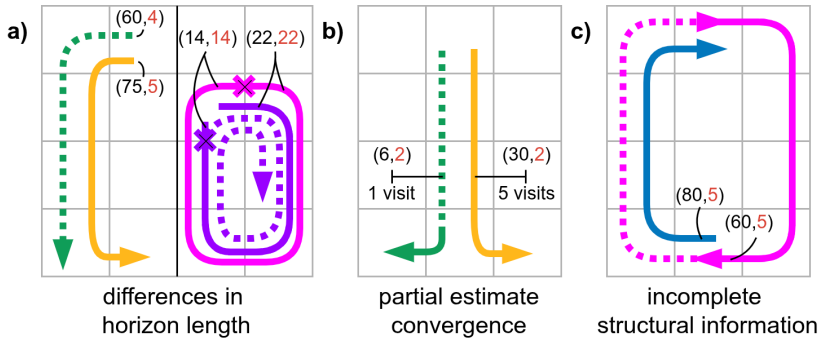


Figure 4: **Sources of Spurious Dominations.** Dotted lines indicate dominated paths; 'x' denotes early termination; tuples: return and length (simplified, length in red). Spurious domination due to **(a)** differences in length of trajectories (left), and cycles and non-stationary trajectories (right); **(b)** partial estimate convergence; **(c)** undetected cycle (outer path).

Spurious (false) dominations arise from three main sources: **(i)** ***differences in horizon length*:** longer trajectories may appear superior simply by accumulating rewards over more steps, even if a shorter trajectory could eventually yield higher returns (Figure 4a, left). This issue is aggravated in environments with positive cycles, where cyclic or non-stationary trajectories can self-dominate their transitions (Figure 4a, right); **(ii)** ***partial estimate convergence*:** frequently visited transitions converge returns rapidly, often spuriously dominating less-sampled transitions that have not yet converged (Figure 4b); and **(iii)** ***incomplete structural information*:** without cycle detection, the agent underestimates the potential for infinite cumulative returns of cyclic paths, allowing finite trajectories to spuriously dominate them (Figure 4c).

To address **(i)**, we augment the objective with a trajectory *length* l component (Equation 5):

$$l(s,a,c) = 1 + \begin{cases} l(s',a',c') & \text{, if } (s',a',c') \in \tau \\ 0 & \text{, otherwise.} \end{cases} \tag{5}$$

This length is added as a return component, acting as a regularizer to enforce fair comparison: in positive-reward environments, adding a negative term associated with the length automatically makes shorter trajectories non-dominated, preserving them until they can be properly compared; similarly, in negative-reward environments, a positive term counteracts compounding penalties, preventing myopic policies, since the case where $l = 1$ minimizes cumulative penalty. We unify both cases via a *normalization strategy*.[1] The regularization also prevents self-domination, as the length preserves cyclic and non-stationary transitions. Post-training, the length component can be discarded during policy selection, leaving only the trade-offs that maximize the original task objectives.

To address **(ii)**, we decouple reward estimation from return estimation. Similar to PQL (Van Moffaert & Nowé, 2014), we maintain an independent estimate of the expected immediate reward $\bar{r}(s,a)$ by

[1] We treat length as a negative reward component and apply a minimum-value shift to enforce a shared sign convention.

normalizing the cumulative reward sum R_Σ by the visit count N (Equation 6):

$$\bar{r}(s,a) = \frac{R_\Sigma(s,a)}{N(s,a)}. \tag{6}$$

Such separation ensures that reward estimates converge independently of the trajectory domination dynamics.

Together, (i) and (ii) already address spurious domination in non-stationary policies. Addressing **(iii)** requires explicitly managing cycle information, which we discuss in Section 3.7.

3.5 Bounding Returns in Infinite Horizon

In infinite-horizon or episodic settings with positive cycles, undiscounted return estimates diverge. Standard RL typically addresses this divergence by applying a *discount factor* or adopting an *average reward formulation*. However, discounting introduces a recency bias that decays the importance of future rewards. As a consequence, the resulting Pareto front could be formed by a different set of solutions with fundamentally different trade-offs. The average reward formulation, on the other hand, would require the agent to directly learn an average reward component for each policy (or trajectory, given policies are not fully defined until learned). Until such averages are stabilized, the learning process would suffer from comparing partial estimates, leading to spurious dominations (as discussed in Section 3.4). Furthermore, average reward formulations typically assume environment ergodicity (i.e., all states are reachable from all states), while our framework makes no such assumption. Finally, for non-stationary policies, the model would have to keep track of moving averages, which destabilizes convergence.

Our method addresses divergence by associating each return with a well-defined length, ensuring local convergence. However, because the horizon remains infinite, trajectories, and thus returns, can still grow without bound. We therefore propose defining a *maximum trajectory length* and decomposing the infinite horizon into a sequence of finite trajectories. This constraint ensures global boundedness while preserving the integrity of the accumulated return.

In practice, the agent normalizes trajectory returns by length, the *length-normalized return* $\bar{G}$ (Equation 7):

$$\bar{G}(s,a,c) = \bar{G}(s',a',c') + \frac{\bar{r}(s,a) - \bar{G}(s',a',c')}{l(s',a',c') + 1}, \tag{7}$$

and the Q-function is equivalently defined as the product of this normalized return and trajectory length l (Equation 8):

$$Q(s,a,c) = G(s,a,c) = \bar{G}(s,a,c) \times l(s,a,c). \tag{8}$$

This normalized formulation enables the comparison of returns across trajectories of different lengths, offering the user a dual interpretation for return: as a reward rate for infinite horizons or as a total return for episodic settings.

Policy execution follows a sequential composition strategy: after completing the selected trajectory, the agent chooses the next one that best aligns the collected return with the desired trade-offs. This strategy also enables interpolated preferences, where the user defines target returns, and the agent selects trajectories that approximate them. Note that performance depends on tuning the maximum length: the horizon must be long enough to capture environment dynamics and provide coverage of length-normalized returns, but capped when extending trajectories yields no further benefit.

3.6 Mapping Trajectories of Stationary Policies

As noted in Section 3.3, the naive coloring scheme is insufficient due to overlaps between non-dominated trajectories. If two trajectories branch from a common path and later overlap (Figure 5

(left), in green), only one retains the color, forcing the other to be pruned. Alternatively, recoloring the conflicting transition (e.g., to yellow) breaks the treatment of potential cycles. As a consequence, the stationary policy is incomplete, missing many valid stationary trajectories.

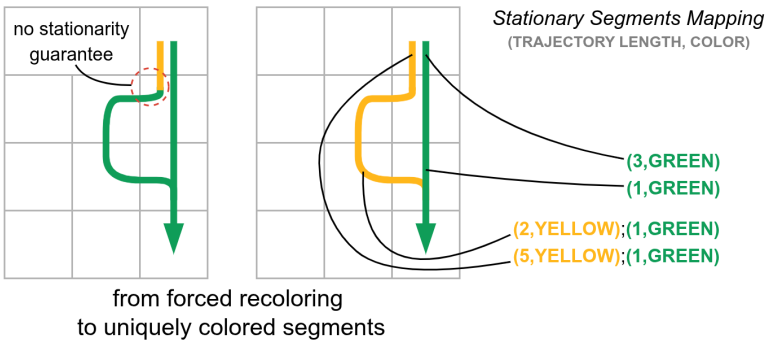


Figure 5: **Managing stationary trajectories.** Dotted lines show dominated paths. **(left)** Stationary trajectory must be recolored; **(right)** We propose uniquely-colored segments and a Stationary Segments Mapping.

To achieve completeness, we propose a segment-based coloring scheme and tracking mechanism. Under this new coloring scheme, we break trajectories into uniquely colored segments $\sigma^{(c)}$, where whenever a trajectory branches into a new path, a unique color is assigned to the newly forming segment. For example, in Figure 5 (right), the trajectory branches from the green segment to start the yellow segment.

To manage these segments, we introduce the *Stationary Segments Mapping (SSM)*, which records the sequence of colored segments and their lengths for each trajectory. In the example, the 'green' trajectory has a single segment *(3, GREEN)*, while 'yellow-green' is defined by *[(5, YELLOW), (1, GREEN)]*, indicating it originates in yellow with $l = 5$ and merges into green with remaining $l = 1$. Accordingly, the policy-tracking rule updates Equation 4 to Equation 9:

$$\pi(a', c' \mid s, a, c, s'), \begin{cases} \text{SCS \& SSM}, & \text{for stationary} \\ c \neq c' & \text{, for non-stationary} \end{cases} \tag{9}$$

where *SCS* is the stationary coloring scheme. To preserve the coloring scheme, no two segments may share the same color. When a trajectory domination occurs, the upstream segment from the split reconnects to the dominant trajectory, and its attributes are updated: return G and length l are recalculated, and a unique color c is assigned to resolve any conflicts.

Finally, to strictly enforce stationarity, we prevent trajectories from becoming non-stationary by explicitly detecting and handling cycles, as detailed in Section 3.7.

3.7 Performing Cycle Detection

A cycle offers the distinct advantage of representing an infinitely defined return while remaining bounded in length. Crucially, as discussed in Section 3.4, failing to explicitly detect cycles renders them vulnerable to spurious domination.

However, detecting cycles is nontrivial due to an inherent ambiguity (illustrated in Figure 6a). When evaluating the TD update from s_{11} to s_{10} (in faded green), it is unclear whether this connection closes a cycle (in orange) or merely extends into a separate, longer trajectory (in blue).

Building on the coloring scheme and tracking mechanism from Section 3.6, we propose a cycle detection mechanism that resolves this ambiguity by distinguishing two scenarios (Figure 6b, indicated by a ✓): **(i)** If the overlap occurs between transitions of the same color (e.g., yellow to yellow), a cycle is unambiguously formed, as we permit only one segment of a given color; **(ii)** If the colors

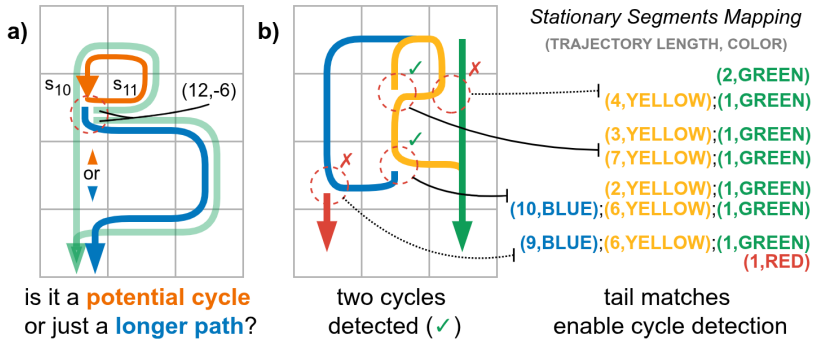


Figure 6: **Performing cycle detection.** Tuples denote return and length (simplified). **(a)** Ambiguity: does the extension close a cycle or a longer path? **(b)** Membership check confirms the cycle.

differ, we determine whether the transitions belong to the same trajectory by consulting the SSM to check whether the shorter trajectory is a suffix of the longer one (e.g., blue to yellow). If not, the paths are independent or have previously diverged, representing distinct trajectories (e.g., red to blue and yellow to green, each marked by a red *X*), and no cycle is formed.

Once a cycle is detected, the agent backpropagates the cycle's return and length information along trajectories that extend from it. This information enriches decision-making by establishing a concrete return guarantee, enabling the policy to specify fully defined trajectories for arbitrarily long horizons or as stable average rewards for infinite horizon settings.

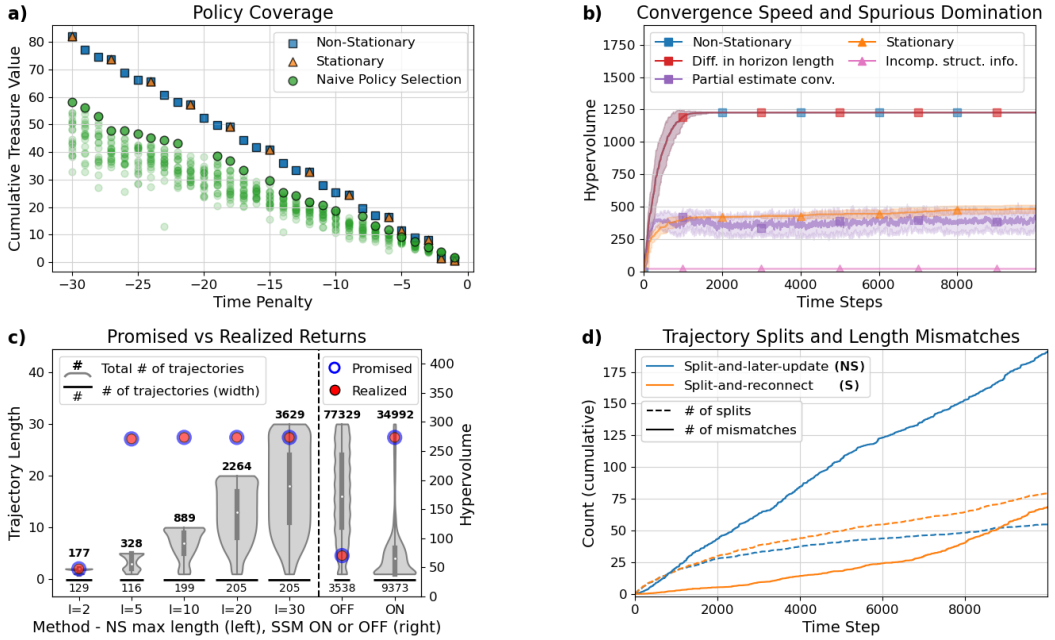


Figure 7: **Ablation study results.** Plots show different aspects: (a) returns for Stationary, Non-Stationary, and naive policies; (b) convergence and spurious domination (shaded area represents standard deviation over 20 seeds); (c) trajectory lengths and promised vs realized returns; (d) trajectory splitting and length mismatches.

4 Comparative Analysis and Ablation Studies

Existing inner-loop multi-policy tabular methods (Van Moffaert & Nowé, 2014; Ruiz-Montiel et al., 2017) do not address the infinite-horizon setting. MPQ-learning (Ruiz-Montiel et al., 2017) is the closest baseline but has structural limitations (see Table 1). Applying it to infinite horizons creates a dilemma: a discount factor renders objectives incomparable (ours are undiscounted), while omitting it causes value divergence. Moreover, previous methods lack safeguards against spurious dominations, making them vulnerable to failures from self-domination and partial estimate convergence.

Since our method is a discrete, exact tabular solution, the goal is to verify the recovery of the Pareto front and to conduct an exhaustive ablation of its core mechanisms. Consequently, we validate our method via targeted ablations (AS1–AS8) and analyze an optimistic approximation of the MPQ-Learning by disabling mechanisms that prevent spurious domination (horizon-length differences (AS3) and partial estimate convergence (AS4)), as shown below in the discussion of Figure 7b.

Table 1: **Structural Comparison:** Applicability to multi-policy MORL. ($\times$ denotes incompatibility, ✓ denotes support).

Capability	**PQL** or **MPQ-learning**	**Ours**
Episodic (w/o (+) cycles)	✓	✓
Episodic (w (+) cycles)	$\times$	✓
Infinite-Horizon	$\times$	✓
Non-Stationary Policies	✓	✓
Stationary Policies	$\times$	✓
Handles Spurious Dominations	$\times$	✓
Performs Cycle Detection	$\times$	✓

Specifically, we analyze key design choices through ablations on the well-known DeepSeaTreasure (DST) problem (Vamplew et al., 2011) (present in MO-Gymnasium (Alegre et al., 2022)), which is the standard benchmark for tabular MORL due to its non-convex Pareto front, on par with the evaluation of prior tabular methods (Van Moffaert & Nowé, 2014; Ruiz-Montiel et al., 2017).

DST is a gridworld that challenges the agent to trade off treasure value against time (choosing between shallow, low-value treasures and deeper, higher-value ones). To adapt DST to an infinite-horizon setting, we remove state terminations and enable cycles via repeated treasure collection. For simplicity, we add a transition from treasure back to the starting state with reward $(0, 0)$, creating a controlled infinite-horizon environment to evaluate stable average reward rates (infinite horizon) and total cumulative reward (finite horizon).

Agents are trained and tested for 10,000 steps, with results averaged over 20 seeds. Each *ablation study* (AS1-AS8) evaluates a specific aspect from Section 3. We present the main results for non-stationary (NS) and stationary (S) policies in Figures 7a–7d, while the full hypothesis, experimental details, metrics, and results are provided in Appendix B.

Figure 7a (AS1, AS2) reports returns for NS, S, and naive policy selection (NPS, i.e., random selection of non-dominated actions). We test whether executed policies match learned target returns (NPS uses the same targets as NS). The results show that NS policies successfully recover the full Pareto front, whereas S policies cover only the convex subset; NPS yields many dominated solutions, since selecting non-dominated actions does not produce a non-dominated policy. While NPS appears to perform well for small time penalties (-1 and -2), such performance is an artifact of stochasticity: NPS deviates from the intended policy and reaches lower-penalty treasures, inflating the average return. In contrast, NS and S agents adhere consistently to their policies.

Figure 7b (AS2–AS5) shows hypervolume[2] convergence for S (triangles) and NS (squares) policies around 1,000 time steps. The figure highlights performance drops caused by spurious domination due to differences in horizon length (red), partial estimate convergence (purple), and incomplete structural information (pink). Notably, disabling trajectory length (AS3) does not degrade performance (red line), as the DST time penalty naturally acts as a trajectory length, balancing the returns. In contrast, partial estimate convergence (AS4) introduces mild instability and significantly lower performance (purple line), and disabling cycle detection in S (AS5) removes critical structural information, severely limiting performance (pink line, near-zero hypervolume).

As MPQ-learning (Ruiz-Montiel et al., 2017) learns non-stationary policies without the safeguards tested in AS3 and AS4, we can define a "best-case" proxy that approximates its theoretical behavior. As in DST, the time penalty mitigates spurious domination, the proxy corresponds to AS4 (purple line in Figure 7b), which still performs significantly worse than the proposed method (blue line).

Figure 7c (AS6, AS7) examines sensitivity to maximum length (l) in NS policies and the impact of SSM in S policies. The violin plots display trajectory length distributions, comparing *promised* returns (presented to the user) against *realized* returns (after 1,000 steps). A match between promised and realized returns (right Y-axis) indicates sustained average rewards over long horizons. In DST, the return $(8.2, -3)$ yields the only optimal average reward; once learned, the hypervolume reaches its maximum, explaining the stabilization for $l \geq 5$. The results confirm that SSM is crucial for stationary policies: without it, cycle detection is disabled, severely limiting long-horizon performance. The figure also shows that SSM causes trajectories to be shorter, as cycle detection halts trajectory growth once a cycle is formed.

Figure 7d (AS8) reports splits (dashed) and mismatches (solid). Splits occur when a trajectory is dominated, while mismatches arise when trajectories end prematurely (i.e., the upstream segment of a dominated trajectory has not yet been updated). We compare two update strategies: *Split-and-Later-Update* (NS) and *Split-and-Reconnect* (S). Since splits are comparable across strategies, the fewer mismatches for Split-and-Reconnect (orange) indicate higher sample efficiency by avoiding updates based on outdated trajectories.

5 Future Work: Extending Design Principles to Deep RL

This work extends tabular multi-policy methods to infinite-horizon settings Extending these principles, which are exact and discrete, to deep RL, where solutions are approximate and continuous, remains an open challenge. Below, we outline potential pathways for this translation.

One promising avenue is recasting PTT and Q-table within an actor–critic framework. The actor models $\pi(a', c'|s, a, s', c)$, replacing discrete color c with a learned transition embedding. For stationary policies, such embeddings serve as policy embeddings. The critic, in turn, incorporates the embedding into value estimation $Q(s, a, c)$.

Graph Neural Network (GNN) (Wu et al., 2019) could implement the SSM, with each colored segment as a node and connections as edges. Although smaller than encoding all transitions, the graph remains large, exceeding current GNN capacity (Wu et al., 2024); capturing cyclic structures is especially challenging for standard message-passing GNNs (Xu et al., 2018; Alon & Yahav, 2020).

For non-stationary policies, we propose an alternative strategy that drops explicit embeddings with a return-conditioned actor, akin to PCN (Reymond et al., 2022). In this formulation, the critic models non-dominated returns via density estimation with deep generative models (Bond-Taylor et al., 2021), representing the Pareto front as a distribution.

[2]Hypervolume measures the objective space dominated by solutions relative to a reference point. Larger values indicate better *convergence* and *diversity* (i.e., proximity and spread along the PF).

6 Conclusion

This work establishes fundamental design principles for tabular multi-policy MORL from episodic to infinite-horizon settings. Our trajectory-centric framework ensures reliable tracking and execution of both stationary and non-stationary policies, integrates cycle detection, and prevents spurious domination from differences in horizon length, partial estimate convergence, and incomplete structural information. Empirical results confirm that each principle is essential for recovering diverse, interpretable, and stable policies.

References

Lucas N. Alegre, Florian Felten, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, Ana L. C. Bazzan, and Bruno C. da Silva. MO-Gym: A library of multi-objective reinforcement learning environments. In *34th Benelux Conference on Artificial Intelligence BNAIC/Benelearn*, 2022.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *ArXiv*, abs/2006.05205, 2020.

Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:7327–7347, 2021. URL https://api.semanticscholar.org/CorpusID:232147810.

Indraneel Das and John E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, 1997.

Conor F. Hayes, Roxana Ruadulescu, Eugenio Bargiacchi, Johan Kallstrom, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai Aravazhi Irissappane, Patrick Mannion, Ann Nowé, Gabriel de Oliveira Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 2021.

Lawrence Mandow and José-Luis Pérez de-la Cruz. Pruning dominated policies in multiobjective pareto q-learning. In *Conferencia de la Asociación Española para la Inteligencia Artificial*, 2018.

Simone Parisi, Matteo Pirotta, and Jan Peters. Manifold-based multi-objective policy search with sample reuse. *Neurocomputing*, 2017.

Mathieu Reymond and Ann Nowé. Pareto-dqn: Approximating the pareto front in complex multi-objective decision problems. In *Adaptive and learning agents workshop at International Conference on Autonomous Agents and Multi-Agent Systems*, 2019.

Mathieu Reymond, Eugenio Bargiacchi, and Ann Nowé. Pareto conditioned networks. In *Adaptive Agents and Multi-Agent Systems*, 2022.

Mathieu Reymond, Conor F Hayes, Lander Willem, Roxana Rădulescu, Steven Abrams, Diederik M Roijers, Enda Howley, Patrick Mannion, Niel Hens, Ann Nowé, et al. Exploring the pareto front of multi-objective covid-19 mitigation policies using reinforcement learning. *Expert Systems with Applications*, 2024.

Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 2013.

Diederik Marijn Roijers, Shimon Whiteson, and Frans A Oliehoek. Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, 2015.

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

Manuela Ruiz-Montiel, Lawrence Mandow, and José-Luis Pérez de-la Cruz. A temporal difference method for multi-objective reinforcement learning. *Neurocomputing*, 2017.

Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 2011.

Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 2014.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Douglas John White. Multi-objective infinite-horizon discounted markov decision processes. *Journal of Mathematical Analysis and Applications*, 1982.

Meng Wu, Mingyu Yan, Wenming Li, Xiaochun Ye, Dongrui Fan, and Yuan Xie. Survey on characterizing and understanding gnns from a computer architecture perspective. *IEEE Transactions on Parallel and Distributed Systems*, 36:537–552, 2024.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32:4–24, 2019. URL https://api.semanticscholar.org/CorpusID:57375753.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ArXiv*, abs/1810.00826, 2018.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Algorithmic details and analysis

A standard single-policy tabular RL method like Q-learning (Watkins & Dayan, 1992), in the infinite-horizon setting, follows the structure in Algorithm 1. This algorithm maintains a Q-table of state–action values, updated incrementally as the agent interacts with the environment. At each step, the agent selects an action via an exploration policy (e.g., ϵ-greedy), observes the reward and next state, and applies the TD update on the Q-table. As usual, the discount factor γ weights future rewards, while the learning rate α controls how quickly new information overrides old estimates. Repeated interactions refine the Q-values, which can then define a greedy policy. In Sections A.1 and A.2, the *Q-table update* is replaced by our non-stationary and stationary *Multi-Policy Update*, respectively, and Section A.3 discusses their time and space complexity.

Algorithm 1 Tabular RL for Infinite-Horizon

Require: State space $\mathcal{S}$, action space $\mathcal{A}$, learning rate α, discount factor γ, exploration policy π_ϵ, maximum steps T

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
for $t = 1$ **to** T **do**
  Observe current state s_t
  Select action a_t according to $\pi_\epsilon(Q, s_t)$
  Execute a_t, observe reward r_t and next state s_{t+1}
  Update Q-table:
  $$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \Big]$$
  $s_t \leftarrow s_{t+1}$
end for
return $Q(s, a)$

A.1 Non-Stationary

Algorithm 2 outlines the update procedure for the non-stationary policies. The method extends standard TD updates to maintain multiple policies simultaneously via the Q-table and a policy tracking table — PTT (from Section 3.2).

Step 1. Retrieve from Q-table the non-dominated set at the next state s', consisting of tuples (s', a', c') whose Q-values are non-dominated.

Step 2. If the set is empty, then s' is being visited for the first time and $Q(s', a', c')$ has no entries. In this case, add a new Q-value estimate for (s, a, c) from reward r with trajectory length $l = 1$.

Step 3. Fetch all PTT entries (s, a, c, s', a', c') associated with the current transition (s, a, s'). Delete PTT entries whose successors (s', a', c') are no longer in the non-dominated set (obsolete). Their Q-value estimates (i.e., (s, a, c)) restart with reward r and trajectory length $l = 1$. Add any entries from the non-dominated set to the PPT, associating them with the current (s, a) and generating a new color c for each association (s, a, c, s', a', c'), thereby extending trajectories from s' to s with distinct colors c_i and adding 1 (one) to the trajectory length.

Step 4. Use each updated PTT entry to update the Q-table according to Equations 7 and 8.

Step 5. Prune dominated entries from (s, a) in the Q-table and then in the PTT, preserving only Pareto-optimal (s, a, c) entries for future updates.

Algorithm 2 Policy Update: Non-Stationary

Require: Current state s, action a, next state s', Q-table, policy tracking table (PTT)

Step 1: Fetch ND entries
next_state_nd_set $\leftarrow$ fetch_nd_set(Q, s') ▷ (s', a', c') entries
Step 2: Initialize Q-table if no ND entries at s'
if next_state_nd_set is empty **then**
    $c \leftarrow$ generate_new_color()
    ADD entry (s, a, c) *to Q-table*
end if
Step 3: Fetch PTT entries for transition (s, a, s'), add new and discard old transitions
ptt_entry_set $\leftarrow$ fetch_ptt_entries(PTT, s, a, s') ▷ (s, a, c, s', a', c') entries
ptt_old_entries $\leftarrow$ ptt_entries_set $-$ next_state_nd_set ▷ Compare (s', a', c'), keep PTT entry
for (s, a, c, s', a', c') in ptt_old_entries **do**
    DELETE entry (s, a, c, s', a', c') *from PPT*
end for(s, a, c, s', a', c')
ptt_new_entries $\leftarrow$ next_state_nd_set $-$ ptt_entries_set ▷ Compare (s', a', c'), keep (s', a', c')
for (s', a', c') in ptt_new_entries **do**
    $c \leftarrow$ generate_new_color()
    ADD entry (s, a, c, s', a', c') *to PTT*
end for
Step 4: Update Q-table for each PTT entry
for (s, a, c, s', a', c') in ptt_entries **do**
    ADD/UPDATE entry (s, a, c) to Q-table ▷ Update done via Equation 7 and 8
end for
Step 5: Clean up dominated entries
state_action_dominated_set $\leftarrow$ *fetch_dominated_set(Q, s, a)*
for (s, a, c) in state_action_dominated_set **do**
    DELETE entry (s, a, c) *from Q-table*
    DELETE entry from PTT which starts with (s, a, c) *as in* (s, a, c, s', a', c')
end for

A.2 Stationary

Algorithm 3 outlines the update procedure for the stationary policies. In addition to the Q-table and PTT, it incorporates the Stationary Segments Mapping (SSM) from Section 3.6.

Step 1. Retrieve from Q-table the non-dominated set at the next state s', consisting of tuples (s', a', c') whose Q-values are non-dominated.

Step 2. If the set is empty, then s' is being visited for the first time and $Q(s', a', c')$ has no entries. In this case, add a new Q-value estimate for (s, a, c) from reward r with trajectory length $l = 1$.

Step 3. Fetch all PTT entries (s, a, c, s', a', c') associated with the current transition (s, a, s'). **Unlike the non-stationary case,** no entries are obsolete: dominated trajectories are reconnected. Add non-dominated successors from Step 1 to the PTT, linking them to (s, a) with the appropriate color c. Assign a new color if (s, a, c) is already reached by another entry; otherwise, reuse the existing color, thereby extending trajectories from s' to s with trajectory length $l = l' + 1$.

Step 4. Update the Q-table using each revised PTT entry, according to Equations 7 and 8.

Step 5. Check for cycles. If one is found, duplicate its transitions and assign them a new shared color; otherwise, update the SSM with the new PTT entry.

Step 6. Prune dominated entries from (s, a) in the Q-table. Reconnect all prior associations from pruned entries to their dominant counterparts in the PTT, updating their Q-Values, and generating new colors as in Step 3. This preserves only Pareto-optimal (s, a, c) entries for future updates.

A.3 Algorithmic Complexity

We summarize the worst-case computational complexity of the stationary and non-stationary policy updates.

The **non-stationary update** is dominated by computing the non-dominated set for the next state (Step 1 in Algorithm 2). Let M denote the number of objectives and $N^{ns}_{s'}$ the total number of Q-value estimates for all actions a' in the next state s', found in *non-stationary policies*. The worst-case complexity is $O(MN^{ns\,2}_{s'}) \sim O(N^{ns\,2}_{s'})$, as the number of objectives is constant. This arises from pairwise comparisons of all Q-value vectors in s' across objectives.

In addition to doing the same work as the non-stationary update, the **stationary update** also iterates over stationary segments (from SSM) when checking for cycles (Step 5 in Algorithm 3) and over outdated segments to update and reconnect them (Step 6 in Algorithm 3). Let S denote the number of stationary segments, $N^{s}_{s'}$ the total number of Q-value estimates for all actions a' in the next state s', found in *stationary* policies, N_{sa} the number of Q-value estimates for the current state-action pair (s, a) used for cycle detection, E_D and E_d the numbers of dominant and dominated entries, respectively, and Ld the dominated trajectory length. The worst-case complexity is $O(MN^{s\,2}_{s'} + SN_{sa}N^{s}_{s'} + L_dE_dE_D) \sim O(N^{s\,2}_{s'} + SN_{sa}N_{s'} + L_dE_dE_D)$, where the first term corresponds to computing the non-dominated set, the second to iterating over stationary segments for cycle detection, and the third to updating dominated segments. Here, $N^{s\,2}_{s'} \geq N_{sa}N^{s}_{s'}$ since N_{sa} counts only the estimates for the current action. In the worst case, S can be as large as the trajectory length of the smaller trajectory in each pair comparison. Since stationary policies form a subset of non-stationary policies, we have $N^{s}_{s'} \leq N^{ns}_{s'}$, which can partially offset the difference in computational complexity.

For both stationary and non-stationary updates, **space complexity** is dominated by the Q-table and PTT. The Q-table stores one entry per non-dominated trajectory for each state-action pair, requiring at most $O(|S||A|N_{\max})$ space, where $N_{\max}$ is the maximum number of non-dominated trajectories per state-action. The PTT stores the transitions leading to these trajectories, with a worst-case bound of $O(|S|^2|A|N_{\max})$, though in practice it is much smaller. Similarly, the SSM stores stationary segments, with space proportional to the number of non-dominated segments. Because the number of stationary segments is smaller than the number of transitions, the SSM is smaller than the PTT.

Algorithm 3 Policy Update: Stationary

Require: Current state s, action a, next state s', Q-table, policy tracking table (PTT), Stationary Segments Mapping (SSM)

Step 1: Fetch ND entries
next_state_nd_set ← fetch_nd_set(Q, s') ▷ (s', a', c') entries
Step 2: Initialize Q-table if no ND entries at s'
if next_state_nd_set is empty **then**
    $c \leftarrow$ generate_new_color()
    ADD entry (s, a, c) *to Q-table*
end if
Step 3: Fetch PTT entries for transition (s, a, s') and add new transitions
ptt_entry_set ← fetch_ptt_entries(PTT, s, a, s') ▷ (s, a, c, s', a', c') entries
ptt_new_entries ← next_state_nd_set − ptt_entries_set ▷ Compare (s', a', c'), keep (s', a', c')
for (s', a', c') in ptt_new_entries **do**
    $c \leftarrow$ fetch_or_generate_color(s', a', c') ▷ Repeat color if first extension, new one otherwise
    ADD entry (s, a, c, s', a', c') *to PTT*
end for
Step 4: Update Q-table for each PTT entry
for (s, a, c, s', a', c') in ptt_entries **do**
    ADD/UPDATE entry (s, a, c) to Q-table ▷ Update done via Equation 7 and 8
end for
Step 5: Check for cycles and update the SSM
for (s, a, c, s', a', c') in ptt_entries **do**
    if cycle is detected in PTT **then**
        Identify all transitions in the cycle
        $c \leftarrow$ generate_new_color()
        for each transition in the cycle **do**
            DUPLICATE the transition
            UPDATE SSM with the new PTT entry
        end for
    else
        UPDATE SSM with the new PTT entry
    end if
end for
Step 6: Clean up dominated entries
dominated_set ← fetch_dominated_set(Q, s, a)
for each d:(s, a, c) in dominated_set **do**
    for each PTT entry ending in (s, a, c), i.e., $(s^d_{prev}, a^d_{prev}, c^d_{prev}, s, a, c)$ **do**
        $c_{\text{new}} \leftarrow$ fetch_or_generate_color() ▷ As in Step 3
        RECONNECT previous links with color c and update Q-table and PTT with c_{new}
    end for
    DELETE Q-table entry (s, a, c)
    DELETE PTT entries starting from (s, a, c)*, i.e.,* (s, a, c, s', a', c')
end for

Regarding the color space $|C|$, for non-stationary policies, the color only needs to be unique within each state–action pair, since the Q-table is already indexed by (s, a) and the PTT only links (s, a, c, s') to (a', c'). Thus, the space of possible colors is bounded by the maximum number of visits to each state–action pair. In contrast, for stationary policies, the color must be unique per policy. By tracking how many transitions each color participates in, we can upper-bound the number of required colors by $|S|$ (the state space), since in the worst case, each state may become the root of a tree of converging trajectories. In practice, this bound is much smaller because colors propagate along trajectories. In the best case, the bound is $|P|$ (the policy space), one color per policy.

B Complete Ablation Plan

Table 2 details the comprehensive ablation study. Each row identifies a specific ablation instance and consolidates the corresponding hypothesis, experimental design, evaluation metrics, and results. The main results and associated figures are presented in the main text.

C Use of large language models (LLMs)

We used Large Language Models (LLMs) to support writing and research. Specifically, they were employed to polish and refine phrasing for clarity and conciseness, to assist in exploring alternative framings of ideas (e.g., refining how concepts and contributions are presented, not generating new research directions), to complement traditional tools (e.g., Google Scholar) in identifying related work, and to provide limited coding support, such as boilerplate code for plotting or debugging assistance. The models did not contribute at the level of a coauthor, and all research design, implementation, analysis, and conclusions are our own.

Table 2: **Ablation plan.** Each row tests a design choice with metrics and expected outcomes.

Id	Ablation	Hypothesis / Experiment / Metrics / Results
AS1	Policy tracking *vs.* state-wise ND selection	Lack of trajectory-level tracking causes agents to follow dominated policies. **Experiment:** NPS (i.e., random non-dominated actions) vs. S and NS policies. **Metrics:** Policy coverage. **Results:** NPS gets less treasure value because it combines state-wise ND actions into trajectories that often yield dominated solutions (Figure 7a).
AS2	Learning the full/convex PF with NS/S policies	Allowing NS policies enables full PF recovery; restricting to S policies limits recovery to its convex subset (thus lower hypervolume). **Experiment:** S vs. NS policies. **Metrics:** Policy coverage; convergence speed. **Results:** NS polices recover the full PF, whereas S ones recover only the convex subset (Figure 7a and 7b).
AS3	Differences in horizon length (spurious)	Longer trajectories spuriously dominate others without normalization. **Experiment:** Disable trajectory length. **Metrics:** Hypervolume over time. **Results:** No observable difference, as the environment already incorporates a length-like component, time penalty (Figure 7b).
AS4	Partial estimate convergence (spurious)	Partial estimates cause frequent low-reward policies to appear superior to sparse high-reward ones. **Experiment:** Couple immediate reward and return estimates. **Metrics:** Hypervolume over time. **Results:** Partial estimates destabilize learning and degrade performance (Figure 7b).
AS5	Incomplete structural information (spurious)	Failure to detect cycles biases Pareto comparisons, making cycles appear dominated by high-return trajectories. **Experiment:** S without cycle detection. **Metrics:** Hypervolume over time. **Results:** Failing to detect cycles sharply degrades performance (Figure 7b).
AS6	Maximum trajectory-length sensitivity and policy selection	Maximum trajectory length affects whether policies sustain stable returns. **Experiment:** Vary max length (2, 5, 10, 20, 30). **Metrics:** Hypervolume from promised (max_length steps) vs. realized (1,000 steps). **Results:** Promised and realized returns align, indicating sustained return over following multiple trajectories (Figure 7c).
AS7	Blocking segments and SSM	SSM prevents blocking and coloring conflicts, enabling stationary policies to be recovered correctly. **Experiment:** Disable SSM. **Metrics:** Hypervolume from promised (max_length steps) vs realized (1,000 steps). **Results:** Promised and realized returns not only align but also improve, confirming correct recovery with SSM (Figure 7c).
AS8	Split-and-reconnect segment updates	Applying segment updates improves sample efficiency and accelerates recovery after dominated trajectories. **Experiment:** Disable reconnection. **Metrics:** # of splits; # of mismatches. **Results:** Split-and-reconnect yields substantially fewer trajectory-length mismatches (Figure 7d).

S - Stationary; NS - Non-stationary; NPS - Naive Policy Selection; PF - Pareto front; ND - non dominated.