Adapting the Behavior of Reinforcement Learning Agents to Changing Action Spaces and Reward Functions

Author template
University
City, Country
university-email

***Abstract*—In Reinforcement Learning, agents learn to solve specific tasks through the exploration of the environment in which they execute. Agents' behavior is specialized towards a given goal, and learnt from agent's experience from the execution of its actions. While effective in optimizing the behavior for a specific goal, the performance of agent's rapidly decreases when the environment conditions change. This paper introduces MORPHIN, a self-adaptive Q-learning agent that enables on-the-fly adaptation without full retraining. MORPHIN leverages proactive environment monitoring and concept-drift detection to trigger dynamic adjustment of learning and exploration parameters only when needed, while preserving prior policy knowledge to mitigate catastrophic forgetting. Additionally, MORPHIN supports seamless incorporation of new actions into the agents action space. We validate our approach on a standard RL benchmark with shifting reward functions and the introduction of a new actions. Additionally we validate our approach in a realistic traffic signal control application to manage road intersections. The results demonstrate the proposed approach evidences superior convergence speed, resource efficiency, and continuous adaptation to evolving conditions, with respect to the baseline.**

***Index Terms*—Reinforcement Learning, Continual Reinforcement Learning, Q-learning, Concept Drift Detection, Adaptive Systems, Traffic Signal Control**

I. Introduction

Reinforcement Learning (RL) is a subset of Machine Learning (ML) in which agents learn by interacting with an environment, through trial and error. An agent receives a scalar reward (positive or negative) for each interaction step, based on the outcome of the action taken. The objective of the agent is to maximize its cumulative long-term reward [1]. Traditional RL algorithms, such as Q-Learning, assume stationary Markov Decision Processs (MDPs), where transition probabilities and reward functions are constant over time [2]. However, this assumption restricts RL agents' effectiveness in real-world environments, which often exhibit non-stationary characteristics like evolving state distributions, varying reward dynamics, and changing action spaces [3].

To overcome these limitations, our work adopts a Continual Reinforcement Learning (CRL) paradigm, where agents continuously adapt to environmental changes [4]. Approaches such as meta-learning [5] and transfer learning [6] have shown promise within CRL. Meta-learning allows an agent to *learn how to learn*, enhancing adaptability in new contexts [7]. Transfer learning leverages prior knowledge to accelerate learning in related tasks [8].

This paper introduces MORPHIN, a Q-learning-based RL algorithm that incorporates adaptive mechanisms inspired by CRL to account for environment changes in the goal/rewards definition or in the action space available to an agent. Our work is motivated by the suitability of RL for self-adaptive systems, given its capacity to dynamically adjust behavior in response to environmental feedback [9]. However, real-world scenarios such as traffic control present substantial challenges due to their inherent non-stationary dynamics [2]. We note, the problem of environment changes to the state space has been addressed [10], while adaptations to the goals (*i.e.,* rewards) or action space remain an open question.

Our method effectively addresses non-stationary environments by continuously monitoring and updating agents' learning strategy.

We validate MORPHIN with two applications. One application in the context of RL, using a standard RL benchmark Gridworld, and one application in the context of self-adaptive systems, with a traffic signal control at city intersections. In the first scenario, environment goals are changed back and forth at defined points in the execution to observe the behavior of the environment in learning (or lack there of) new behavior. Additionally, we present a second independent scenario introducing new actions to the agent. In the second scenario we combine the changing goals and introduction of new actions to observe if the agent is effective in detecting both the new goal and action incorporating them into the new learned policy. The use of signal phases continuously influence state transitions and rewards [2]. By demonstrating the effectiveness of MORPHIN under these conditions, we underline its suitability for self-adaptive systems, particularly those relying on streaming data where concept drift may occur, making efficient adaptation essential.

This paper is structured as follows: Section II introduces theoretical foundations relevant to our implementation, along with a motivating case study that illustrates the problem addressed in this work. Section III describes the MORPHIN algorithm in detail. Section IV presents our validation methodology across the implemented experiments, including traffic signal control, and discusses the results. Section V reviews

related work and situates our contribution within the current state of the art. Finally, Section VI summarizes our conclusions and outlines directions for future research.

II. Background

A. Reinforcement Learning

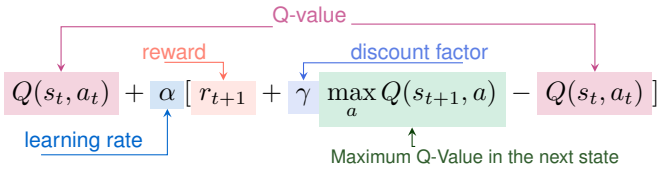
In RL, agents learn to map environmental situations (environment states) to actions that maximize a numerical reward signal received from the environment in the long term [1]. RL problems are formulated as Markov Decision Processes (MDPs), defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where: $\mathcal{S}$ is the state space, comprising all relevant states of the environment; $\mathcal{A}$ is the action space, i.e., the set of all actions the agent can perform to affect the environment; $P(s_{t+1} \mid s_t, a_t)$ is the transition probability from state s_t to s_{t+1} under action a_t; $R(s_t, a_t)$ is the reward function, providing the numerical signal (r) that encodes the positive or negative impact of taking action a_t in state s_t at each execution step t; and $\gamma \in [0, 1]$ is the discount factor, which determines the importance of future rewards.

In stationary settings, P and R remain fixed over time, and the agents objective is to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected discounted return. However, real-world environments often violate this stationarity assumption: transition dynamics and/or reward functions may shift due to evolving conditions. We model such scenarios as *non-stationary* MDPs, represented by a sequence

$$\{\mathcal{M}_t\}_{t=1}^{\infty} \quad \text{with} \quad \mathcal{M}_t = \langle \mathcal{S}, \mathcal{A}_t, P_t, R_t, \gamma \rangle,$$

where P_t and/or R_t change at unknown time steps t ("concept drifts"). In this work, we further consider the novel scenario where the agent's action space $\mathcal{A}$ may also change over time. Detecting and adapting to these drifts is central to CRL [3, 4], which treats learning as an ongoing process across a continually changing sequence of tasks.

Q-learning [11] is a widely used model-free approach in RL. The long-term quality of an action performed at a given state is computed iteratively in a series of steps and is represented by a Q value, $Q(s, a)$. Formally, each execution step t captures information from the environment and maps it to a state $s_t \in \mathcal{S}$ in its state space. It then selects an action $a_t \in \mathcal{A}$ from its action space and executes it. The agent receives a reward r_t from the environment when it moves to the next state $s_{t+1} \in \mathcal{S}$. The reward is used to update the optimality of performing the action a_t at state s_t. The agent's goal is to learn a policy (i.e. the best-fit action for each state) that maximizes the reward of the long-run behavior. The learning rate α determines how much new experiences overwrite previously learned experiences, and the discount factor γ determines how much future rewards are discounted so that agents prioritize immediate actions and can plan the best long-term actions. At each time step t, the Q value of an action a_{t+1} taken in state s_{t+1}, $Q(s_{t+1}, a_{t+1})$, is updated by the Bellman learning equation as follows:



While Q-learning converges under stationary conditions (with appropriate decay of α), it can struggle when P_t or R_t change over time. Our work builds on this foundation by incorporating online concept-drift detection and adaptive updates, enabling the agent to remain effective in non-stationary MDPs. We focus particularly on shifts in the reward function R_t, which may also induce changes in the action space $\mathcal{A}_t$.

B. Motivating example

To motivate the adaptation of agent's behavior to changing goals and the acquisition of new actions, we use two scenarios of the Gridworld benchmark as a running example.

Our Gridworld configuration, consists of a matrix (of 9×9) where every cell is associated with a reward of -1, except for a single goal state that has a reward of $+100$ as shown in the left-hand side of Fig. 1. The agent begins each episode at the grid's center and may move in four directions: `up`, `down`, `left`, or `right`. Initially, the goal is placed in the top-left corner.

In the first scenario, we allow the goal state of the environment to change. Such changes are unknown to the agents beforehand [12] and maybe due to the reallocation of objectives [3] (*e.g.,* reorganizing products in a warehouse for robotic fulfillment systems), or the shift to new objectives [13] (*e.g.,* UAV drones changing their mission due to flight plan changes, or changing in leadership for flight formations).

Additionally, we consider a scenario of adding walls to the Gridworld, blocking the agent's movement according to its initial set of actions (Fig. 5). For such scenario, we enable the agent to extended its action space by acquiring new actions that provide additional capabilities to the agent (*i.e.,*, jumping). Such situations are common in robotics, particularly with modular or evolving robots [14, 15], or in swarm robotics [16], where agents acquire new capabilities through the physical attachment of components, the ability to combine actions with nearby robots, or the transfer of responsibilities and behaviors among agents. In all these cases, the agent is expected to adapt to such changes and efficiently update its policy to leverage the newly acquired behaviors.

III. RL Agents with Adaptive Behavior

This section introduces MORPHIN, our proposed approach for enabling RL agents to dynamically adapt their behavior in response to evolving environmental conditions, specifically changes in the goal state and available agent actions. Our method enables agents to pursue shifting objectives throughout their operational lifespan and acquire new capabilities as tasks change over time. MORPHIN is made of two main parts: (1) monitoring the environment for changes, (2) adapting agents' behavior to new goals and incorporating new actions. The implementation of our work is publicly available.[1]

[1] Available at: https://anonymous.4open.science/r/morphin_rl

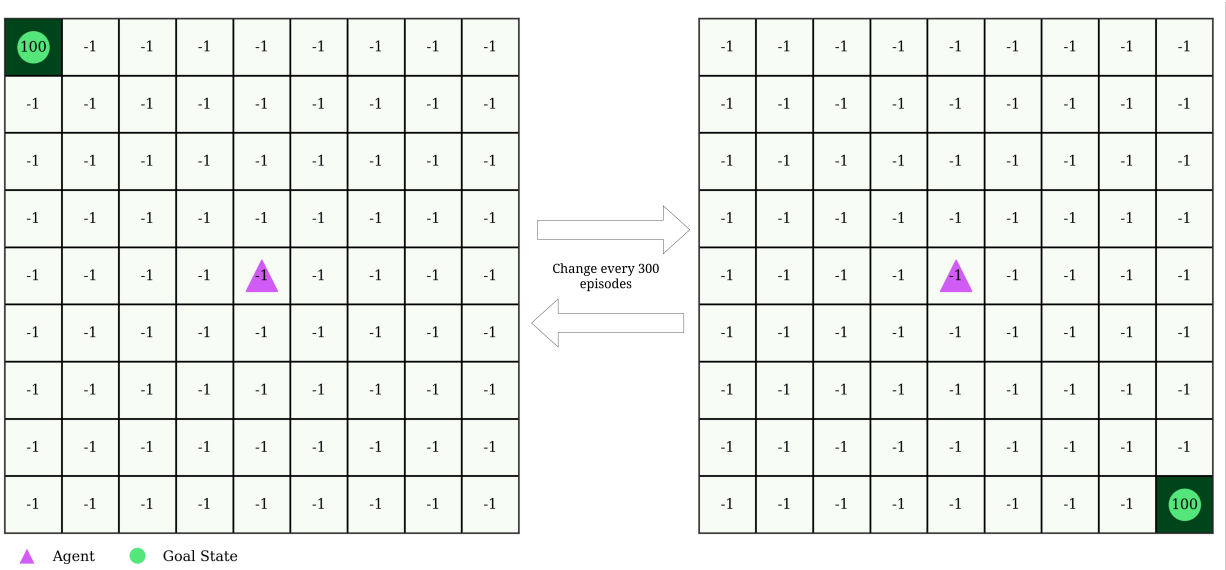


Fig. 1: Non-Stationary Gridworld: Concept drift is induced by relocating the reward. The agent starts at the center and must reach the goal, which alternates between the top-left and bottom-right corners every 300 episodes.

The conceptual foundation of MORPHIN aligns with the ideas proposed by Abel *et al.* [4] for CRL, emphasizing continual adaptation rather than convergence to a fixed solution. We extend tabular Q-learning by integrating adaptive mechanisms that enable agents to dynamically adjust their learning strategies in response to detected environmental changes. Similar to the approach described by Norman *et al.* [17], every time a change is detected, be that a significant change in the rewards or the introduction of new actions, our agent thoroughly explores the environment while retaining information of previously learned policies. The exploration process continues until a new policy stabilizes, to then exploit the acquired knowledge. Applying such strategy agents can rapidly convergence to new environment configurations, without entirely discarding previously learned knowledge.

To enable agents to adapt their behavior to new conditions, we introduce an adaptive mechanisms that adjust the agent's learning rate (α) and exploration rate (ε) in conjunction with a concept drift detection strategy. By continuously monitoring and responding to changes in the environment, agents can autonomously modify their learning process at run time, enabling them to learn new behavior to attain their goals, even if these change. This capability positions our approach within the class of self-adaptive systems (SAS) [18], ensuring continual adaptation and resilience in non-stationary contexts.

A. Environment monitoring and drift detection

The first step for agents to adapt their behavior to an evolving environment is to continuously monitor the environment. RL environment monitoring is intrinsic through the interaction with the environment, as agents continuously observed rewards after each action execution. To detect changes in the environment, *i.e.,* drift, we implement the Page-Hinkley Test (PH-test) [19, 20]. The PH-test calculates the cumulative difference between observed rewards and the running mean reward, incorporating a sensitivity parameter (δ). A concept drift (environmental change) is flagged when the cumulative difference surpasses a predefined threshold. Selecting an appropriate threshold value is crucial, as it determines the sensitivity of drift detection. Higher thresholds result in more conservative detection, while lower thresholds increase sensitivity to changes, this value must be selected based on the magnitude order of rewards and posible changes over them.

Building on the concept of drift detection, and following the ideas of Mignon *et al.* [19], we adaptively increase the exploration rate (ε) whenever the PH-test detects a concept drift. This approach promotes exploration immediately following environmental changes, enabling the agent to acquire new knowledge by temporarily adopting an exploration-focused policy. To ensure adequate exploration, the agent maintains an elevated exploration rate until rewards stabilize (*i.e.,* the cumulative difference is below the threshold, and no further drifts are detected). Once stable, the exploration rate ε uses a decay policy, leading the agent to exploit its behavior.

This adaptive drift detection mechanism ensures the agent maintains an effective balance between exploring new environment dynamics and leveraging previously acquired knowledge. It is crucial to ensure a high exploration rate after concept drift is detected consistently, so the agent can learn a new policy (*i.e.,* behavior) without forgetting previously acquired knowledge. Fig. 2, shows the behavior of the dynamic exploration rate (ε^*) in our non-stationary Gridworld running example.

B. Adaptation to Changing Goals and new Actions

The key to enable agents is to, on the one hand, keeping knowledge gained under previous environment conditions, and

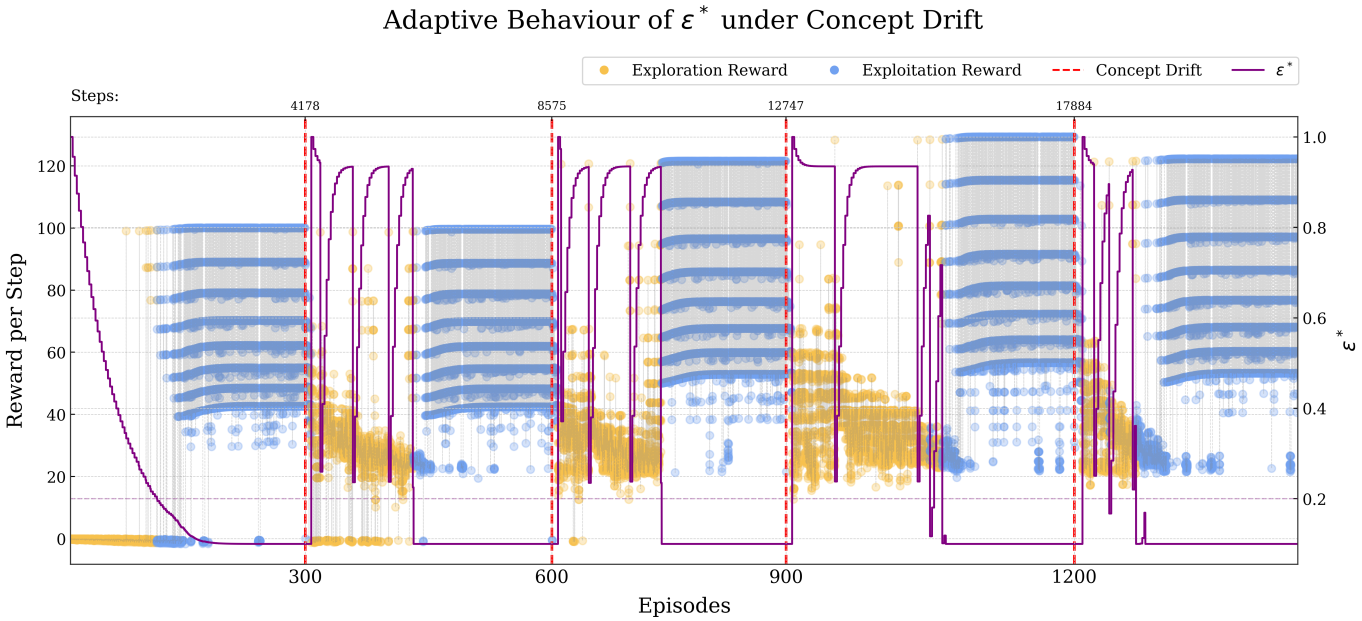


Fig. 2: MORPHIN's adaptive behavior of the dynamic exploration rate (ε^*) in Non-Stationary Gridworld experiments. Observe how ε^* increases whenever a concept drift is detected by the PH-test and remains elevated until the agent consistently achieves the expected rewards (i.e., no further drift and stable rewards). Once stability is reached, ε^* decays toward its minimum value, allowing the agent to exploit the acquired knowledge.

to dynamically adjusting the learning rate (α) in response to detected drift. The learning rate is modified dynamically using the Temporal Difference (TD) error defined in Equation (1).

$$TD_{error} = r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \quad (1)$$

The TD error quantifies the difference between the agent's predicted reward and the actual reward received. The dynamic learning rate (α^*) is then adjusted based on the TD error following Equation (2).

$$\alpha^* = \alpha + (\alpha_{\max} - \alpha) \cdot \frac{1}{1 + e^{-(|TD_{error}| - k)}} \quad (2)$$

In the equation, α corresponds to the base learning rate (e.g., 0.1), and $\alpha_{\max}$ is the upper bound of the learning rate (*e.g.,* 0.9 specified during the creation of the agent). This method introduces a new learning parameter k, which controls the sensitivity of the learning rate to the TD error, with higher values resulting in reduced sensitivity. Parameter k must be carefully (and empirically) tuned according to the specific characteristics of the environment and learning context. High TD errors induce a larger learning rate, enabling faster updates during exploration, whereas lower TD errors yield a more stable and conservative learning process during exploitation phases. Fig. 3 illustrates the behavior of the dynamic learning rate (α^*). In the figure it is possible to observe that the agent is able to detect the concept drift and enable exploration short after the drift occurs. Moreover, after the second concept drift, the exploration period of the agent shortens. This is due fact that agents do not loose their knowledge after a concept drift is detected. As a consequence, agents take more informed decisions after every episode, leading to higher rewards.

The adaptation in the case of the introduction of new actions to an agent, follows the same process as the adaptation to changing goals. Agents can be given new capabilities with the introduction of actions as a direct signal to the agent, in which case the dynamic adaptation process is triggered instantly to enable the exploration taking into account the new action. Another possibility is that the action is obtained unannounced, in which case the agent can detect the new action during the monitoring, and then use the dynamic learning rate.

Whichever method used, agents using the dynamic learning rate strategy, agents explore the environment anew, enabling the discovery of new or improved behavior due to the acquired actions. The overall performance of the agent improves or keeps the same, as the goals remain unchanged.

Finally note that if both the goals change, and the agent acquires new actions simultaneously, The proposed strategy can detect both situations as concept drift, and trigger the learning process to effectively resolve the two variation points.

IV. VALIDATION

A. Evaluation scenarios

To evaluate the effectiveness of MORPHIN in adapting to changing goals and new acquired actions we use two different scenarios. First, we use the Gridworld benchmark as a proof of concept evaluation and running example of MORPHIN. Second we use a traffic-signal control scenario to validate the adaptation of agents' behavior in real-world scenarios.

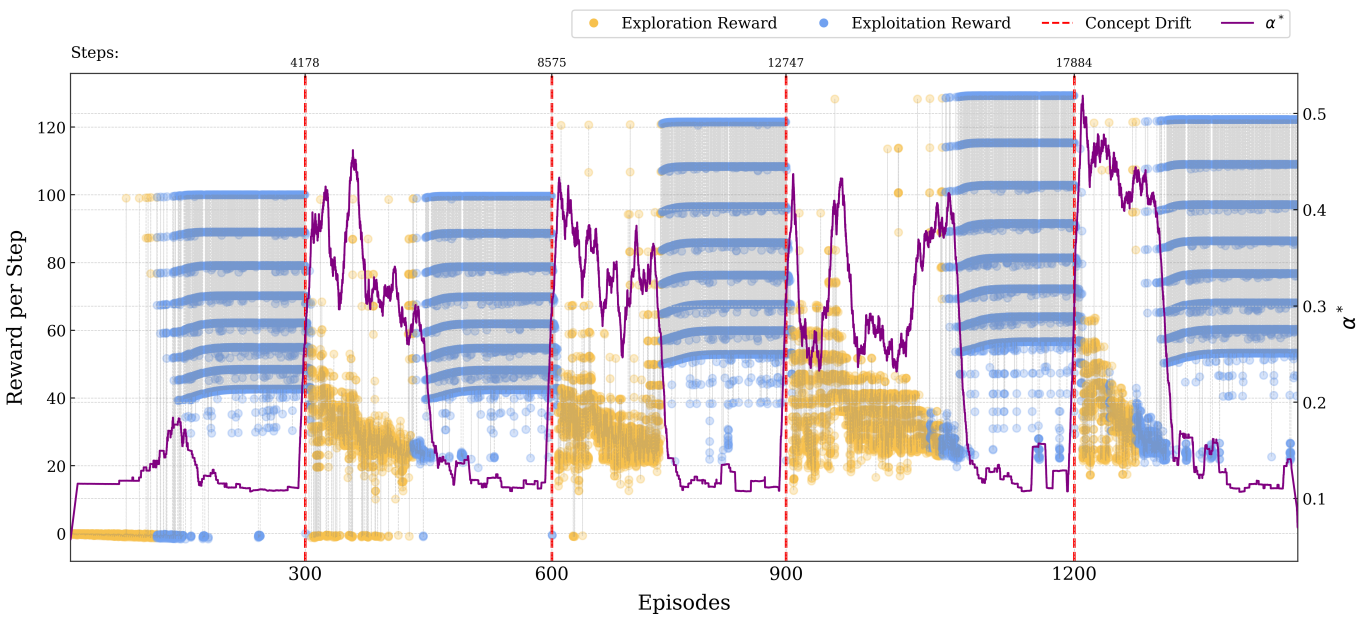


Fig. 3: MORPHIN's adaptive behavior of the dynamic learning rate (α^*) in Non-Stationary Gridworld experiments. Notice how α^* increases during exploration phases (when the TD error is high) and decreases during exploitation phases (when the TD error is low), enabling rapid convergence during exploration and stable learning during exploitation.

1) Gridworld: The Gridworld scenario is defined as described in Section II-B. We execute the Gridworld agents for $1,500$ episodes with a maximum of 36 steps per episode (until the episode concludes). The agents are executed for $1,000$ independent runs, to assure significance of the presented results.

The Gridworld scenario is split into two independent adaptation cases. In the first case, the environment's goal changes every 300 episodes, which for the agent should have learned the optimal policy. On every 300th episode, we change the goal to the diagonally opposite corner (*e.g.,* bottom-right corner on the first change). The goal is changed a total of five times throughout the $1,500$ episodes. After the goal change, the agent should detect the concept drift and converge to an optimal policy for the new goal state.

In the second case, we modified the set of actions the agent can take in episode 300. The action acquired by the agent is the ability to jump over walls. After detecting the change, the agent should incorporate the new action into the optimal policy, leading to episodes executing less steps.

2) Traffic-signal control: Intelligent traffic management is a canonical application of self-adaptive systems [9], and recent meta-RL work has shown benefits when reward functions change with traffic saturation [2]. Here, we model two crossing lanes (C1 vertical, C2 horizontal) with Poisson arrivals and dynamically switch congestion levels (changing λ parameter) to induce concept drift.

In this scenario, the changes to the environment goals and agent actions take place simultaneously. The agent is given a new set of actions following a concept drift detected in a new reward function. This enables the agent to efficiently learn how to utilize the new actions and optimize traffic flow by adapting its policy to varying congestion levels. The agent acquires new capabilities without forgetting previously learned behavior, maintaining adaptability to changing environmental conditions.

We implement a custom `TrafficEnv` (extending OpenAI's Gym library [21]). The system state (c_1, c_2) represents the number of queued vehicles on lanes C1 and C2 respectively, where state values are bounded in the interval $[0, max\_state]$. At each time step the agent selects one of three signal phases (actions): $\{(5,2),\ (2,5),\ (3,3)\}$, indicating service capacities (percentage of the lane cleared per step) for C1 and C2, respectively. During each step, service is applied to C1, new vehicles arrive to C2 (a Poisson distribution with rate λ_2), then service is applied to C2, followed by arrival of new vehicles to C1 (a Poisson distribution with rate λ_1). Any unused service capacity incurs a penalty:

$$\text{penalty} \ = \ 3 \times (\text{waste}_{C1} + \text{waste}_{C2}) \,,$$

discouraging over-serving lanes when queues are small.

The *dynamic reward* combines congestion cost and service penalty, defined as $r - \text{penalty}$, where:

$$r = \begin{cases} -(2c_1 + c_2) & \text{if } c_1 > 7 \ \wedge \ c_1 > c_2, \\ -(c_1 + 2c_2) & \text{if } c_2 > 7 \ \wedge \ c_2 > c_1, \\ -(c_1 + c_2) & \text{otherwise} \end{cases}$$

The agents execute for a total of $10,000$ episodes with 30 steps per episode. We induce two goal changes triggering a concept drift by changing arrival rate (λ_1 and λ_2 for the Poisson distributions) of vehicles: at episode $3,000$ we update the rates (λ_1, λ_2) from $(4, 2) \rightarrow (5, 7)$, and at episode $8,000$ we update the rates from $(5, 7) \rightarrow (3, 1)$.

Note that when a drift is detected, agents acquires two new signal phases (*i.e.,* actions): $(7, 3)$ and $(3, 7)$, allowing for finer control under high congestion. Old actions are never removed.

B. Experimental setting

All experiments are run on an Intel Core i5 with 64GB RAM. The experiments use version 3.12 of Python with the Gym library (0.26.2).

For the evaluation in both application domains we compare the standard Q-learning algorithm (used as a baseline) and MORPHIN.

- *Standard Q-learning* agents are defined with a fixed $\alpha = 0.1$, and ε decayed exponentially to 0.01 from 0.9.
- MORPHIN agents use the PH-test to detect drifts in episode-rewards, resets $\varepsilon = 1$ upon detection, and adaptively adjusts α based on the TD-error (*cf.*. Section III).

C. Evaluation results

1) Gridworld: When comparing the results between MORPHIN and Q-learning, we observe MORPHIN consistently outperforms the base line in adapting to changing goals and acquiring new policies without forgetting previously learned knowledge. As shown in Fig. 4, after $1,500$ episodes, MORPHIN maintains high Q-values for all previous goal configurations, whereas traditional Q-learning must overwrite its Q-values each time the goal relocates. Notably, MORPHIN preserves prior knowledge by promoting exploration for a sufficient period after each detected drift and by scaling updates according to the TD-error magnitude. This approach not only prevents catastrophic forgetting but also enables the agent to handle both contradictory and non-contradictory overlapping subtasks—such as when goals are randomly sampled in nearby regions rather than at the exact same cell.[2]

MORPHIN introduces a modest overhead due to the detection of concept drift and adaptive updates. Nonetheless, the runtime efficiency the agents is actually lower. Using our experimental setting $1,000$ runs take approximately 5 minutes for MORPHIN, while it takes around 8 minutes to complete all runs with the baseline Q-learning implementation. This efficiency gain results directly from the reduced number of steps needed to reach the goal after each drift. Specifically, MORPHIN converges in significantly fewer steps than the standard agent – requiring about $1.9\times$ fewer iterations after the first conceptual drift and $1.7\times$ fewer over the full $1,500$ episodes; which highlights the practical advantages of our adaptive mechanisms in dynamic environments.

[2]Animated simulations of these and other scenarios are available at: https://anonymous.4open.science/r/morphin_rl/simulations/memory_stm_v4_exp.mp4

Fig. 4 shows the state coverage and accumulated rewards for each of the Gridworld states after the $1,500$ episodes. The figure confirms (similar to the Figures 3 and 2) that, the agent is able to detect concept drift and learned to behave in the light of new environment conditions. In the heatmap, lighter colors represent a higher reward, showing how the lower right part of the grid has been explored successfully. Moreover, with or experiment design confirms the benefits of not forgetting, as after the second goal change, the visited states continue to have the same information as initially discovered (in proportion to the grid). At the end, the agent is more motivated to go towards the new goal, as that part of the environment has already been explored with good rewards around the new goal.

In the second case, when a new action is introduced, the MORPHIN agent incorporates the new action as part of the optimal policy, reducing the actions taken from 14 to 4, by jumping over walls and other states. Fig. 5 shows the Q-value heatmaps after 400 episodes (300 episodes until the new action is introduced, and 100 episodes to find the new policy including the acquired action). Upon introduction of the new action, MORPHIN increases exploration and effectively learns to adjust its policy, leveraging the extended action set. In contrast, the baseline Q-learning agent only manages to apply the jump capability in previously unexplored states, as it has a higher chance of being used in such states than in states is a set policy. This results in a suboptimal policy after (taking 14 steps to reach the goal) as the action space changes.

This demonstrates how the adaptive mechanisms in MORPHIN enable effective learning of new actions and adaptation to an expanded action space, while the standard Q-learning agent fails to do so.

2) Traffic-signal control: We now focus on the results of using MORPHIN in the traffic-signal control scenario under non-stationary congestion.

Fig. 6 shows the cumulative mean reward (over the last 500 episodes) for both the baseline agent and the MORPHIN agent. In this scenario there are two moments for concept drift, one at episode $3,000$, and one at episode $8,000$. In the figure, the green vertical line marks the detection of the concept drift using the PH-test. We observe the first concept drift is effectively detected by episode $3,004$. However, the second concept drift is not detected. The reason for the failure to detect the concept drift in the second case is that the new arrival rates generated in to disturb the agent behavior have a reward distribution that is a subset of the original rewards, and therefore remain under the sensitivity threshold for the PH-test.

In Fig. 6 we observe that after the first concept drift, MORPHIN promptly increases exploration, and incorporates its new actions $(7, 3)$ and $(3, 7)$ in the agent's policy within a couple hundred episodes. With the new actions, performance starts to build up, effectively managing the new traffic conditions. In contrast, the baseline Q-learning agent demonstrate a deep performance loss, with only a slight performance gain over time. When the second concept drift occurs, restoring the traffic balance, both agents present a rapid performance

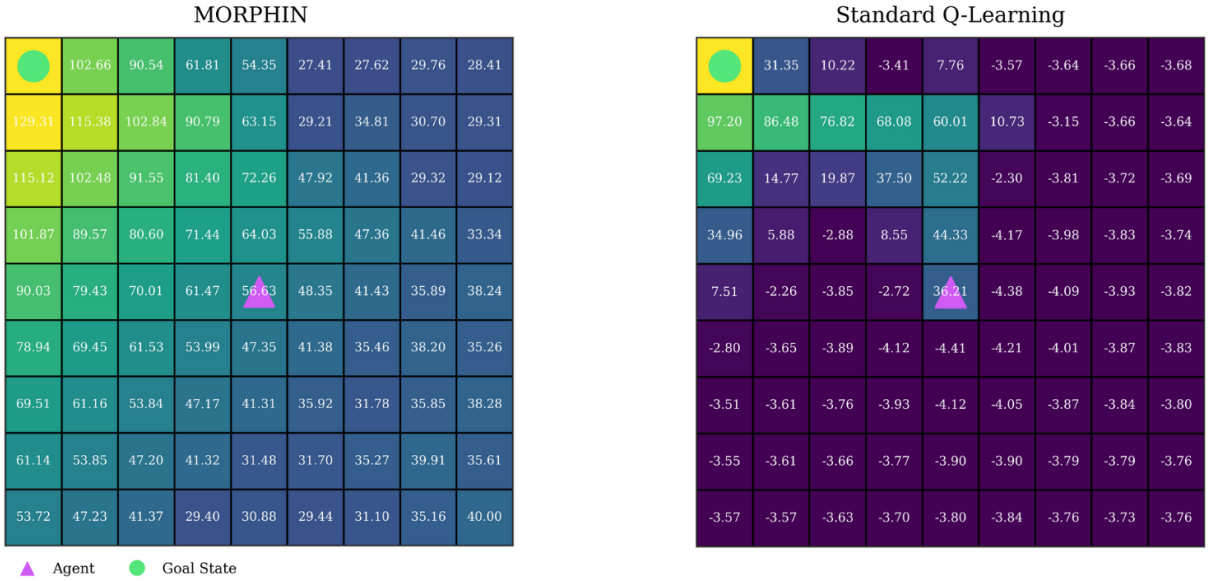


Fig. 4: Comparison of Q-value heatmaps after 1,500 episodes in a Non-Stationary Reward Environment: (left) MORPHIN preserves high Q-values across alternating goal configurations; (right) Standard Q-learning must overwrite past knowledge to learn each new goal.

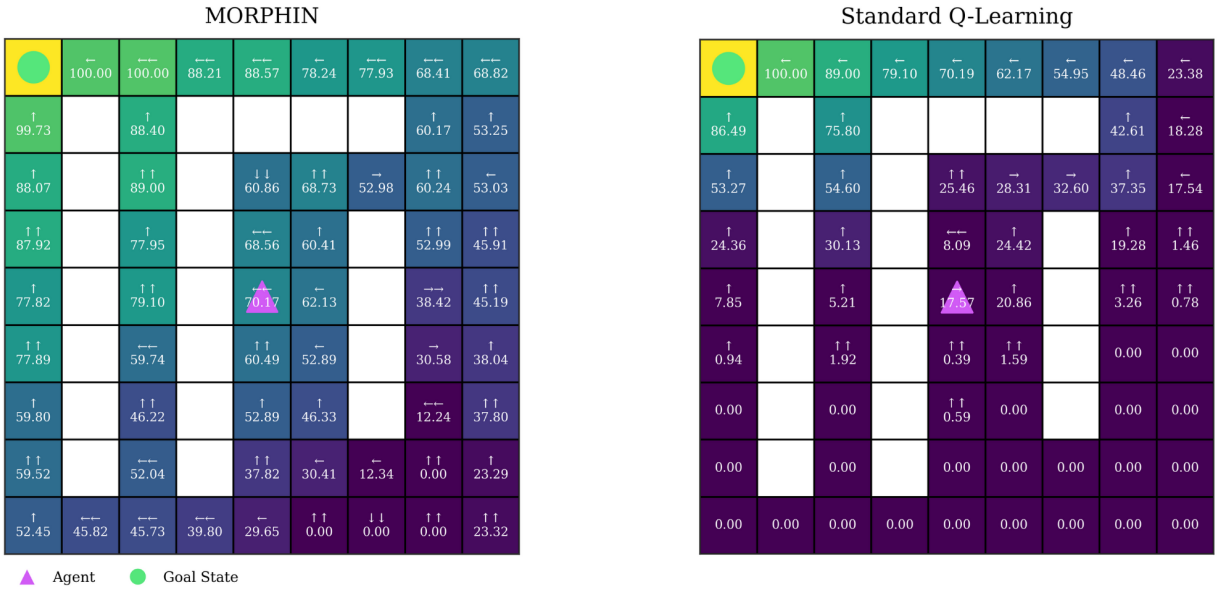


Fig. 5: Comparison of Q-value heatmaps after 400 episodes with a mutable action space: (left) MORPHIN effectively learns to use the extended set of actions, achieving an optimal policy for this configuration; (right) Standard Q-learning remains in a suboptimal policy for the new action space. Double arrows indicate the new actions available to the agent (jumping).

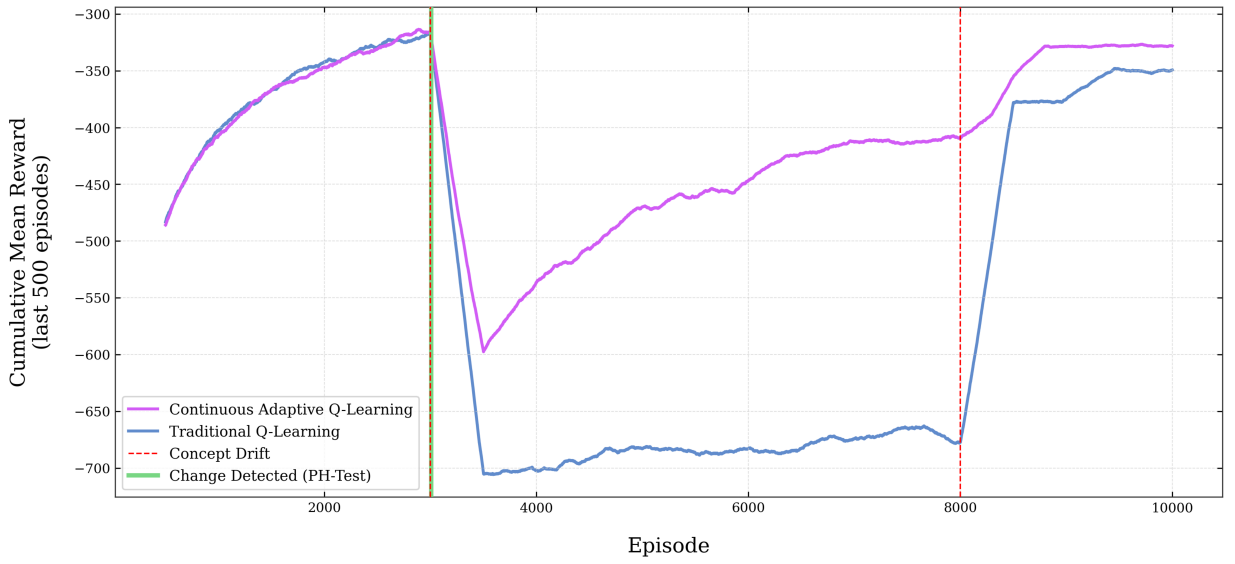


Fig. 6: Learning performance for traffic-signal control under non-stationary congestion. MORPHIN rapidly recovers after the first drift thanks to extension of action set after PH-Test detection, while traditional Q-learning suffers prolonged underperformance until the initial distribution is restored.

improvement, as they are able to quickly exploit the initial policy.

Finally, we observe the built-in penalty for wasted green time ensures MORPHIN does not over-serve one lane as queues reduce, automatically balancing old and new phases without discarding any actions.

This traffic-signal control case study confirms that MORPHIN can effectively:

1) Detect and react to non-stationary conditions via the PH-test, boosting exploration as needed and exploiting previous knowledge whenever possible; making for adaptable agents.
2) Seamlessly incorporate new signal phases, analogous to real-world reprogramming of traffic lights, without removing prior actions.
3) Leverage a dynamic reward (with penalization) to avoid excessive servicing when traffic is light, resulting in resource-efficient policies.

This enables traffic controllers to update phase timings in real time, *i.e.,*, to be self-adaptive, as MORPHIN allows continuous automated adjustment to changing traffic patterns without manual retuning or agent retraining.

V. Related Work

RL inherently deals with the critical trade-off between exploration and exploitation, which significantly influences the performance of the learned policy. An agent must explore sufficiently to avoid settling for suboptimal solutions, yet excessive exploration can lead to inefficient training. Hence, determining an optimal balance between these two strategies is crucial for achieving high-quality solutions [1].

Many adaptive strategies have been proposed to manage this balance dynamically. Tokic [22] and Mignon *et al.* [19] introduce adaptive implementations of the classic ε-greedy policy, highlighting the effectiveness of dynamically adjusting ε values rather than maintaining them statically. Mignon *et al.* [19] demonstrate how an adaptive approach enhances performance in both stationary and non-stationary environments by employing the PH-test for detecting environmental concept drifts.

Building on the exploration-exploitation dilemma, Norman *et al.* [17] propose First-Explore, a meta-RL approach utilizing distinct policies dedicated to exploration and exploitation. Unlike conventional methods that directly optimize cumulative rewards, First-Explore trains these two separate policies and later combines them to form an inference policy that strategically explores initially for k episodes, sacrificing immediate rewards for greater cumulative future gains. This strategy specifically addresses the limitations faced by existing methods that tend to prematurely converge to suboptimal solutions due to inadequate early exploration.

Hyperparameter optimization, particularly the learning rate, also plays a crucial role in training effective RL models. A well-known example is the Adam optimizer [23], which dynamically adjusts the learning rate based on past gradient information to facilitate more efficient training convergence. Extending this line of research, Donancio *et al.* [24] propose a dynamic learning rate approach tailored for deep RL scenarios. Their method adaptively selects optimal learning rates at different training stages, demonstrating substantial performance improvements by considering the non-stationarity inherent to RL tasks.

Transfer learning strategies, like the Transferred Q-Learning [8], have demonstrated improvements in convergence rates. By reusing previous knowledge from similar tasks,

showing that these methods effectively accelerate the learning process.

The methods mentioned abovefocused on the adaptivity of learning parameters and the reuse of prior knowledgeclosely align with the concept of CRL [3]. Abel *et al.* [4] define CRL as a setting in which the best agents never stop learning, contrasting this with traditional RL, which typically treats learning as the identification of a static solution rather than a process of continuous adaptation. Recent work has begun to explore CRL specifically through the lens of Q-learning. Bagus *et al.* [25] provide a systematic empirical studies of Continual Q-learning, using a decomposition of the original task into overlapping but non-contradictory sub-tasks to evaluate the effectiveness of continual learning mechanisms. Another approach for Continual Q-Learning comes from Araújo *et al.* [26], who propose Adaptive Q-Learning (AQL) and its single-partition variants SPAQL and SPAQL-TS. These algorithms dynamically refine stateaction partitions during training and demonstrate strong sample efficiency in continuous control problems such as CartPole, without relying on fixed descretization.

Various studies have shown RL as a potent tool for self-adaptive systems in real-world applications [9]. Notable examples of RL implementations include the use of adaptive ε-greedy policies to dynamically adjust exploration-exploitation trade-offs for Internet of Things (IoT) security in edge computing [27], the integration of RL with active learning and concept drift detection mechanisms (like PH-Test) for network monitoring to effectively identify potential threats [20], or the use of RL macro actions to continuously learn adaptation strategies [28].

Our approach unifies and extends three key research directions identified in the state of the art: (1) decoupling exploration and exploitation through specialized policies; (2) online adaptation of learning parameters based on performance feedback; and (3) proactive environment monitoring and concept drift detection to trigger adaptive responses. Uniquely, MORPHIN integrates these mechanisms within a CRL framework that responds to detected drifts and action-space expansions, resetting exploration and updating learning parameters only when necessary. This coordinated strategy enables continual adaptation without excessive retraining. Furthermore, addressing challenges such as those highlighted by Bagus *et al.* [25], our method mitigates catastrophic forgetting by preserving and reusing prior policy knowledge across evolving configurations, ensuring accelerated convergence even in the presence of overlapping contradictory subtasks.

VI. Conclusion and Future Work

This paper presents MORPHIN, a self-adaptive approach to enable Q-learning agents to effectively detect and adapt to non-stationary environments. In particular, with this work we address the problem of agents' adaptation to changes in their objectives or goal states, as well as changes in their action space by incorporating new actions. The adaptive capabilities of MORPHIN agents are aligned with the principles of self-adaptive systems, enabling the agent to autonomously adjust its behavior in response to evolving environmental conditions. Agent adaptation to changing goals and actions is realized by leveraging concept drift detection, the introduction of the dynamic learning rate, and the dynamic adjustment of the exploration parameters. MORPHIN proves effective in adapting agents' behavior to the environment, resulting in a sustained performance, in contrast of the baseline Q-learning agents, which observe a significant performance drop upon environment changes.

The adaptability strategy proposed with MORPHIN is in line with the ideas of CRL [4], treating learning as a continuous process rather than a one-time optimization. Notably, MORPHIN addresses one of the major challenges put forward in CRL: the problem of catastrophic forgetting. MORPHIN retains prior policy knowledge while adapting to new configurations, enabling knowledge reuse and accelerating convergence as a consequence, even when faced with overlapping contradictory subtasks Bagus *et al.* [25]. The observed performance increase when using MORPHIN is between $1.7\times$ and $1.9\times$.

We validate the effectiveness of MORPHIN in two application domains, a proof of concept RL benchmark, and a realistic traffic signal control scenarios. In both scenarios, the MORPHIN agent successfully adapts to the changing environment conditions, switching of goals or introduction of new actions in the first case, and changing traffic patterns and performance requirements in the second case. In both cases, MORPHIN evidences a performance improvement in contrast to the baseline Q-learning implementation. The introduction of new actions (be that as specific signals as in Gridworld, or in response to drift detection in the traffic signal application), MORPHIN demonstrated its ability to acquire new capabilities without discarding existing ones. This is particularly relevant in real-world traffic systems, where signal configurations can be updated in real time; equipping them with adaptive intelligence to allow the system for continuous optimization without the need for manual system retuning. Finally, for the traffic signal control application, we recognize that the use of the dynamic reward function, including penalties for over-serving, proved effective in promoting resource-efficient policies.

In conclusion, MORPHIN stands as a promising approach for self-adaptive systems requiring lightweight, real-time learning solutions in environments with limited computational resources. MORPHIN is the first tabular self-adaptive agent to seamlessly integrate concept drift detection, dynamic hyperparameter adaptation, and on-the-fly action-space expansion within a unified framework. This approach addresses a significant gap in existing methods and represents a step forward toward robust continual learning agents capable of operating in highly dynamic domains.

Our work present an initial evaluation an adaptation of RL agents to changing environment conditions. As avenues of future work, we aim to generalize its core principles from tabular methods to deep learning architectures. This includes incorporating memory-based plasticity to support knowledge

reuse across tasks and enabling online test-time adaptation, allowing agents to respond to new situations without requiring full retraining.

Moreover, we will extend the evaluation of our work to different application domains combining both goal changes and the introduction of new actions to increase the generalization of our results. Among the extension to the evaluation we envision an empirical evaluation of the introduced learning parameter k, the removal of agent actions, and the combination of environment changes under different contexts.

Finally, we plan to integrate MORPHIN with complementary mechanisms, such as active learning, and deploy it in edge devices for real-world applications (*e.g.,* IoT, or robotics).

REFERENCES

[1] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning, An Introduction*, Second Edition. MIT Press, 2018, p. 550, ISBN: 9780262039246.

[2] G. Kim, J. Kang, and K. Sohn, "A metareinforcement learning algorithm for traffic signal control to automatically switch different reward functions according to the saturation level of traffic flows," *Computer-Aided Civil and Infrastructure Engineering*, vol. 38, Sep. 2022. DOI: 10.1111/mice.12924.

[3] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, *Towards continual reinforcement learning: A review and perspectives*, 2022. arXiv: 2012.13490 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2012.13490.

[4] D. Abel, A. Barreto, B. V. Roy, D. Precup, H. van Hasselt, and S. Singh, *A definition of continual reinforcement learning*, 2023. arXiv: 2307.11046 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2307.11046.

[5] L. M. Zintgraf, L. Feng, C. Lu, M. Igl, K. Hartikainen, K. Hofmann, and S. Whiteson, "Exploration in approximate hyper-state space for meta reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2021, pp. 12 991–13 001.

[6] F. Zhuang *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. PP, pp. 1–34, Jul. 2020. DOI: 10.1109/JPROC.2020.3004555.

[7] J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson, *A survey of meta-reinforcement learning*, 2024. arXiv: 2301.08028 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2301.08028.

[8] E. Y. Chen, M. I. Jordan, and S. Li, *Transferred q-learning*, 2022. arXiv: 2202.04709 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2202.04709.

[9] E. Henrichs, V. Lesch, M. Straesser, S. Kounev, and C. Krupitzer, "A literature review on optimization techniques for adaptation planning in adaptive systems: State of the art and research directions," *Information and Software Technology*, vol. 149, p. 106 940, 2022, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2022.106940. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922000891.

[10] M. Guériau, N. Cardozo, and I. Dusparic, "Constructivist approach to state space adaptation in reinforcement learning," in *International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, ser. SASO'19, Umea, Sweden: IEEE, Jun. 2019.

[11] C. J. C. H. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[12] N. Cardozo and I. Dusparic, "Adaptation to unknown situations as the holy grail of learning-based self-adaptive systems: Research directions," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS'21, IEEE, 2021, pp. 252–253.

[13] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *International conference on machine learning*, PMLR, 2018, pp. 1515–1528.

[14] A. Eiben, "Evolving robot software and hardware," in *Proceedings of the IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS'20, 2020, pp. 1–4.

[15] K. Miras, E. Ferrante, and A. E. Eiben, "Environmental influences on evolvable robots," *PloS one*, vol. 15, no. 5, e0233848, 2020.

[16] M. Schranz, M. Umlauft, M. Sende, and W. Elmenreich, "Swarm robotic behaviors and current applications," *Frontiers in Robotics and AI*, vol. 7, p. 36, 2020.

[17] B. Norman and J. Clune, *First-explore, then exploit: Meta-learning to solve hard exploration-exploitation trade-offs*, 2024. arXiv: 2307.02276 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2307.02276.

[18] T. Wong, M. Wagner, and C. Treude, *Self-adaptive systems: A systematic literature review across categories and domains*, 2022. arXiv: 2101.00125 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2101.00125.

[19] A. Mignon and R. L. A. Rocha, "An adaptive implementation of -greedy in reinforcement learning," *Procedia Computer Science*, vol. 109, pp. 1146–1151, Dec. 2017. DOI: 10.1016/j.procs.2017.05.431.

[20] S. Wassermann, T. Cuvelier, P. Mulinka, and P. Casas, "Adaptive and reinforcement learning approaches for online network monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. PP, pp. 1–1, Nov. 2020. DOI: 10.1109/TNSM.2020.3037486.

[21] M. Towers *et al.*, *Gymnasium: A standard interface for reinforcement learning environments*, 2024. arXiv: 2407.17032 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2407.17032.

[22] M. Tokic, "Adaptive -greedy exploration in reinforcement learning based on value differences," Sep. 2010, pp. 203–210, ISBN: 978-3-642-16110-0. DOI: 10.1007/978-3-642-16111-7_23.

[23] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1412.6980.

[24] H. Donancio, A. Barrier, L. South, and F. Forbes, "Dynamic learning rate for deep reinforcement learning: A bandit approach," Oct. 2024. DOI: 10.48550/arXiv.2410.12598.

[25] B. Bagus and A. Gepperth, "A study of continual learning methods for q-learning," in *International Joint Conference on Neural Networks (IJCNN)*, IEEE, Jul. 2022, pp. 19. DOI: 10.1109/ijcnn55064.2022.9892384. [Online]. Available: http://dx.doi.org/10.1109/IJCNN55064.2022.9892384.

[26] J. P. Araújo, M. A. T. Figueiredo, and M. A. Botto, *Control with adaptive q-learning*, 2020. arXiv: 2011.02141 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2011.02141.

[27] A. Kumar and D. Singh, "Adaptive epsilon greedy reinforcement learning method in securing iot devices in edge computing," *Discover Internet of Things*, vol. 4, Nov. 2024. DOI: 10.1007/s43926-024-00080-7.

[28] N. Cardozo and I. Dusparic, "Auto-cop: Adaptation generation in context-oriented programming using reinforcement learning options," *Information and Software Technology*, vol. 164, p. 107 308, 2023.