A Survey of Reinforcement Learning Algorithms for Dynamically Varying Environments

SINDHU PADAKANDLA, Department of Computer Science and Automation, Indian Institute of Science

Reinforcement learning (RL) algorithms find applications in inventory control, recommender systems, vehicular traffic management, cloud computing, and robotics. The real-world complications arising in these domains makes them difficult to solve with the basic assumptions underlying classical RL algorithms. RL agents in these applications often need to react and adapt to changing operating conditions. A significant part of research on single-agent RL techniques focuses on developing algorithms when the underlying assumption of stationary environment model is relaxed. This article provides a survey of RL methods developed for handling dynamically varying environment models. The goal of methods not limited by the stationarity assumption is to help autonomous agents adapt to varying operating conditions. This is possible either by minimizing the rewards lost during learning by RL agent or by finding a suitable policy for the RL agent that leads to efficient operation of the underlying system. A representative collection of these algorithms is discussed in detail in this work along with their categorization and their relative merits and demerits. Additionally, we also review works that are tailored to application domains. Finally, we discuss future enhancements for this field.

CCS Concepts: • **Computing methodologies** → **Sequential decision making**; *Online learning settings*; *Control methods*; *Planning under uncertainty;*

Additional Key Words and Phrases: Reinforcement learning, sequential decision-making, non-stationary environments, Markov decision processes, regret computation, meta-learning, context detection

ACM Reference format:
Sindhu Padakandla. 2021. A Survey of Reinforcement Learning Algorithms for Dynamically Varying Environments. *ACM Comput. Surv.* 54, 6, Article 127 (July 2021), 25 pages.
https://doi.org/10.1145/3459991

1 INTRODUCTION

Resurgence of **artificial intelligence (AI)** and advancements in it has led to automation of physical and cyber-physical systems [45], cloud computing [75], communication networks [58], robotics [42], and so on. Intelligent automation through AI requires that these systems be controlled by smart *autonomous agents* with least manual intervention. Many of the tasks in the above listed applications are of *sequential decision-making* nature, in the sense that the autonomous agent monitors the *state* of the system and decides on an *action* for that state. This action when exercised on the system *changes* the state of the system. Further, in the new state, the agent again needs

Author's address: S. Padakandla, Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, Karnataka, India, 560012; email: sindhupr@iisc.ac.in.

0360-0300/2021/07-ART127 $15.00
https://doi.org/10.1145/3459991

to choose an action (or control). This repeated interaction between the autonomous agent and the system is sequential and the change in state of the system is dependent on the action chosen. However, this change is uncertain and the future state of the system cannot be predicted. For example, a recommender system [25] controlled by an autonomous agent seeks to predict "rating" or "preference" of users for commercial items/movies. Based on the prediction, it recommends items-to-buy/videos-to-watch to the user. Recommender systems are popular on online stores, video-on-demand service providers, and so on. In a recommender application, the *state* is current genre of videos watched or books purchased, and so on, and the agent decides on the set of items to be recommended for the user. Based on this, the user chooses the recommended content or just ignores it. After ignoring the recommendation, the user may go ahead and browse some more content. In this manner, the state *evolves* and every action chosen by the agent captures additional information about the user.

It is important to understand that there must be a *feedback* mechanism that recognizes when the autonomous agent has chosen the *right* action. Only then can the autonomous agent *learn* to select the right actions. This is achieved through a *reward (or cost)* function, which ranks an action selected in a particular state of the system. Since the agent's interaction with the system (or *environment*) produces a sequence of actions, this sequence is also ranked by a pre-fixed *performance criterion*. Such a criterion is usually a function of the rewards (or cost) obtained throughout the interaction. The goal of the autonomous agent is to find a sequence of actions for every initial state of the system such that this performance criterion is optimized in an average sense. **Reinforcement learning (RL)** [68] algorithms provide a mathematical framework for sequential decision making by autonomous agents.

In this article, we consider an important challenge for developing autonomous agents for real-life applications [20]. This challenge is concerned with the scenario when the environment undergoes changes. Such changes necessitate that the autonomous agent continually track the environment characteristics and adapt/change the learnt actions to ensure efficient system operation. For example, consider a vehicular traffic signal junction managed by an autonomous agent. This is an example of intelligent transportation system, wherein the agent selects the green signal duration for every lane. The traffic inflow rate on lanes varies according to time of day, special events in a city, and so on. If we consider the lane occupation levels as the *state*, then the lane occupation levels are influenced by traffic inflow rate as well as the number of vehicles allowed to clear the junction based on the green signal duration. Thus, based on traffic inflow rate, some particular lane occupation levels will be more probable. If this inflow rate varies, then some other set of lane occupation levels will become more probable. Thus, as this rate varies, so does the state evolution distribution. It is important that under such conditions, the agent select appropriate green signal duration based on the traffic pattern and it must be adaptive enough to change the selection based on varying traffic conditions.

Formally, the system or environment is characterized by a *model* or *context*. The model or context comprises of the state evolution probability distribution and the reward function - the first component models the uncertainty in state evolution, while the second component helps the agent learn the right sequence of actions. The problem of varying environments implies that the environment context changes with time. This is illustrated in Figure 1, where the environment model chooses the reward and next state based on the current "active" context i, $1 \leq i \leq n$. More formal notation is described in Section 2.

1.1 Contributions

- Many streams of work in current RL literature attempt to solve a single underlying problem—that of learning policies that ensure proper and efficient system operation in case of

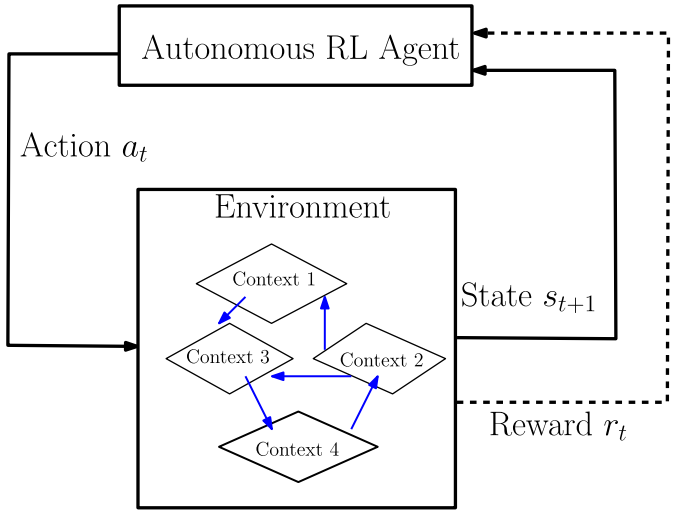

Fig. 1. Reinforcement learning with dynamically varying environments. The environment is modeled as a set of contexts and evolution of these contexts is indicated by blue arrows. At time t, the current state is s_t and RL agent's action a_t changes the state to s_{t+1} and reward r_t is generated.

dynamically varying environments. In this work, we provide a general problem formulation for this scenario, encompassing all cases of **Markov decision process (MDP)** problems.

- Further, this article provides a detailed discussion of the reinforcement learning techniques for tackling dynamically changing environment contexts in a system. The focus is on a single autonomous RL agent learning a sequence of actions for controlling such a system.
- Additionally, we provide an overview of challenges and benefits of developing new algorithms for dynamically changing environments. The benefits of such an endeavour is highlighted in the application domains where the effect of varying environments is clearly observed.

1.2 Overview

The remainder of the article is organized as follows. Section 2 presents the basic mathematical foundation for modelling a sequential decision-making problem in the MDP framework. It also briefly states the assumptions that are building blocks of RL algorithms. In Section 3, we formally introduce the problem, provide a rigorous problem statement and the associated notation. Section 4 describes the benefits of developing algorithms for dynamically varying environments. It also identifies challenges that lie in this pursuit. Section 5 describes the solution approaches proposed till now for the problem described in Section 3. This section discusses two prominent categories of prior works. Section 6 discusses relevant works in continual and meta learning. In both Sections 5 and 6, we identify the strengths of the different works as well as the aspects that they do not address. Section 7 gives a brief overview of application domains that have been specifically targeted by some authors. Section 8 concludes the work and elaborates on the possible future enhancements with respect to the prior work. Additionally, it also describes challenges that research in this area should address.

2 PRELIMINARIES

RL algorithms are based on a stochastic modelling framework known as MDP [9, 57]. In this section, we describe in detail the MDP framework.

2.1 Markov Decision Process : A Stochastic Model

A MDP is formally defined as a tuple $M = \langle S, A, P, R \rangle$, where S is the set of states of the system, A is the set of actions (or decisions). $P : S \times A \to \mathcal{P}(S)$ is the conditional transition probability function. Here, $\mathcal{P}(S)$ is the set of probability distributions over the state space S. The transition function P models the uncertainty in the evolution of states of the system based on the action exercised by the agent. Given the current state s and the action a, the system evolves to the next state according to the probability distribution $P(\cdot|s, a)$ over the set S. At every state, the agent selects a feasible action for every *decision epoch*. The decision horizon is determined by the number of decision epochs. If the number of decision epochs is finite (or infinite), then the stochastic process is referred to as a *finite (or infinite)-horizon* MDP, respectively. $R : S \times A \to \mathbb{R}$ is the reward (or cost) function that helps the agent learn. The environment context comprises of the transition probability and reward functions. If environments vary, then they share the state and action spaces but differ only in these functions.

2.2 Decision Rules and Policies

The evolution of states, based on actions selected by agent until time t, is captured by the "history" variable h_t. This is an element in the set H_t, which is the set of all plausible histories upto time t. Thus, $H_t = \{h_t = (s_0, a_0, s_1, a_1, \ldots, s_t) : s_i \in S, a_i \in A,\ 0 \leq i \leq t\}$. The sequence of decisions taken by agent is referred to as *policy*, wherein a policy is comprised of *decision rules*. A randomized, history-dependent decision rule at time t is defined as $u_t : H_t \to \mathcal{P}(A)$, where $\mathcal{P}(A)$ is the set of all probability distributions on A. Given u_t, the next action at current state s_t is picked by sampling an action from the probability distribution $u_t(h_t)$. If this probability distribution is a degenerate distribution, then the decision rule is called *deterministic decision rule*. Additionally, if the decision rule does not vary with time t, then we refer to the rule as a *stationary decision rule*. A decision rule at time t dependent only on the current state s_t is known as a state-dependent decision rule and denoted as $d_t : S \to \mathcal{P}(A)$. A deterministic, state-dependent and stationary decision rule is denoted as $d : S \to A$. Such a rule maps a state to its feasible actions. When the agent learns to make decisions, basically it learns the appropriate decision rule for every decision epoch. A policy is formally defined as a sequence of decision rules. Type of policy depends on the common type of its constituent decision rules.

2.3 Value Function : Performance Measure of a Policy

Each policy is assigned a "score" based on a pre-fixed performance criterion (as explained in Section 1). For ease of exposition, we consider state-dependent deterministic decision rules only. For a finite-horizon MDP with horizon T, the often used performance measure is the expected total reward criterion. Let $\pi : S \to A$ be a deterministic policy such that for a state s, $\pi(s) = (d_1(s), \ldots, d_T(s))$, $\forall s \in S$. The value function of a state s with respect to this policy is defined as follows:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{T} R(s_t, d_t(s_t))|s_0 = s\right], \tag{1}$$

where the expectation is w.r.t. all sample paths under policy π. A policy π^* is optimal w.r.t. the expected total reward criterion if it maximizes Equation (1) for all states and over all policies. A related criterion for finite-horizon MDP with horizon T is known as *regret*. This performance criterion is directly concerned with the rewards gained during system evolution, i.e., its more emphasis is on the rewards collected rather than on finding the policy that optimally controls a

system. The regret is usually defined for a finite-horizon system as follows:

$$\text{Regret} = V_T^*(s_0) - \sum_{t=0}^{T-1} R(s_t, a_t), \tag{2}$$

where $V_T^*(s_0)$ is the optimal expected T-step reward that can be achieved by any policy when system starts in state s_0.

For infinite horizon MDP, the often used performance measures are the expected sum of discounted rewards of a policy and the average reward per step for a given policy. Under the expected sum of discounted rewards criterion, the value function of a state s under a given policy $\pi = (d_1, d_2, \ldots)$ is defined as follows:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, d_t(s_t)) | s_0 = s\right]. \tag{3}$$

Here, $0 \leq \gamma < 1$ is the discount factor and it measures the current value of a unit reward that is received one epoch in the future. A policy π^* is optimal w.r.t. this criterion if it maximizes Equation (3). Under the average reward per step criterion, the value function of a state s under a given policy $\pi = (d_1, d_2, \ldots)$ is defined as follows (if it exists):

$$V^\pi(s) = \lim_{N \to \infty} \frac{1}{N} \mathbb{E}\left[\sum_{t=0}^{N-1} R(s_t, d_t(s_t)) | s_0 = s\right]. \tag{4}$$

The goal of the autonomous agent (as explained in Section 1) is to find a policy π^* such that either Equation (3) or Equation (4) is maximized in case of infinite horizon or Equation (1) in case of finite horizon, for all $s \in S$.

2.4 Algorithms and their Assumptions

RL algorithms are developed with basic underlying assumptions on the transition probability and reward functions. Such assumptions are necessary, since RL algorithms are examples of *stochastic approximation* [11] algorithms. Convergence of the RL algorithms to the optimal value functions hold when the following assumptions are satisfied.

Assumption 1. $|R(s, a)| < \mathcal{B} < \infty, \forall a \in A \; \forall s \in S$.

Assumption 2. *Stationary P and R, i.e., the functions P and R do not vary over time.*

Assumption 1 states that the reward values are bounded. Assumption 2 implies that the transition probability and reward functions do not vary with time.

We focus on model-based and model-free RL algorithms in this survey. Model-based RL algorithms are developed to learn optimal policies and optimal value functions by estimating P and R from state and reward samples. Model-free algorithms do not estimate P and R functions. Instead these directly either find value function of a policy and improve or directly find the optimal value function. RL algorithms utilize *function approximation* to approximate either the value function of a policy or the optimal value function. Function approximation is also utilized in the policy space. Deep neural network architectures are also a form of function approximation for RL algorithms [22].

In this article, we use the terms "dynamically varying environments" and "non-stationary environments" interchangeably. In the non-stationary environment scenario, Assumption 2 does not hold true. Since previously proposed RL algorithms [10, 68] are mainly suited for stationary environments, we need to develop new methods that autonomous agents can utilize to handle non-stationary environments. In the next section, we formally describe the problem of non-stationary

environments using the notation defined in this section. Additionally, we also highlight the performance criterion commonly used in prior works for addressing learning capabilities in dynamically varying environments.

3 PROBLEM FORMULATION

In this section, we formulate the problem of learning optimal policies in non-stationary RL environments and introduce the notation that will be used in the rest of the article. Since the basic stochastic modeling framework of RL is MDP, we will describe the problem using notation introduced in Section 2.

We define a family of MDPs as $\{M_k\}_{k\in\mathrm{N}^+}$, where $M_k = \langle S, A, P_k, R_k\rangle$, where S and A are the state and action spaces, while P_k is the conditional transition probability kernel and R_k is the reward function of MDP M_k. The autonomous RL agent observes a sequence of states $\{s_t\}_{t\geq 0}$, where $s_t \in S$. For each state, an action a_t is chosen based on a policy. For each pair (s_t, a_t), the next state s_{t+1} is observed according to the distribution $P_k(\cdot|s_t, a_t)$ and reward $R_k(s_t, a_t)$ is obtained. Here $0 < k \leq t$. Note that, when Assumption 2 is true, $P_k(\cdot|s_t, a_t) = P(\cdot|s_t, a_t),\ \forall k \in \mathrm{N}^+$ (as in Section 2). The RL agent must learn optimal behaviour when the system is modeled as a family of MDPs $\{M_k\}_{k\in\mathrm{N}^+}$.

The decision epoch at which the environment model/context changes is known as *changepoint*, and we denote the set of changepoints using the notation $\{T_i\}_{i\geq 1}$, which is an increasing sequence of random integers. Thus, for example, at time T_1, the environment model will change from say M_{k_0} to M_{k_1}, at T_2 it will change from M_{k_1} to say M_{k_2} and so on. With respect to these model changes, the non-stationary *dynamics* for $t \geq 0$ will be

$$P(s_{t+1} = s'|s_t = s, a_t = a) = \begin{cases} P_{k_0}(s'|s,a), & t < T_1 \\ P_{k_1}(s'|s,a), & T_1 \leq t < T_2 \\ \vdots \end{cases} \tag{5}$$

and the reward for $(s_t, a_t) = (s, a)$ will be

$$R(s,a) = \begin{cases} R_{k_0}(s,a), & t < T_1 \\ R_{k_1}(s,a), & T_1 \leq t < T_2 \\ \vdots \end{cases} \tag{6}$$

The extreme cases of the above formulation occur when either $T_{i+1} = T_i + 1,\ \forall i \geq 1$ or $T_1 = \infty$. The former represents a scenario where model dynamics change in every epoch. The latter is the stationary case. Thus, the above formulation is a generalization of MDPs as defined in Section 2. Depending on the decision making horizon, the number of such changes will be either finite or infinite. With changes in context, the performance criterion differs, but Equations (1)–(4) give away some hints as to what they can be. Additionally, since Assumption 2 does not hold true, it is natural to expect that a stationary policy may not be optimal. Hence, it is important to expand the policy search space to the set of all history-dependent, randomized time-varying policies.

Given the family of MDPs $\{M_k\}_{k\in\mathrm{N}^+}$, one *objective* is to learn a policy $\pi = (u_1, u_2, \ldots)$ such that the long-run expected sum of discounted rewards, i.e., $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, u_t(H_t))|H_0 = h_0]$ is maximized for all initial histories $h_0 \in H_0$. Here $u_t(H_t)$ gives a probability distribution over actions and $R(s_t, u_t(H_t))$ denotes the expected reward when an action is sampled from this distribution. For finite horizon MDPs, the objective equivalent to Equation (1) can be stated in a similar fashion. The same follows for Equations (2) and (4) for the infinite-horizon case as well, where the policy search space will be randomized, history-dependent and time-varying.

It should be noted that the space of history-dependent, randomized policies is a large intractable space. Searching this space for a suitable policy is hard. Additionally, in the model-free RL case, how do we learn value functions with only state and reward samples? In the next section, we explore these issues and discuss prior approaches in connection with the problem of non-stationary environments in RL. Some are methods designed for the case when model-information is known, while others are based on model-free RL. All regret-based approaches usually are model-based RL approaches, which work with finite-horizon systems. Approaches based on infinite-horizon systems usually are control methods, i.e., the main aim in such works is to find an approximately optimal policy for a system exposed to changing environment parameters.

4 BENEFITS AND CHALLENGES OF RL IN NON-STATIONARY ENVIRONMENTS

In this section, we will indicate what are the benefits of tackling non-stationary environments in RL algorithms. These benefits straddle across single-agent and multi-agent scenarios.

4.1 Benefits

RL is a machine learning paradigm that is more similar to human intelligence, compared to supervised and unsupervised learning. This is because, unlike supervised learning, the RL autonomous agent is not given samples indicating *what* classifies as good behaviour and what is not. Instead the environment only gives a feedback recognizing *when* the action by the agent is good and when it is not. Making RL algorithms efficient is the first step toward realizing general artificial intelligence [73]. Dealing with ever-changing environment dynamics is the next step in this progression, eliminating the drawback that RL algorithms are applicable only in domains with low risk, for example, video games [65] and pricing [59].

Multi-agent RL [12] is concerned with learning in prescence of multiple agents. It can be considered as an extension of single-agent RL, but encompasses unique problem formulation that draws from game theoretical concepts as well. When multiple agents learn, they can be either competitive to achieve conflicting goals or cooperative to achieve a common goal. In either case, the agent actions are no longer seen in isolation, when compared to single-agent RL. Instead the actions are ranked based on what effect an individual agent's action has on the collective decision making. This implies that the dynamics observed by an individual agent changes based on other agents' learning. So, as agents continually learn, they face dynamically varying environments, where the environments are in this case dependent on joint actions of all agents. Unlike the change in transition probability and reward functions (Section 3), when multiple agents learn, the varying conditions is a result of different regions of state-action space being explored. Thus, non-stationary RL methods developed for single-agent RL can be extended to multi-agent RL as well.

4.2 Challenges

- Sample efficiency: Algorithms for handling varying environment conditions will definitely have issues w.r.t. sample efficiency. When environment changes, then learning needs to be quick, but the speed will depend on the state-reward samples obtained. Hence, if these samples are not informative of the change, then algorithms might take longer to learn new policies from these samples.
- Computation power: Single-agent RL algorithms face *curse-of-dimensionality* with increased size of state-action spaces. Deep RL [22] use **graphical processing units (GPU)** hardware for handling large problem size. Detecting changing operating conditions puts additional burden on computation. Hence, this will present a formidable challenge.
- Theoretical results: As stated in Section 2, without Assumption 2, it is difficult to obtain convergence results for model-free RL algorithms in non-stationary environments. Thus,

providing any type of guarantees on their performance becomes hard. A step in this direction is the work [17] that proves that if the accumulated changes in transition probability or reward function remain *bounded* over time and such changes are insignificant, then value functions of policies of all contexts are "close" enough. However, a more rigorous theoretical pursuit is required.

5 CURRENT SOLUTION APPROACHES

Solution approaches proposed till now have looked at both finite horizon (see Section 5.1) as well as infinite horizon (Section 5.2) cases. Prior approaches falling into these categories are described in the following subsections.

5.1 Finite Horizon Approaches

Finite horizon approaches to dealing with non-stationary environment are References [19, 30, 32, 38, 51]. These study MDPs with varying transition probability and reward functions. The performance criterion is the regret Equation (2) and the goal of these algorithms is to minimize the regret over a finite horizon T. Since decision horizon is finite, the number of changes in the environment is utmost $T-1$. Additionally, a stationary policy need not be optimal in this scenario. So, regret needs to be measured with respect to the best time-dependent policy starting from a state s_0. Basically, regret measures the sum of missed rewards when compared to the best policy (time-dependent) in hindsight.

5.1.1 Comparison of the works. How do References [19, 30, 32, 38, 51] compare with each other?

- All have similar objective—i.e., to minimize the regret during learning. Unlike infinite-horizon results, which maximize the long-run objective and also provide methods to find optimal policy corresponding to this optimal objective value, regret-based learning approaches minimize regret during learning phase only. There are no known theoretical results to obtain a policy from this optimized regret value. Moreover, the regret value is based on the horizon length T.
- References [19, 30, 32, 38, 51] slightly differ with regard to the assumptions on the pattern of environment changes. Reference [32] assumes that the number of changes is known, while References [38, 51] do not impose restrictions on it. The work on Contextual MDP [30] assumes a finite, known number of environment contexts. Reference [19] assumes that only the cost functions change and that they vary arbitrarily.
- Other than the mathematical tools used, the above works also differ with respect to the optimal time-dependent policy used in the computation of the regret. The optimal policy is average-reward optimal in Reference [32], while it is total-reward optimal in References [19, 30, 38, 51]. Reference [30] differs by letting the optimal policy to be a piecewise stationary policy, where each stationary policy is total-reward optimal for a particular environmental context.

5.1.2 Details. We now describe each of the above works in detail. **Contextual MDP (CMDP)** is introduced by Reference [30]. A CMDP is a tuple $\langle C, S, A, Y(C)\rangle$, where C is the set of contexts and $c \in C$ is the context variable. $Y(C)$ maps a context c to a MDP $M_c = \langle S, A, P_c, R_c, \xi_0^c\rangle$. Here, P_c and R_c are same as P_k, R_k, respectively, as defined in Section 3. ξ_0^c is the distribution of the initial state s_0. The time horizon T is divided into H episodes, with an MDP context $c \in C$ picked at the start of each episode. This context chosen is latent information for the RL controller. After the context is picked (probably by an adversary), a start state s_0 is picked according to the distribution ξ_0^c and episode sample path and rewards are obtained according to P_c, R_c. Suppose the episode

variable is h and r_{ht} is the reward obtained in step t of episode h. Let T_h, which is a stopping time, be the episode horizon. The regret is defined as follows:

$$\text{Regret}_{\text{CMDP}} = \sum_{h=1}^{H} J_h^* - \sum_{h=1}^{H} \sum_{t=1}^{t_h} r_{ht},$$

where $J_h^* = J_h^{\pi_c^*} = \mathbb{E}[\sum_{t=0}^{T_h} r_{ht} | s_0 \sim \xi_0^c, \pi_c^*]$ and π_c^* is the optimal policy for context c. Note that c is hidden and hence the above regret notion cannot be computed, but can only be estimated empirically. The CECE algorithm proposed by Reference [30] clusters each episode into one of the contexts in C, based on the partial trajectory information. Depending on the cluster chosen, the context is explored and rewards are obtained.

CMDP [30] necessitates the need to measure "closeness" between MDPs, which enables the proposed CECE algorithm to cluster MDP models and classify any new model observed. The clustering and classification of MDPs requires a distance metric for measuring how close are two trajectories to each other. [30] defines this distance using the transition probabilities of the MDPs. Using this distance metric and other theoretical assumptions, this work derives an upper bound on the regret, which is linear in T. The mathematical machinery used to show this is complex. Moreover, the distance measure used considers only the distance between probability distributions. However, the reward functions are important components of MDP and varies with the policy. It is imperative that a distance measure is dependent on reward functions too.

The UCRL2 [32] and its improvement, variation-aware UCRL2 [51] are model-based regret minimization algorithms, which estimate the transition probability function as well as the reward function for an environment. These algorithms are based on the diameter information of MDPs, which is defined as follows:

$$D_M = \max_{s \neq s'} \min_{\pi : S \to A} \mathbb{E}[W(s'|s, \pi)], \tag{7}$$

where M is the environment context and W is the first time step in which s' is reached from the initial state s. For a given pair of states (s, s') and a policy π, $\mathbb{E}[W(s'|s, \pi)]$ gives the average time the chain takes to move from state s to s'. Note that this average time need not be symmetric w.r.t. the states in the pair. The minimum of this average time w.r.t. all policies indicates how each policy controls the movement between a given pair of states. This information is useful to know when one state belongs to a sort of "good" region of the state space and the other belongs to a "bad" region of the state space. These regions are dictated by the rewards obtained. It tells us how fast does the chain move into or away from "bad" regions. By computing the maximum of this transition time between all pairs of states, the agent can get an idea about the spread of the MDP and which states are better reachable, and so on. This is the information captured in the diameter $D(M)$ of MDP context M .

Both algorithms keep track of the number of visits as well as the empirical average of rewards for all state-action pairs. Using a confidence parameter, confidence intervals for these estimates are maintained and improved. The regret is defined as follows:

$$\text{Regret}_{\text{UCRL2}} = T\rho^* - \sum_{t=1}^{T} r_t,$$

where r_t is the reward obtained at every step t and ρ^* is the optimal average reward defined as follows:

$$\rho^* = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^{T} r_t^*\right],$$

r_t^* is he reward obtained at every step when optimal policy π^* is followed.

UCRL2 learning proceeds in episodes, wherein the complete horizon T is divided into episodes. For each episode i, based on the transition probability and reward estimates, a set of plausible MDPs $\mathcal{M}_i$ is defined in terms of a confidence parameter. From this set, an optimistic MDP $\tilde{M}_i$ and the corresponding optimal policy $\tilde{\pi}_i$ are selected using value iteration. This policy is executed for the episode i. Over iterations the set $\mathcal{M}$ is modified to reflect the actual underlying MDP. The regret during learning is measured based on the policies $\tilde{\pi}_i$ chosen upto horizon T. When environment model changes utmost L times, then UCRL2 restarts learning with a confidence parameter that is dependent on L. Variation-aware UCRL2 [51] proceeds in a similar fashion. However, in Reference [51], the confidence sets used to determine the set $\mathcal{M}_i$ in episode i is dependent on the individual variations in reward and transition probabilities. In both versions of UCRL2, when the environment changes, the estimation restarts leading to a loss in the information collected. Both algorithms give sublinear regret upper bound dependent on the diameter of the MDP contexts. The regret upper bound in Reference [51] is additionally dependent on the reward and transition probability variation terms.

UCRL2 and variation-aware UCRL2 restart learning repetitively. This implies that in simple cases where the environment model alternates between two contexts, these methods restart with large confidence sets, leading to increased regret. Even if this information of alternating contexts is provided, these algorithms will necessarily require a number of iterations to improve the confidence sets for estimating transition probability and reward functions.

Online learning-based approaches [64] for non-stationary environments are proposed by References [19, 38]. MD^2 [19] assumes that the transition probabilities are stationary and known to the agent, while the cost functions vary (denoted L_t) and are picked by an adversary. The goal of the RL agent is to select a sequence of vectors $w_t \in C_V$, where $C_V \in \mathbb{R}^d$ is a convex and compact subset of $\mathbb{R}^d$. The chosen vectors must reduce the regret, which is defined as follows:

$$\text{Regret}_{\text{MD2}} = \sum_{t=1}^{T} \langle L_t, w_t \rangle - \min_{w \in C_V} \sum_{t=1}^{T} \langle L_t, w \rangle,$$

where $\langle \cdots \rangle$ is the usual Euclidean inner product. Thus, without information of L_t, w_t can be chosen only by observing the history of cost samples obtained. For this, the authors propose solution methods based on *Mirror Descent* and *Exponential Weights* algorithms. Reference [38] considers time-varying reward functions and develops a distance measure for reward functions, based on total variation. Using this, regret upper bound is derived, which depends on this distance measure. Further, Reference [38] adapts *Follow the Leader* algorithm for online learning in MDPs.

UCRL2, variation-aware UCRL2, and online learning approaches discussed here are model-based approaches that do not scale well to large state-action space MDPs. The diameter D_M (see (7)) varies with the model and in many cases can be quite high, especially if the MDP problem size is huge. In this case, the regret upper bound is destined to be very large.

5.2 Infinite Horizon Approaches

Works based on infinite-horizon are References [1, 6, 14, 15, 17, 18, 28, 37, 52, 77]. These are oriented toward developing algorithms that learn a good control policy in non-stationary environment models.

5.2.1 Details. [14] proposes a stochastic model for MDPs with non-stationary environments. These are known as **hidden-mode MDPs (HM-MDPs)**. Each mode corresponds to a MDP with stationary environment model. When a system is modeled as HM-MDP, then the transitions between modes are hidden from the learning agent. State and action spaces are common to all modes - but each mode differs from the other modes w.r.t. the transition probability and reward functions.

Algorithm for solving [15] HM-MDP assumes that model information is known. It is based on a Bellman equation developed for HM-MDP, which is further used to design a value iteration algorithm based on dynamic programming principles for this model.

A model-based algorithm for non-stationary environments is proposed by Reference [18]. It is a context detection-based method known as RLCD. Akin to UCRL2, RLCD estimates transition probability and reward functions from simulation samples. However, unlike UCRL2, it attempts to infer whether underlying MDP environment parameters have changed or not. The active model/context is tracked using a predictor function. This function utilizes an error score to rank the contexts that are already observed. The error score dictates which context is designated as "active," based on the observed trajectory. At every decision epoch, the error score of all contexts is computed and the one with the least error score is labeled as the current "active" model. A threshold value is used for the error score to instantiate data structures for new context, i.e., a context that is not yet observed by the learning agent. If all the contexts have an error score greater than this threshold value, then data structures for a new context are initialized. This new context is then selected as the active context model. Thus, new model estimates and the associated data structures are created on-the-fly. This is the main contribution of RLCD. For every context, the action is picked using either Prioritized sweeping or Dyna algorithm [68], which are well researched algorithms.

Change detection-based approaches for learning/planning in non-stationary RL is proposed by References [6, 28, 52]. The identification of active context based on the error score is the crux of RLCD method. Reference [28] improves RLCD by incorporating change detection techniques for identification of active context. Similar to RLCD, this method estimates the transition and reward functions for all contexts. Suppose the number of active context estimates maintained by Reference [28] is j. At time t, a number $S_{i,t}, \forall\, i,\ 1 \leq i \leq j$ is computed. Let $\tilde{P}_i$ and $\tilde{R}_i$ be the transition probability and reward function estimates of context i, where $1 \leq i \leq j$. $S_{i,t}$ is updated as follows:

$$S_{i,t} = \max\left(0, S_{i,t-1} + \ln \frac{\tilde{P}_i(s_{t+1}|s_t, a_t)\tilde{R}_i(s_t, a_t, s_{t+1})}{P_0(s_{t+1}|s_t, a_t)R_0(s_t, a_t, s_{t+1})}\right),$$

where P_0 is the fixed transition function for a uniform model—one that gives equal probability of transition between all states for all actions and R_0 is set to 0 for all state-action pairs. A change is detected if $\max_{1 \leq i \leq j} S_{i,t} > c$, where c is a threshold value. $\tilde{R}_i$ is updated as the moving average of simulated reward samples. $\tilde{P}_i$ is updated based on maximum likelihood estimation. The updation of $\tilde{P}_i$ and $\tilde{R}_i$ are same as in Reference [18]. Also, similar to References [18, 28] utilizes prioritized sweeping algorithm for selecting policy of each context.

Reference [6] shows that in full information case, i.e., when complete model information is known, the change detection approach of Reference [28] leads to loss in performance with delayed detection. Based on this observation, with the full information assumption, Reference [6] designs a **two-threshold policy switching** method (denoted as **TTS**). Given the information that the environment switches from context i to context j, TTS computes the **Kullback-Leibler (KL)** divergence of two contexts $P_i^{\pi_i}$ and $P_j^{\pi_i}$ w.r.t. policy π_i, even though the policy π_i is optimal for context i. When a sample tuple $(s_t, \pi_i(a_t), s_{t+1})$ comprising of current state, current action and next state is obtained at time t, the MDP controller computes the CUSUM [66] value SR_t as follows:

$$SR_{t+1} = (1 + SR_t)\frac{P_j^{\pi_i}(s_{t+1}|s_t, \pi_i(a_t))}{P_i^{\pi_i}(s_{t+1}|s_t, \pi_i(a_t))}, \qquad SR_0 = 0. \tag{8}$$

If SR_t is higher than a threshold value c_1, then it implies that the tuple $(s_t, \pi_i(a_t), s_{t+1})$ is highly likely to be originated in the context j, but it necessitates adequate exploration. Hence in every state, the action that maximizes the KL divergence between $P_j^{\pi_i}$ and $P_i^{\pi_i}$ is fixed as the exploring

action. This policy is denoted as π_{KL} and sample tuples starting from time $t+1$ are obtained using π_{KL}. Simultaneously SR_t is also updated. When SR_t crosses threshold c_2, where $c_2 > c_1$, TTS switches to π_j, which is the optimal policy for MDP with P_j as the transition probability function. The CUSUM statistic SR_t helps in detecting changes in environment context.

Reference [52] proposes a model-free RL method for handling non-stationary environments based on a novel change detection method for multivariate data [55]. Similar to Reference [6], this work assumes that context change pattern is known. However, unlike Reference [6], Reference [52] carries out change detection on state-reward samples obtained during simulation and not on the transition probability functions. The Q-learning (QL) algorithm (see References [68, 74]) is used for learning policies, but maintains a separate Q value table for each of the environment contexts. During learning, the state-reward samples, known as *experience tuples*, are analyzed using the multivariate change detection method known as ODCP. When a change is detected, based on the known pattern of changes, the RL controller starts updating the Q values of the appropriate context. This method is known as *Context QL* and is more efficient in learning in dynamically varying environments, when compared to QL.

A variant of QL, called as **Repeated Update QL (RUQL)** is proposed in Reference [1]. This adaptation of QL repeats the updates to the Q values of a state-action pair by altering the learning rate sequence of QL. Though this variant is simple to implement, it has the same disadvantage as QL, i.e., poor learning efficiency in non-stationary environments.

Online-learning-based variant of QL for arbitrarily varying reward and transition probability functions in MDPs is proposed by Reference [77]. This algorithm, known as Q-FPL, is model-free and requires the state-reward sample trajectories only. With this information, the objective of the algorithm is to control the MDP in a manner such that regret is minimized. The regret is defined as the difference between the average reward per step obtained by Q-FPL algorithm and the average reward obtained by the best stationary, deterministic policy. Formally, we have

$$\text{Regret}_{\text{Q-FPL}} = \sup_{\sigma: S \to A} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[r_t(s_t, \sigma(s_t))] - \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[r_t(s_t, a_t))],$$

where $r_1, r_2, \ldots$ are the arbitrary time-varying reward functions and a_t is the action picked by Q-FPL. σ is a stationary deterministic policy. Q-FPL partitions the learning iterations into intervals and in each interval, the Q values are learnt from the reward samples of that interval. These Q values are stored and are used to pick actions for the next interval by using the *Follow the Perturbed Leader* strategy [64]. At the end of every interval, Q values are reset to zero and not updated during the future intervals. The regret bounds for Q-FPL are derived by Reference [77].

The **risk-averse-tree-search (RATS)** algorithm [37] assumes minimal information regarding the evolution of environment contexts. However, it is a model-based algorithm that is developed for **non-stationary MDP (NSMDP)** as defined in Section 3. This algorithm assumes that the transition probability and reward function changes slowly with time. This "slowness" is formalized as a Lipschitz continuity requirement on these two functions, so that each of these two functions is Lipschitz continuous w.r.t. time. This assumption implies that for small durations of time, the transition probability and reward functions do not change drastically. This relaxation allows the RL agent to work with a "snapshot" of the NSMDP model. The snapshot of the NSMDP model at time t is nothing but the MDP context active at time t. Given this snapshot model and the current state s_{t_0}, the RATS algorithm (a tree-search algorithm) builds a tree of reachable states from the current state s_{t_0}. For building this tree, the algorithm utilizes the snapshot MDP model. However, the action for the current epoch is chosen based on the best response to the worst-case snapshot model at the next epoch. The snapshot model at the next epoch for the reachable states (s'_{t+1})

is estimated and then the appropriate action selected. Based on the maximum depth $d_{\max}$ of the search tree, the learning agent has to estimate a snapshot model for each level (only for reachable states) $t_0+1, \ldots, t_0+d_{\max}$ and select the action a_0 accordingly. The action selected a_0 is then used to perform a real transition. Since a search tree needs to be built and extensive information pertaining to snapshot models is required, the RATS algorithm is not scalable to larger MDP problems.

5.2.2 Remarks.

- The algorithms for solving HM-MDPs [15] are computationally intensive and are not practically applicable. With advances in deep RL [22], there are better tools to make these computationally more feasible. HM-MDPs are a special case of **partially observable Markov decision processes (POMDP)** [68], wherein the state is not accessible to the learning agent. In Equations (5) and (6), if along with state s the time of context change, i.e., k_i is also treated as a state variable, then because k_i is not observable, we obtain a POMDP formulation. Thus, the state of this new POMDP formulation, denoted as $\tilde{s}$ is composed of two components. The first being the original state s and the other being the context change epoch k_i. Thus, $\tilde{s} = (s, k_i)$. Such a formulation requires that non-stationary environment conditions be solved using POMDP solution tools. These solution tools are scarce, owing to the complexity of the problem (see Reference [68] for details). However, recent advances in POMDP solution techniques give promising directions for further research [29, 48].
- RLCD [18] does not require apriori knowledge about the number of environment contexts and the context change pattern, but is highly memory intensive, since it stores and updates estimates of transition probabilities and rewards corresponding to all detected contexts.
- Reference [6] is a model-based algorithm and hence it is impossible to use it when model information cannot be obtained. However, this algorithm can be utilized in model-based RL. But certain practical issues limit its use even in model-based RL. One is that pattern of model changes needs to be known apriori. Additionally, its two-threshold switching strategy is dependent on CUSUM statistic for change detection and more importantly on the threshold values chosen. Since [6] does not provide a method to pre-fix suitable threshold values, it needs to be always selected by trial and error. This is impossible to do, since it will depend on the reward values, sample paths, and so on.
- Extensive experiments while assessing the two threshold switching strategy put forth the following issue. This issue is with reference to Equation (8), where the fraction $\frac{P_j^{\pi_i}(s_{t+1}|s_t,\pi_i(a_t))}{P_i^{\pi_i}(s_{t+1}|s_t,\pi_i(a_t))}$ is computed. Suppose for the policy π_i it so happens that $P_j^{\pi_i}(s_{t+1}|s_t,\pi_i(a_t)) = P_i^{\pi_i}(s_{t+1}|s_t,\pi_i(a_t))$ and optimal policy of P_j is $\pi_j \neq \pi_i$, we will have $\frac{P_j^{\pi_i}(s_{t+1}|s_t,\pi_i(a_t))}{P_i^{\pi_i}(s_{t+1}|s_t,\pi_i(a_t))} = 1$ and SR_t will grow uncontrollably and cross every pre-fixed threshold value. Thus, in this normal case itself, the detection fails, unless threshold value if pre-fixed with knowledge of the changepoint! Thus, Reference [6] is not practically applicable in many scenarios.
- RUQL [1] faces same issues as QL—it can learn optimal policies for only one environment model at a time and cannot retain the policies learnt earlier. This is mainly because both QL and RUQL update the same set of Q values, even if environment model changes. Further, QL and RUQL cannot monitor changes in context—this will require some additional tools as proposed by Reference [52]. The Context QL method retains the policies learnt earlier in the form of Q values for all contexts observed. This eliminates the need to re-learn a policy leading to better sample efficiency. This sample efficiency is however attained at the cost of memory requirement—Q values need to be stored for every context and hence the method is not scalable.

Table 1. A Summary of Current Solution Approaches for Non-stationary Environments

Algorithm	Decision Horizon	Model Information Requirements	Mathematical Tool Used	Policy Retention
CECE [30]	Finite	Model-based	Clustering and Classification	—
UCRL2 [32]	Finite	Model-based	Confidence Sets	—
Variation-aware UCRL2 [51]	Finite	Model-based	Confidence Sets	—
MD^2 [19]	Finite	Partially Model-based	Online learning	—
FTL [38]	Finite	Model-based	Online learning	—
Hidden-mode MDP [14, 15]	Infinite	Model-based	Multiple modes	Yes
RLCD [18]	Infinite	Model-based	Error score	Yes
Extension of RLCD [28]	Infinite	Model-based	Change detection	Yes
TTS [6]	Infinite	Model-based	Change detection	—
Context QL [52]	Infinite	Model-free	Change detection	Yes
RUQL [1]	Infinite	Model-free	Step-size manipulation	No
Q-FPL [77]	Infinite	Model-free	Online learning	Yes
RATS [37]	Infinite	Model-free	Decision tree search	Yes

The prior approaches discussed in this section are summarized in Table 1. The columns assess decision horizon, model information requirements, mathematical tools used and policy retaining capability. A "-" indicates that the column heading is not applicable to the algorithm. In the next section, we describe works in related areas that are focussed on learning across different tasks or using experience gained in simple tasks to learn optimal control of more complex tasks. We also discuss how these are related to the problem we focus on.

6 RELATED AREAS

6.1 Continual Learning

Continual learning algorithms [54] have been explored in the context of deep neural networks. However, it is still in its nascent stage in RL. The goal in continual learning is to learn across multiple tasks. The tasks can probably vary in difficulty, but mostly they are the same problem domain. For example, consider a grid world task, wherein the RL agent must reach a goal position from a starting position by learning the movements possible, any forbidden regions, and so on. Note that the goal position matters in this task, since the agent learns to reach a given goal position. If the goal position changes, then it is a completely new task for the RL agent, which now has to find the path to the new goal position. Thus, both tasks though being in the same problem domain are different. When the RL agent has to learn the optimal policy for the new grid world, it should make sure to not forget the policy for the old task. Hence, continual learning places emphasis on resisting forgetting [54].

An agent capable of **continual, hierarchical, incremental learning and development (CHILD)** is proposed in Reference [60]. This work introduces continual learning by stating the properties expected out of such a RL agent and combines **temporal transition hierarchies (TTH)** algorithm with QL. The TTH method is a constructive neural network-based approach that predicts probabilities of events and creates new neuronal units to predict these events and their contexts. This method updates the weights, activations of the existing neuronal units and also creates new ones. It takes as input the reward signal obtained in the sample path. The output gives the Q values, which are further utilized to pick actions. This work provides extensive

experimental results on grid world problems where learning from previous experience is seen to outperform learning from scratch. The numerical experiments also analyze TTH's capability of acquiring new skills, as well as retaining learnt policies.

Reference [33] derives motivation from synaptic plasticity of human brain, which is the ability of the neurons in the brain to strengthen their connections with other neurons. These connections (or synapses) and strengths form the basis of learning in brain. Further, each of the neurons can simultaneously store multiple memories, which implies that synapses are capable of storing connection strengths for multiple tasks! Reference [33] intends to replicate synaptic plasticity in neural network architectures used as function approximators in RL. For this, the authors use a biologically plausible synaptic model [8]. According to this model, the synaptic weight at time t is dependent on a weighted average of the history of it's modifications upto time t. This is continuous in time, but can be approximated using a particular chain model of N communicating dynamic variables denoted as $\{g_{i,i+1} : 1 \leq i \leq N\}$. Each of these variables is dependent on the values of its immediate neighbouring values. Thus, g_i is modified using values of g_{i+1} and g_{i-1}. The actual synaptic value is read off from the first variable (i.e., g_1) in the chain.

This chain model, which gives the synaptic weight at current time by accounting for all previous changes, is incorporated to tune the Q values and parameters of the neural networks. In case of Q-learning, for each state-action pair, the dynamic variables in the chain are the multiple Q-values of the pair. Thus, the chain model encodes Q-values at different time-scales. Similarly for large state-action space, neural network architectures can be again encoded using the chain model. Here, each of the parameters of the network is modeled using the chain. Thus, if the number of parameters is M and a N-variable chain model is used, then the complexity of the network shoots up to $O(MN)$. That is, the number of parameters to be tuned will be $O(MN)$. The advantage of using the chain model is that when multiple tasks are to be learned, the neural networks modeled using the chain model learn fast and reach optimal reward levels quite soon, when compared to vanilla **deep Q-networks (DQN)** [46]. However, since a single network is used to learn, policy retention is not possible and the RL agent equipped with the chain modeled network has to re-learn from scratch everytime the environment changes.

Policy consolidation-based approach [34] is developed to tackle forgetting of policies. It operates on the same synaptic model as Reference [33], but consolidates memory at the policy level. Policy consolidation means that the current behavioural policy is distilled into a cascade of hidden networks that record policies at multiple timescales. As seen in Reference [33], each neural network parameter is modeled using a N-variable chain model. If we take the kth variable of all chains, then these parameters itself form a neural network. Hence, in this manner, the neural network modeled using the chain model, gives rise to N policies. These recorded policies affect the behavioural policy by feeding into the policy network. The distance between the parameters of two such networks can be used as a substitute for the distance between policies (represented by the networks). This substitute measure is also incorporated in the loss function used for training the actual policy network. This is known as "policy consolidation." Similar to Reference [33], the complexity of Reference [34] is also $O(MN)$. Also, the smoothening of neural network parameters achieved by multi-timescale recording helps in convergence of the policy network.

The CHILD [60] method is akin to RLCD [18] and Context QL [52], both of which also have separate data structures for each model. Thus, in combination with change detection, the CHILD algorithm can be used for dynamically varying environments as well. Developing biologically inspired algorithms [33, 34] is a novel idea. This has been also explored in many areas in supervised learning as well. However, to develop robust performance that is reliable, adequate experimentation and theoretical justifications is needed. The above works lack this and at best can be considered as just initial advancements in this stream of work.

6.2 Learning to Learn: Meta-learning Approaches

Meta-learning as defined in Section 1 involves reusing experience and skills from earlier tasks to learn new skills. If RL agents must meta-learn, then we need to define what constitutes experience and what is the previous data that is useful in skill development. Thus, we need to understand what constitutes *meta-data* and how to learn using meta-data. Most of the prior works are targeted toward **deep reinforcement learning (DRL)**, where only deep neural network architectures are used for function approximation of value functions and policies.

A general model-agnostic meta-learning algorithm is proposed in Reference [21]. The algorithm can be applied to any learning problem and model that is trained using a gradient-descent procedure, but is mostly tested on deep neural architectures, since their parameters are trained by back propagating the gradients. The main idea is to get hold of an internal representation in these architectures that is suitable for a wide variety of tasks. Further, using samples from new tasks, this internal representation (in terms of network parameters) is fine-tuned for each task. Thus, there is no "learning from scratch," but learning from a basic internal representation is the main idea. The assumptions are that such representations are functions of some parameters and the loss function is differentiable w.r.t. those parameters. The method is evaluated on classification, regression and RL benchmark problems. However, it is observed by Reference [43] that the gradient estimates of MAML have high variance. This is mitigated by introducing surrogate objective functions which are unbiased.

A probabilistic view of MAML is given by Reference [2]. A fixed number of trajectories from a task T and according to a policy parameterized by θ is obtained. The loss function defined on these trajectories is then used to update the task-specific parameters ϕ. This is carried out using the gradient of the loss function, which is obtained from either policy gradient [69] or TRPO [63]. The same formulation is extended to non-stationary settings where Reference [2] assumes that tasks themselves evolve as a Markov chain.

Learning good directed exploration strategies via meta-learning is the focus in Reference [27]. The algorithm developed by the authors, known as MAESN, uses prior experience in related tasks to initialize policies for new tasks and also to learn appropriate exploration strategies as well. This is in comparison to Reference [21], where only policy is fine tuned. The method assumes that neural network parameters are denoted θ and per-task variational parameters are denoted as ω_i, $1 \leq i \leq N$, where N is the number of tasks. On every iteration through the task training set, N tasks are sampled from this training set according to a distribution ρ. For a task, the RL agent gets state and reward samples. These are used to update the variational parameters. Further, after the iteration, θ is updated using TRPO [63] algorithm. Numerical experiments for MAESN are carried out on robotic manipulation and locomotion tasks.

6.3 Remarks

The continual learning and meta-learning approaches discussed here are summarized in Table 2. We make some additional remarks here.

- We would like to compare continual learning algorithms with approaches in Section 5. Algorithms like References [1, 51] do not resist *catastrophic forgetting*, because training on new data quickly erases knowledge acquired from older data. These algorithms restart with a fixed confidence parameter schedule. In comparison to this, Reference [52] adapts Q-learning for non-stationary environments. It resists catastrophic forgetting by maintaining separate Q values for each model. This work provides empirical evidence that policies for all models are retained. However, there are issues with computational efficiency and the method needs to be adapted for function-approximation-based RL algorithms.

Table 2. Continual Learning and Meta-learning Approaches

Genre	Mathematical Tool Used	Policy Retention
Continual Learning	Temporal Transition Hierarchies	Yes
	Benna-Fusi Plasticity Model [33]	No
	Policy consolidation [34]	No
Meta-Learning	Shared neural network layers [21]	Yes
	Task distribution as Markov chain [2]	No
	Exploration strategy tuning [27]	Yes

- Explainability of generalization power of deep neural network architectures is still an open problem. The meta-RL approaches are all based on using these architectures. Thus, the performance of these algorithms can only be validated empirically. Also, most of the works described in this section lack theoretical justification. Only the problem formulation involves some mathematical ideas, but none of the results are theoretical in nature. However, applied works like Reference [31] can be encouraged, but only if such works provide some performance analysis of meta-RL algorithms.

7 APPLICATION DOMAINS

Reinforcement learning finds its use in a number of domains—for example, in operations research [62], games [65], robotics [35], and intelligent transportation and traffic systems [76]. However, in most of the prior works in these applications, the assumption is that the environment characteristics and dynamics remain stationary. The number of prior works developing application-specific non-stationary RL algorithms is limited. This is due to the fact that adapting RL to problems with stationary environments is the first simple step toward more general RL controllers, for which scalability is still an issue. Only recent advances in deep RL [22] has improved their scalability to large state-action space MDPs. Improved computation power and hardware, due to advances in high-performance computing, has led to better off-the-shelf software packages for RL. Advancements in computing has led to better implementations of RL—these use deep neural architectures [22] and parallelization [39, 47] for making algorithms scalable to large problem sizes. Single-agent RL algorithms are now being deployed in a variety of applications owing to the improved computing infrastructure. One would also expect that easing of the stationary assumptions on RL environment models would also further increase the need for high computation power. But, due to these advances in computing infrastructure, there is hope to extend RL to applications where non-stationary settings can make the system inefficient (unless there is adaptation).

In this section, we survey the following representative application domains: transportation and traffic systems, cyber-physical systems, digital marketing and inventory pricing, recommender systems and robotics. In these representative domains, we cover works that propose algorithms to specifically deal with dynamically varying environments. Most of these prior works are customized to their respective applications.

7.1 Transportation and Traffic Systems

Traffic systems are either semi-automated or fully automated physical infrastructure systems that manage the vehicular traffic in urban areas. These are installed to improve flow of vehicular traffic and relieve congestion on urban roads. With the resurgence of AI, these systems are being fully automated using AI techniques. AI-based autonomous traffic systems use computer vision, data analytics and machine learning techniques for their operation. Improvement in computing power

for RL has catapulted its use in traffic systems, and, RL-based traffic signal controllers are being designed [52, 56, 61]. Non-stationary RL traffic controllers are proposed by References [52, 61].

Soilse [61] is a RL-based intelligent **urban traffic signal controller (UTC)** tailored to fluctuating vehicular traffic patterns. It is phase-based, wherein a phase is a set of traffic lanes on which vehicles are allowed to proceed at any given time. Along with the reward feedback signal, the UTC obtains a degree of pattern change signal from a **pattern change detection (PCD)** module. This module tracks the incoming traffic count of lanes at a traffic junction. It detects a change in the flow pattern using moving average filters and CUSUM [53] tool. When a significant change in traffic density is detected, learning rates and exploration parameters are changed to facilitate learning. Context QL [52], as described in Section 5, tackles non-stationary environments. This method is evaluated in an autonomous UTC application. The difference in the performance of QL [74] and Context QL is highlighted by numerical experiments [52]. This difference indicates that designing new methods for varying operating conditions is indeed beneficial.

Intelligent transportation systems (ITS) employ information and communication technologies for road transport infrastructure, mobility management and for interfaces with other modes of transport as well. This field of research also includes new business models for smart transportation. Urban aerial transport devices like **unmanned aerial systems (UAS)** are also part of ITS. For urban services like package delivery, law enforcement and outdoor survey, an UAS is equipped with cameras and other sensors. To carry out these tasks efficiently, UAS takes photos and videos of densely human populated areas. Though information gathering is vital, there are high chances that the UAS intrudes privacy. Reference [3] considers this conflicting privacy-information criteria. The problem is that UAS may fly over areas that are densely populated and take pictures of humans in various locations. Though the UAS can use human density datasets to avoid densely populated locations, human concentration is still unpredictable and may change depending on weather, time of day, events, and so on. Thus, the model of human population density tends to be non-stationary. Reference [3] proposes a model-based RL path planning problem that maintains and learns a separate model for each distribution of human density.

7.2 Cyber-Physical Systems and Wireless Networks

Cyber-physical systems (CPS) are integration of physical processes and their associated networking and computation systems. A physical process, for example, a manufacturing plant or energy plant, is controlled by embedded computers and networks, using a closed feedback loop, where physical processes affect computation and vice-versa. Thus, autonomous control forms an innate part of CPS. Many prior works address anomaly detection in CPS [50], since abnormal operation of CPS forces the controllers to deal with non-stationary environments.

CPS security [16] also arises from anomaly detection. The computation and networking systems of CPS are liable to **denial of service (DoS)** and malware attacks. These attacks can be unearthed only if sensors and/or CPS controller can detect anomalies in CPS operation. In this respect, [16] proposes a statistical method for operational CPS security. A modification of the Shiryaev-Roberts-Pollak procedure is used to detect changes in operation variables of CPS, which can detect DDoS and malware attacks.

The data from urban infrastructure CCTV networks can be leveraged to monitor and detect events like fire hazards in buildings, organized marathons on urban roads, crime hot-spots, and so on. Reference [7] uses CCTV data along with social media posts data to detect events in an urban environment. This multimodal dataset exhibits change in properties before, after and during the event. Specifically, Reference [7] tracks the counts of social media posts from locations in the vicinity of a geographical area, counts of persons and cars on the road. These counts are modeled as Poisson random variables and it is assumed that before, after and during a running marathon event,

the mean rates of the observed counts changes. A **hidden Markov model (HMM)** is proposed with the mean count rates as the hidden states. This HMM is extended to stopping time POMDP and structure of optimal policies for this event detection model is obtained.

[5] considers improving user experience in cellular wireless networks by minimizing **Age of Information metric (AoI)**. This metric measures the freshness of information that is transmitted to end users ("user equipments") in a wireless cellular network. A multi-user scheduling problem is formulated that does not restrict the characteristics of the wireless channel model. Thus, a non-stationary channel model is assumed for the multi-user scheduling problem and the objective is to minimize transmission of stale information from the base stations to the user equipments. For this, an infinite-state, average-reward MDP is formulated. Optimizing this MDP is infeasible and hence this work finds a simple heuristic scheduling policy that is capable of achieving the lowest AoI.

7.3 Digital Marketing and Inventory Pricing

Digital marketing and inventory pricing are connected strategies for improving sale of goods and services. In current times, many online sites complement inventory pricing with digital marketing to attract more buyers and hence improve sales. Digital marketing moves away from conventional marketing strategies in the sense that it uses digital technologies like social media, websites, display advertising, and so on, to promote products and attract customers. Thus, it has more avenues for an improved reach when compared to conventional marketing. Inventory pricing is concerned with pricing the goods/services that are produced/developed to be sold. It is important that to gain profits, the manufacturer prices products/services according to the uncertain demand for the product, the production rate, and so on.

A pricing policy for maximizing revenue for a given inventory of items is the focus of Reference [59]. The objective of the automated pricing agent is to sell a given inventory before a fixed time and maximize the total revenue from that inventory. This work assumes that the demand distribution is unknown and varies with time. Hence, this gives rise to non-stationary environment dynamics. This work employs QL with eligibility traces [68] to learn a pricing policy.

Reference [70] studies off-policy policy evaluation method for digital marketing. The users of an online product site are shown customized promotions. Every such marketing promotion strategy uses the customer information to decide which promotions to display on the website. Reference [70] proposes a MDP model with user information as the state and the promotions to be displayed as the action. The reward gained from promotions is measured by tracking the number of orders per visit of the customer. The proposed method is shown to reduce errors in policy evaluation of the promotion strategy.

7.4 Recommender Systems

Recommender systems/platforms are information filtering systems that predicts the preferences that a user would assign to a product/service. These systems have revolutionized online marketing, online shopping and online question-answer forums, and so on. Their predictions are further aimed at suggesting relevant products, movies, books etc to online users. These systems now form the backbone of digital marketing and promotion. Many content providers like Netflix, YouTube, Spotify, Quora, and so on, use them as content recommenders/playlists.

A concept drift-based model management for recommender systems is proposed by Reference [40]. This work utilizes RL for handling concept drift in supervised learning tasks. Supervised learning tasks see shifts in input-label correspondence, feature distribution due to ever changing dynamics of data in real world. Each feature distribution and input-label correspondence is represented as a model and whenver there is a shift in the underlying data, this model needs to be

retrained. A MDP is formulated for taking decisions about model retraining, which decides when to update a model. This decision is necessary, because the model of a given system influences the ability to act upon the current data and any change in it will affect its influence on current as well as future data. If new models are learned quickly, then the learning agent may be simply underfitting data and wasting computational resources on training frequently. However, if the agent delays model retraining, then the prediction performance of model might decrease drastically. Thus, given the current model, current data, the MDP-based RL agent decides when and how to update the model. A similar work using variants of DQN [46] is proposed in Reference [13].

7.5 Robotics

Robotics is the design, development, testing, operation and the use of robots. Its objective is to build machines that are intelligent, can assist humans in various tasks and also perform tasks that are beyond human reach. Robots can also be designed to provide safety to human operations. Robots are now being utilized in outer space missions, medical surgery, meal delivery in hospitals [49], and so on. However, often robots need to adapt to non-stationary operating conditions—for example, a ground robot/rover must adapt its walking gait to changing terrain conditions [44] or friction coefficients of surface [41].

Robotic environments characterized by changing conditions and sparse rewards are particularly hard to learn, because, often, the reinforcement to the RL agent is a small value and is also obtained at the end of the task. Reference [41] focuses on learning in robotic arms where object manipulation is characterized by sparse-reward environments. The robotic arm is tasked with moving or manipulating objects that are placed at fixed positions on a table. In these tasks, often, dynamic adaptation to the surface friction and changed placement of objects on the table is tough. Reference [41] adapts the TRPO algorithm for dealing with these changing operating conditions. The robotic arm RL agent is modeled as a continuous state-action space MDP. In a continuous state-action space setting, the policy is parameterized by Gaussian distribution. Reference [41] proposes a strategy to adjust the variance of this Gaussian policy to adapt to environment changes.

Hexapod locomotion in complex terrain is the focus of Reference [4]. This approach assumes that the terrain is modeled using N discrete distributions and each such distribution captures the difficulties of that terrain. For each such terrain, an expert policy is obtained using deep RL. Conditioned on the state history, a policy from this set of expert policies is picked leading to an adaptive gait of hexapod.

7.6 Remarks

All prior works discussed in this section are specifically designed for their respective applications. For example, Soilse [61] predicts the change in lane inflow rates and uses this to infer whether environment context has changes or not. This technique is limited to the traffic model and more so if lane occupation levels are the states of the model. Similar is the case with a majority of the other works as well. It is tough to extend the above works to more general settings. Some works that are generalizable are References [3, 4, 41, 52, 70]. All these works are summarized in Table 3. The methods suggested in these works can be adapted to other applications as well, provided some changes are incorporated. For example, Reference [52] should be extended to continuous state-action space settings by incorporating function approximation techniques. This will improve its application to tougher problems. Reference [3] utilizes Gaussian process tool to build a model-based RL path planner for UAS. This can be extended to model-free settings using Reference [71] or other works on similar lines. Extending Reference [70] to policy improvement techniques like actor-critic [36] and policy gradient [69] is also a good direction of future work.

Table 3. Summary of Works Specific to Applications

Application Domain	Specific Application	RL Algorithm	Mathematical Tool Used
Traffic Systems	Signal Control [61]	QL	CUSUM
	UAS path planning [3]	Model-based	Confidence Sets
Cyber-Physical Systems	WSN [50]	—	Change Detection
	CCTV Networks [7]	POMDP Model-based	HMM
Digital Marketing	Pricing [59]	QL	—
	Ad placement [70]	Model-based	—
Robotics	Object manipulation [41]	TRPO	Policy distribution adjustment
	Hexapod Locomotion [4]	DQN	Expert Policies

The RL algorithm used is indicated in the third column and the specific mathematical tool used for tackling changing environments is indicated in the last column.

8 FUTURE DIRECTIONS

The previous sections of this survey introduced the problem, presented the benefits and challenges of non-stationary RL algorithms, as well as introduced prior works. This survey article also categorized earlier works. In this section, we describe the possible directions in which the prior works can be enhanced. Following this, we also enumerate challenges that are not addressed by the prior works and that warrant our attention.

Prior approaches can be improved in the following manner:

- The regret-based approaches described in Section 5.1 are useful in multi-armed bandit-type settings where efficient learning with minimal loss is the focus. Since these are not geared toward finding good policies, these works do not prove to be directly useful in RL settings, where control is the main focus. However, the ideas they propose can be incorporated to guide initial exploration of actions in approaches like References [1, 52].
- Relaxing certain theoretical assumptions like non-communicating MDPs [23], multi-chain MDPs [67], and so on, can further improve the applicability of regret-based approaches in control-based approaches.
- Most of the model-based and model-free approaches in Section 5 are not scalable to large problem sizes. This is because each of these methods either consume lot of memory for storing estimates of model information [14, 15, 17–19, 28, 30, 38, 51], or consume compute power for detecting changes [6, 52]. Reference [37] uses compute power for building large decision trees as well. These phenomenal compute power and memory requirements render these approaches to be non-applicable in practical applications that typically function with restricted resources. An option is to offload the compute and memory power requirements onto a central server. Another option is to incorporate function approximation in the representation of value functions and policies.
- Tools from statistics—like, for example, quickest change detection [66], anomaly detection can prove to be indispensable in the problem of non-stationary RL. Also introducing memory retaining capacity in deep neural network architectures will can be a remedy for resisting catastrophic forgetting.
- References [6, 52, 61] assume that the pattern of environment changes is known and can be tracked. However, practically it is often difficult to track such changes. For this, tracking [26] methods can be used.

Next, we discuss additional challenges in this area.

- Need to develop algorithms that are sensitive to changes in environment dynamics and adapt to changing operating conditions seamlessly. Such algorithms can be extended to continual RL settings.
- In the literature, there is a lack of Deep RL approaches to handle non-stationary environments, which can scale with the problem size. Meta learning approaches [2, 21, 27, 43] exist, but these are still in the initial research stages. These works are not sufficiently analyzed and utilized. More importantly, these are not explainable algorithms.
- Some applications like robotics [35] create additional desired capabilities like for example, *sample efficiency*. When dealing with non-stationary environment characteristics, the number of samples the RL agent obtains for every environment model can be quite limited. In the extreme case, the agent may obtain only one sample trajectory, which is observed in robotics arm manipulation exercises. In such a case, we expect the learning algorithm to be data efficient and utilize the available data for multiple purposes—like learn good policies as well as detect changes in environment statistics.
- While encountering abnormal conditions, a RL autonomous agent might violate safety constraints, because the delay in efficiently controlling the system in abnormal conditions can lead to some physical harm. For example, in self-driving cars, a sudden or abrupt change in weather conditions can lead to impaired visual information from car sensors. Such scenarios mandate that the RL agent, though still learning new policies, must keep up with some nominal safe bahaviour. Thus, this can lead to works that intersect safe RL [24] and non-stationary RL algorithms.

9 CONCLUSIONS

RL in dynamically varying environments is a young, growing field with many research attempts targeting specific applications like robotics, autonomous driving, and so on. It mainly integrates results from basic Markov decision process theory, statistics, control, and (of late) neural networks. The promise of this niche research area is to support seamless learning across various environment models and also to build onto theory that will prove to be useful in **multi-agent RL (MARL)**. MARL is also dependent on the area we focus on in this work, because multiple RL agents interact in MARL and non-stationarity occurs as a result of agents learning simultaneously.

In this survey, we have discussed in detail the problem definition for this niche area, problem formulation, and many prior approaches to deal with the associated issues. Algorithms for finite horizon, infinite horizon were discussed. This categorization was essential, because, as can be observed clearly from Section 5, the objective of these categories are quite different from each other. We also covered related areas like meta-learning and continual learning as these areas have very similar problem definitions. Thus, it is worthwhile to explore related areas as well.

As discussed in Section 8, many avenues are open to advance the research in this area. Also, a wide customization of algorithms to applications will be an important way forward as well.

REFERENCES

[1] Sherief Abdallah and Michael Kaisers. 2016. Addressing environment non-stationarity by repeating Q-learning updates. *J. Mach. Learn. Res.* 17, 46 (2016), 1–31.

[2] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. 2018. Continuous Adaptation via meta-learning in nonstationary and competitive environments. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=Sk2u1g-0.

[3] R. Allamaraju, H. Kingravi, A. Axelrod, G. Chowdhary, R. Grande, J. P. How, C. Crick, and W. Sheng. 2014. Human aware UAS path planning in urban environments using nonstationary MDPs. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'14)*. 1161–1167.

[4] Teymur Azayev and Karel Zimmerman. 2020. Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification. *J. Intell. Robot. Syst.* 99 (2020), 659–671. https://doi.org/10.1007/s10846-020-01162-8

[5] Subhankar Banerjee, Rajarshi Bhattacharjee, and Abhishek Sinha. 2020. Fundamental limits of age-of-information in stationary and non-stationary environments. In *Proceedings of the 2020 IEEE International Symposium on Information Theory (ISIT'20)*. 1741–1746. DOI: 10.1109/ISIT44484.2020.9174054

[6] Taposh Banerjee, Miao Liu, and Jonathan P. How. 2017. Quickest change detection approach to optimal control in Markov decision processes with model changes. In *Proceedings of the American Control Conference (ACC'17)*. 399–405.

[7] T. Banerjee, G. Whipps, P. Gurram, and V. Tarokh. 2018. Sequential event detection using multimodal data in non-stationary environments. In *Proceedings of the 21st International Conference on Information Fusion (FUSION'18)*. 1940–1947.

[8] Marcus K. Benna and Stefano Fusi. 2016. Computational principles of synaptic memory consolidation. *Nature Neurosci.* 19, 12 (2016), 1697.

[9] D. P. Bertsekas. 2013. *Dynamic Programming and Optimal Control* (4 ed.). Vol. II. Athena Scientific, Belmont, MA.

[10] Dimitri P. Bertsekas and John N. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

[11] Vivek S. Borkar. 2009. *Stochastic Approximation: A Dynamical Systems Viewpoint*. Vol. 48. Springer, Berlin.

[12] L. Busoniu, R. Babuska, and B. De Schutter. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst., Man Cybernet., Part C (Appl. Rev.)* 38, 2 (2008), 156–172.

[13] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. 2018. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1187–1196. https://doi.org/10.1145/3219819.3220122

[14] Samuel P. M. Choi, Dit-Yan Yeung, and Nevin Lianwen Zhang. 2000. An environment model for nonstationary reinforcement learning. In *Advances in Neural Information Processing Systems*. 987–993.

[15] Samuel P. M. Choi, Dit-Yan Yeung, and Nevin L. Zhang. 2000. Hidden-mode Markov decision processes for nonstationary sequential decision making. In *Sequence Learning*. Springer, Berlin, 264–287.

[16] Angelo Coluccia and Alessio Fascista. 2018. An alternative procedure to cumulative sum for cyber-physical attack detection. *Internet Technology Letters* 1, 3 (2018), e2. https://doi.org/10.1002/itl2.2

[17] Balázs Csanád Csáji and László Monostori. 2008. Value function-based reinforcement learning in changing Markovian environments. *J. Mach. Learn. Res.* 9 (June 2008), 1679–1709.

[18] Bruno C. da Silva, Eduardo W. Basso, Ana L. C. Bazzan, and Paulo M. Engel. 2006. Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd International Conference on Machine Learning*. 217–224.

[19] Travis Dick, Andras Gyorgy, and Csaba Szepesvari. 2014. Online learning in Markov decision processes with changing cost sequences. In *Proceedings of the International Conference on Machine Learning*. 512–520.

[20] Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. 2019. Challenges of real-world reinforcement learning. Retrieved from https://abs/1904.12901.

[21] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML'7)*. JMLR.org, 1126-1135.

[22] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. 2018. *An Introduction to Deep Reinforcement Learning*. Vol. 11. 219–354 pages. https://doi.org/10.1561/2200000071

[23] Ronan Fruit, Matteo Pirotta, and Alessandro Lazaric. 2018. Near optimal exploration-exploitation in non-communicating Markov decision processes. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates, 2998–3008.

[24] Javier García and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* 16, 42 (2015), 1437–1480.

[25] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Info. Syst.* 6, 4 (Dec. 2016), 19. https://doi.org/10.1145/2843948

[26] Oleg N. Granichin and Viktoriya A. Erofeeva. 2018. Cyclic stochastic approximation with disturbance on input in the parameter tracking problem based on a multiagent algorithm. *Autom. Remote Control* 79, 6 (2018), 1013–1028.

[27] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. 2018. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*. MIT Press, 5302–5311.

[28] Emmanuel Hadoux, Aurélie Beynier, and Paul Weng. 2014. Sequential decision-making under non-stationary environments via sequential change-point detection. In *Learning over Multiple Contexts*. Nancy, France.

[29] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. PMLR, 2555–2565.

[30] Assaf Hallak, Dotan Di Castro, and Shie Mannor. 2015. Contextual Markov decision processes. In *Proceedings of the 12th European Workshop on Reinforcement Learning (EWRL'15)*.
[31] Y. Jaafra, A. Deruyver, J. L. Laurent, and M. S. Naceur. 2019. Context-aware autonomous driving using meta-reinforcement learning. In *Proceedings of the 18th IEEE International Conference On Machine Learning and Applications (ICMLA'19)*. 450–455. https://doi.org/10.1109/ICMLA.2019.00084
[32] Thomas Jaksch, Ronald Ortner, and Peter Auer. 2010. Near-optimal regret bounds for reinforcement learning. *J. Mach. Learn. Res.* 11 (Apr. 2010), 1563–1600.
[33] Christos Kaplanis, Murray Shanahan, and Claudia Clopath. 2018. Continual reinforcement learning with complex synapses. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. PMLR, 2497–2506.
[34] Christos Kaplanis, Murray Shanahan, and Claudia Clopath. 2019. Policy consolidation for continual reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. PMLR, 3242–3251.
[35] Jens Kober and Jan Peters. 2014. *Reinforcement Learning in Robotics: A Survey*. Springer International Publishing, 9–67. https://doi.org/10.1007/978-3-319-03194-1_2
[36] Vijay R. Konda and John N. Tsitsiklis. 2003. On actor-critic algorithms. *SIAM J. Control Optim.* 42, 4 (Apr. 2003), 1143–1166. https://doi.org/10.1137/S0363012901385691
[37] Erwan Lecarpentier and Emmanuel Rachelson. 2019. Non-stationary Markov decision processes, a worst-case approach using model-based reinforcement learning. In *Advances in Neural Information Processing Systems*. MIT Press, 7214–7223.
[38] Y. Li and N. Li. 2019. Online learning for Markov decision processes in nonstationary environments: A dynamic regret analysis. In *Proceedings of the American Control Conference (ACC'19)*. 1232–1237. https://doi.org/10.23919/ACC.2019.8815000
[39] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. PMLR, 3053–3062.
[40] Elad Liebman, Eric Zavesky, and Peter Stone. 2018. A Stitch in Time—Autonomous model management via reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'18)*. International Foundation for Autonomous Agents and Multiagent Systems, 990–998.
[41] X. Lin, P. Guo, C. Florensa, and D. Held. 2019. Adaptive variance for changing sparse-reward environments. In *Proceedings of the International Conference on Robotics and Automation (ICRA'19)*. 3210–3216.
[42] H. Liu, S. Liu, and K. Zheng. 2018. A reinforcement learning-based resource allocation scheme for cloud robotics. *IEEE Access* 6 (2018), 17215–17222.
[43] Hao Liu, Richard Socher, and Caiming Xiong. 2019. Taming MAML: Efficient unbiased meta-reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. PMLR, 4061–4071.
[44] G. Mesesan, J. Englsberger, G. Garofalo, C. Ott, and A. Albu-Schäffer. 2019. Dynamic walking on compliant and uneven terrain using dcm and passivity-based whole-body control. In *Proceedings of the IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids'19)*. 25–32.
[45] H. Mirzaei Buini, S. Peter, and T. Givargis. 2017. Adaptive embedded control of cyber-physical systems using reinforcement learning. *IET Cyber-Phys. Syst.: Theory Appl.* 2, 3 (2017), 127–135.
[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
[47] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion stoica. 2018. Ray: A distributed framework for emerging AI Applications. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, 561–577.
[48] Masashi Okada, Norio Kosaka, and Tadahiro Taniguchi. 2020. PlaNet of the bayesians: Reconsidering and improving deep planning network by incorporating bayesian inference. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'20)*. IEEE, 5611–5618. https://doi.org/10.1109/IROS45743.2020.9340873
[49] Michael A. Okyere, Richmond Forson, and Felix Essel-Gaisey. 2020. Positive externalities of an epidemic: The case of the coronavirus (COVID-19) in China. *J. Med. Virol.* 92, 9 (2020), 1376–1379. DOI: 10.1002/jmv.25830
[50] C. O'Reilly, A. Gluhak, M. A. Imran, and S. Rajasegarar. 2014. Anomaly detection in wireless sensor networks in a non-stationary environment. *IEEE Commun. Surveys Tutor.* 16, 3 (2014), 1413–1432.
[51] Ronald Ortner, Pratik Gajane, and Peter Auer. 2019. Variational regret bounds for reinforcement learning. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence*.
[52] Sindhu Padakandla, Prabuchandran K. J., and Shalabh Bhatnagar. 2020. Reinforcement learning algorithm for non-stationary environments. *Appl. Intell.* 50, 11 (01 Nov 2020), 3590–3606. https://doi.org/10.1007/s10489-020-01758-5
[53] E. S. Page. 1954. Continuous inspection schemes. *Biometrika* 41, 1/2 (1954), 100–115.

[54] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Netw.* 113 (2019), 54–71. https://doi.org/10.1016/j.neunet.2019.01.012
[55] K. J. Prabuchandran, Nitin Singh, Pankaj Dayama, and Vinayaka Pandit. 2019. Change point detection for compositional multivariate data. Retrieved from https://arXiv:1901.04935.
[56] K. J. Prabuchandran, A. N. Hemanth Kumar, and S. Bhatnagar. 2014. Multi-agent reinforcement learning for traffic signal control. In *Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems (ITSC'14)*. 2529–2534.
[57] Martin L. Puterman. 2005. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (2nd ed.). John Wiley & Sons, New York, NY.
[58] Y. Qian, J. Wu, R. Wang, F. Zhu, and W. Zhang. 2019. Survey on reinforcement learning applications in communication networks. *J. Commun. Info. Netw.* 4, 2 (June 2019), 30–39. https://doi.org/10.23919/JCIN.2019.8917870
[59] Rupal Rana and Fernando S. Oliveira. 2014. Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning. *Omega* 47 (2014), 116–126. https://doi.org/10.1016/j.omega.2013.10.004
[60] Mark B. Ring. 1998. CHILD: A first step towards continual learning. In *Learning to Learn*. Springer, 261–292.
[61] A. Salkham and V. Cahill. 2010. Soilse: A decentralized approach to optimization of fluctuating urban traffic using reinforcement learning. In *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems*. 531–538.
[62] Manuel Schneckenreither and Stefan Haeussler. 2019. Reinforcement learning methods for operations research applications: The order release problem. In *Machine Learning, Optimization, and Data Science*, Giuseppe Nicosia, Panos Pardalos, Giovanni Giuffrida, Renato Umeton, and Vincenzo Sciacca (Eds.). Springer International Publishing, 545–559.
[63] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning*. 1889–1897.
[64] Shai Shalev-Shwartz. 2012. Online learning and online convex optimization. *Found. Trends Mach. Learn.* 4, 2 (2012), 107–194. https://doi.org/10.1561/2200000018
[65] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. 2019. A Survey of Deep Reinforcement Learning in Video Games. Retrieved from https://arXiv:1912.10944.
[66] A. Shiryaev. 1963. On optimum methods in quickest detection problems. *Theory Probabil. Appl.* 8, 1 (1963), 22–46.
[67] Tao Sun, Qianchuan Zhao, and Peter B. Luh. 2009. A rollout algorithm for multichain Markov decision processes with average cost. In *Positive Systems*. Springer, 151–162.
[68] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press, Cambridge, MA.
[69] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99)*. MIT Press, Cambridge, MA, 1057–1063.
[70] Philip S. Thomas, Georgios Theocharous, Mohammad Ghavamzadeh, Ishan Durugkar, and Emma Brunskill. 2017. Policy evaluation for nonstationary decision problems, with applications to digital marketing. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*. AAAI Press, 4740–4745.
[71] M. Turchetta, A. Krause, and S. Trimpe. 2020. Robust model-free reinforcement learning with multi-objective bayesian optimization. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA'20)*. 10702–10708. DOI: 10.1109/ICRA40945.2020.9197000
[72] Joaquin Vanschoren. 2019. *Meta-Learning*. Springer International Publishing, 35–61. https://doi.org/10.1007/978-3-030-05318-5_2
[73] Pei Wang and Ben Goertzel. 2012. *Theoretical Foundations of Artificial General Intelligence*. Vol. 4. Springer.
[74] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Mach. Learn.* 8, 3-4 (1992), 279–292.
[75] Cheng-Zhong Xu, Jia Rao, and Xiangping Bu. 2012. URL: A unified reinforcement learning approach for autonomic cloud management. *J. Parallel Distrib. Comput.* 72, 2 (2012), 95 – 105. https://doi.org/10.1016/j.jpdc.2011.10.003
[76] Kok-Lim Alvin Yau, Junaid Qadir, Hooi Ling Khoo, Mee Hong Ling, and Peter Komisarczuk. 2017. A Survey on reinforcement learning models and algorithms for traffic signal control. *ACM Comput. Surv.* 50, 3 (June 2017), 38. https://doi.org/10.1145/3068287
[77] J. Y. Yu and S. Mannor. 2009. Arbitrarily modulated Markov decision processes. In *Proceedings of the 48th IEEE Conference on Decision and Control Conference (CDC'09)*. 2946–2953.

Received July 2020; revised January 2021; accepted April 2021