Reinforcement Learning in Non-Stationary Environments

Sindhu Padakandla*, Prabuchandran K.J and Shalabh Bhatnagar

***Abstract*—Reinforcement learning (RL) methods learn optimal decisions in the presence of a stationary environment. However, the stationary assumption on the environment is very restrictive. In many real world problems like traffic signal control, robotic applications, one often encounters situations with non-stationary environments and in these scenarios, RL methods yield sub-optimal decisions. In this paper, we thus consider the problem of developing RL methods that obtain optimal decisions in a non-stationary environment. The goal of this problem is to maximize the long-term discounted reward achieved when the underlying model of the environment changes over time. To achieve this, we first adapt a change point algorithm to detect change in the statistics of the environment and then develop an RL algorithm that maximizes the long-run reward accrued. We illustrate that our change point method detects change in the model of the environment effectively and thus facilitates the RL algorithm in maximizing the long-run reward. We further validate the effectiveness of the proposed solution on non-stationary random Markov decision processes, a sensor energy management problem and a traffic signal control problem.**

***Note to Practitioners*— This paper is motivated by the problem of efficient energy management in a energy harvesting (EH) sensor node. The sensor is typically powered using a energy buffer that stores energy harvested from the ambient environment. The amount of energy harvested from the environment is stochastic in nature, and is usually modeled as a random quantity whose values are sampled from a probability distribution. The goal of the controller here is to enable the sensor node to work perpetually and efficiently. This it needs to do by intelligently deciding on the amount of energy to be utilized from the energy buffer in order to transmit the data stored in the data buffer to a central server. This induces a tradeoff between the delay in the transmitted data and the amount of energy drawn from the buffer. A natural problem arises when the modelling assumption on the energy harvested from a probability distribution changes, as it happens in case of for e.g., solar energy. A sensor can harvest more solar energy around noon, but energy collection dips during evening time and is zero during night. Under such change in the statistics of the energy harvesting process, how does one enable the sensor node to achieve efficient operation ? Our paper suggests a general RL method that can help in intelligent automation of physical systems where there is a need for optimal management of resources under changing system conditions. Using this method an automated system controller can know when the operating conditions have changed and accordingly adjust its behaviour to improve the system's efficiency and performance. Preliminary simulations suggest that this approach is feasible and effective. Our method depends on the data collected from the system and assumes sufficient data could be obtained on a fixed system condition in order to detect change in the system condition effectively.**

***Index Terms*—Markov decision processes, Reinforcement Learning, Non Stationary Environments, Change Detection**

I. Introduction

Autonomous agents are increasingly being designed for sequential decision-making tasks under uncertainty in various domains. For e.g., in traffic signal control [1], an autonomous agent decides on the green signal duration for all lanes at a traffic junction, while in robotic applications, human-like robotic agents are built to dexterously manipulate physical objects [2]. The common aspect in these applications is the evolution of the *state* of the system based on decisions by the agent. In traffic signal control for instance, the *state* is the current congestion levels at a junction and the agent decides on the green signal duration for all lanes at the junction, while in a robotic application, the *state* can be motor angles of the joints etc and the robot decides on the torque for all motors. The key aspect is that the decision by the agent affects the immediate next state of the system, the *reward (or cost)* obtained as well as the future states. Further, the sequence of decisions by the agent is ranked based on a fixed *performance criterion*, which is a function of the rewards obtained for all decisions made. The central problem in *sequential decision-making* is that the agent must find a sequence of decisions for every state such that this performance criterion is optimized. Markov decision processes, dynamic programming (DP) and reinforcement learning (RL) [3], [4], [5] provide a rich mathematical framework and algorithms which aid an agent in sequential decision making under uncertainty.

In this paper, we consider an important facet of real-life applications where the agent has to deal with non-stationary rewards and non-stationary transition probabilities between system states. For example, in vehicular traffic signal control, the traffic inflow rate in some (or all) lanes are different during peak and off-peak hours. The varying traffic inflow rates makes some lane queue length configurations more probable compared to other configurations, depending on the peak and off-peak traffic patterns. It is paramount that under such conditions, the agent select appropriate green signal duration taking into account the different traffic patterns. Also, in robotic navigation, the controller might have to vary robotic arm/limb joint angles depending on the terrain or weather conditions to ensure proper locomotion, because the same joint angles may give rise to different movement trajectories in varying terrains and weather conditions. When environment dynamics or rewards change with time, the agent must quickly adapt its policy to maximize the long-term cumulative rewards collected and to ensure proper and efficient system operation as well. We view this scenario as illustrated in Fig. 1, where the environment changes between models $1, \ldots, n$ dynamically.

The authors are with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India. *Corresponding author (e-mail: sindhupr@iisc.ac.in).

The epochs at which these changes take place are unknown to (or hidden from) the agent controlling the system. The implications of the non-stationary environment is this: when the agent exercises a control a_t at time t, the next state s_{t+1} as well as the reward r_t are functions of the *active* environment model dynamics.

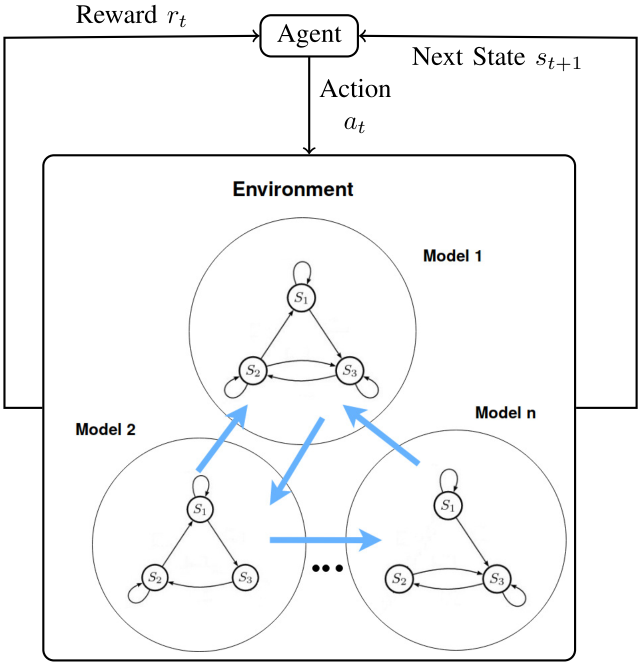

Fig. 1: Non-Stationary RL Framework

Motivated by the real-world applications where changing environment dynamics (and/or rewards, costs) are frequently observed, we focus on developing a model-free RL method that learns optimal policies for non-stationary environments.

A. Related Work

Very few prior works have considered the problem of developing RL algorithms for non-stationary environment models. [6], [7] proposed modeling changing environments in terms of hidden-mode MDPs (HM-MDPs), wherein each *mode (or context)* captures a stationary MDP setting and mode transitions are hidden. All modes share the state and action spaces, but differ either in transition probability function of system states and/or reward function. The methods described in [7] requires information about these functions for each of the modes. Additionally, algorithms which find optimal policies for systems modeled as HM-MDP are computationally intensive and are not practically applicable.

A context detection based RL algorithm (called as RLCD) is proposed in [8]. RLCD algorithm estimates transition probability and reward functions from simulation samples, while predictors are used to assess whether these underlying MDP functions have changed. The active context which could give rise to the current state-reward samples is selected based on an error score. The error score of all contexts estimated till the current epoch is computed. The context which minimizes this error score is designated as the current active model. If all the contexts have a high error score, a new context is estimated. RLCD does not require apriori knowledge about the number of environment contexts, but is highly memory intensive, since it stores and updates estimates of transition probabilities and rewards corresponding to all detected contexts. Moreover, the predictor which allows detection of new contexts is heuristic and is not easy to interpret.

Theoretical framework for RL in fast changing environments based on (ϵ, δ)-MDP is developed in [9]. In this framework, if the accumulated changes in transition probability or reward function remains *bounded* over time and is insignificant, then [9] shows that changes in the optimal value function is also negligible. A model-based method for detecting changes in environment models was proposed in [10], while [11] proposes an extension to model-free RLCD method. Both these works employ quickest change detection [12] methods to detect changes in the transition probability function and /or reward function. The approach in [11] executes the optimal policy for each MDP, while parallely using CUSUM technique to find changes. [10] shows that such an approach leads to loss in performance with delayed detection. It designs a two-threshold switching policy based on KL divergence that detects changes faster, although with a slight loss in rewards accrued. However, [10] is limited in scope, since it assumes that complete model information of all the contexts is known. Hence, the work is not applicable in model-free RL settings. Moreover, [10] does not specify any technique for selecting the threshold values used in the switching strategy, even though the method proposed is completely reliant on the threshold values chosen.

A variant of Q-learning (QL), called as Repeated Update QL (RUQL) was proposed in [13]. It essentially repeats the updates to the Q values of a state-action pair and is shown to have learning dynamics which is better suited to non-stationary environment tasks, on simulation experiments. However, the RUQL faces same issues as QL - that it can learn optimal policies for only one environment model at a time. QL and RUQL update the same set of Q values, even if environment model changes. Additionally, unlike [8], [10], [11], QL and RUQL do not incorporate any mechanism for monitoring changes in environment. So, the agent cannot know whether the model has changed, or whether the model was previously observed. The lack of detection leads to the situation that if the agent encounters a previously learnt model, it has to re-learn the policy, since, there is no mechanism of storing previously learnt policies either.

While all the prior works have provided significant insights into the problem, there are still issues with computational efficiency. Moreover, a reliable technique based on sound theoretical justification is required to find changes in environment statistics, which can be used to adapt policies for the different environment models.

B. Our Contributions

The primary contribution of this paper is to propose a model-free RL algorithm for handling non-stationary environments. In this work, we adapt Q-learning (QL) [14] to learn optimal policies for different environment models. A RL agent employs QL algorithm to learn optimal policies when environment model information is not available, but state and rewards can be obtained through a generative model (i.e., simulation). Q-learning assumes stationary environment model (see [3]), but we adapt it to learn optimal policies for varying environment models. The method we propose utilizes data samples collected during learning to detect changes in the model. We leverage results of change detection on these samples to estimate policy for the new model or improve the policy learnt, if the model had been previously experienced. The resultant method is an online method which can learn and store the policies for the different environment contexts. Note that like [10] we assume that model-change patterns are known and employ a novel algorithm to detect the switches in environment statistics. However, unlike [10], [11] which track changes in probability transition function, we track changes in state and reward samples. Tracking changes in state-reward samples is advantageous when compared to monitoring the transition probability function and the reward function, because in the model-free scenario, we do not have access to these two functions. Hence, monitoring changes in these functions in the model-free case will require the maintenance of their estimates similar to RLCD [8].

The rest of the paper is organised as follows. We give a brief background on Markov decision process (MDP) framework in the next section. This section describes the basic definitions and assumptions made by DP algorithms for solving MDPs. Section III describes the problem along with the notation which will be used in the rest of the paper. We propose a RL method for non-stationary environments in Section IV. Section V shows numerical results on different application domains and analyzes the results. Section VI provides concluding remarks.

II. Preliminaries

A Markov decision process (MDP) [3] 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). $P : S \times A \times S \to [0, 1]$ is the transition probability function. The transition function P models the uncertainty in the evolution of states of the system based on the action exercised by the agent. The evolution is uncertain in the sense that given the current state s and the action a, the system evolves into next state according to the probability distribution $P(s, a, \cdot)$ over the set S. Actions are selected at *decision epochs* by the agent based on their *feasibility* in a state. A decision epoch is the time instant at which the agent selects an action and the number of such epochs determines the decision horizon of the agent. When the number of decision epochs is infinite, we refer to M as an *infinite-horizon* MDP. Depending on the application, each action yields a numerical reward (or cost), which is modeled by the function $R : S \times A \to \mathbb{R}$. Transition function P and reward function R define the *environment model* in which the system operates and the agent interacts with this environment. The interaction comprises of the action selection by the agent for the state and the environment presenting it with the future state and reward (or cost) for the action selected.

A deterministic decision rule $d : S \to A$ maps a state to its feasible actions and it models the agent's action choice for every state. The agent picks a decision rule for very decision epoch. A stationary deterministic Markovian policy $\pi = (d, d, \ldots)$ for an infinite-horizon MDP is a sequence of decision rules, where the deterministic decision rule does not change with the decision epochs. The value function $V^{\pi} : S \to \mathbb{R}$ associated with a policy π is the expected total discounted reward obtained by following the policy π and is defined as

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

for all $s \in S$. $0 \leq \gamma < 1$ is the discount factor and it measures the current value of a unit reward that is received one epoch in the future. The value function is the performance criterion to be optimized in the sequential decision-making problem modeled as MDP. Thus, the objective is to find a stationary policy $\pi^* = (d^*, d^*, \ldots)$ such that

$$V^{\pi^*}(s) = \max_{\pi \in \Pi^{SD}} V^{\pi}(s), \qquad \forall s \in S \tag{2}$$

where Π^{SD} is the set of all stationary deterministic Markovian policies. An optimal stationary deterministic Markovian policy satisfying (2) is known to exist under the following assumptions:

Assumption 1: $|R(s, a)| \leq C < \infty$, $\forall a \in A \; \forall s \in S$

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

Dynamic programming (DP) [3] techniques iteratively solve (2) and provide an optimal policy and the optimal value function for the given MDP based on the above assumptions, when model information P, R is known. Model-free reinforcement learning (RL) [5] algorithms on the other hand obtain the optimal policy when Assumptions 1 and 2 hold, but model information is not available. In non-stationary environments scenario, Assumption 2 is invalid. Clearly classical RL algorithms cannot help in learning optimal policies when Assumption 2 does not hold true. In the next section, we discuss why a stationary policy is no longer optimal when Assumption 2 does not hold true. Additionally, in the next section, we formally describe the problem of non-stationary environments and discuss how to develop RL algorithms which can tackle non-stationarity.

III. Problem Formulation

In this section, we formulate the problem of learning optimal policies in MDP environments with model changes and introduce the notation that will be used in the rest of the paper. We define a family of MDPs $\{M_\theta\}$, where θ takes values from a finite index set Θ. For each $\theta \in \Theta$, we define $M_\theta = \langle S, A, P_\theta, R_\theta\rangle$, where S and A are the state and action spaces, while P_θ is the transition probability kernel and R_θ is the reward function as defined before. The 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 chosen according to the active environment model. We refer to the decision epochs at which the environment model changes as the *changepoints* and denote them using the set $\{T_i\}_{i\geq 1}$. For example, suppose at time T_1, the environment model changes from say M_{θ_0} to M_{θ_1}, at T_2 it changes from M_{θ_1} to say M_{θ_2} and so on. With respect to these model changes, the non-stationary dynamics for $t \geq 0$ will be

$$P(s_{t+1} = s' | s_t = s, a_t = a) = \begin{cases} P_{\theta_0}(s, a, s'), \text{ for } t < T_1 \\ P_{\theta_1}(s, a, s'), \text{ for } T_1 \leq t < T_2 \end{cases} \tag{3}$$

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

$$R(s, a) = \begin{cases} R_{\theta_0}(s, a), \text{ for } t < T_1 \\ R_{\theta_1}(s, a), \text{ for } T_1 \leq t < T_2 \end{cases} \tag{4}$$

We define the randomized history-dependent decision rule at time t as $u_t : H_t \rightarrow \mathcal{P}(A)$, where H_t is the set of all possible histories at time t and $\mathcal{P}(A)$ is the set of all probability distributions on A. An element of H_t is of the form $h_t = (s_0, a_0, s_1, a_1, \ldots, s_t)$. u_t is history dependent since distribution $q_{u_t} \in \mathcal{P}(A)$ picked is dependent on the sequence of states and actions observed upto time t. Given this rule, the next action at current state s_t is picked by sampling an action from q_{u_t}. If we consider only the current state as the history, then $h_t = s_t$ and the decsion rule will be Markovian. The deterministic decision rule $d_t : S \rightarrow A$ defined earlier is then equivalent to u_t when H_t is just s_t and $\mathcal{P}(A)$ is a degenerate probability distribution over A.

Given the family of MDPs $\{M_\theta\}$, the *objective* is to learn a policy $\pi = (u_1, u_2, \ldots)$ such that the expected sum of discounted rewards accumulated over the infinite horizon, i.e., $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, u_t(H_t)) | H_0 = h_0\right]$ is maximized for all initial histories $h_0 \in H_0$. Since, Assumption 2 is not satisfied when model changes, a stationary Markovian deterministic policy may no longer be optimal for this objective and one may have to search over randomized history-dependent policies, which is an intractable problem. But, can we reduce this search space?

Remark 1: Suppose we assume that there is a single changepoint T_1 at which the MDP model changes from M_0 to M_1. Let $\pi_0^* = (d_0^*, d_0^*, \ldots)$ and $\pi_1^* = (d_1^*, d_1^*, \ldots)$ be the optimal policies of M_0 and M_1 respectively. *With an abuse of notation, we denote each decision rule in these optimal policies itself as π_0^* and π_1^*.* If the time T_1 is exactly known, then the agent can select actions using π_0^* till $T_1 - 1$. At T_1, the agent can start following policy π_1^* to collect the rewards. This is the ideal case as the agent cannot achieve a larger return using any other strategy (because π_0^* and π_1^* are optimal for M_0 and M_1 resp.) This is known as the *oracle* strategy. However, since the changepoint is unknown, the next best solution will be to detect the change reliably. The agent can then hope to achieve an expected return which is close to what the oracle strategy yields. This observation leads us to restrict the search space of policies.

As remarked above, when environment model information is known, the agent can precompute optimal policies using DP iterative algorithms and use the appropriate optimal policy for the active model, if the change can be detected reliably. However, in the model-free case, other issues arise which are interconnected and these are described below:

1) Model information is not available - the transition function P and reward function R are not known. Only samples of state and reward from simulation are available. Hence, we cannot detect changes using transition functions P and/or reward function R.
2) Detecting context changes - Since model information is not accessible, how to use the state and reward samples to find changes in contexts ? Changes can occur exclusively in P or exclusively in R, so we require a unified method to detect changes in either of them.
3) Which policy to follow to follow during the learning phase ? Even if we devise a method to detect changes using state-reward samples, which policy do we follow to collect the samples while learning ?

In the next section, we explore these issues and provide solutions to address them. We mainly provide solutions for the model-free RL setting, suggesting what policies to be used while learning and how to detect changes. Hence, our method also works for the model-based setting and is described in the next section.

IV. Our Approach

In this section, we describe the RL method to deal with changes in environment models. The method adapts a change detection algorithm [15] to find changes in the pattern of state-reward tuples observed while learning a policy.

A. Our Proposed Solution

The change in P and R ultimately manifests itself in a change in sample paths. Taking a cue from this, we consider change detection on state-reward sequences. By tracking variations in the state-reward tuples, our method not only captures variations which occur combinedly in P and R, but also those which occur exclusively in P or exclusively in R. For the proposed solution, we assume some structure in how the context changes occur. These assumptions are listed below:

Assumption 3: In our method, which is detailed in this section, we assume that atleast once the environment model changes. Additionally, we assume that the pattern of model change(s) is also known.

Assumption 4: The environment context changes are not too frequent, i.e., we get sufficient state-reward samples for every context before the environment switches to some other context.

To evaluate our method, we need a performance metric. Since, in the model-free scenario, the reward obtained is the only indicator of how well the agent is performing, we compare the long-term discounted rewards collected by our RL method with the same quantity that the oracle strategy yields. We refer to the difference between these two quantities as the *regret*, which is similar to the definition in [16]. Following the above assumptions and the performance metric, we describe next, our RL method for non-stationary environments.

1) Experience tuples: Unlike previous approaches [11], [10] which track variations in the functions P and R, our method captures variations in the occurences of states and rewards. We define an experience tuple e_t at epoch t as the triplet consisting of the current state s_t, current immediate reward (or cost) obtained r_t and the next state s_{t+1}. So, $e_t = \langle s_t, r_t, s_{t+1} \rangle$. The set of experience tuples $\{e_t : 1 \leq t \leq B\}$, where B is a batch size, is input to the changepoint detection algorithm which is described next.

2) Change detection using experience tuples: We adapt the changepoint detection algorithm proposed in [15] for data consisting of experience tuples. [15] describes an online parametric Dirichlet changepoint (ODCP) algorithm for unconstrained multivariate data. ODCP algorithm transforms any discrete or continuous data into compositional data and utilizes Dirichlet parameter likelihood testing to detect change points. Multiple changepoints are detected by performing a sequence of single changepoint detections.

ODCP uses appropriate metric while detecting change points. One can also adapt other change detection algorithms like E-Divisive Change Point detection (ECP) [17]. However, ECP uses Euclidean distance based metric to detect change points, which may not be suitable for discrete and compositional data that do not follow Euclidean geometry. Thus, ODCP reliably estimates change points compared to ECP.

ODCP requires the multivariate data to be i.i.d samples from a distribution. However, we utilize it in the Markovian setting. This is possible under the assumption that we have sufficient data under each of the fixed Markovian probability distribution. Let us suppose we choose the actions according to randomized policy $\pi = \{u, u, \ldots, \}$, where $q_{u(s)} \in \mathcal{P}(A)$. With an abuse of notation, we denote each decision rule as π, with $\pi(s) \in \mathcal{P}(A)$. Let $\psi(a|s)$ be the probability that action a is selected in state s according to the decision rule $\pi(s)$. Let $\phi^{\pi}(\cdot)$ denote the steady state distribution under policy π. Under this condition, the tuple (s, r, s') namely the state, reward and the next state will be distributed as follows,

$$(s, r, s') \sim \phi^{\pi}(s)\psi(\cdot|s)P(s, \psi(\cdot|s), s')R(s, \psi(\cdot|s), s').$$

Now we could treat the tuples (s_t, r_t, s_{t+1}) as i.i.d data and utilize the ODCP algorithm.

3) Sampling mechanism for collecting experience tuples: When changepoints are unknown, an acceptable solution is to find changepoints reliably and minimize the regret as defined in Section III. Hence, we need to formulate techniques to sample state-reward tuples without incurring much loss in rewards collected during learning. The RL algorithm must detect a change in environment model which occurs when the policy is still being learnt. In such a case, it can switch to the optimal policy w.r.t the active environment model. The techniques we formulate are applicable for both model-based (known P and R) and model-free (unknown P and R) RL settings.

1) Model-based RL Setting: P_θ, R_θ are known. A sampling mechanism needs to explore different actions other than the optimal action in order to detect changes. However, such exploration should be appropriately controlled, otherwise it will increase the average regret. As part of our solution we prescribe that experience tuples be collected using the following specific randomized policy: at each state s, the agent should follow optimal action prescribed by π_0^* with probability $(1-\epsilon)$ and a random action with probability ϵ, where $\epsilon > 0$. Thus, the policy used is $\pi = (u, u, \ldots)$, where $u : S \rightarrow \mathcal{P}(A)$, $q_u(\pi_0^*(s)) = 1-\epsilon$ and $q_u(a) = \frac{\epsilon}{|A|-1}$, $a \in A \setminus \{\pi_0^*(s)\}$. We call this as ϵ-policy.
2) Model-free RL Setting: P_θ, R_θ are unknown. We propose the use of Q-learning (QL) [14], a model-free iterative RL algorithm to obtain the experience tuples. QL estimates the optimal Q-values for all state-action pairs of a MDP. Q-value of a state-action pair w.r.t policy π is defined as the expected discounted return starting from state s, taking action a and following policy π thereafter. The QL iteration [14] requires that all state-action pairs be explored for an infinite number of times, so that the Q-value of each pair can be accurately estimated, based on the reward obtained at each step of the algorithm. To ensure this, an exploration strategy is used. As part of our solution, we prescribe that experience tuples be collected using either of the following strategies:
   - ϵ-greedy: At state s, with probability $(1-\epsilon)$, the action maximizing the Q-value estimate of state s at iteration k, i.e., $\arg\max_b Q_k(s, b)$ is selected, while with probability ϵ, a random action in A is selected.
   - UCB [18]: At state s, an action a is selected as follows
   $$a = \arg\max_b \left(Q_k(s, b) + C\sqrt{\frac{\log N(s)}{N(s, b)}} \right),$$
   where $Q_k(s, \cdot)$ is the estimate of the Q-value at iteration k, $N(s)$ tracks the number of times state s is visited and $N(s, b)$ is the number of times action b has been picked when state s is visited. C is a constant.

Once samples are collected and changes are detected (which can be done in an online fashion), the policy can be changed to the optimal policy of the appropriate model.

B. Q-learning for different environment contexts

For model-free RL case, we use the result of the ODCP algorithm to learn optimal policies for the next model from the samples obtained after the changepoint is detected. We call this method as Context QL. The concept of Context QL is in a way similar to RLCD [8], where new models are instantiated whenever a change is detected and P and R for the new model are estimated based on samples obtained after the change. However, unlike RLCD, Context QL employs the ODCP algorithm to detect changes. Furthermore, Context QL updates Q values of the relevant model whenever a change is detected and does not attempt to estimate P and R for the new model. Additionally, if the method obtains samples from a prior observed model, it updates the Q values corresponding to that model. Thus, in this manner, the information which was learnt and stored earlier (in the form of Q values), is not lost. The Context QL pseudocode is given in Algorithm 1.

Algorithm 1 Context QL

```
1: $\{T_1, T_2, \ldots, T_n\}$, $\{M_1, M_2, \ldots, M_k\}$
2: **Input:** Model change pattern, $M_{j_1} \to M_{j_2}$, $M_{j_2} \to M_{j_3}$, $\ldots, M_{j_{n-1}} \to M_{j_n}$
3: Context number, $c = 1$
4: Initialize Q values $Q(m, s, a) = 0$, $\forall m \in 1, \ldots, k$, $\forall (s, a) \in S \times A$
5: Initial state $s_1 = s$, $\tau^* = 1$
6: **for** $i = 1$ to $T_1 - 1$ **do**
7:     Get $a_i$ according to $\epsilon$-greedy or UCB exploration
8:     Obtain next state $s_{i+1}$ according to $M_{j_1}$ dynamics
9:     Get reward $r_i$ according to $M_{j_1}$ reward function
10:    Update Q value $Q(j_c, s_i, a_i)$
11:    $e_i = \langle s_i, s_{i+1}, r_i \rangle$
12:    $\tau$ = ODCP($\{e_t : \tau^* \le t \le i\}$)
13:    **if** $\tau$ is not Null **then**
14:        Increment $c$
15:        $\tau^* = \tau$
16:    **end if**
17: **end for**
   ...
18: **for** $i = T_{n-1}$ to $T_n$ **do**
19:    Get $a_i$ according to $\epsilon$-greedy or UCB exploration
20:    Obtain next state $s_{i+1}$ according to $M_{j_n}$ dynamics
21:    Get reward $r_i$ according to $M_{j_n}$ reward function
22:    Update Q value $Q(j_c, s_i, a_i)$
23:    $e_i = \langle s_i, s_{i+1}, r_i \rangle$
24:    $\tau$ = ODCP($\{e_t : \tau^* \le t \le i\}$)
25:    **if** $\tau$ is not Null **then**
26:        Increment $c$
27:        $\tau^* = \tau$
28:    **end if**
29: **end for**
```

The Context QL algorithm takes as input the pattern of changes in the environment models $M_1, \ldots, M_k$, so that the correct set of Q values are updated. For e.g. suppose the agent knows that model changes from say M_0 to M_1 and then to M_2. Then Context QL updates Q values pertaining to model M_0 initially. Later when first change is detected, it updates Q values of M_1, followed by updates to Q values of M_2 when another change is detected.

The algorithm initializes a context counter c, which keeps track of the current active context, according to the changes detected. It maintains Q values for all known contexts $1, \ldots, k$ and initializes the values to zero. The learning begins by obtaining experience tuples e_t according to dynamics and reward function of context M_{j_1}. The state and reward obtained are stored as experience tuples, since model/context information is not known. The samples can be analyzed for context changes in batch mode or online mode, which is denoted as a function call to ODCP in the algorithm pseudocode. If ODCP detects a change, then the counter c is incremented, signalling that the agent believes that context has changed. The lines 6-16 represent this learning phase when context M_{j_1} is active. Similar learning takes place for other contexts as well (upto line 27).

In the next section, we describe numerical experiments to validate the effectiveness of Algorithm 1 in non-stationary environments. Our setup ranges from simulations on random MDP models to previously proposed applications that utilize classical RL algorithms to optimize performance of certain physical systems.

V. Experimental Results

In this section, we evaluate our method for accuracy in the changepoints detected and the reward accrued compared to what the oracle strategy yields. The experience tuples are collected from randomly-generated MDPs (in Section V-1), from a sensor application (see Section V-2) and a traffic application (see Section V-3). All numerical experiments are carried out using R statistical package and Python programming language.

We also compare the accuracy of changepoints detected by ODCP [15] and E-Divisive (ECP) [17] for the data consisting of experience tuples. ECP [17] is also a changepoint detection algorithm for multivariate data and uses a classical divergence measure between two distributions based on characteristic functions and the energy statistic.

1) Random MDP: We test our method on different Random MDP models generated using *MDPtoolbox* [19]. First, the methods are tested for single changepoint detection followed by multiple changepoints detection. The results below are grouped according to this.

(a) *Single Changepoint Detection:* We consider model changes from M_0 to M_1. Let τ^* be the changepoint detected by all the methods. Table I summarizes the results. It shows the mean, median and standard deviation of the changepoints averaged over 20 monte carlo simulations. All the setups analyze 2000 samples (i.e., experience tuples, 3-dimensional), with actual changepoint $T_1 = 1000$. We collect samples using a randomized policy (Section IV-A3) with $\epsilon = 0.1$ for both model-based and model-free cases.

Algorithms, Changepoint	Mean of τ^*	SD of τ^*	Median of τ^*
Model-based, ODCP with ϵ-policy	1008	24	1007
Model-based, ECP with ϵ-policy	1003	84	1000
ODCP, QL with UCB	1045	76	1039
ECP, QL with UCB	1080	100	997
ODCP, QL with ϵ-greedy	1009	36	1004
ECP, QL with ϵ-greedy	997.16	44	994

TABLE I: Performance comparison of our method with QL and ϵ-policy sampling and ECP.

The average regret for the first two setups, i.e., 2000 samples with equal number of samples for M_0 and M_1 is shown below:

Method	Mean	SD	Median
Model-based, ODCP ϵ-policy	20	15	16.6
Model-based, ECP ϵ-policy	39	26	52

TABLE II: Regret of ECP and ODCP ϵ-policy based method.

(b) *Multiple Changepoints Detection:* We evaluate accuracy of [15] and [17] on MDP for multiple changepoints. In the experiments, model alternates thrice between M_0 and M_1 starting with M_0. With 2000 samples, changepoints are fixed at $T_1 = 500$, $T_2 = 1000$ and $T_3 = 1500$. Averaged over 20 monte carlo simulations, mean of $\tau_1^* = 520$, mean of $\tau_2^* = 1059$ and mean of $\tau_3^* = 1510$ for our method, while ECP identifies only the first changepoint with mean 855. ECP fails to detect the second and third changepoints.

2) Energy management (EM) in a single sensor node with finite buffer: We consider the model described in [20] which proposes a EM MDP model for a sensor node with energy harvesting capabilities. Sensor node has a finite energy buffer to store the energy harvested from the ambient environment and a finite data buffer to store the data generated. The authors assume that energy units are harvested at a mean rate of λ_E, while data bits are generated at a mean rate of λ_D. The state of the system comprises of the current levels of energy and data buffers and the RL agent needs to decide on the number of energy units to be used for transmission. The actual number of data bits transmitted is a non-linear function of the number of energy units utilized. The RL agent needs to minimize the long-term discounted cost by finding a suitable policy. The immediate cost per step is the queue length of data buffer after successful transmission. In [20], model information is unknown and hence QL is used to find optimal EM policies.

A sensor which is designed to harvest energy from the ambient environment, like for e.g., solar energy, has to appropriately modify its policy based on how λ_E changes with day timings. We assume that the sensor monitors a physical system which generates data at a fixed rate that does not change over time.

A change in λ_E gives rise to non-staionary environments. We consider this scenario and show that our method is effective in handling changing mean rate of energy harvest, when compared to QL, RUQL. In our experiments, the exploration strategy used is ϵ-greedy with $\epsilon = 0.1$. We analyze our method and QL, RUQL for regret when the environmental model changes once. The number of iterations for learning phase is set to 4000 with a changepoint at 2000. The results are below:

Method	Mean	SD	Median
Context QL	498.75	48.78	500
RUQL	803.7	121.3673	800.5
QL	675.5	50.96	677

TABLE III: Regret performance of our method, RUQL and QL with ϵ-greedy exploration.

3) Traffic Signal Control: As highlighted in Section I, vehicular traffic signal control is a sequential decision-making problem. In the experiments, we show that our method is effective in finding changes in vehicular patterns and learn the optimal policies for the same. The experimental setup consists of a single junction with 4 incoming lanes. The traffic junction is illustrated in Fig. 2, Fig.3 which are snapshots of the simulations carried out in VISSIM [21]. The junction is controlled by a signal. We model the traffic signal duration control as a MDP following [1]. The state of the junction is the information consisting of queue lengths of all incoming lanes and the current phase. The phase indicates which incoming lane should be given the green signal. In order to tackle the state space dimensionality, we aggregate the queue lengths of lanes as low $= 0$, medium $= 1$ and high $= 2$. Hence, if a lane congestion level is one-third of its length, then we say that the aggregated state of that lane is 0. If the lane congestion level is higher than one-third of the lane distance but lesser than two-thirds the distance, then aggregated state of that lane is 1. For high congestion levels, the aggregated state is 2. With this lane queue length aggregation scheme, the state space dimensionality is reduced to $3^4 \times 4$. The actions for the signal controller is a set of green signal durations $\{20, 25, \dots, 70\}$ in seconds. The immediate cost is the sum of the lane queue lengths and the RL agent must minimize the long-term discounted cost.

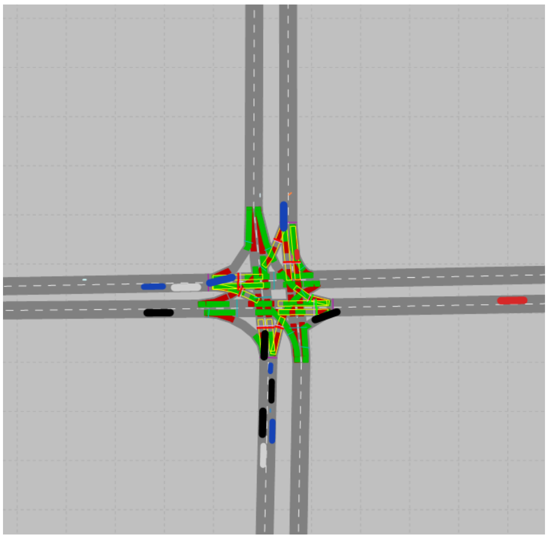

Fig. 2: Illustration of the vehicular traffic junction - the junction has 4 incoming lanes, each having a lane distance of 150 m. The green coloured areas at the junction indicate conflict zones, while the red coloured areas indicate reduced-speed zones. As seen, the vehicular input volume on the lanes is low.

The traffic RL agent learns a policy using QL. We train the traffic RL agent for 10^6 simulation seconds with a change in vehicular input volumes after the simulation has run for half the time. A change in the input vehicular volumes causes a change in the environment dynamics. The implications of the varying input vehicular volumes is illustrated in Fig. 2 and 3. The results for regret are as shown below:

Method	Mean	SD	Median
Context QL	1100.022	34	1000
QL	1400.1	67	1300

TABLE IV: Regret performance of our method and QL with ϵ-greedy exploration.

4) Discussion: As observed in Table I, III, the results of our method are promising. The average regret for our method is less than regret of ECP, which shows that our method can be used as an optimal control method to reduce losses incurred in applications with changing P and R. Our method basically captures the change in joint distribution of (s, a, r) tuples. The

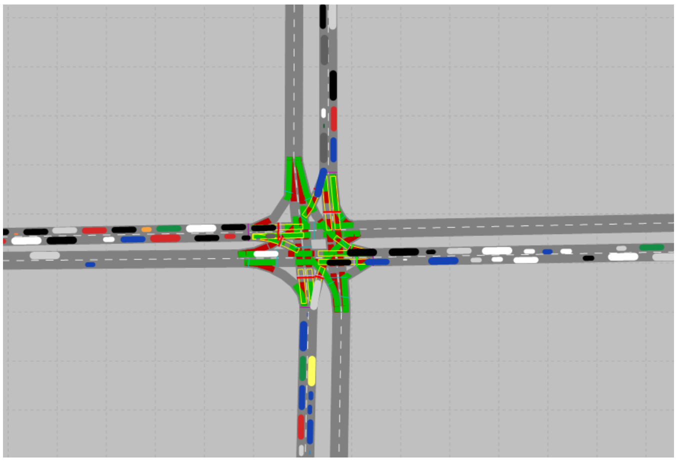

Fig. 3: A snapshot of the simulation with high vehicular input volume on all lanes. This scenario is encountered after environment model dynamics (P) change due to a change in vehicular input volumes.

testing of statistical significance of Z^* in [15] is based on permutation tests and is hence flexible. Moreover, our method is model-free and detects changes in P and R with a unified technique. ECP method can also be used to obtain state-reward tuples. However, the divergence measure used for estimating a candidate changepoint in ECP is based on euclidean distances (see [17]) and is not suitable for our scenario. Hence ECP is unable to detect changepoints in certain cases and finds multiple changepoints in sensor EM MDP case. Also, in simulation runs concerning single changepoint detection, it was observed that ECP does not consistently find changepoints - in some runs it finds none and in other runs it finds too many changepoints.

VI. Conclusion

This work develops a RL method based on QL for changes in environment models. A novel change detection algorithm for experience tuples is used to determine changes in environment models. The numerical experiments in various applications show that the method is promising since the change is detected quickly without causing high loss in rewards collected.

References

[1] L. A. Prashanth and S. Bhatnagar, "Reinforcement learning with average cost for adaptive control of traffic lights at intersections," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct 2011, pp. 1640–1645.

[2] M. Andrychowicz *et al.*, "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[3] D. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Belmont,MA: Athena Scientific, 2013, vol. II.

[4] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2005.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[6] S. P. Choi, D.-Y. Yeung, and N. L. Zhang, "An Environment Model for Nonstationary Reinforcement Learning," in *Advances in neural information processing systems*, 2000, pp. 987–993.

[7] ——, "Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making," in *Sequence Learning*. Springer, 2000, pp. 264–287.

[8] B. C. da Silva, E. W. Basso, A. L. C. Bazzan, and P. M. Engel, "Dealing with Non-stationary Environments using Context Detection," in *Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 217–224.

[9] B. C. Csáji and L. Monostori, "Value function based reinforcement learning in changing markovian environments," *J. Mach. Learn. Res.*, vol. 9, pp. 1679–1709, jun 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1390681.1442787

[10] T. Banerjee, M. Liu, and J. P. How, "Quickest change detection approach to optimal control in markov decision processes with model changes," in *2017 American Control Conference, ACC 2017, Seattle, WA, USA, May 24-26, 2017*, 2017, pp. 399–405.

[11] E. Hadoux, A. Beynier, and P. Weng, "Sequential Decision-Making under Non-stationary Environments via Sequential Change-point Detection," in *Learning over Multiple Contexts (LMCE)*, Nancy, France, Sep. 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01200817

[12] A. Shiryaev, "On optimum methods in quickest detection problems," *Theory of Probability & Its Applications*, vol. 8, no. 1, pp. 22–46, 1963.

[13] S. Abdallah and M. Kaisers, "Addressing environment non-stationarity by repeating q-learning updates," *Journal of Machine Learning Research*, vol. 17, no. 46, pp. 1–31, 2016. [Online]. Available: http://jmlr.org/papers/v17/14-037.html

[14] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[15] K. J. Prabuchandran, N. Singh, P. Dayama, and V. Pandit, "Change Point Detection for Compositional Multivariate Data," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1901.04935

[16] J. Y. Yu and S. Mannor, "Online learning in markov decision processes with arbitrarily changing rewards and transitions," in *2009 International Conference on Game Theory for Networks*, May 2009, pp. 314–322.

[17] D. S. Matteson and N. A. James, "A nonparametric approach for multiple change point analysis of multivariate data," *Journal of the American Statistical Association*, vol. 109, no. 505, pp. 334–345, 2014.

[18] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec 2016, pp. 1–8.

[19] "MDPtoolbox," https://cran.r-project.org/web/packages/MDPtoolbox/MDPtoolbox.pdf.

[20] K. J. Prabuchandran, S. K. Meena, and S. Bhatnagar, "Q-learning based energy management policies for a single sensor node with finite buffer," *Wireless Communications Letters, IEEE*, vol. 2, no. 1, pp. 82–85, 2013.

[21] "PTV Planning Transport Verkehr AG. 2004. User's Manual, VISSIM 4.0, Karlsruhe, Germany. ," http://vision-traffic.ptvgroup.com/en-uk/training-support/support/ptv-vissim/.