Incremental Reinforcement Learning with Prioritized Sweeping for Dynamic Environments

Zhi Wang, Chunlin Chen, *Member, IEEE,* Han-Xiong Li, *Fellow, IEEE,* Daoyi Dong, *Senior Member, IEEE,*
Tzyh-Jong Tarn, *Life Fellow, IEEE*

Abstract—In this paper, a novel incremental learning algorithm is presented for reinforcement learning (RL) in dynamic environments, where the rewards of state-action pairs may change over time. The proposed incremental reinforcement learning (IRL) algorithm learns from the dynamic environments without making any assumptions or having any prior knowledge about the ever-changing environment. First, IRL generates a detector-agent to detect the changed part of the environment (drift environment) by executing a virtual RL process. Then, the agent gives priority to the drift environment and its neighbor environment for iteratively updating their state-action value functions using new rewards by dynamic programming. After the prioritized sweeping process, IRL restarts a canonical learning process to obtain a new optimal policy adapting to the new environment. The novelty is that IRL fuses the new information into the existing knowledge system incrementally as well as weakening the conflict between them. The IRL algorithm is compared to two direct approaches and various state-of-the-art transfer learning methods for classical maze navigation problems and an intelligent warehouse with multiple robots. The experimental results verify that IRL can effectively improve the adaptability and efficiency of RL algorithms in dynamic environments.

Index Terms—Dynamic environments, environment drift, incremental reinforcement learning, intelligent warehouses, prioritized sweeping.

I. INTRODUCTION

REINFORCEMENT learning (RL) [1] is an important approach to machine learning, control engineering, operations research, etc. RL theory addresses the problem of how an autonomous active agent can learn to approximate an optimal behavioral strategy while interacting with its environment.

Z. Wang is with the Department of Control and Systems Engineering, Nanjing University, Nanjing 210093, China and with the Department of System Engineering and Engineering Management, City University of Hong Kong, Hong Kong, China (email: njuwangzhi@gmail.com).

C. Chen is with the Department of Control and Systems Engineering, Nanjing University, Nanjing 210093, China (email: clchen@nju.edu.cn).

H.-X. Li is with the Department of System Engineering and Engineering Management, City University of Hong Kong, Hong Kong, China and with the State Key Laboratory of High Performance Complex Manufacturing, Central South University, Changsha, Hunan, China (email: mehxli@cityu.edu.hk).

D. Dong is with the School of Engineering and Information Technology, University of New South Wales, Canberra, ACT 2600, Australia (email: daoyidong@gmail.com).

T.-J. Tarn is with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA (email: tarn@wuauto.wustl.edu).

In RL, the learning environment of an autonomous agent is often modeled as a Markov Decision Process (MDP) using an interactive cycle of sensing, acting and learning with rewards. RL algorithms, such as temporal difference (TD) [2] and Q-learning [3], have been widely used in intelligent control and industrial applications [4–11]. Recent developments of deep reinforcement learning [12, 13] make RL the state-of-art techniques for artificial intelligence.

Traditionally, the research on RL has been focused on stationary problems, where the environment remains unchanged during the whole learning process. However, in many real-world applications, the environments are often dynamic, where the agent's states, available actions, state transition functions, and corresponding rewards may change over time, such as for robot navigation problems [14] or multi-agent reinforcement learning (MARL) problems [15]. There are mainly two kinds of approaches dealing with RL in dynamic environments. The first kind uses the *trick* of designing parameters, such as a new definition for the state space [14], a new reward evaluation method [16] and the fuzzification of RL parameters [17]. Despite the simplicity, it relies on specific scenarios for pre-processing parameters, and cannot obtain good generalization performance over diverse RL problems in dynamic environments. The other kind is transfer learning [18]. The core idea is that the experience gained in learning to perform a set of source tasks can help improve the learning performance in a related but different task. Many methods have been proposed for inter-task transfer, such as probabilistic policy reuse [19], learning a portable shaping function via shared features [20], policy transfer using reward shaping [21], scalable transfer expectations [22], stochastic abstract policy [23], and Bayesian policy reuse [24]. Transfer learning has recently been investigated in the domain of MARL and learning in dynamic environments, while it requires a relatively large set of source knowledge to obtain a good knowledge base for a target task.

A new mechanism is necessary for learning algorithms to track and adapt to the ever-changing environment. Thus, the concept of incremental learning emerges by fusing the new information into the existing knowledge system. Incremental learning has been widely addressed in machine learning and intelligent control communities to cope with learning tasks where training samples become available over time or the learning environment is ever-changing [25]. Various algorithms have been suggested for incremental learning in different areas to deal with non-stationary data, including unsupervised learning [26], supervised learning [27], machine vision [28, 29], evolutionary algorithms [30, 31], and human-

robot interaction [32]. However, few results have been reported regarding incremental learning for RL problems in dynamic environments. The agent tends to execute the previous converged optimal policy and may be trapped. It is hard to find the real optimal policy adapting to the current new environment. The simplest way is to restart an RL process from scratch whenever an environment change is detected, but it may not be feasible in many data-intensive real applications due to limited memory and computational resources. It should be more efficient to develop new approaches that can fuse the new information into the existing knowledge system in an incremental way to adapt to the ever-changing environment.

In this paper, we address the problem of RL in dynamic environments, where the reward functions may change over time. A novel incremental reinforcement learning (IRL) algorithm is designed to guide the previous optimal policy to a new one adapting to the new environment. First, an RL agent computes the value functions and optimal policy in the original environment. When the environment changes, an RL detector-agent is generated to detect the changed part of the environment, as called *drift environment*. Then, the value functions are updated for the drift environment and its neighbor environment using dynamic programming, which is called *prioritized sweeping of drift environment*. Finally, the agent starts a new RL process with the partly updated value functions to a new optimal policy, aiming at fusing the new information (drift environment) into the existing system (previous optimal policy) in an incremental way.

To comprehensively evaluate the proposed IRL algorithm, various experiments are carried out for classical maze navigation problems and an intelligent warehouse system. We compare the IRL algorithm with two direct methods, including 1) RL without π_{old}^* method: restart a completely new learning process from scratch without using any existing knowledge (i.e. previous optimal polity π_{old}^* from the original environment); 2) RL with π_{old}^* method: restart a learning process directly with existing knowledge, but without prioritized sweeping in advance. In addition, IRL is further compared with various state-of-the-art transfer learning methods.

This paper is organized as follows. In Section II, the basic concepts of RL are introduced and an overview of different approaches is presented for incremental learning in various research areas. In Section III, the framework of IRL is presented, and then related techniques and specific algorithms are given in detail. Experimental results on maze benchmark problems are demonstrated in Section IV and the applications to an intelligent warehouse problem are given in Section V. Finally, Section VI presents concluding remarks.

II. BACKGROUND

In this section, RL is first introduced and then an overview of incremental learning in various areas is provided with the motivation of introducing its counterpart in RL.

A. Reinforcement Learning

Standard RL theories are based on the concept of Markov decision process (MDP). An MDP is denoted as a tuple $\langle S, A, R, P \rangle$, where S is the state space, A is the action space, R is the reward function, and P is the state transition function. A policy is defined as a function $\pi : S \rightarrow Pr(A)$, i.e. a probability distribution in the state-action space. The objective is to estimate the optimal policy π^* that satisfies $J_{\pi^*} = \max_\pi J_\pi = \max_\pi \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in (0, 1]$ is the discount factor and r_t is the reward at time-step t, and J_π is the expected total reward.

MDPs with known state transition functions and reward functions can be solved optimally using dynamic programming. A value function $Q(s, a)$ represents the estimate of the expected return attainable from executing the action a in the state s. Its computation (i.e. value iteration) repeatedly sweeps through the state-action space. The value function of each state-action pair is updated according to

$$Q(s, a) \leftarrow \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma \max_{a'} Q(s', a')], \quad (1)$$

until the largest change, Δ, for any state-action pair is smaller than a preset constant threshold. After the algorithm converges, the optimal policy is obtained by simply taking the greedy action in each state s as $a^* = \arg\max_a Q^*(s, a), \forall s \in S$.

When the agent has no prior knowledge of the environment, Q-learning (a widely used RL algorithm) can achieve optimal policies from delayed rewards. At a time step t, the agent observes the state s_t, and then chooses an action a_t using a certain exploration strategy. After executing the action a_t, the agent receives a reward r_{t+1} (a reflection of how good that action is in a short-term sense), and gets into the next state s_{t+1}. Then the agent will choose the next action a_{t+1}. Let α_t be the learning rate, the one-step updating rule of Q-learning can be described as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)). \quad (2)$$

More details about Q-learning can be found in [1, 3].

B. Incremental Learning

Incremental learning has attracted wide interests from different areas of machine learning and intelligent control. In unsupervised/supervised learning cases, it focuses on continual learning from data which is available only in small batches over a period of time. The new information is learned while the previously acquired knowledge is preserved. Well-performed examples include adaptive resonance theory [26] and learn++ family of algorithms [27]. In the areas of machine vision, evolutionary algorithms and human-robot interaction, incremental learning focuses on learning from and adapting online to dynamic environments. Well-performed instances include incremental principal component analysis for robust visual tracking [28] and face recognition [29], population-based incremental learning for dynamic optimization problems [30], incremental learning for evolutionary-algorithms-based classification [31] and human-robot interaction [32].

As summarized in TABLE I, all these incremental learning schemes incrementally fuse the new information into the existing knowledge system, and emphasize on the learning performance in dynamic environments. However, few practical implementations of incremental learning for RL have

TABLE I
INCREMENTAL ALGORITHMS IN VARIOUS APPLICATIONS

Research Areas	Unsupervised Learning	Supervised Learning	Machine Vision	Evolutionary Algorithms
Incremental Learning	Generating new decision clusters for new patterns that are sufficiently different from previous instances	Classifiers are incrementally trained on coming batches of data and combined with some form of weighted majority voting	Updates the feature space incrementally by rotating the eigen-axes and increasing the dimensions	Competing the best probability reevaluated in the new environment with the current working probability vector for future iterations
Background	Intrinsic batch flow of big data	Intrinsic batch flow of big data	Dynamic environments (change of the objects' appearance)	Dynamic environments (change of fitness functions)
Well-performed Examples	Adaptive resonance theory (ART)	Learn++ family of algorithms	Incremental principal component analysis (IPCA)	Population-based incremental learning (PBIL)

been systematically investigated. Although there are several results proposing the concept of incremental learning in the RL domain [33–36], they focused on the bootstrap property within a single learning process in a stationary environment, which is different from the incremental learning setting for dynamic environments in this paper. Compared to the various areas discussed above, RL has its own properties, such as the characteristics of sequential decision-making and learning from delayed rewards. Hence, it is desired to develop new incremental learning schemes for RL, which motivates our idea of incremental reinforcement learning (IRL) for dynamic environments.

III. INCREMENTAL REINFORCEMENT LEARNING

As an example, imagine a search-and-rescue robot to detect injured people inside damaged buildings. The falling bricks or steels may block the shortest path to the wounded that the robot has already learned. Since relearning the new environment from scratch is too time-consuming, it is reasonable and effective to revise the optimal policy incrementally after learning the changed part first. In this type of problems, incremental learning can be achieved and the learning process can be accelerated to adapt to the dynamic environment. The concept of IRL has been initialized in our conference paper [37] and a formal framework for IRL is presented in detail with related techniques and specific algorithms in this section. In particular, we consider dynamic environments where the reward functions may change over time.

A. Framework for IRL

In a standard Q-learning process, we store the tuple $\langle s, a, r, s' \rangle$ every time the agent interacts with the environment, where r and s' are the reward and next state after executing the action a in the state s. After convergence, we use the set of all tuples to compute the optimal value functions set $Q(S, A)$ and construct the model of the environment $E(S, A, R, P)$. As shown in Fig. 1, the RL agent observes different rewards at time t and t' when executing the same action a in the same state s. We call the dynamic process where the environment is changing over time as "environment drift", and the changed part of the environment is defined as "drift environment".

Different from learning in a static environment, the optimal policy may not always stay stable in dynamic environments.
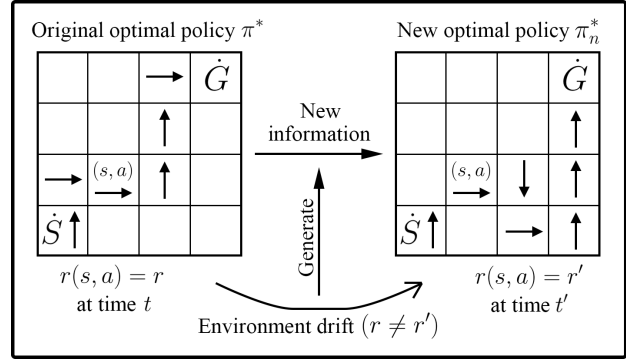

Fig. 1. Illustration of IRL. $\dot{S}$ is the start point and $\dot{G}$ is the goal point; (s, a) is a given state-action pair; the arrows in the grids indicate the optimal actions in corresponding states.

Algorithm 1: Framework for IRL
___
1 Start a Q-learning process to compute the optimal policy π^* and construct the original environment model $E(S, A, R, P)$.
2 Generate a detector-agent to detect the drift environment when the environment changes.
3 Fuse the new information into the existing knowledge system by prioritized sweeping of drift environment.
4 Restart a Q-learning process to compute the new optimal policy π_n^* and construct the new environment model $E_n(S_n, A_n, R_n, P_n)$.
___

Due to environment drift, the RL agent may not be able to reach the target state following the previous optimal policy; or there is an even better policy than the previous optimal one. The objective for IRL in dynamic environments is to revise π^* to a new one π_n^* that adapts to the new environment by fusing the new information (generated by environment drift) into the existing knowledge system (optimal policy learned from the original environment) incrementally.

Unfortunately, if the RL agent directly re-learns a new optimal policy based on the old one, it may be trapped in the new environment since it has converged to an optimum in the original environment. Due to environment drift, there exists a *conflict* between the new information and the existing knowledge, which may cost the agent an extraordinary amount of computational time to fix this conflict. Therefore, it is
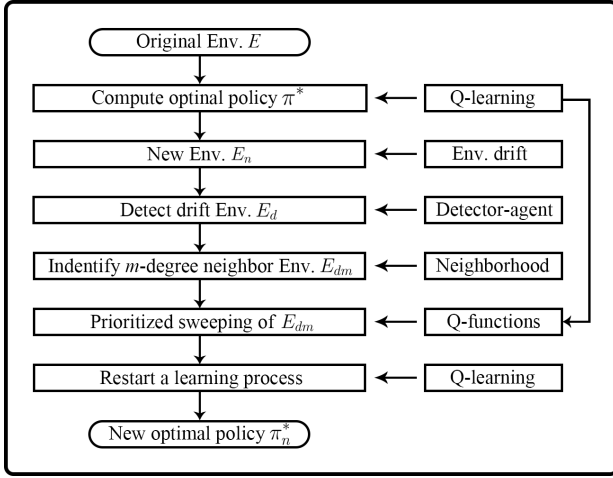
Fig. 2. Flow diagram of the integrated incremental reinforcement learning algorithm. Abbreviation Env. stands for environment.

necessary to first weaken the conflict before making use of the existing knowledge. Since the conflict is caused by environment drift, we primarily update the value functions regarding the drift environment and its neighbor environment using new rewards observed in the new environment. This process is called as *prioritized sweeping of drift environment*, aiming at fusing the new information (new rewards) into the existing knowledge (previous optimal value functions), as well as weakening the conflict between them. Finally, based on the value functions that have been prioritized swept, the agent restarts an RL process to compute the new optimal policy π_n^* adapting to the new environment.

In this paper, we adopt the widely used Q-learning as the implementation of all test algorithms. The framework of IRL is summarized as in Algorithm 1 and illustrated by the flow diagram as shown in Fig. 2, which will be implemented with specific techniques and algorithms introduced in the following subsections.

Remark 1. *IRL shares some similarities with transfer learning regarding transferring knowledge from one scenario (original environment) to another (new learning process). However, transfer learning focuses on providing a good knowledge base for the target task, while IRL emphasizes the temporal property of adding the new information to the existing knowledge system and fine-tuning the previous optimal policy for the new environment. The key representation of such a difference is the prioritized sweeping process of drift environment. It guides the previous optimal policy to a new one, while transfer learning just acquires the transferred knowledge.*

B. Detection of Drift Environment

Environment drift refers to the change of the environment parameters over time. In supervised classification [27], it corresponds to "concept drift", which refers to the change of feature-based probabilities (evidence) of data. In visual tracking research [28], it corresponds to the significant variation of the objects' appearance or surrounding illumination. In evolutionary computation [30], it corresponds to the change of

the fitness function, design variables and environmental conditions. Drift environment generates new data and provides new information for the learner. As RL learns through interactions with the environment, i.e. using the reward r received when the action a is taken in the state s, environment drift in RL implies that the corresponding reward r of a certain state-action pair (s, a) has changed over time. The drift environment is defined as the set of all state-action pairs whose rewards in the new environment differ from those in the original one.

Definition 1 (Drift Environment). *Assume that the original and the new environment models are constructed as $E(S, A, R, P)$ and $E_n(S_n, A_n, R_n, P_n)$, respectively. Given a state-action pair (s, a), $s \in S \cap S_n$, $a \in A \cap A_n$, $r \in R$ and $r_n \in R_n$, if the one-step reward $r(s, a) \neq r_n(s, a)$, then the state-action pair (s, a) has drifted. The set of all state-action pairs that have drifted is defined as "drift environment", and can be formulated as $E_d(S_d, A_d, R_d, P_d)|r_n(s_d, a_d) \neq r(s_d, a_d)$, where $s \in S \cap S_n$, $a \in A \cap A_n$.*

Remark 2. *According to the one-step value iteration of Q-learning, the reward change will influence the value functions of this state-action pair and its neighbor state-action pairs. In the long term of learning, it will guide the previous optimal policy to a new one for the new environment. In this sense, environment drift in RL shares some similarities with "concept drift" in supervised classification [27], where concept drift guides the fine-tuning of the classification boundaries.*

Remark 3. *Drift environment differs from a completely new environment. In IRL, the original environment and the new one share some parts of state-action space. However, in a completely new environment, the state-action space has no guaranteed connections with the original one. An agent should start the learning process from scratch and the learning in a completely new environment requires replacement learning (where existing knowledge becomes irrelevant, i.e. reconstructing), whereas learning with a drift environment requires supplemental learning (where the new information is fused into the existing knowledge system, i.e. fine-tuning).*

In dynamic environments, the learning tasks require some level of feedback about the incoming data or the learning performance at any given time [27]. This feedback is used to discern how the new information contained in the coming data differs from existing knowledge. In IRL, the function of determining when such a change (environment drift) has occurred is known as the *detection of drift environment*. The goal is to detect the part of state-action space whose rewards have changed, i.e. the drift environment E_d. We need to observe the rewards in both the original and the new environments to obtain the changed part. Since the original environment model E has been constructed after the convergence of the previous learning process, we generate a detector-agent to explore the new environment by executing a virtual RL process. The RL detector-agent observes the rewards by fully exploring the new environment with equal probability all the time.

Assume that the total number of all state-action pairs in the original environment is N_{sum}, and the number of the traversed state-action pairs (also contained in the original environment)

Algorithm 2: Detection of Drift Environment

Input: The number of state-action pairs in E: N_{sum}; the exploration threshold ρ

Output: The drift environment E_d

1 In E_n, initialize $\pi : p^\pi(s,a) = \frac{1}{|A(s)|}, \forall s \in S, \forall a \in A(s)$, and $|A(s)|$ denotes the cardinality of the action set
2 Initialize the number of state-action pairs traversed by the detector-agent: $N_e \leftarrow 0$
3 **while** $\frac{N_e}{N_{sum}} \leq \rho$ **do**
4   Initialize $t = 1, s_t$
5   **while** s_t *is not terminal* **do**
6          $a_t \leftarrow$ action a_i with probability $p(s_t, a_i)$
7   Execute a_t, observe r_{t+1} and s_{t+1}
8   **if** $(s_t, a_t) \in E(S, A, R, P)$ **then**
9   **if** (s_t, a_t) *is new to the detector* **then**
10                 $N_e \leftarrow N_e + 1$
11   recall r'_{t+1} in E
12   **if** $r_{t+1} \neq r'_{t+1}$ **then**
13   Mark (s_t, a_t) as a drift pair
14   **end**
15   **end**
16   **end**
17         $t \leftarrow t + 1$
18   **end**
19 **end**
20 Gather all the marked drift pairs to construct E_d

---

explored by the detector-agent is N_e. When $\frac{N_e}{N_{sum}} \geq \rho$, where ρ is a predetermined threshold close to but less than 1, the detector-agent ends the detection process for exploring the new environment. We compare the rewards of state-action space in the new environment with those in the original environment, and construct the drift environment. The detailed drift detection process is summarized as in Algorithm 2.

Remark 4. *The detector-agent aims at exploring the state-action space in the new environment as much as possible, rather than executing a real RL process. Therefore, there is no need to update the value functions and the probabilistic action selection policy. Since the environment has changed, the detector-agent may not be capable of traversing all the state-action pairs contained in the original environment. Hence, in the detection procedure, we usually set the exploration threshold ρ as a constant near to 1 (e.g. 0.95). In order to achieve the predetermined exploration goal, for a given state-action pair in the new environment, the detector-agent needs to traverse it only once. Compared with a standard RL process, the computational complexity of drift detection may be neglected.*

C. Prioritized Sweeping of Drift Environment

In the original environment, we can construct the environment model $E(S, A, R, P)$ with known state transition functions and reward functions. With the environment drift, the new environment has some similarities/connections with the original one, especially those parts that have not drifted over

time. Though we cannot construct the complete model of the new environment temporarily, we can use the predictive model of the original environment to concurrently train the new policy. In the first half of the incremental learning process, the existing knowledge (the original environment model) is used for a new learning process. However, due to the environment drift, the agent tends to execute the previous optimal policy and may be trapped, which will make it hard to find the real optimal policy. In other words, the new information (drift environment) tends to be in conflict with the existing knowledge (previous optimal policy). Hence, a new mechanism is necessary to weaken the conflict in advance and a prioritized sweeping process is applied for the drift environment in the second half of the incremental learning process.

According to the updating rule of Q-learning in Eq. (2), the value functions of the drift environment and its neighbor environment will change in the new environment. In the long term, the change of drift environment influences the whole state-action space from the near to the distant gradually and will lead to a new optimal policy. Since the drift environment is the source of new information, we can update the state-action space of drift environment by dynamic programming with top priority, using the value functions from the original environment and the rewards from the new environment. In this way, the IRL approach fuses the new information into the existing knowledge system (previous optimal value functions), as well as weakening the conflict between them. Then the value functions of the neighbor state-action space will also change under the influence of the drift environment.

Definition 2 (Neighborhood Degree). *The neighborhood degree (denoted as* NEI*) between the state-action pairs* (s, a) *and* (s', a') *is defined as the step distance (i.e. the least steps of actions) from state s to s' with sequential actions starting with action a, where $a \in A(s)$ and $a' \in A(s')$. Specifically, the* NEI *between the same state-action pairs is defined as* 0 *and the* NEI *between two adjacent state-action pairs is* 1.

The set of state-action pairs, whose neighborhood degree to a given pair (s, a) is below or equal to m, is defined as the m-degree neighbor environment of (s, a):

$$E_{nei}^m(S_{nei}, A_{nei}, R_{nei}, P_{nei}) | \text{NEI}[(s_{nei}, a_{nei}), (s, a)] \leq m, \tag{3}$$

where $(s_{nei}, a_{nei}) \in (S_{nei}, A_{nei})$ and $\text{NEI}[\cdot]$ is the neighborhood degree between the two state-action pairs.

As for the drift environment, the set of state-action pairs whose neighborhood degree to any pair (s_d, a_d) in drift environment is below or equal to m is defined as the m-degree neighbor environment of drift environment:

$$E_{dn}^m(S_{dn}, A_{dn}, R_{dn}, P_{dn}) | \text{NEI}[(s_{dn}, a_{dn}), (s_d, a_d)] \leq m, \tag{4}$$

where $(s_{dn}, a_{dn}) \in (S_{dn}, A_{dn})$ and $(s_d, a_d \in (S_d, A_d)$.

After updating the drift environment, we then update the state-action space of its m-degree neighbor environment with the second priority. It can be viewed as a process of spreading the changes caused by the drift environment to its neighbors and even to the whole state-action space gradually. In this process, the new information is fused into the existing knowledge

Algorithm 3: Prioritized Sweeping of Drift Environment

Input: $Q^*(s,a), \forall (s,a) \in (S,A)$ in E;
       a small threshold η
Output: $Q_{dn}(s_{dn}, a_{dn}), \forall (s_{dn}, a_{dn}) \in (S_{dn}, A_{dn})$
1   $E_d \leftarrow$ the drift environment using Algorithm 2
2   $E_{dn}^m \leftarrow$ m-degree neighbor environment
3   Initialize $E_{dn}^m : Q_{dn}(s_{dn}, a_{dn}) \leftarrow Q^*(s_{dn}, a_{dn})$,
    $\forall (s_{dn}, a_{dn}) \in (S_{dn}, A_{dn}) \bigcap (S,A)$
4   Initialize: $\Delta_{max} \leftarrow \infty$
5   **while** $\Delta_{max} \geq \eta$ **do**
6      $\Delta_{max} \leftarrow 0$
7   **for** $(s_{dn}, a_{dn}) \in (S_{dn}, A_{dn})$ **do**
8          $Q_{temp} \leftarrow \sum_{s'_{dn}} p(s'_{dn}|s_{dn}, a_{dn})[$
           $r_{dn}(s_{dn}, a_{dn}, s'_{dn}) + \gamma \max_{a'_{dn}} Q_{dn}(s'_{dn}, a'_{dn})]$
9          $\Delta \leftarrow |Q_{temp} - Q_{dn}(s_{dn}, a_{dn})|$
10         $Q_{dn}(s_{dn}, a_{dn}) \leftarrow Q_{temp}$
11   **if** $\Delta > \Delta_{max}$ **then**
12             $\Delta_{max} \leftarrow \Delta$
13   **end**
14   **end**
15 **end**

Algorithm 4: Incremental Reinforcement Learning

Input: The learning rate α; the discount factor γ
Output: The new optimal policy π_n^* for E_n
1   In E, initialize $Q(s,a), \forall (s,a) \in (S,A)$ and π randomly
2   **while** *Not converged* **do**
3   Initialize $t=1, s_t$
4   **while** s_t *is not terminal* **do**
5   Select a_t using ϵ-greedy or Softmax
6          $\delta_{t+1} \leftarrow r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$
7          $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_{t+1}$
8          $t \leftarrow t + 1$
9   **end**
10 **end**
11 Obtain the optimal policy π^* and value functions Q^*
12 $E_d \leftarrow$ the drift environment using Algorithm 2
13 $E_{dn}^m \leftarrow$ m-degree neighbor environment
14 Sweep E_{dn}^m using Algorithm 3, and obtain Q_{dn}
15 Initialize $E_n, \forall (s_n, a_n) \in (S_n, A_n)$:

$$Q_n(s_n, a_n) \leftarrow \begin{cases} Q_{dn}(s_n, a_n), & \forall (s_n, a_n) \in (S_{dn}, A_{dn}) \\ Q^*(s_n, a_n), & \forall (s_n, a_n) \notin (S_{dn}, A_{dn}) \end{cases}$$

16 Initialize π_n according to Q_n
17 **while** *Not converged* **do**
18   Initialize $t=1, s_{nt}$
19   **while** s_{nt} *is not terminal* **do**
20   Select a_{nt} using ϵ-greedy or Softmax
21          $\delta_{n(t+1)} \leftarrow$
          $r_{n(t+1)} + \gamma \max_{a'_n} Q_n(s_{n(t+1)}, a'_n) - Q_n(s_{nt}, a_{nt})$
22          $Q_n(s_{nt}, a_{nt}) \leftarrow Q_n(s_{nt}, a_{nt}) + \alpha \delta_{n(t+1)}$
23          $t \leftarrow t + 1$
24   **end**
25 **end**

---

system starting from the drift environment. The process of updating the state-action space of the drift environment and its neighbor environment with priority by dynamic programming is called as *prioritized sweeping of drift environment*, which is shown as in Algorithm 3.

Remark 5. *Prioritized sweeping [38] is a specific algorithm that does asynchronous Bellman backups in areas where the value functions would make the largest change. The agent uses each state-action-reward-state experience to not only update the policy, but also simultaneously learn a predictive model of the environment to concurrently train the policy. In the IRL algorithm, compared to the original environment, the value functions of the drift environment would have the largest change in the new environment, and we give top priority to this area for updating. We adopt the name of* prioritized sweeping *in our method regarding the conceptual similarity.*

D. Integrated IRL Algorithm

With the detection and prioritized sweeping techniques for drift environment, an integrated IRL algorithm is shown as in Algorithm 4. First, in the original environment, the RL agent obtains the optimal policy π^* and value functions $Q(s,a), \forall (s,a) \in (S,A)$, and constructs the environment model $E(S,A,R,P)$ through a standard Q-learning process in Lines 1-13. Second, after any environment drift is detected in the form of changed rewards, the drift environment is constructed as $E_d(S_d, A_d, R_d, P_d)$ in Line 14, and generates new data in the new environment. Third, the value functions of the drift part in the new environment tend to have the largest change since the drift environment is the source of new data, while the value functions of the neighbor environment have relatively smaller change on the influence of the drift environment. Therefore, we give priority to the m-degree neighbor

environment of drift environment $E_{dn}^m(S_{dn}, A_{dn}, R_{dn}, P_{dn})$ to sweep the value functions using dynamic programming in Lines 15-16. Finally, we initialize the value functions of the new environment with the combination of the value functions of Q_{dn} and Q^* in Line 17. The part of E_{dn}^m with prioritized sweeping stands for the new information along with the weakened conflict. The part of the original environment E stands for the existing knowledge to accelerate the learning process in the new environment. Based on this mechanism of fusing the new information into the existing knowledge system, the agent restarts a standard Q-learning process and computes the new optimal policy π_n^* in an incremental way in Lines 18-29.

Remark 6. *Generally, there are two more direct methods for RL in dynamic environments: 1) Regenerate a completely new learning process without using any existing knowledge (RL without π_{old}^* method); 2) Restart an RL process directly based on the existing knowledge without prioritized sweeping (RL with π_{old}^* method). In principle, the RL without π_{old}^* method works, since it learns from scratch whenever the environment changes. However, it is infeasible in many data-intensive applications due to limited memory and computational resources.*

On the other hand, the RL with π_{old}^* method would spend an extraordinary amount of time to fix the conflict and get out of the trap of previous optimal policy in spite of making use of existing knowledge. The new learning process of IRL is executed after inexplicitly initializing the value functions computed from the original environment and the sweeping process, which provides a jump-up improvement and a right direction towards the new optimum for the new learning process. The whole learning scheme is a trade-off between exploitation of the designed initialization and exploration of the new environment. In the worst case, IRL is comparable to a standard Q-learning process where we initialize the value functions arbitrarily and learn from scratch, since there is no benefit from the exploitation of the existing knowledge. Therefore, the convergence of the proposed IRL algorithm is the same as that of the classical Q-learning. The learning performance of IRL is at least comparable to a classical Q-learning process, while in many cases, it is much better.

Remark 7. *The proposed IRL algorithm consists of two key parts. One is to make use of the existing knowledge and the other is to fuse the new information into the existing knowledge system. Since we use new rewards observed by the detector-agent in the sweeping process and previous optimal value functions of the original environment, the new information is fused into the existing knowledge system through iteratively updating the value functions of this inexplicit environment model. We use dynamic programming for the prioritized sweeping process with the partly known environment model in spite of its inexplicitness and incompleteness. Thereafter, the RL agent restarts a new learning process and converges to a new optimal policy π_n^* with accelerated speed.*

IV. Experimental Results on Maze Benchmarks

Several groups of experiments on classical maze navigation problems are carried out to evaluate the proposed algorithm. We compare IRL to two direct methods: RL without π_{old}^* method and RL with π_{old}^* method. Additionally, we use the PRQ-learning [19] as the main transfer learning method for comparison, since it is most relevant and comparable to our IRL algorithm. Besides, we also compare IRL with various state-of-the-art transfer learning methods including PTS (policy transfer using reward shaping [21]), SAP (stochastic abstract policy [23]), and BPR (Bayesian policy reuse [24]).

A. Experimental Settings

For all the experiments, the 2D coordinate (i, j) of the map stands for the state s and in each state the agent has four possible actions a: up, down, left and right. When executing an available action a in state s and then transiting to state s', the agent receives an immediate reward $r(s, a)$ of 100 (when the agent reaches the target state), -100 (if the agent hits on the walls or the obstacles and bounces to its previous states) or -0.1 (otherwise). The parameter settings for the one-step Q-learning are the same for all algorithms: learning rate $\alpha = 0.01$, discount factor $\gamma = 0.95$. For each group of experiments, different exploration strategies including ϵ-greedy and Softmax methods [1] are applied to investigate
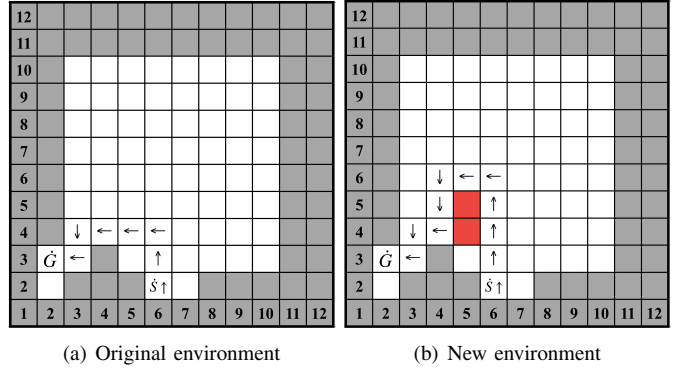

(a) Original environment   (b) New environment

Fig. 3. The simple maze environments.

the performance of the proposed IRL algorithm. For PRQ-learning, $\psi_h = 1.0$, $\upsilon = 0.95$, and τ is initialized to 0, and incremented by 0.05 in each episode. More details about the definitions of these parameters in PRQ-learning can be found in [19]. All the experiments are carried out on an Intel(R) Core(TM) i5-65000U 3.20 GHz PC with 8G main memory under Windows 7. And the experimental results presented in this paper are averaged over 100 runs.

B. Simple Maze

As shown in Fig. 3, the simple maze consists of a 12×12 grid map. $\dot{S} : (6, 2)$ is the start point and $\dot{G} : (2, 3)$ is the goal point, and the gray grids stand for the walls or the obstacles. In the new environment, the grids $(5, 4)$ and $(5, 5)$ are set as obstacles indicating environment drift.

First, we test the algorithms with ϵ-greedy exploration strategy. By experience, we set $\epsilon = 0.1$ first, and after the algorithms converge to around the new optimal policy (about 800 episodes) we decrease it to 0. As shown in Fig. 4, compared with RL without π_{old}^* method, RL with π_{old}^* method achieves an even worse performance, since the conflict is so *serious* that the agent needs more time to fix it. PRQ-learning achieves an intermediate performance between the two direct methods, since it seeks a trade-off between exploring the new environment and exploiting the past policy in the original environment. The two direct methods and PRQ-learning have spent a large number of learning steps before converging to the new optimal policy, while the learning steps of IRL stay very close to the minimum during the whole new learning process. The reason is that, after prioritized sweeping of the drift environment, the initial distribution of value functions has been very close to the optimal one for the new environment and the agent takes only small effort for converging to the new optimal policy.

We also test the algorithms using the Softmax exploration strategy with different initial temperature parameters ($\tau = 30, 25, 20, 15$, respectively). Fig. 5 shows the performance with initial $\tau = 30$ and $\tau = 15$. When τ decreases from 30 to 15 gradually, the results show that: (i) the performance is almost the same for RL without π_{old}^* and PRQ-learning; (ii) for RL with π_{old}^*, more exploitation of existing knowledge leads to greater conflict between new information and existing knowl-
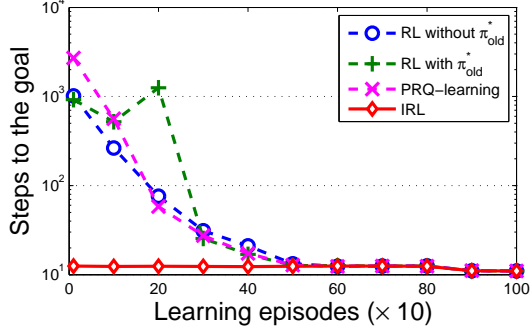
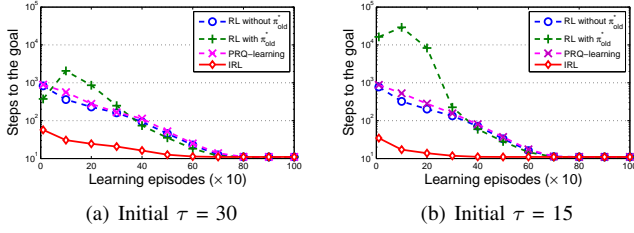Fig. 4. Performance of ϵ-greedy exploration strategy in the simple maze.

(a) Initial $\tau = 30$

(b) Initial $\tau = 15$

Fig. 5. Performance of the Softmax exploration strategy in simple maze with different initial values of the temperature parameter τ.

(a) Original environment

(b) New environment

Fig. 6. Implementation of a hexapod robot navigating in the simple maze.

edge, and the learning performance decreases significantly as the initial exploration ratio τ decreases; (iii) for IRL, with weakened conflict in advance, more exploitation of existing knowledge will accelerate the learning process and the learning performance tends to improve a little as the initial exploration ratio τ decreases.

Compared with the ϵ-greedy strategy, the Softmax strategy depends more on the distribution of the value functions (the existing knowledge). The process of prioritized sweeping of drift environment aims at adjusting the previous distribution of value functions in order to guide it towards the right optimal one in the new environment. Therefore, by means of closer relationship with the distribution of value functions, IRL with Softmax exploration strategy obtains better results than that with ϵ-greedy strategy from the exploitation of the existing knowledge and the prioritized sweeping process.

The learning results are further implemented using a CR-6 hexapod robot in the same maze environments as shown in Fig. 3. The specifications of the CR-6 hexapod robot are: 1) three degrees of freedom (DOF) in each leg; 2) an HC-SR04 sonar sensor in each leg, with a sensitive range of $0.02 \sim 4.50$m; 3) an OV7670 camera at the top, with a rate of 30 frame/s, 0.3M pixels and the USB interface. In practice, the mobile robot scans the room using the camera to draw 2D maps of the environments. The maps obtained by the simultaneous localization and mapping (SLAM) are rasterized and imported to the computing platform for learning the corresponding optimal policies. Following the learned policies, the robot walks from the start point to the goal by the control units. Consistent with the simulation part, the real navigation performance is shown as in Fig. 6 and successfully verifies the learned optimal policies in the original and the
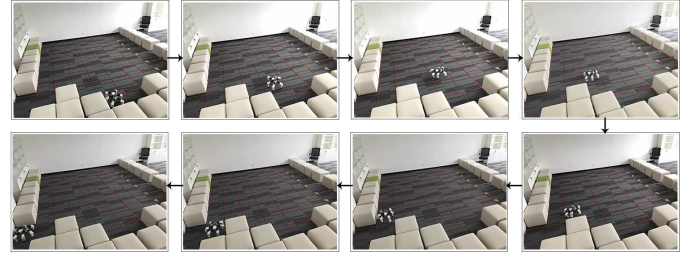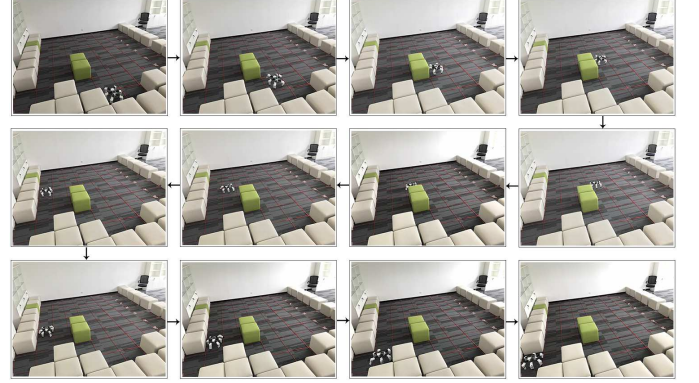
new environments, respectively.

C. Benchmark Maze

A 22×22 maze problem [38] as shown in Fig. 7 is adopted to test the proposed IRL algorithm for complex problems. As an example, the grid $(11, 9)$ can be set as an obstacle indicating environment drift, which leads to previous converged optimal policy to a new one in the new environment. We utilize a statistical analysis method to address the issue of stochastic environment drift. In each test, we randomly select the grid cells to change: a blank cell becomes an obstacle or an obstacle becomes navigable. Both the dynamic drifts can be attributed to the type of reward change. The number of cells changed at one time is no more than 3. We repeat the process for 30 times and average the results to demonstrate the performance of all the test algorithms.

In the first stage, we compare our IRL algorithm to the two direct methods and PRQ-learning. As shown in Fig. 8, the advantage of IRL for the complex maze example is even greater than that for the simple maze example regarding the computational time, since IRL can exploit more from the existing knowledge in larger state-action spaces. RL with π_{old}^* method has better performance than RL without π_{old}^* method, which is opposite to the simple maze case. Intuitively, in the complex maze with a larger state-action space, the conflict is not so serious and the agent still benefits a little from exploiting existing knowledge in spite of fixing the conflict with many trials. PRQ-learning still gains an intermediate performance between the two direct methods.

In the second stage, the proposed IRL algorithm is compared to various state-of-the-art transferring learning methods
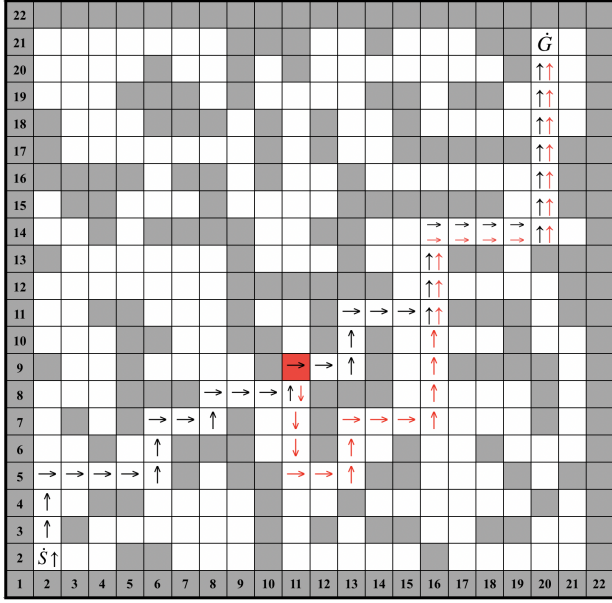
Fig. 7. Model of the complex maze. Grid $(11, 14)$ indicates the environment drift, which leads to the previous optimal policy (black arrows) changing to a new one (red arrows).
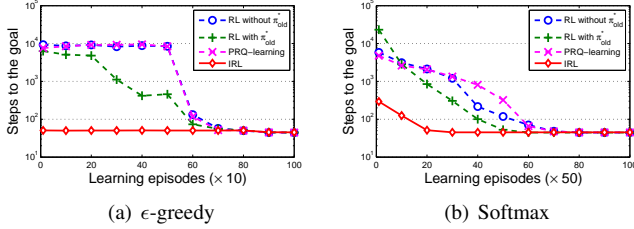

Fig. 8. Performance with different exploration strategies in the complex maze.

using both exploration strategies, as shown in Fig. 9. It can be observed that all the tested transfer learning algorithms did not achieve good performance compared with the IRL algorithm. Transfer learning aims at transferring the *universal* knowledge learned from a set of source tasks to a target task. It helps improve the learning performance in a *generalization* sense. PRQ-learning, as well as other most advanced transfer learning algorithms, is intrinsically a trade-off between exploitation of the transferred past policies (RL with π_{old}^*) and exploration of the new environment (RL without π_{old}^*). Hence, it does not help improve the learning performance in these experiments. In addition, the *negative* transferred knowledge may hinder the new learning process due to the conflict. On the other hand, IRL first detects the drift part that potentially contains negative transfer and then it adjusts the previous optimal policy to a new one towards the right direction.

Additionally, we estimate the average running time of all the tested algorithms as shown in TABLE II. It can be clearly observed that, compared to all the other algorithms, our IRL algorithm achieves a significant reduction in running time and accelerates the RL process in dynamic environments. With respect to the theoretical analysis in Section III-B, it can be verified experimentally that the drift detection process requires
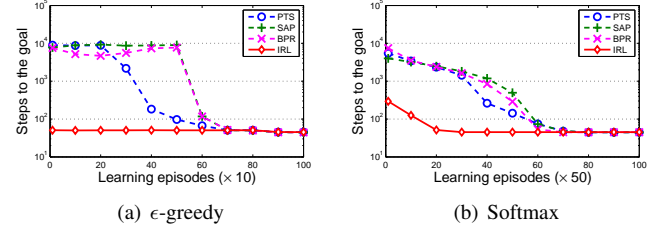

Fig. 9. Further comparison with various transfer learning algorithms.

TABLE II
RUNNING TIME (SECONDS) OF ALL TESTED ALGORITHMS. D1: RL without π_{old}^*, D2: RL with π_{old}^*, DD: drift detection, PS: prioritized sweeping.

	D1	D2	PRQ	PTS	SAP	BPR	IRL DD	IRL PS	IRL RL
ϵ-greedy	84.7	26.5	103.4	43.2	101.9	77.2	1.1	1.6	1.0
Softmax	150.4	575.7	125.8	109.2	146.5	226.5	1.1	3.6	6.3

only a relatively small amount of computational time.

## V. APPLICATION TO AN INTELLIGENT WAREHOUSE

To further test the performance of the proposed IRL algorithm, we apply it to an intelligent warehouse system. Recently multi-robot systems have been widely studied for various potential applications [39, 40] and autonomous guided vehicles (AGVs) have been used to perform tasks in intelligent warehouses for more than fifty years [41, 42]. For example, as shown in Fig. 10, the large-scale Kiva robots system [41] has been a promising application for commercial automatic transportation in Amazon warehouses in recent years.

In this paper we adopt a 12-robot simulation platform as an example to test the proposed IRL algorithm. As shown in Fig. 11, each storage shelf consists of several inventory pods and each pod consists of several resources. By order, a robot lifts and carries a pod at a time along with a pre-planned path from the starting point S_i to the goal G_i ($i = 1, 2, ..., 12$), delivers it to the specific service stations which are appointed in the order and finally returns the pod back. In real applications, the storage shelves and the picking stations may change over time. Moreover, in multi-agent reinforcement learning (MARL) problems [15], the environment of an RL robot may change due to the movements of other RL robots. All these factors will cause environment drift (rewards change) and the mobile robots have to adapt to dynamic environments in the real world.
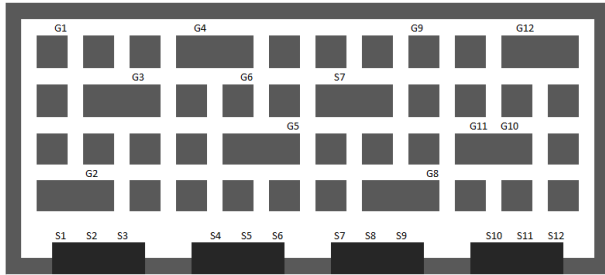
We test the IRL algorithm on all the 12 robots in the simulated dynamic environment, and average the performance of the 12 robots as shown in Fig. 12. All the experimental settings are the same as those introduced in Section IV-A. By comparison, IRL gains a much better performance regarding the average learning steps in each episode towards the optimal policy. Compared with the RL without π_{old}^* method, IRL can make use of existing knowledge so that the agent does not have to spend too many trials to explore the new environment from scratch. Compared with the RL with π_{old}^* method, IRL can weaken the conflict in advance so that the agent does not

Fig. 10. A portion of a Kiva warehouse [41].
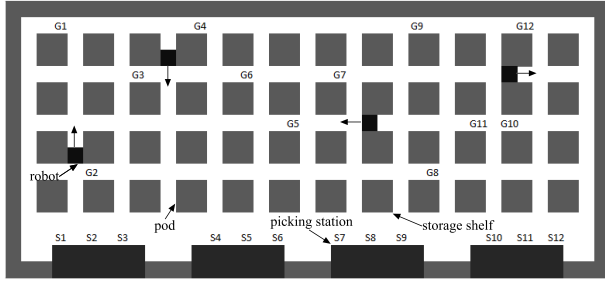
(a) Original environment

(b) New environment

Fig. 11. Model of a Kiva intelligent warehouse system with 12 mobile robots.
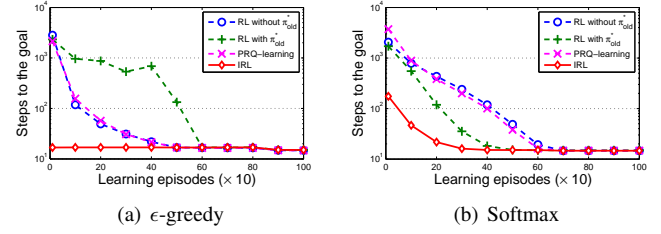

(a) ϵ-greedy

(b) Softmax

Fig. 12. The averaged learning performance of 12 robots with different exploration strategies for the intelligent warehouse system.

to implement a canonical learning process to obtain a new optimal policy adapting to the new environment. An IRL agent fully utilizes the existing knowledge system while avoiding being trapped by the previous converged policy, thus enabling a fast adaptation to the changes of the external environment. The proposed IRL algorithm provides an appealing option for saving a significant amount of computational resources, while the dynamic environment scenario is supposed to hold for many challenging real-world domains.

We adopt the widely used Q-learning algorithm as the basic implementation for IRL, which can be extended to other specific RL algorithms (e.g. SARSA [43]) in principle. Our future work will focus on the statistical analysis of IRL in more complex dynamic environments and automatical optimal selection of the neighborhood degree m.

need to spend a large amount of time to fix the conflict and get out of the trap of the previous optimal policy. Compared with PRQ-learning and other transfer learning methods, IRL takes both of the two advantages. By means of fusing the new information into the existing knowledge system and weakening the conflict between them in advance, IRL learns the non-stationary knowledge over time in an incremental way and dramatically improves the adaptability and efficiency of RL in dynamic environments.

VI. CONCLUSIONS

In this paper, we propose a systematic incremental reinforcement learning (IRL) algorithm under the challenging scenario where the learning environment changes in the reward function. First, a detector-agent is generated to detect the drift environment by executing a virtual RL process. Then, IRL gives priority to the drift environment and its neighbor environment for value iteration, using the newly observed rewards. After this priority sweeping process, IRL continues

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction (2nd Edition)*. MIT press, Cambridge, MA, 2018.

[2] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

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

[4] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, no. Dec, pp. 1107–1149, 2003.

[5] K.-S. Hwang, C.-Y. Lo, and G.-Y. Lee, "A grey synthesis approach to efficient architecture design for temporal difference learning," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 6, pp. 1136–1144, 2011.

[6] Y. Wang, H. Lang, and C. W. De Silva, "A hybrid visual servo controller for robust grasping by wheeled mobile robots," *IEEE/ASME Transactions on Mechatronics*, vol. 15, no. 5, pp. 757–769, 2010.

[7] E. Rombokas, M. Malhotra, E. A. Theodorou, E. Todorov, and Y. Matsuoka, "Reinforcement learning and synergistic control of the act hand," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 2, pp. 569–577, 2013.

[8] Y. Wang and C. W. de Silva, "Sequential Q-learning with Kalman filtering for multirobot cooperative transportation," *IEEE/ASME Transactions on Mechatronics*, vol. 15, no. 2, pp. 261–268, 2010.

[9] D. Dong, C. Chen, H. Li, and T.-J. Tarn, "Quantum reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 5, pp. 1207–1220, 2008.

[10] C. Chen, D. Dong, H.-X. Li, J. Chu, and T.-J. Tarn, "Fidelity-based probabilistic Q-learning for control of quantum systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 5, pp. 920–933, 2014.

[11] D. Dong, C. Chen, J. Chu, and T.-J. Tarn, "Robust quantum-inspired reinforcement learning for robot navigation," *IEEE/ASME Transactions on Mechatronics*, vol. 17, no. 1, pp. 86–97, 2012.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[14] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 1, pp. 135–149, 2011.

[15] L. Zhou, P. Yang, C. Chen, and Y. Gao, "Multiagent reinforcement learning with sparse interactions by negotiation and knowledge transfer," *IEEE Transactions on Cybernetics*, vol. 47, no. 5, pp. 1238–1250, 2017.

[16] A. K. Agogino and K. Tumer, "Analyzing and visualizing multiagent rewards in dynamic and stochastic domains," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 320–338, 2008.

[17] M. Rahimiyan and H. R. Mashhadi, "An adaptive Q-learning algorithm developed for agent-based computational modeling of electricity market," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 5, pp. 547–556, 2010.

[18] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[19] F. Fernández, J. García, and M. Veloso, "Probabilistic policy reuse for inter-task transfer learning," *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 866–871, 2010.

[20] G. Konidaris, I. Scheidwasser, and A. Barto, "Transfer in reinforcement learning via shared features," *Journal of Machine Learning Research*, vol. 13, no. May, pp. 1333–1371, 2012.

[21] A. Fachantidis, I. Partalas, G. Tsoumakas, and I. Vlahavas, "Transferring task models in reinforcement learning agents," *Neurocomputing*, vol. 107, pp. 23–32, 2013.

[22] T. T. Nguyen, T. Silander, Z. Li, and T.-Y. Leong, "Scalable transfer learning in heterogeneous, dynamic environments," *Artificial Intelligence*, vol. 247, pp. 70–

94, 2017.

[23] M. L. Koga, V. Freire, and A. H. Costa, "Stochastic abstract policies: Generalizing knowledge to improve reinforcement learning," *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 77–88, 2015.

[24] B. Rosman, M. Hawasly, and S. Ramamoorthy, "Bayesian policy reuse," *Machine Learning*, vol. 104, no. 1, pp. 99–127, 2016.

[25] H. He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data," *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 1901–1914, 2011.

[26] G. A. Carpenter and S. Grossberg, "The art of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol. 21, no. 3, pp. 77–88, 1988.

[27] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[28] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, vol. 77, no. 1-3, pp. 125–141, 2008.

[29] C.-X. Ren and D.-Q. Dai, "Incremental learning of bidirectional principal components for face recognition," *Pattern Recognition*, vol. 43, no. 1, pp. 318–330, 2010.

[30] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 5, pp. 542–561, 2008.

[31] S.-U. Guan and F. Zhu, "An incremental approach to genetic-algorithms-based classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 2, pp. 227–239, 2005.

[32] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 330–345, 2012.

[33] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," in *Machine Learning Proceedings*. Elsevier, 1994, pp. 226–232.

[34] T. Mori and S. Ishii, "Incremental state aggregation for value function estimation in reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 5, pp. 1407–1416, 2011.

[35] R. Zajdel, "Epoch-incremental reinforcement learning algorithms," *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 3, pp. 623–635, 2013.

[36] T. Taniguchi and T. Sawaragi, "Incremental acquisition of behaviors and signs based on a reinforcement learning schemata model and a spike timing-dependent plasticity network," *Advanced Robotics*, vol. 21, no. 10, pp. 1177–1199, 2007.

[37] Z. Wang, C. Chen, H.-X. Li, D. Dong, and T.-J. Tarn, "A novel incremental learning scheme for reinforcement learning in dynamic environments," in *Intelligent Control and Automation (WCICA), 2016 12th World Congress on*.

IEEE, 2016, pp. 2426–2431.

[38] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, no. 1, pp. 103–130, 1993.

[39] M. A. Neumann and C. A. Kitts, "A hybrid multirobot control architecture for object transport," *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 6, pp. 2983–2988, 2016.

[40] A.-R. Merheb, V. Gazi, and N. Sezer-Uzol, "Implementation studies of robot swarm navigation using potential functions and panel methods," *IEEE/ASME Trans. Mechatronics*, vol. 21, no. 5, pp. 2556–2567, 2016.

[41] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, p. 9, 2008.

[42] X. Zhai, J. E. Ward, and L. B. Schwarz, "Coordinating a one-warehouse n-retailer distribution system under retailer-reporting," *International Journal of Production Economics*, vol. 134, no. 1, pp. 204–211, 2011.

[43] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems.* University of Cambridge, Department of Engineering Cambridge, England, 1994, vol. 37.

Zhi Wang received the B.E. degree in automation from the Department of Control and Systems Engineering, Nanjing University, Nanjing, China, in 2015. He is currently pursuing the Ph.D. degree in Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong, China.

His current research interests include reinforcement learning, machine learning, and robotics.

Chunlin Chen (S'05-M'06) received the B.E. degree in automatic control and Ph.D. degree in control science and engineering from the University of Science and Technology of China, Hefei, China, in 2001 and 2006, respectively. He is currently a professor and the head of the Department of Control and Systems Engineering, School of Management and Engineering, Nanjing University, Nanjing, China. He was with the Department of Chemistry, Princeton University from September 2012 to September 2013. He had visiting positions at the University of New South Wales and City University of Hong Kong.

His current research interests include machine learning, intelligent control and quantum control. He is Co-chair of Technical Committee on Quantum Cybernetics, IEEE Systems, Man and Cybernetics Society.

Han-Xiong Li (S'94-M'97-SM'00-F'11) received his B.E. degree in aerospace engineering from the National University of Defense Technology, China in 1982, M.E. degree in electrical engineering from Delft University of Technology, The Netherlands in 1991, and Ph.D. degree in electrical engineering from the University of Auckland, New Zealand in 1997.

He is a professor in the Department of SEEM, City University of Hong Kong. He has a broad experience in both academia and industry. He has authored 2 books and about 20 patents, and published more than 200 SCI journal papers with h-index 42 (web of science). His current research interests include process modeling and control, system intelligence, distributed parameter systems, and battery management system.

Dr. Li serves as Associate Editor for IEEE Transactions on SMC: System, and was associate editor for IEEE Transactions on Cybernetics (2002-2016) and IEEE Transactions on Industrial Electronics (2009-2015). He was awarded the Distinguished Young Scholar (overseas) by the China National Science Foundation in 2004, a Chang Jiang professorship by the Ministry of Education, China in 2006, and a national professorship in China Thousand Talents Program in 2010. He serves as a distinguished expert for Hunan Government and China Federation of Returned Overseas Chinese. He is a Fellow of the IEEE.

Daoyi Dong received a B.E. degree in automatic control and a Ph.D. degree in engineering from the University of Science and Technology of China, Hefei, China, in 2001 and 2006, respectively. Currently, he is an Associate Professor and Scientia Fellow at the University of New South Wales, Canberra, Australia. He was with the Institute of Systems Science, Chinese Academy of Sciences and with the Institute of Cyber-Systems and Control, Zhejiang University. He had visiting positions at Princeton University, NJ, USA, RIKEN, Wako-Shi, Japan and The University of Hong Kong, Hong Kong.

His research interests include quantum control, multiagent systems and machine learning. Dr. Dong was awarded an ACA Temasek Young Educator Award by The Asian Control Association and is a recipient of an International Collaboration Award and an Australian Post-Doctoral Fellowship from the Australian Research Council. He is also a co-recipient of Guan Zhao-Zhi Award at The 34th Chinese Control Conference, and the Best Theory Paper Award at The 11th World Congress on Intelligent Control and Automation. He serves as an Associate Editor of IEEE Transactions on Neural Networks and Learning Systems. He is the General Chair of the 1st Quantum Science, Engineering and Technology Conference.

Tzyh-Jong Tarn (M'71-SM'83-F'85-LF'05) received the D.Sc. degree in control system engineering from Washington University in St. Louis, St. Louis, MO. He is a Senior Professor in the Department of Electrical and Systems Engineering at Washington University in St. Louis.

Prof. Tarn received the NASA Certificate of Recognition for the creative development of a technical innovation on robot arm dynamic control by computer in 1987. He also received the Best Paper Award at the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. He is the first recipient of both the Nakamura Prize and the Ford Motor Company best paper award at the Japan/USA Symposium on Flexible Automation in 1998. In addition, he is the recipient of the prestigious Joseph F. Engelberger Award of the Robotic Industries Association in 1999, the Auto Soft Lifetime Achievement Award in 2000, and the Pioneer in Robotics and Automation Award in 2003 from the IEEE Robotics and Automation Society.

Prof. Tarn served as the President of the IEEE Robotics and Automation Society (1992-1993), the Director of the IEEE Division X (1995-1996), and was a Member of the IEEE Board of Directors (1995-1996).