A Novel Incremental Learning Scheme for Reinforcement Learning in 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, we develop a novel incremental learning scheme for reinforcement learning (RL) in dynamic environments, where the reward functions may change over time instead of being static. The proposed incremental learning scheme aims at automatically adjusting the optimal policy in order to adapt to the ever-changing environment. We evaluate the proposed scheme on a classical maze navigation problem and an intelligent warehouse system in simulated dynamic environments. Simulation results show that the proposed scheme can greatly improve the adaptability and applicability of RL in dynamic environments compared to several other direct methods.**

I. Introduction

REINFORCEMENT learning (RL) [1] has attracted many research interests in machine learning, operations research, control engineering, and other related disciplines. In RL, the learning agent interacts with an initially unknown environment and modifies its action policies to maximize its cumulative payoffs. RL algorithms, such as temporal difference (TD) algorithms [2] and Q-learning algorithms [3], have been widely investigated in intelligent control [4]-[8], and more state-of-art, practical applications with large state-action spaces [9]-[11].

Traditional RL techniques have focused on stationary tasks, whose environments is fixed during the whole learning process. As the real-world situation is more dynamic and keeps changing, a RL agent is actually exposed to a changing environment and its solution needs to evolve to adapt to various changes [12]-[13]. To deal with the changing environment, an agent has to learn non-stationary knowledge incrementally and adapt to the new environment gradually. In recent years, incremental learning has attracted much research attention in the literature. Incremental learning has been widely addressed in machine learning and intelligent control communities to cope with learning tasks where the training samples become available over time or the learning environment is ever-changing, including unsupervised learning [15], supervised learning [16], machine vision [17], evolutionary algorithms [18] and human-robot interaction [19].

This work was supported by the National Natural Science Foundation of China (Nos. 61273327, 61374092 and 61432008), by the GRF project from RGC of Hong Kong (CityU: 11205615) and by the Australian Research Council (DP130101658).

Z. Wang is with the Department of Control and Systems Engineering, School of Management and 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 and with the Research Center for Novel Technology of Intelligent Equipments, 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@gmail.com).

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).

There are few existing results reported regarding the incremental learning for RL due to the dynamic environments. In this paper, a novel incremental learning scheme is proposed for RL in dynamic environments. The novel approach is designed to guide the previous optimal policy to a new one adapting to the new environment when the environment changes. First, a RL agent (e.g., Q-learning) computes the value functions and optimal policy in the original environment. When the environment changes, we generate a RL detector-agent to detect the changed part of environment (as called *drift environment*). Then we primarily update the value functions for the drift environment and its neighboring 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 adapting to the new environment, aiming at fusing new information (drift environment) into the existing knowledge system (previous optimal policy) in an incremental way.

In order to demonstrate the performance of the incremental learning scheme for dynamic environments, simulated experiments are carried out for a classical maze navigation problem and an intelligent warehouse system. We compare the proposed algorithm with two direct methods, including 1) RL without π^*_{old} method: restart a completely new RL learning process from scratch without using any existing knowledge (i.e., previous optimal polity π^*_{old} for the original environment); 2) RL with π^*_{old} method: restart a RL process directly with existing knowledge, but without prioritized sweeping in advance.

The rest of this paper is organized as follows. Section II provides some backgrounds of RL. Section III presents the incremental learning scheme for RL in dynamic environments. Simulation results are demonstrated in Section IV. Conclusions are given in Section V.

II. Reinforcement Learning

Standard RL theories are based on the concept of Markov decision process (MDP). A 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 probability. The goal of RL is to learn an optimal policy π^*, so that the expected sum of discounted reward of each state will be maximized

$$J_{\pi^*} = \max_{\pi} J_{\pi} = \max_{\pi} E_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t], \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor and r_t is the reward at time-step t, $E_{\pi}[\cdot]$ stands for the expectation with respect to the policy π and the state transition probabilities, and J_{π} is the expected total reward.

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

$$Q(s, a) \leftarrow \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma \max_{a'} Q(s', a')] \tag{2}$$

until the largest change $\varDelta$ in the value of any state-action pair is smaller than a preset constant threshold, where $p(s'|s, a)$ is the probability of state transition from s to s' after executing action a and $r(s'|s, a)$ is the corresponding reward. After the algorithm converges, the optimal policy is followed by simply taking the greedy action in each state s as

$$a^* = \arg\max_{a} Q^*(s, a), (\forall s \in S). \tag{3}$$

As for model-free cases where 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 certain time step t, the agent observes the state s_t, and then chooses an action a_t. After executing action a_t, the agent receives a reward r_{t+1} and gets into the next state s_{t+1}. Then the agent will choose the next action a_{t+1} according to the best known knowledge. Let α_t be the learning rate, the one-step updating rule of Q-learning can be described as

$$Q(s_t, a_t) \leftarrow (1-\alpha_t)Q(s_t, a_t)+\alpha_t(r_{t+1}+\gamma \max_{a'} Q(s_{t+1}, a')). \tag{4}$$

III. Incremental Learning Scheme

RL learns knowledge from delayed rewards when interacting with the external environment. We assume that the environment changes when any changes of reward functions are detected. When the environment changes, the objective for incremental learning in dynamic environments is to guide the original optimal policy π^* to a new one π_n^* that adapts to the new environment by fusing new information into the existing knowledge system incrementally.

A. Environment Drift

Environment drift refers to a change of the environment model over time. It 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.

Assume that $E(S, A, R, P)$ is the original environment model obtained by the RL agent and $E_n(S_n, A_n, R_n, P_n)$ is the new environment model that will be obtained. Given a state-action pair $(s, a), s \in S, a \in A, s \in S_n, a \in A_n$, if $r(s, a) \neq r_n(s, a), r \in R, r_n \in R_n$, which means that when taking action a in state s, the one-step reward $r_n(s, a)$ obtained in the new environment is different from the reward $r(s, a)$ obtained in the original environment, then we get the message that this given state-action pair (s, a) has drifted. The set of all state-action pairs which have drifted is called "drift environment", which can be formulated as

$$E_d(S_d, A_d, R_d, P_d)|r_n(s_d, a_d) \neq r(s_d, a_d), \tag{5}$$

where $s_d \in S$, $a_d \in A$, $s_d \in S_n$ and $a_d \in A_n$.

B. Drift Detection

Algorithm 1 Drift Detection

```
1: $N_{sum} \leftarrow$ number of all state-action pairs in the original environment
2: In the new environment, initialize the probabilistic action selection policy $\pi : P^{\pi}|p^{\pi}(s_t, a_t) = \frac{1}{|A_t(s_t)|}$, where $a_t \in A_t(s_t)$ is any available action in state $s_t$, and $|\cdot|$ is the number of elements in a set
3: Initialize the number of state-action pairs traversed by the detector-agent: $N_e \leftarrow 0$
4: repeat (for each episode)
5:     Initialize $t = 1$, $s_t$
6:     repeat (for each step of episode)
7:         $a_t \leftarrow$ action $a_i$ with probability $p(s_t, a_i)$ for $s_t$
8:         Take $a_t$, observe reward $r_{t+1}$, and next state $s_{t+1}$
9:         if $(s_t, a_t) \in E(S, A, R, P)$ then
10:            if $(s_t, a_t)$ has not been traversed by the detector-agent in the new environment before then
11:                $N_e \leftarrow N_e + 1$
12:                recall $r'_{t+1}$ observed in $E(S, A, R, P)$
13:                if $r_{t+1} \neq r'_{t+1}$ then
14:                    Mark $(s_t, a_t)$ as a drift state-action pair
15:                end if
16:            end if
17:        end if
18:        $t \leftarrow t + 1$
19:    until $s_{t+1}$ is terminal
20: until $N_e/N_{sum} \geq \rho$ (a threshold close to 1)
21: Gather all the marked drift pairs to generate the model of drift environment $E_d$
```

The goal of drift detection 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 both in the original and the new environment to get the changed part. Since the original environment model E has been obtained by the RL agent, the only remaining thing is to observe rewards in the new environment. Due to having no prior knowledge about the new environment, we generate a detector-agent to explore it 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.

Let us assume 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) explored by the detector-agent is N_e. When $\frac{N_e}{N_{sum}} \geq \rho$, where ρ is a preset threshold close to but less than 1, the detector-agent ends the detection process for exploring the new environment. Finally, we compare the rewards of state-action space in the new environment with those in the original environment, and obtain the changed part, i.e., drift environment. The detailed drift detection process is summarized as in Algorithm 1.

C. Prioritized Sweeping of Drift Environment

There has been research on combining the advantages of model-based algorithms and model-free algorithms for accelerating RL learning process, such as Dyna-like frameworks [20] and prioritized sweeping [21]. In those proposed schemes, 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. On the setting of our incremental learning scheme in this paper, though we can not get the explicit model of the new environment temporarily, we can use the predictive model of the original environment to concurrently train the new policy.

Algorithm 2 Prioritized Sweeping of Drift Environment

```
1:  $Q(s,a)(\forall s \in S, a \in A) \leftarrow$ optimal value functions in the original environment
2:  $E^m_{dn} \leftarrow m$-degree neighboring environment of drift environment
3:  Initialize $E^m_{dn}$ : $Q_{dn}(s_{dn}, a_{dn}) \leftarrow Q(s_{dn}, a_{dn})$, where $(s_{dn}, a_{dn}) \in (S_{dn}, A_{dn}), (s_{dn}, a_{dn}) \in (S, A)$
4:  repeat (for each iterative episode)
5:      $(s_{dn}, a_{dn}) \leftarrow$ the first state-action pair in $E^m_{dn}$
6:      $\Delta_{max} \leftarrow 0$
7:      repeat (for each state-action pair in $E^m_{dn}$)
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 if
14:         $(s_{dn}, a_{dn}) \leftarrow$ the next state-action pair in $E^m_{dn}$
15:     until all the pairs in $E^m_{dn}$ have been traversed
16: until $\Delta_{max} < \theta$ (a small threshold)
```

Since the drift environment is the source of environmental changes, we can update the state-action space corresponding to drift environment by dynamic programming with top priority, using the value functions from the original environment and the rewards from the new environment. Then the value functions of the neighboring state-action space will also change under the influence of drift environment.

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 state 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.

We define the set of state-action pairs whose neighborhood degree to anyone pair (s_d, a_d) in drift environment is below or equal to m, as the m-degree neighboring environment of drift environment.

$$E^m_{dn}(S_{dn}, A_{dn}, R_{dn}, P_{dn})|NEI[(s_{dn}, a_{dn}), (s_d, a_d)] \leq m, \tag{6}$$

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 neighboring environment with the second priority. It can be viewed as a process of spreading the changes caused by drift environment to its neighbors and even to the whole state-action space gradually. In this process shown in Algorithm 2, new information is fused into the existing knowledge starting from the drift environment incrementally.

D. Integrated Incremental Learning Algorithm

With the detection and prioritized sweeping techniques for drift environment, an integrated incremental learning scheme is presented as shown in Algorithm 3.

First, in the original environment, the RL agent obtains the optimal policy π^*, value functions $Q(s, a)$ ($\forall s \in S, a \in A$) and the environment model $E(S, A, R, P)$ through a standard Q-learning process (line 1-11 of Algorithm 3).

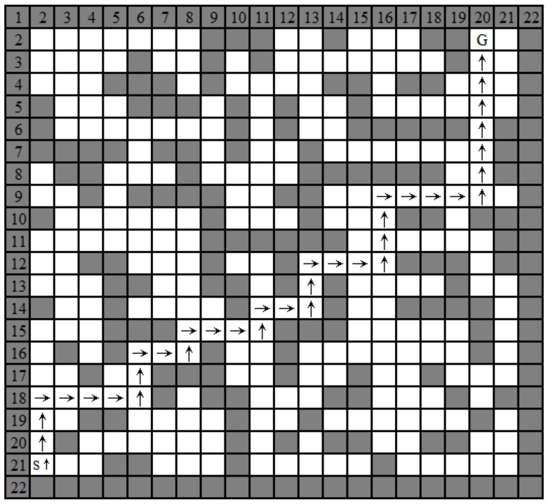
Second, after any environment drift is detected, the new environment differs from the original environment in the form of rewards changing. The different part where the rewards have changed is denoted as drift environment $E_d(S_d, A_d, R_d, P_d)$ (line 12 of Algorithm 3), which generates new data in the new environment.

Third, compared with the original environment, the value functions of the drift part in the new environment tend to have the largest changes since drift environment is the source of new data. Then, the value functions of the neighboring environment will have relatively smaller changes on the influence of drift environment. Therefore, we give priority to the m-degree neighboring environment of drift environment $E^m_{dn}(S_{dn}, A_{dn}, R_{dn}, P_{dn})$ to sweep the value functions using dynamic programming (line 13 of Algorithm 3).

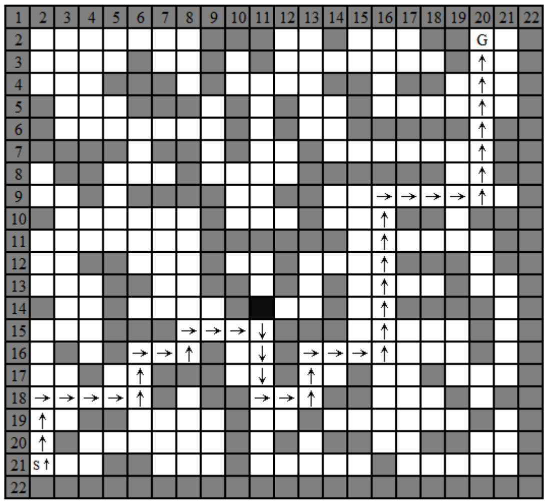
Finally, we initialize the value functions of the new environment with the combination of the value functions of Q_{dn} and Q. The part of E^m_{dn} with prioritized sweeping stands for the new information. The part of the original environment E stands for existing knowledge to accelerate the learning process in the new environment. Based on this mechanism of fusing 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 (line 15-25 of Algorithm 3).

Algorithm 3 Integrated Incremental Learning Algorithm

```
1: In the original environment $E$, initialize $Q(s,a)$ ($\forall s \in S, a \in A$) and the policy $\pi : P^{\pi}$ arbitrarily
2: repeat (for each episode)
3:     Initialize $t = 1, s_t$
4:     repeat (for each step of episode)
5:         $a_t \leftarrow$ action $a_i$ with probability $p(s_t, a_i)$ for $s_t$
6:         Take $a_t$, observe reward $r_{t+1}$, and next state $s_{t+1}$
7:         $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_t(r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t))$
8:         Update policy $\pi$ using $\epsilon-$greedy or Softmax
9:         $t \leftarrow t+1$
10:    until $s_{t+1}$ is terminal
11: until the optimal policy $\pi^*$ is computed
12: In the new environment $E_n$, generate a detector-agent to detect the drift environment $E_d$ using Algorithm 1
13: Obtain the m-degree neighboring environment of the drift environment $E^m_{dn}$, and sweep $E^m_{dn}$ using Algorithm 2
14: Initialize the value functions of the new environment $Q_n(s_n,a_n) (\forall s_n \in S_n, a_n \in A_n)$ :
        $Q_n(s_{dn},a_{dn}) \leftarrow Q_{dn}(s_{dn},a_{dn}), (s_{dn},a_{dn}) \in (S_{dn},A_{dn})$
        $Q_n(s_n,a_n) \leftarrow Q(s_n,a_n), (s_n,a_n) \notin (S_{dn},A_{dn})$
15: Initialize the policy $\pi_n : P_n^{\pi_n}$ according to $Q_n(s_n,a_n)$ ($\forall s_n \in S_n, a_n \in A_n$)
16: repeat (for each episode)
17:     Initialize $t = 1, s_{nt}$
18:     repeat (for each step of episode)
19:         $a_{nt} \leftarrow a_{ni}$ with probability $p(s_{nt}, a_{ni})$ for $s_{nt}$
20:         Take $a_{nt}$, observe $r_{n(t+1)}$, $s_{n(t+1)}$
21:         $Q_n(s_{nt},a_{nt}) \leftarrow Q_n(s_{nt},a_{nt}) + \alpha_t(r_{n(t+1)} + \gamma \max_{a_n'} Q(s_{n(t+1)},a_n') - Q_n(s_{nt},a_{nt}))$
22:         Update policy $\pi_n$ using $\epsilon-$greedy or Softmax
23:         $t \leftarrow t+1$
24:     until $s_{n(t+1)}$ is terminal
25: until the new optimal policy $\pi^*_n$ is computed
```



(a) Original environment



(b) New environment

Fig. 1. The model of the maze navigation.

IV. Simulation Experiments

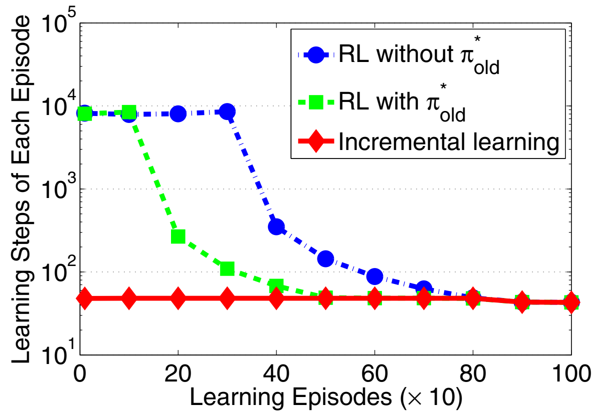
In this section, we present several experiments on a classical maze navigation problem and an intelligent warehouse system with a MATLAB simulation platform to test the proposed incremental learning algorithm. In order to highlight the performance of incremental learning, in each group of experiments we compare it with the two direct methods: RL without π^*_{old} method and RL with π^*_{old} method.

For all these experiments, the 2D coordinate (i, j) of the map stands for the state s and in each state the agent has four available 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 (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 the performance of the proposed incremental learning scheme. All the experimental results presented in this paper are averaged over 100 runs.

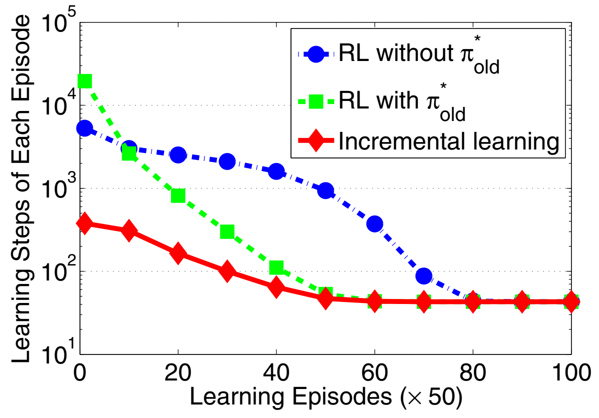
A. Maze Navigation

As shown in Fig. 1, the maze consists of a 22×22 grid map with complicated topography used in the research on prioritized sweeping [21]. After obtaining the optimal path in the original environment, we set the grid $(14, 11)$ as an obstacle to indicate the environment drift, and compute the new optimal policy in the new environment.

As shown in Fig. 2, the two direct methods have spent a large number of learning steps before converging to the new optimal policy, while the learning steps of incremental learning stay very close to the minimum during the whole new learning process. This is because after prioritized sweeping of the drift environment, the initial distribution of value functions has been very close to the optimal one that we shall compute for the



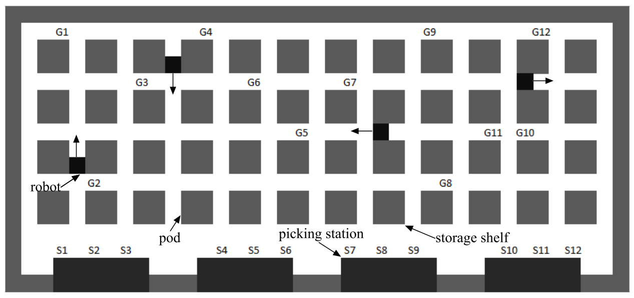
(a) ϵ-greedy



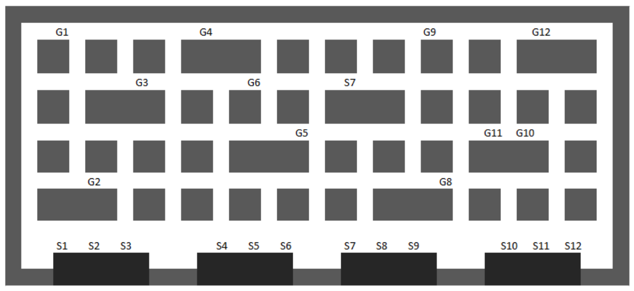
(b) Softmax

Fig. 2. Learning performance of different exploration strategies in the maze.

Fig. 3. Portion of a Kiva warehouse [23].



(a) Original environment



(b) New environment

Fig. 4. The model of a Kiva intelligent warehouse system with 12 mobile robots.

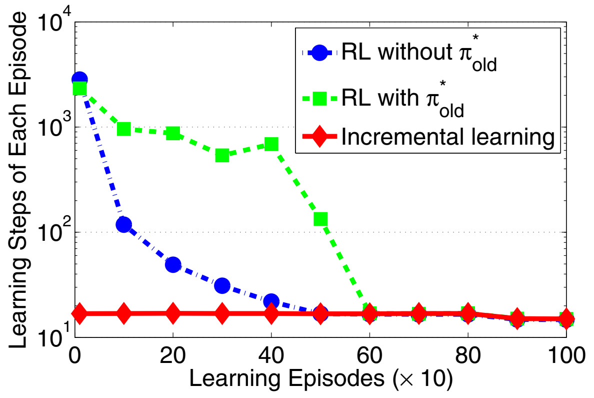
new environment. Hence, the agent takes only a few learning efforts towards convergence to the new optimal policy.

B. An Intelligent Warehouse System Example

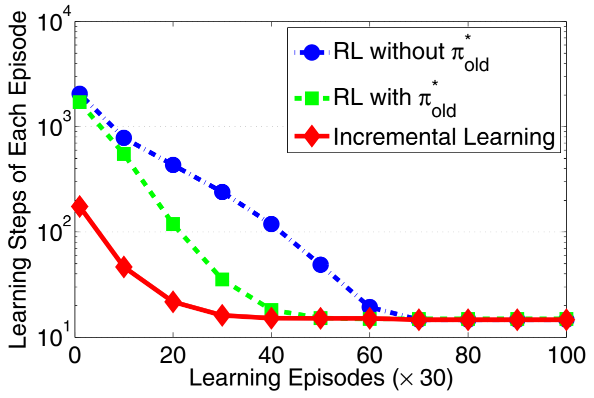
Autonomous guided vehicles (AGVs) have been used to perform tasks in intelligent warehouses for more than fifty years [8]-[14], [22]-[23]. For example, as shown in Fig. 3, the large-scale Kiva robots system [22]-[23] has been a promising application for commercial automatic transportation in Amazon warehouses in recent years. In this paper we make a 12-robot simulation platform as an example to test the proposed incremental learning algorithm.

As shown in Fig. 4, 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 a preplanned path from the starting point S_i to the goal G_i ($i = 1, 2, ..., 12$), deliver 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 [9], [14], the environment of a RL robot may change due to movements of other RL robots. All these factors will cause environment drift. Hence, it requires the capability for the mobile robots to adapt to dynamic environments in the real world.

We test the incremental learning algorithm on all the 12 robots in the simulated dynamic environment, and average the performance of the 12 robots. As shown in Fig. 5, incremental learning gains a much better performance regarding the average learning steps in each episode towards the optimal policy. Compared to RL without π^*_{old} method, the incremental learning scheme can make use of existing knowledge so that the agent do not have to spend too many trials to explore the new environment from scratch. Compared to RL with π^*_{old} method, the incremental learning scheme can guide the previous optimal policy to a new one in an appropriate direction. As shown in TABEL I, by means of fusing new information into the existing knowledge system, incremental learning learns the non-stationary knowledge over time in an incremental way and dramatically improves the adaptability and applicability of RL for dynamic environments.



(a) ϵ-greedy



(b) Softmax

Fig. 5. The averaged learning performance of 12 robots with different exploration strategies for the intelligent warehouse system.

TABLE I
COMPARISON AMONG RL WITHOUT π^*_{old} (RL I), RL WITH π^*_{old} (RL II) AND IRL. AES (E+02), ASS (E+04): AVERAGE EPISODES AND AVERAGE STEPS TOWARDS OPTIMAL POLICY

Scenarios	Maze				Warehouse			
	ϵ-greedy		Softmax		ϵ-greedy		Softmax	
	AEs	ASs	AEs	ASs	AEs	ASs	AEs	ASs
RL I	8	268	8	639	5	9	7	72
RL II	5	101	6	521	6	37	5	60
IRL	2	5	6	52	2	2	4	7

V. CONCLUSIONS

In this paper, we developed a novel incremental learning algorithm for RL in dynamic environments which can automatically track and adapt to the ever-changing environment. The novelty of the proposed incremental learning scheme is the process of prioritized sweeping of drift environment, which guides the previous optimal policy to a new one in an appropriate direction and fuses new information into the existing knowledge system in an incremental way. Experimental results also prove that the incremental learning scheme can dramatically accelerate the learning process in the ever-changing environment. Due to the variability of the dynamic environments in many real-world applications, the incremental learning scheme is of great significance in the way of saving computational resources and adapting to dynamic environments well.

REFERENCES

[1] R. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[2] R. Sutton, "Learning to predict by the methods of temporal difference," *Machine Learning*, vol.3, pp.9-44, 1988.

[3] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol.8, pp.279-292, 1992.

[4] F. Y. Wang, H. G. Zhang and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE Computational Intelligence Magazine*, vol.4, no.2, pp.39-47, 2009.

[5] X. Xu, D. Hu and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE Transactions on Neural Networks*, vol.18, no.4, pp.973-992, 2007.

[6] D. Dong, C. Chen, H. X. Li and T. J. Tarn, "Quantum reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol.38, pp.1207-1220, 2008.

[7] C. Chen, D. Dong, H. X. Li, J. Chu, T. J. Tarn, "Fidelity-based probabilistic Q-Learning for control of quantum systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol.25, pp.920-933, 2014.

[8] 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, pp.86-97, 2012.

[9] L. Busoniu, R. Babuska and B. De Schutter, "A comprehensive survey of multiagent reinforcement learing," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol.38, no.2, pp.156-172, 2008.

[10] J. Ni, M. Liu, L. Ren and S. X. Yang, "A multiagent Q-learning-based optimal allocation approach for urban water resource management system," *IEEE Transactions on Automation Science and Engineering*, vol.11, no.1, pp.204-214, 2014.

[11] J. Wu, X. Xu, P. Zhang, and C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in Grid computing," *Future Generation Comput. Syst.*, vol.27, no.5, pp.430-439, 2011

[12] J. M. A. Kareem, 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.

[13] M. J. Er and C. Deng, "Obstacle avoidance of a mobile robot using hybrid learning approach," *IEEE Transactions on Industrial Electronics*, vol.52, pp.898-905, 2005.

[14] L. Zhou, Y. Shi, J. Wang and P. Yang, "A balanced heuristic mechanism for multirobot task allocation of intelligent warehouses," *Mathematical Problems in Engineering*, vol.2014, Article ID 380480, 2014.

[15] G. A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol.21, pp.77-88, 1988.

[16] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol.22, pp.1517-1531, 2011.

[17] 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, pp.125-141, 2008.

[18] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol.12, pp.542-561, 2008.

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

[20] R. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," *In Proc. 7th International Conference on Machine Learning*, pages 216-224, Morgan Kaufmann, 1990.

[21] A. Moore and C. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol.13, pp.103-130, 1993.

[22] P. R. Wurman, R. D'Andrea and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol.29, no.1, pp.9-19, 2008.

[23] R. D'Andrea and P. Wurman, "Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities," *IEEE International Conference on Technologies for Practical Robot Applications*, pp.80-83, Woburn MA, 10-11 Nov. 2008.