A Novel Exploration/Exploitation Policy Accelerating Learning in Both Stationary and Non-Stationary Environment Navigation Tasks

Wiem Zemzem*, Moncef Tagina
National School of Computer Science, University of Manouba, Tunisia.

* Corresponding author. Tel.: (+216)99171757; email: wiem.zemzem@ensi-uma.tn
Manuscript submitted February 13, 2015; accepted May 3, 2015.
doi: 10.17706/ijcee.2015.7.3.149-158

Abstract: In this work, we are addressing the problem of an autonomous mobile robot navigating in a large, unknown and dynamic environment using reinforcement learning abilities. This problem is principally related to the exploration/exploitation dilemma, especially the need to find a solution letting the robot detect the environmental change and also learn in order to adapt to the new environmental form without ignoring knowledge already acquired. Firstly, a new exploration/exploitation policy (EEP) is proposed. Unlike existing EEPs such as ε-greedy and Boltzmann, the new EEP doesn't only rely on the information of the actual state but also uses those of the eventual next states. Secondly, as the environment is large, an exploration favoring least recently visited states is added to the proposed EEP in order to accelerate learning. The simulated experiments, using a ball-catching problem, show that combining this policy with Qlearning is more effective and efficient compared with ε-greedy in stationary environments and UQ-learning in non stationary environments.

Key words: Autonomous mobile robot, exploration/exploitation policy, large, unknown and dynamic environment, reinforcement learning.

1. Introduction

Autonomous mobile robots systems have drawn considerable attentions to both industry and academia in the recent years. These robots are smart machines designed to navigate in an unknown area and carry a certain task without any human intervention. They can be used in many applications such as, space and underwater exploration, surveillance and personal services [1]. In order to increase autonomy in robotic systems, many researchers are interested in designing robots with learning capabilities. To this end, reinforcement learning and dynamic programming for optimal control have been widely studied [2]-[8].

An RL agent learns by interacting with its environment and observing the results of these interactions. One of the challenges that arise in RL is the tradeoff between exploration and exploitation. The agent faces the following dilemma: either it chooses a random action in order to explore new area and enlarge its knowledge about the environment (exploration), or it chooses an action which looks optimal given the past observations and rewards (exploitation) [8]-[10]. Algorithms such as ε-greedy, Boltzmann distribution, simulated annealing, Probability Matching and Optimistic Initial Values are proposed to solve this dilemma [10]. However, they are designed for simply assuming the environment is stationary.

Learning in non stationary environments is yet morechallenging. In this case, the agent needs to

constantlyexplore the environment in order to integrate the most recentchanges into its knowledge of the world but such explorationshould not be done excessively that performance is greatlydegraded. Many researchers have been interested in thisproblem [11]-[24]. As examples, DynaQ+ [3], RBE [4], [11]and RB-DAE [11] has been applied to dynamic environments,but it takes a large amount of time for training. UQlearningis also proposed to quickly adapt toenvironment changes, but it maintains a considerable rate ofexploration along the whole learning experience. Other workslike [14]-[16] maintain periodic intervals of exploration. Thisrandom exploration is useful in the case of optimizing the oldpath, detecting the short path if it exists and avoiding beingstuck near corners. Unfortunately, this exploration can occureven if no change is held which may cause the non stabilityof the system's convergence. In another hand, exploring untildetecting the new position of the target like in [15] isn'tsufficient in the case of a single agent with an infrequentlymoving target; the exploration must be done during severalepisodes in order to propagate the promising information toa large number of state/action pair and move away from theold target's position.

In this paper, a new EEP addressing the problem of autonomous navigation in a large, unknown and dynamic environment is proposed. Unlike existing policies, the new EEP, called ε-greedy-MPA (the ε-greedy policy favoring the most promising actions), doesn't only rely on the information of the actual state but also uses those of the eventual next states. By this way, the exploration will be increased only when needed, especially, at the beginning of learning and when the environment changes. On another hand, as the environment is large, the agent needs to choose the most promising subset of states/actions to explore. To do that, an exploration favoring least recently visited states is added to the proposed EEP in order to accelerate learning. Using the ε-greedy-MPA, the agent is able to rapidly adapt to the new environmental form while maintaining its previous knowledge.

The rest of the paper is organized as follows. Problemstatement is described in Section 2. In Section 3, we present the UQ-Learning algorithm that willbe compared with our proposed method in the experimentalsection. Section 4 is dedicated to present the ε-greedy-MPA policy. Several experiences are conducted in Section 5 showing the efficiency of our proposals. Some concluding remarks and future works are discussed in Section 6.

2. Problem Statement

In this paper, we consider the problem of a single robot navigating in a discrete, large, unknown and dynamic environment. The robot is expected to find an optimal path from the starting position to a destination point (containing a ball) while avoiding unknown obstacles. The dynamic of the environment is due to a possible movement of the ball and/or obstacles making the current path no longer available. The environment is considered unknown in the sense that the robot doesn't know the current positions of the goal and obstacles, neither how they move.

The general state transition process of the system is a tuple of (S, A, T, R), where S is a discrete set of environmental states, A is a discrete set of agent actions, $R: S \times A \to \mathbb{R}$ is a reward function and $T: S \times A \to \Pi(S)$ is a state transition function ($\Pi(S)$ is a probability distribution over S). where S is a discrete set of environmental states, A is a discrete set of agent actions, $R: S \times A \to \mathbb{R}$ is a reward function and $T: S \times A \to \Pi(S)$ is a state transition function ($\Pi(S)$ is a probability distribution over S) [3].

It is assumed that the robot knows its initial position, can locate itself using its on-board sensors such as encoders and can detect the target or obstacles using sonar sensors.

3. The UQ-Learning Method

In this section, we presents the UQ-learning method, proposedin [9], as an hybrid approach dealing with non stationaryenvironments. This method, derived from the well-known Qlearning method [6], aims to

reduce the amount oftraining's time observed with the RBE method [11] and tobetter solve action selection problem by using reinforcementlearning and utility function. Reinforcement learning can providethe information of environment and utility function is usedto balance the exploration-exploitation dilemma. The Qlearning method (Algorithm1), the action selection strategy (Algorithm 2) and the learning algorithm (Algorithm 3) are described in Fig. 1. The utility-based reinforcement learning approach will be compared with our proposed method in the experimental section.

Algorithm 1: Qlearning algorithm

1: Initialize the table entry Q(s, a) to zero
2: Observe the current state s
3: Select an action a and execute it
4: Receive immediate reward r
5: Observe the new state s′
6: Update the table entry for Q(s, a) as follows:
$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r + \gamma \max_a Q(s_{t+1}, a))$
Where α is the current learning rate,
$0 < \gamma < 1$ is the discount factor
7: $s \leftarrow s'$
8: Go to step 2

Algorithm 2: SelectAction(s) algorithm

1: Calculate the utility using the formula:
$U(s,a) = \delta\, h(s) + (1-\delta)\, f_Q(s,a)$
2: Select action a_{max} according to
$U_{max}(s,a) = max_{a \in A} U(s,a)$
3: Select an action a_r in random
4: Generate a random number $\varepsilon \in [0,1]$
5: If $U_{max}(s, a_{max}) < \varepsilon$, choose action a_r
else choose action a_{max}

Algorithm 3: UQ-learning(U,Q) algorithm

1: initialize the table Q
2: initialize the table U
3: Observe current state s
4: Repeat {for each episode}
5: Repeat
6: Choose an action a as follows:
SelectAction(s) // Algorithm 2
7: Receive immediate reward r
8: Observe the new state s'
9: Update the table entry for Q(s, a) as follows:
$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
where α is the current learning rate
10: Update the table entry for U(s, a) as follows:
$U(s,a) = \delta\, h(s) + (1-\delta)\, f_Q(s,a)$, where δ is the balance parameter
11: $s \leftarrow s'$
12: Until the end condition is satisfied (s' is goal or reach a given iteration number)
13: Until no more episodes

Fig. 1. Learning methods.

4. A New EEP Dealing with Non-Stationary Environment

The goal of our work is to solve the exploration/exploitation dilemma in non stationary environments. We aim to establish a single agent learning approach that's able to learn the new environmental form using the least possible amount of exploration and without ignoring previous knowledge (without forcing the agent to learn from scratch).

One of the most sophisticated ways to trade off between exploration and exploitation is the ε-greedy strategy. This policy consists in selecting the greedy action (one that maximizes $Q(s, a)$) all but ε of the time and selecting a random action ε of the time, where $0 \leq \varepsilon \leq 1$. It's defined as follows:

$$\text{action} = \begin{cases} \text{a random action with probability } \varepsilon \\ \arg\max_a Q(s,a) \quad \text{with probability } 1-\varepsilon \end{cases} \tag{1}$$

The ε-greedy performs well in the case of stationary environments [3] but fails to solve dynamic problems since there's no particular explorations to detect environmental changes.

To overcome these limitations, we modify the ε-greedy policy as the promising action becomes the one having the highest Qvalue and leading to a more promising state. Our proposed policy, called the ε-greedy-MPA (the ε-greedy policy favoring the most promising actions), is defined as follows:

Algorithm 4: the ε-greedy-MPA policy

1: Generate a random number explore_1 ϵ [0, 1]
2: If (explore_1 $<$ ε)
3: Choose a random action in A
4: else
5: Define a list L
6: $\forall a \in A$, if $Q(s,a) = \max_a Q(s)$ then $L \leftarrow a$
7: If $|L| = 1$ and the action a, stored in L, leads to the goal state
8: $\text{action}_{\text{chosen}} = a$
9: else
10: Define a second list L'
11: $\forall a \in L$:
12: Determine the eventual state s' resulting from executing action a in the actual state s
13: If ($\max_a Q(s) < \max_a, Q(s')$)
14: L'←a
15: If ($|L'| >= 1$)
16: randomly choose an action from L'
17: else (L' is empty)
18: Generate a random number explore_2 ϵ [0, 1]
19: If (explore_2 $< p$)
20: randomly choose an action in the state s
21: else
22: select from all possible actions in the states the action leading to the least recently visited state with avoiding known obstacles

The Algorithm 4 summarizes our proposed action selection strategy. The list L stores the most promising actions in the current state s while the list L' stores the actions of L leading to a more promising next states'.

As mentioned in step 8 of the Algorithm 4, in the case that the list L contains a single action a leading to the goal state, this action a will be immediately chosen without defining the second list L'. Consequently, the position of the goal must be known without really executing a in s. One possible solution is to store the position of the goal s_{target} as follows:

- Initially, s_{target}=null;
- After reaching the goal, update s_{target}; s_{target}← actual_position;
- If the robot visits the ancient goal position without hitting the food source, update s_{target}; s_{target}← null.

According to step 12, the agent has to identify the next state s' resulting from executing action a in the actual state s. Action a isn't really experienced and according to reinforcement learning properties, an agent can determine the characteristics of any state only after visiting it. However, in our case study, the robot moves in a discrete and a two-dimensional space, defined by the coordinates (x, y), by performing the following actions: "up", "right", "down" and "left". So, to calculate the coordinates of the next state, it doesn't need to move neither toknow the nature of that state (an obstacle, the food source or an intermediatestate). In the contrary, knowing the next action to perform, the coordinates ofthe next state is automatically deduced from those of the actual state (For example: Next-action= "up"→ Next-state=(x−1, y)).

As noted in step 13, each action in the list L will be stored in the second list L' only if the robot trying the action a in the actual state s will move to a more promising state s', i.e, a state s' having a higher

Qvalue than the current state *s*. The chosen action will be then one of those stored in *L'*. In the worse case (*L'* is empty), the agent will resort to an exploration:

In order to promote the exploration of new areas, we can alternate between a random selection and a selection favoring least recently tested states. To do that, a table Anc: S→ℝ is maintained, storing the seniority of each visited state. $\mathrm{Anc}(s) = t$ means that the last visit of state s was occurred at time t. As mentioned in step 22, when the agent chooses the next action to perform according to the least visited states, it should avoid known obstacles. This is in the aim to avoid stuck near corners (for example: *L* obstacle or the extremities of the environment, etc). The random exploration (step 20) is without avoiding known obstacles. This is in order to detect moving obstacles and to find new areas that become recently reachable.

To avoid obstacles, a specific distribution of reward is needed ($R < 0$ if the robot hits an obstacle, $R > 0$ if the robot detects the source food and $R = 0$ otherwise). As a result, actions related to negative Qvalues are actions leading to collision with obstacles. Finally, the exploration with probability ε (step 3) is maintained in order to optimize the constructed policy after the convergence of the learning algorithm and even if no change is detected during several time steps.

5. Experiments and Analysis

In this section, we present three validation scenarios which evaluate how well our method performs in stationary as well as non-stationary environments. We employ Simbad, a Java 3d robot simulator, for simulating these testing scenarios [25].

We consider a robot navigation task in a non-stationary maze: a scenario that has a large state-action space as well as temporal non-stationarity. The agent navigates in a 30×30 grid world including passable grids and blocks. It is expected to find an optimal path from the starting position to a destination point (containing the ball). However, during the learning phase, the maze can be abruptly changed to another form and the ball can also move to a new location. Therefore, the agent has to adapt to such changes.

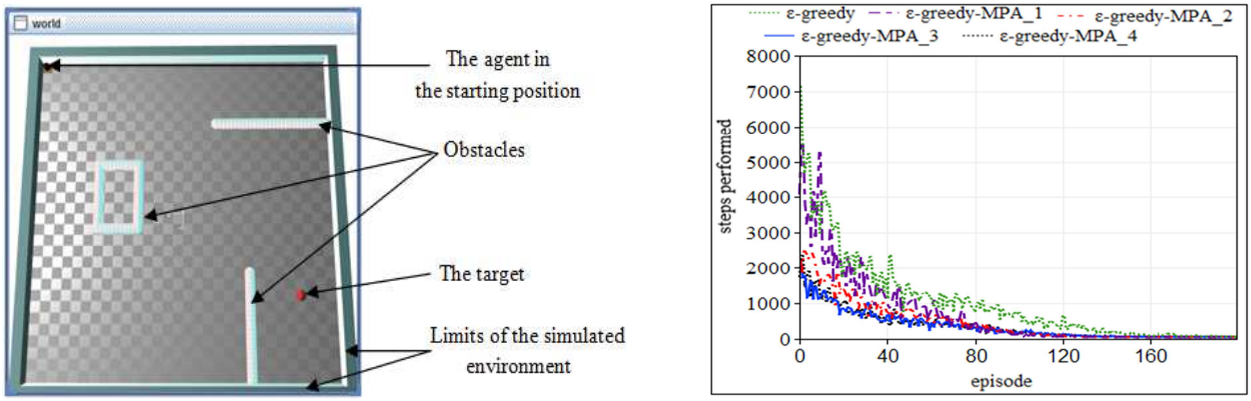
The agent learns using the Qlearning algorithm (Algorithm 1) and theε-greedy-MPA policy (Algorithm 4). To evaluate the new proposed EEP, the parameter settings are defined as follows: The agent's Q-value is setup to be zero initially, the probability ε is defined as 0.05 while γandαare defined as 0.9 for all experiences. For the rewards, it is defined as 0 for each regular action without hitting any obstacles, or the ball. If the robot hits an obstacle, it gets a penalty of −100. If it finds the target, it gets a reward of 100. The robot is initially located at the starting position. Its possible actions are: 'left', 'right', 'up', and 'down'. Once it detects the target, it returns to the starting position and a new episode begins.

5.1. Testing a Stationary Environment

We aim to evaluate theε-greedy-MPA policy in the case of a stationary environment. Fig. 2(a) describes the environment of the test. This EEP is compared to the classic ε-greedy policy (1). The impact of favoring the exploration of least recently visited states (LRVS) as well as the impact of avoiding known obstacles on learning are also evaluated.

Table 1 describes the different variants of the ε-greedy-MPA policy that we consider in this evaluation as well as the number of time steps needed on each episode for each tested policy. The probability p (see step 19 in Algorithm 4) is set as 0.5 in all tested policies except theε-greedy-MPA1 where p=1.

The average of twenty experiences is recapitulated in Fig. 2(a) and Table 1. We can see that each tested policy converges to a nearly optimal solution. However, using the ε-greedy-MPA, the learning is significantly improved. By this policy, the agent finds a promising path in less iterations and less episodes than using the ε-greedy.



(a) (b)

Fig. 2. Testing a stationnary environment. (a) Environment of the test over time (average of 20 experiences). (b) The number of time steps needed on each episode.

Table 1. Variants of the ε-Greedy-MPA Policy

Policies	Exploitation	Random exploration		Exploration of LRVS		Number of time steps needed for the system's convergence (average of 20 experiences)		
		With avoiding known obstacles	Without avoiding known obstacles	With avoiding known obstacles	Without avoiding known obstacles	Iterations before conver-gence	Iterations during first 121 episodes	Iterations during 200 episodes
ε-greedy-MPA1	X	X				127098	128547	132837
ε-greedy-MPA_2	X		X		X	89441	89746	94003
ε-greedy-MPA_3	X	X		X		65939	66085	71004
ε-greedy-MPA_4	X		X	X		66209	67131	72361
ε-greedy	X		X			199390	189856	201716

Considering the different variants of our proposed EEP, the ε-greedy-MPA_3 is the faster because it takes advantage from all previous knowledge (known obstacles, less recently visited states and most promising actions). This policy can still be useful in a non stationary environment only if the change is caused by the movement of the ball and not the displacement of obstacles. More clearly, the probability to detect if a state previously occupied by an obstacle is recently liberated is only 0.05 which is related to the parameter ε. The opportunity of detection is then very low. Increasing the value of ε is not promising since the exploration will increase causing the degradation of learning performance.

Despite the avoidance of known obstacles, the ε-greedy-MPA_1 policy seems to be the worst tested variant. This is because the least recently visited states are ignored here. The exploration favoring LRVS accelerates considerably learning but it blocks the agent near corners. This blockage becomes temporary by alternating between exploring LRVS and the random exploration like the case of the ε-greedy-MPA_2 policy. However, learning is further accelerated by avoiding known obstacles when exploring LRVS because this will prevent the collision with recently detected obstacles, especially in area containing a lot of blocks (obstacles). That's why the ε-greedy-MPA_4 policy is faster than the ε-greedy-MPA_2 policy.

5.2. Testing Non-Stationary Environments

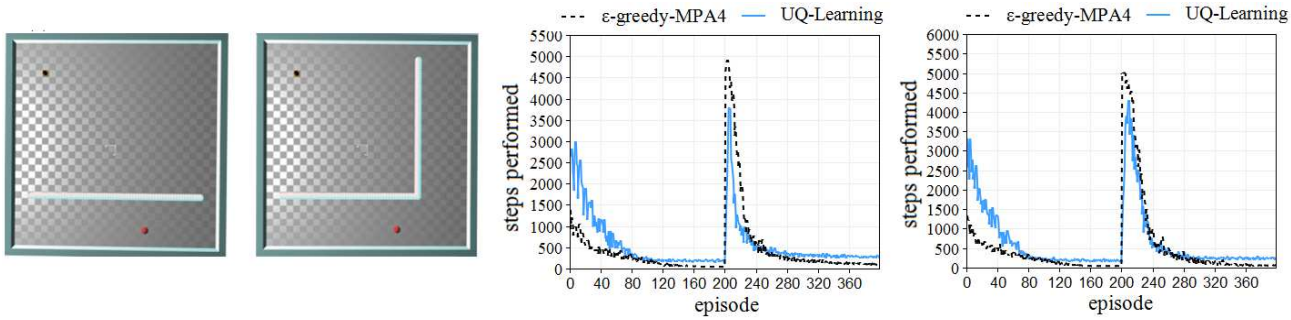
In what follows, we use the ε-greedy-MPA_4 policy to demonstrate the effectiveness of our learning method in the case of dynamic environments. Two dynamic scenarios are considered: in the first scenario, the environmental change is caused by the introduction of new obstacles while the second scenario studies the case of a moving target. Our method is compared to the UQ-learning method (Algorithms 2 and 3).

For the UQ-learning method, we use the following parameter settings, as described in [9]:

- Balance parameter $\delta = 0.4$
- Function $h(s)$ is: $h(s) = e^{-k\frac{t}{T}}$, where t is the number of steps from starting state to current state s, T is the limited number of steps and k is the adjust parameter, T=5000 and k=200 in our experiments.
- Function $f_Q(s,a)$ is: $f_Q(s,a) = \frac{e^{Q(s,a)}}{\sum_{a' \in A} e^{Q(s,a')}}$, where Q is the Qvalue.

5.2.1. Adding obstacles

In this scenario, two forms of obstacles are tested. In the first test, the environment is initially empty (without obstacles) and after 200 episodes a linear obstacle is added. The same is done in the second test but with adding an L obstacle, as shown in Fig. 3(a) and Fig. 3(b).

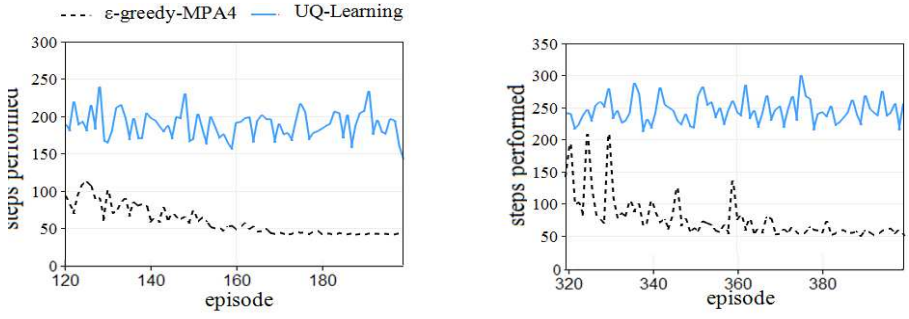


(a) (b) (c) (d)

Fig. 3. Adding new obstacles. (a) Test 1: adding a linear obstacle. (b) Test 2: adding a L obstacle. (c) Learning performance relating to Test 1. (d) Learning performance relating to Test 2.

Fig. 3(c) and Fig. 3(d) show the learning performance obtained from the average of 20 experiences. We can see that our proposed policy accelerates considerably the convergence of the system before the introduction of obstacles (episodes 0-200); Combining Qlearning with the ε-greedy-MPA policy outperforms the UQ-learning in stationary environments.

When the environment changes, bothε-greedy-MPA and UQ-learning enable the learner to adapt to the new environment configuration. The UQ-learning is a little more quickly because of its high level of exploration. However, this exploration prevents the agent using the UQ-learning method to consistently choose the optimal path when the system doesn't change. Fig. 4 presents the length of paths taken by the agent after convergence in the case of adding an L obstacle. We can see that the UQ-learning takes more steps to reach the goal during the episodes 120-200 and 320-400 whereas the ε-greedy-MPA allows the agent to consistently choose a nearly optimal path from a certain episode. The same is observed with the linear obstacle.



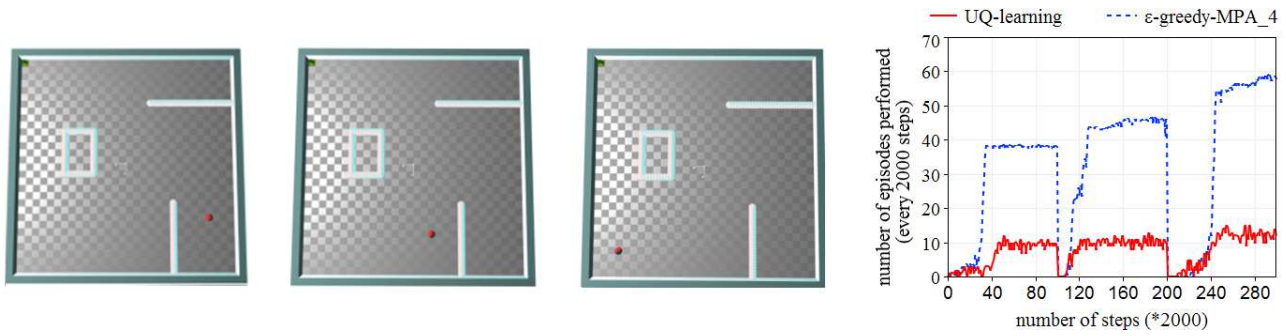
(a) (b)

Fig. 4. Length of paths taken by the agent after convergence. (a) Before adding the L obstacle (during episodes120-200). (b) After the introduction of the L obstacle (during episodes 320-400).

5.2.2. Testing a moving target

In this scenario, the environment contains several fixed obstacles and a temporary moving ball. Initially, it is in the form of Fig. 5(a). After 2.10^5 time steps, the environment changes to the second form (Fig. 5(b)). Another change is occurred after 4.10^5 and leads to the form of Fig. 5(c).

Fig. 5(d) shows the number of ball captured every 2000 time steps. We can see that both ε-greedy-MPA and UQ-learning enable the learner to find the novel location of the ball and construct a short path from the starting position to the current position of the target. However, when the environment doesn't change, the agent using the ε-greedy-MPA policy is able to capture more balls over time than that using UQlearning. This is due to the fact that even when the ball remains stationary, the agent using the UQlearning method continues to explore new area far from the optimal path whereas using the ε-greedy-MPA policy, the exploration is increased at each displacement of the target and is progressively decreased until the construction of the new nearly optimal path. Once the new path is built, only exploration with the probability ε is maintained. This is useful to further optimize the constructed path which explains the increase of captures after convergence. In addition to that, in the 3rd environmental form, the agent using the ε-greedy-MPA policy was able to circumvent the rectangular obstacle and built a semi-optimal path to the left of this obstacle.



(a) (b) (c) (d)

Fig. 5. Testing a moving ball. (a) Initial ball's position. (b) Second position. (c) Third position. (d) Number of balls captured (every 2000 steps).

6. Conclusion and Future Works

In this paper, we have presented a new action selection strategy, called ε-greedy-MPA, to solve the exploration/exploitation dilemma in the case of unknown, large and dynamic environments. Our proposed method was validated in a single robot navigation task. We have shown that combining the ε-greedy-MPA policy with the classic Qlearning method accelerates significantly the convergence of the system in the case of stationary environment and that is better than ε-greedy and UQ-learning. Conducted experiments also demonstrate that our proposed policy outperforms the UQ-learning method in the case of dynamic environments due to its ability to increase the rate of exploration only if needed which ensures the adaptation to infrequently environment changes as well as the convergence of the system to a nearly optimal solution when the environment remains stationary.

We expect to further improve our work by extending it with capabilities for solving more complex scenarios. Possible test cases include cooperative multi-agent system and continuous state spaces.

References

[1] Jaradat, M. A. K., Al-Rousan, M., & Quadan, L. (2011). Reinforcement based mobile robot navigation in

dynamic environment. *Journal of Robotics and Computer-Integrated Manufacturing, 27(1)*, 135-149.

[2] Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Journal of Discrete Event Dynamic Systems, 13(4),* 341-379.

[3] Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[4] Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the 7th International Conference on Machine Learning* (pp. 216-224). Morgan Kaufmann.

[5] Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227-303.

[6] Watkins, C. J., & Dayan, P. (1992). Q-learning. *Journal of Machine learning, 8(3-4),* 279-292.

[7] Muhammad, J., & Bucak, O. (2013). An improved Q-learning algorithm for an autonomous mobile robot navigation problem. *Proceedings ofInternational Conference on TAEECE* (pp. 239-243). IEEE.

[8] Coggan, M. (2004). Exploration and exploitation in reinforcement learning. McGill University.

[9] Chen, K., Lin, F., Tan, Q., & Shi, Z. (2009). Adaptive action selection using utility-based reinforcement learning. *Proceedings of GRC* (pp. 67-72). IEEE.

[10] Yogeswaran, M., & Ponnambalam, S. G. (2012). Reinforcement learning: Exploration–exploitation dilemma in multi-agent foraging task. *Journal of Operations Research & Decision Theory, 49(3),* 223-236.

[11] Zhang, K., & Pan, W. (2006). The two facets of the exploration-exploitation dilemma. *Proceedings of the IEEE/WIC/ACM International Conference on IAT* (pp. 371-380). IEEE Computer Society.

[12] Silva, B. C. da, Basso, E. W., Perotto, F. S., Bazzan, A. L. C., & Engel, P. M. (2006). Improving reinforcement learning with context detection. *Proceedings of the 5th International Joint Conference on AAMAS* (pp. 810-812). ACM.

[13] Silva, B. C. da, Basso, E. W., Bazzan, A. L., & Engel, P. M. (2006). Dealing with non-stationary environments using context detection. *Proceedings of the 23rd International Conference on ML* (pp. 217-224). ACM.

[14] Panait, L., & Luke, S. (July 2004). A pheromone-based utility model for collaborative foraging. *Proceedings of the 3rd International Joint Conference on AAMAS: Vol. 1* (pp. 36-43). IEEE Computer Society.

[15] Hrolenok, B., Luke, S., Sullivan, K., & Vo, C. (2010). Collaborative foraging using beacons. *Proceedings of the 9th International Conference on AAMAS: Vol. 3* (pp. 1197-1204). International Foundation for Autonomous Agents and Multiagent Systems.

[16] Shen, Y., & Zeng, C. (2008). An adaptive approach for the exploration-exploitation dilemma in non-stationary environment. *Proceedings of the International Conference on CSSE: Vol. 1* (pp. 497-500). IEEE.

[17] Choi, S. P., Yeung, D. Y., & Zhang, N. L. (2001). Hidden-mode markov decision processes for nonstationary sequential decision making. In R. Son, & C. L. Giles, (Eds.), *Sequence Learning* (pp. 264-287). Springer Berlin Heidelberg.

[18] Doya, K., Samejima, K., Katagiri, K. I., & Kawato, M. (2002). Multiple model-based reinforcement learning. *Journal of Neural Computation, 14(6),* 1347-1369.

[19] Phommasak, U., Kitakoshi, D., Mao, J., & Shioya, H. (2014). A policy-improving system for adaptability to dynamic environments using mixture probability and clustering distribution. *Journal of Computer and Communications, 2(4)*, 210-219.

[20] Singh, S. P. (July 1992). Reinforcement learning with a hierarchy of abstract models. *Proceedings of the International Conference on AAAI* (pp. 202-207).

[21] Fernández, F., & Borrajo, D. (2008). Two steps reinforcement learning. *International Journal of*

Intelligent Systems, 23(2), 213-245.

[22] Peng, J. & Williams, R. J. (1996). Increment multi-step Qlearning. *Journal of Machine Learning, 22(1),* 283-291.

[23] Lauer, M., & Riedmiller, M. (July 2004). Reinforcement learning for stochastic cooperative multi-agent systems. *Proceedings of the 3rd International Joint Conference on AAMAS: Vol. 3* (pp. 1516-1517). IEEE Computer Society.

[24] Jin, Z., Liu, W., & Jin, J. (2009). A state-cluster based Q-learning. *Proceedings of the 5th International Conference on NC: Vol. 1* (pp. 44-49). IEEE.

[25] Simbad Project Homepage. (May 2011). Form http://simbad.sourceforge.net/

Wiem Zemzem was born in Paris, France, on August 29, 1987. She received her computer engineering degree in 2011 and the master degree of artificial intelligence and decision support from the National School of Computer Sciences, Manouba University, in Tunisia, 2012.

She is currently working towards a PhD degree in computer science within the COSMOS Research Laboratory at the National School of Computer Sciences. Her research interests include robotics and artificial intelligence.

Moncef Tagina was born in Sfax, Tunisia, on July 6, 1968, and went to the Ecole Centrale de Lille (France), where he studied engineering and obtained his master degree of automatic in 1992 and his PhD degree in 1995. He obtained his "University Habilitation" from National School of Engineers of Tunis in 2002. Since 2003 he has been a professor at the National School of Computer Sciences (Tunisia).

He taught one year at Ecole Centrale de Lille before moving in 1996 to University of Science of Monastir (Tunisia). Since 2003, he has taught at the National School of Computer Sciences. His research interests include bond graph modelling, systems monitoring, robotics and artificial intelligence.