2014 IEEE International Conference on Systems, Man, and Cybernetics
October 5-8, 2014, San Diego, CA, USA

Curiosity-based Topological Reinforcement Learning

Muhammad Burhan Hafez, Loo Chu Kiong
Faculty of Computer Science and Information Technology
University of Malaya
Kuala Lumpur, Malaysia
burhan.hafez@gmail.com, ckloo.um@um.edu.my

Abstract—**Recent works involved in enhancing the learning convergence of reinforcement learning (RL) in mobile robot navigation have investigated methods to obtain knowledge from efficiently exploring the robot's environment. In RL, this knowledge is highly desirable to reduce the high number of interactions required for updating the value function and to eventually find an optimal or suboptimal policy for the agent. In this work, we propose a curiosity-based topological RL (CBT-RL) algorithm that makes use of the topological relationships among the observed states of the environment in which the agent acts. This algorithm builds an incremental topological map of the environment using Instantaneous Topological Map (ITM) model, which we use for facilitating value function updates as well as providing a guided exploration. We evaluate our algorithm against the original Q-Learning and Influence Zone algorithms in static and dynamic environments.**

Keywords— Reinforcement Learning; Convergence Acceleration; Topological Map; Guided Exploration.

I. INTRODUCTION

Reinforcement Learning (RL) allows an autonomous agent to learn to perform a task in initially unknown environment. It directs the agent to continuously take actions that maximize the received rewards from the environment. This sequential decision making is inherently related to the problem of autonomous robot navigation, in which the aim is to find an optimal path between two states in the environment while avoiding obstacles. The RL agent optimizes its behavior, unlike the supervised learning algorithms, without external supervision and using only its collected experiences. Accordingly, the goal of the agent is to learn an optimal policy through exploration, which requires enormous interactions with the environment, and it becomes very difficult in large state spaces and in online applications.

Many approaches from many different perspectives have been taken to accelerate the learning convergence of RL. Some of these are model-based [1] [2] and others are model-free [3] [4] [5]. Q-Learning is a well-known example of the model-free approaches and it guarantees to find an optimal policy for the agent having visited every state of the environment infinitely often. Throughout the literature, many methods were proposed to improve and speed up Q-Learning, using eligibility traces [5] [6], exploiting the generalization ability of the supervised learning algorithms [7] [8], or using topological representation of the environment [9] [10]. However, although most of these methods achieve a considerable learning performance, they made no consideration to the exploration part of the RL algorithm which is a significant component of the learning process, and that makes the policy produced by those methods far from being optimal especially in the early stages of learning. Thus, we propose a new Reinforcement Learning algorithm that uses a topological model of the environment to accelerate learning an optimal policy by optimizing the exploration strategy of the agent.

The paper is organized as follows: Section 2 gives an overview of the Q-Learning algorithm. The use of Instantaneous Topological Map (ITM) to model the agent's environment is presented in Section 3. The proposed method is then demonstrated in Section 4. Comparisons and simulation results are shown in Section 5. We conclude the paper in Section 6 by discussing the obtained results and the future direction of the study.

II. MODEL FREE RL AND Q-LEARNING

RL problem is formalized using *Markov Decision Process (MDP)*, which is a five-tuple (S, A, T, R, γ), where S is a set of states, A is a set of actions, $T : S \times A \times S \rightarrow [0, 1]$ a state-transition distribution, $R : S \times A \times S \rightarrow \Re$ is a reward function and γ is a discount factor; $0 \leq \gamma < 1$.

A policy π is defined as a mapping from states to actions; $\pi : S \rightarrow A$. The value of each state is given by a value function, as follows:

$$V^{\pi}(s) = \mathrm{E}[\mathrm{R}(s_0) + \gamma \mathrm{R}(s_1) + \gamma^2 \mathrm{R}(s_2) + \cdots \mid s_0 = s, \pi] = \sum_{s' \in S} (T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V^{\pi}(s'))) \quad (1)$$

The value function, $V^{\pi}(s)$, is the expected sum of discounted rewards the agent receives starting at state s and executing policy π. The goal of the RL agent is to take actions that maximize that value function in order to learn an optimal policy π^*.

Since RL does not have a complete knowledge of the underlying MDP, there were many algorithms to build an approximate model for the MDP, and these are called *Model-based*. On the other hand, *Model-free* algorithms do not estimate the MDP but rather estimate the state values directly based on the collected samples resulting from taking actions according to some fixed policy π. Each sample takes the following form:

$$sample_i = r + \gamma V^{\pi}(s'_i) \quad (2)$$

where r is the reward received on experiencing a transition (s, $\pi(s)$, s'). Averaging these samples gives the value estimate of the state s as follows:

$$V^{\pi}(s) = \frac{1}{n}\sum_i sample_i \quad (3)$$

where n is the number of collected samples from state s. Rather than keeping all samples to be averaged each time a new sample is collected, an exponentially-weighted average is used and updated on observing a new sample, as follows:

$$V^{\pi}(s) = V^{\pi}(s) + \alpha\left(sample - V^{\pi}(s)\right) \quad (4)$$

where α is a weighting term (the learning rate); $0 \leq \alpha \leq 1$. This equation is called *Temporal-Difference Learning (TD-Learning)* [11] because it depends on the difference, $sample - V^{\pi}(s)$, between the new sample and the current value estimate to update and learn the value estimate of state s. Using TD-Learning, Watkins proposed a method to assign values to state-action pairs and named it *incremental dynamic programming* [4] [12]. The method was later called Q-Learning, since its use is to learn state-action values (Q-values). The action-value function of the Q-Learning agent is updates as follows:

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma V(s') - Q(s,a)\right) \quad (5)$$

The value of the next state s' is computed by taking the maximum over all its Q values; $V(s') = \max_{a'} Q(s'_i, a')$. As this value update is applied independently of any policy, the Q-Learning is considered an off-policy TD-Learning method.

III. Topology learning for modeling the environment

Topology learning is a set of unsupervised learning techniques developed to identify data clusters and regularities in the input space, and were frequently used to produce a low-dimensional representation of a high-dimensional space. The information provided by the topology learning methods in terms of neighborhood relationships is of a great importance for autonomous navigation applications [13] [14].

Kohonen's self-organizing feature map (SOFM) was the first algorithm to learn a mapping from the input space to an output space represented by a network of neurons, utilizing competitive learning [15] [16]. The Neural Gas (NG) proposed by Martinetz and Schulten overcomes the SOFM deficiency of using a fixed and predetermined network topology [17]. This model uses competitive Hebbian learning (CHL) to develop edges between the neighboring nodes and to generate the final topology. This was further improved by the work of Fritzke where he introduced an incremental generation of the topology called Growing Neural Gas (GNG) [18]. This incrementally built topology eliminates the need for defining a specific number of nodes *a priori*. Despite the potential of building the topological representation of the RL agent's environment using GNG algorithm, it remains inadequate for such a task. This is because the GNG is based on the assumption that the input data (stimuli) are statistically uncorrelated, which is untrue assumption in robotics where the stimuli are generated along continuous trajectories.

The approach that constructs a topological map from such correlated stimuli is the Instantaneous Topological Map (ITM) [19]. Instead of being stuck in a single node for long time in the case of GNG causing slow adaptation, this approach makes the creation of new nodes resulting from traversing a trajectory faster and efficient, using fewer adaptation parameters.

ITM is defined by a set of neurons i with each neuron represented by a weight vector w_i, and a set of edges which is simply considered as a set of the neighboring nodes $N(i)$ for each node i. ITM implements Delaunay triangulation for the placement of nodes and edges in the map. According to [19], The ITM starts with two connected nodes and each time a new stimulus ξ is provided, the map performs the following adaptation steps:

1. Matching: compute the nearest node n and the second-nearest node s with regard to ξ based on the Euclidean distance measure. $n = \arg\min_i \|\xi - w_i\|$, $s = \arg\min_{j \neq i} \|\xi - w_j\|$.
2. Reference vector adaptation: move the nearest node n by a small rate ε towards ξ, $\Delta w_n = \varepsilon(\xi - w_n)$.
3. Edge adaptation: connect n and s by adding an edge between them (if they are not connected), and for every node $m \in N(n)$, check if the edge between n and m is non-Delaunay. If so, remove that edge. If there are no more edges originating from m, remove the node m as well.
4. Node adaptation: if the distance between ξ and n is greater than some threshold e_{max}, i.e., ($\|\xi - w_n\| > e_{max}$) and ξ lies outside the circle defined by the diameter linking n and s, then add a new node y with $w_y = \xi$, and create a new edge between n and y. Then, if the distance between n and s becomes smaller than $\frac{1}{2}\, e_{max}$, then remove s.

Since the creation of new nodes is no longer time-dependant and the non-Delaunay edges are removed instantly without waiting until they become obsolete, like in NG and GNG models, the ITM can adapt much faster to the observed states of the agent's environment.

In the next section we describe the use of ITM model to facilitate computing the value function for our Topology-based RL agent.

IV. THE CBT-RL

Considering the Eq. (5), we see that in Q-Learning the update to the value estimate of state *s* after taking action *a* depends on the reward experienced *r* and the value estimate of the observed state *s'*, which is *V(s')*. This means that this update contributes, in a decreasing magnitude, to the value estimate of all the previously observed states on the trajectory linking the start state to the current state *s*. However, in Q-Learning this update is limited to one state *s* when observing the transition *(s,a,r,s')*, and only contributes later to the proceeding states over the subsequent iterations, causing slow convergence rate. To overcome this limitation in Q-Learning, a topological representation of the environment is built using the ITM model discussed earlier.

Our algorithm consists of two successive phases: task learning and exploration optimization. In task learning phase, the ITM model is used to build a topological representation of the environment for accelerating the value function updates. In exploration optimization phase, we propose an internal reward function to guide the exploration based on the state values derived from the nodes of the ITM map.

A. *Task learning*:

To overcome the limitation of performing a single value update per interaction in the original Q-Learning algorithm, we build a topological representation of the agent's environment using the ITM model discussed earlier. The RL agent interacts with its environment and once a new state is observed, a new node will be created in the topological map which corresponds to the observed state. When the RL agent takes action *a* at state s_{n-1}, as shown in Fig. 1 , the value estimate of this state $V(s_{n-1})$ is updated according to Eq.(5). Then, this updated estimate is propagated to the neighboring nodes in the first topological neighborhood identified by $N(s_{n-1})$ using the edges developed so far. This allows the identified nodes to have their value estimates updated as well. The neighboring nodes, in turn, propagate their updated values backwards to their neighbors and so on. However, in contrast to the Influence Zone method [10] in which all the nodes in all the identified neighborhoods update their values, we restrict the update to the nodes whose corresponding states have been traversed by the agent in the current learning experience (we call them active nodes). This way we can avoid overestimating the state values, because updating the values of states which have not been visited in the current experience can result in an incorrect policy for the RL agent. Consequently, multiple updates are now performed in each interaction with the environment rather than just a single update, which leads to a faster convergence.

Each node *n* of the employed ITM consists of the following:

1- Reference vector: the geometrical position of the corresponding state in the environment.
2- Set of edges: the neighboring nodes of n, *N(n)*.
3- Set of Q-values: each Q-value corresponds to an edge, and the value of the node is the maximum of its Q-values.
4- Identification flag: indicates whether the node has already undertaken a state-action value function update or not.
5- Activation flag: indicates whether the node has been traversed in the current learning experience (learning trial) or not.

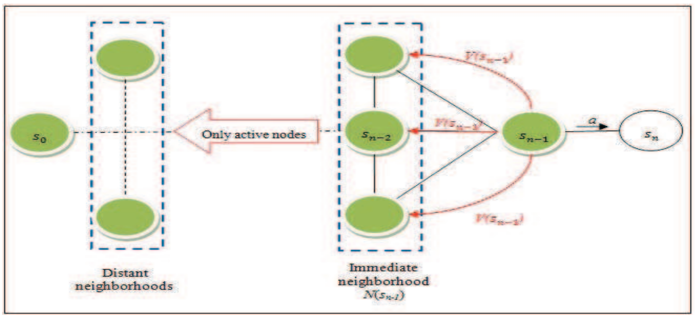

Fig. 1. Backward propagation of state value estimate through topological edges after an interaction with the environmet

B. *Exploration Optimization:*

In our approach we model the internal state of the agent by its current learned policy derived from its current value function. This is done by first defining the value of a policy according to Ng and Jordan [20], as follows:

$$V(\pi) = E_{s_0 \sim D}\,[V^{\pi}(s_0)] \qquad (6)$$

where *D* is the initial-state distribution, and the expectation *E* is taken with respect to s_0 drawn from *D*. We can rewrite Eq. (6) as:

$$V(\pi) = \sum_{s_0 \in S} D(s_0)\; V^{\pi}(s_0) \qquad (7)$$

where *S* is the set of external states. Thus, we consider $V(\pi)$ as an evaluation of the policy π which the agent has learned so far. When the agent transitions from one external state to another, its current policy driven from the state value estimates, which are updated after that transition, changes to a new policy. Based on this change in the internal state and the change in the perception of the agent (provided by the topological map), we propose an internal reward function as follows:

$$r_{int} = V\,(\pi') - V(\pi) + per_{err} \qquad (8)$$

This internal reward function represents how much the current learned policy has changed, $V\,(\pi) \rightarrow V\,(\pi')$, as a result of a transition in the external state space. This is realized by taking the difference between the values of the two policies before and after the transition occurs. The function also includes the perception error, which is the difference between the current state's position ξ and the position of the nearest node in the topological map; $per_{err} = |\xi - Nearest(\xi)|$. Using Eq.(7), we can rewrite Eq.(8) as:

$$r_{int} = \sum_{s_0 \in S} D(s_0)\left[V^{\pi'}(s_0) - V^{\pi}(s_0)\right] + per_{err} \qquad (9)$$

Since a single transition in the external state space between two time steps *t-1* and t produces a change to the value estimate of the current state and all the states in its topological neighborhoods, and these changes are independent of the initial state distribution, we omit the distribution D from Eq. (9). Accordingly, the internal reward function is as follows:

$$r_{int}(t) = \sum_{s_0 \in S} [V_t(s_0) - V_{t-1}(s_0)] + per_{err} \qquad (10)$$

We can interpret our internal reward as consisting of two key components: the learning progress (the change in the learned policy) and the perception improvement (the change in the external state); $r_{int}(t)$ = Learning_progress + Perception improvement.

The first component encourages the agent to take actions that lead to an improvement in the learned policy (rewarding the progress made) while the second encourages the agent to take actions that lead to places it has no or little knowledge about. Overall, this internal feedback acts as a curiosity signal for the agent and gives a guided exploration strategy in which the policies evolve until an optimal or suboptimal policy is learned.

In order to integrate the internal reward in our algorithm, we use two Q-learning value functions. The first Q function estimates the external state values using the external rewards received from the environment and is called the *task function*. The second Q function, which is task-independent, estimates the internal state values using the self-generated internal reward r_{int} defined earlier and is called the *exploration function*.

The optimal policy of the exploration function defines the optimal exploration for the RL agent to use while learning to solve its task. The agent takes actions according to the exploration function throughout the learning process. After the learning ends, the agent takes actions according to the task function to perform the task it has learned.

Fig. 2 shows the proposed algorithm architecture and the data flow between the task layer and the exploration layer which correspond to task learning and exploration optimization phases explained earlier.

Algorithm 1 CBT-RL Algorithm

```
1:  While stop ≠ true do
2:      $s \leftarrow s_0$
3:      While $s \neq$ goal do
4:          Take action $a$ at state s according to the policy
            of the exploration function
5:          Observe state $s'$ and external reward $r$
6:          Create a new node $o'$ corresponds to $s'$
7:          if $o' \not\subset Map$ then
8:              Add $o'$ to $Map$
9:          end if
10:         Update $Map$ according to ITM adaptation steps
11:         $Per_{err} \leftarrow w_{o'} - w_{n'}$: $n'$ is the nearest node to $o'$ in $Map$
12:         Compute the temporal difference value,
            TD_val $\leftarrow r + \gamma V_{task}(s') - Q_{task}(s,a)$, and update the
            state-action value estimate using TD-val
13:         if TD-val > $\theta$ then
14:             for each $n_i \in N(s)$ do
15:                 $Q_{task}(n_i,a_i) \leftarrow Q_{task}(n_i,a_i) + \alpha\,(r_{s_i\rightarrow s} + \gamma$
                    $V_{task}(maxNode) - Q_{task}(n_i,a_i))$: $maxNode$ is the
                    neighboring node with maximum value (with
                    respect to the node $n_i$)
16:                 accumulator $\leftarrow$ accumulator + $V_{task}^{new}(n_i)$ -
                    $V_{task}^{old}(n_i)$
17:                 $s \leftarrow n_i$
18:                 Goto 14 (Repeat the process for $n_i$'s neighboring
                    nodes)
19:             end for
20:             internal_r $\leftarrow$ accumulator + $Per_{err}$
21:             $Q_{exp}(s,a) \leftarrow Q_{exp}(s,a) + \alpha\,($internal_r $+ \gamma V_{exp}(s') -$
                $Q_{exp}(s,a))$
22:         end if
23:         $s \leftarrow s'$
24:     end while
25: end while
26: Exit
```

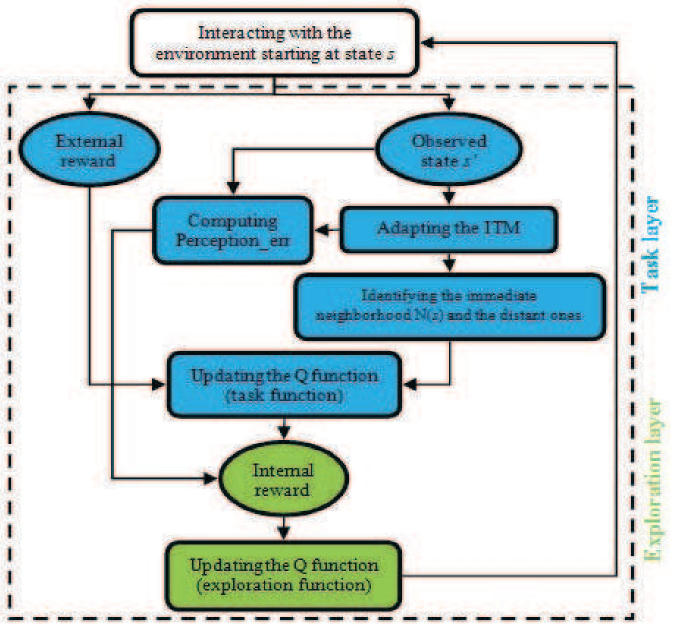
Interacting with the environment starting at state s
External reward
Observed state s'
Computing Perception_err
Adapting the ITM
Identifying the immediate neighborhood N(s) and the distant ones
Updating the Q function (task function)
Internal reward
Updating the Q function (exploration function)
Task layer
Exploration layer

Fig. 2. The Architecture of CBT-RL Agent

A detailed description of the algorithm's global parameters is given in Table 1.

Table 1: Global parameters of CBT-RL algorithm

	Description
Stop	The stopping condition can be either maximum number of learning trials given to the agent, or a minimum-error threshold, with which the total difference between the current and the predicted value estimates over all the states is compared.
Map	It is the topological map incrementally built by using the ITM algorithm.
Threshold θ	The purpose of this threshold is to reduce the number of updates per learning trial, so that a negligible TD_val causes no update propagation to the neighboring nodes.

V. Results and Evaluation

In the following experiments we compare the performance of our proposed algorithm to the original Q-Learning and the Influence Zone algorithms in static and dynamic environments of increasing levels of complexity. The learning rate α is 0.5, the discount factor γ is 0.8, the node insertion threshold of the ITM e_{max} is 0.5, the temporal difference threshold θ is 0.1 and the ε-greedy action-selection parameter (in Q-learning & Influence Zone) is 0.5. The learning rate and discount factor of the exploration function of our algorithm are 0.2 and 0.99 respectively.
The agent selects one of the possible actions corresponding to the eight compass directions. The reward signal the agent receives is either: (-1) for hitting an obstacle or boundary, (+1) for reaching a goal state or (0) for wandering around.

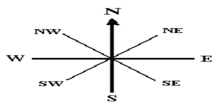

Fig. 3. Possible actions

Fig. 4 shows a static grid-world environment of size 20×20 to which we apply the three algorithms. The red, black and green squares refer to an initial state, an obstacle and a goal state, respectively. In this experiment, we test the agent's learned policy after each learning trial by letting the agent act according to the policy derived from the learned task function. In the test phase the agent is given up to 2000 steps to reach a goal state starting at the initial state.

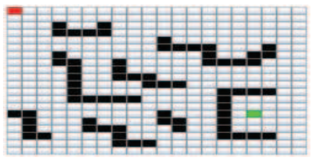

Fig. 4. Static Environment

The number of steps to reach the goal in the Q-learning is 1000 on average as shown in Fig. 5. Influence Zone's agent takes five learning trials to converge to a suboptimal policy reaching the goal with 27 steps. CBT-RL, on the other hand, needs only two learning trials to learn an optimal policy with 24 steps to the goal.

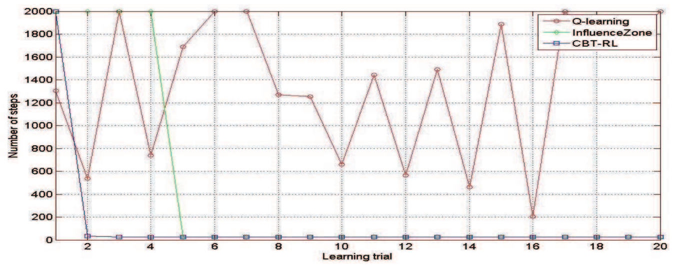

Fig. 5. Number of steps to the goal using the learned policy

To see how CBT-RL reacts to stochastic environment changes (from the agent perspective), we use a 4-cage configuration shown in Fig. 6 where the agent learns a path from a start to a goal state. But after 199 learning trials, the short path to the goal becomes blocked. Fig. 7 shows changes in the environment topology during the learning process, where the red nodes represent the environment's free states.

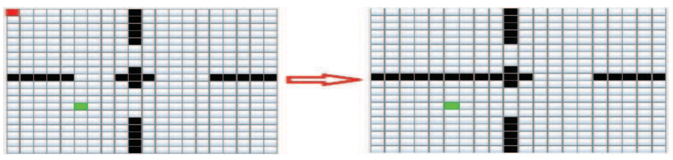

Fig. 6. Dynamic Environment (one downward path blocked stochastically)

In this experiment we test the agent on its learned policy every 25 learning trials, with the agent given up to 2000 trials to reach the goal. The number of steps to the goal has risen in the three algorithms after the change occurred at the learning trial 200 as shown in Fig. 8. But while the Q-learning and the Influence Zone require many learning trials to learn a new trajectory, CBT-RL adapts more quickly to such a stochastic change in the topology, converging to an optimal policy with 21 steps to the goal at the end of the learning process.

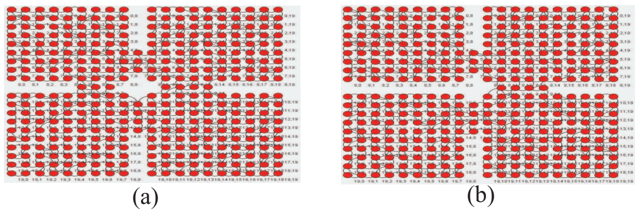

Fig. 7. ITM map: (a) before the change occurs, (b) after the change occurs

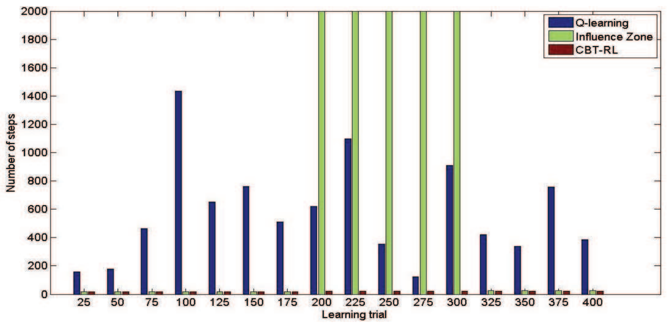

Fig. 8. Number of steps to the goal using the learned policy

In the last experiment, we set the number of learning trials to 500. Every 25 trial during the first 400 trials, the environment changes stochastically to one of the four configurations shown in Fig. 9, with the number of the selected configuration drawn randomly from a uniform distribution over the range [1,4].

As shown in Fig. 10, Q-learning and Influence Zone failed significantly to adapt to this challenging stochastic topology change. The agent in the Influence Zone could not continue the learning process, permanently taking actions that lead to some previous goal position. However, CBT-RL has been able to adapt quickly to this change in the goal position and its neighborhood, learning an optimal policy that leads to the goal position of the final selected configuration with just 20 steps.

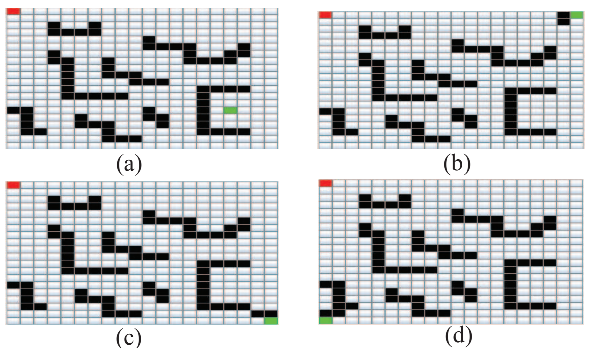

Fig. 9. Stochastically changing environment (goal and its surrounding obstacles change between the four configurations)

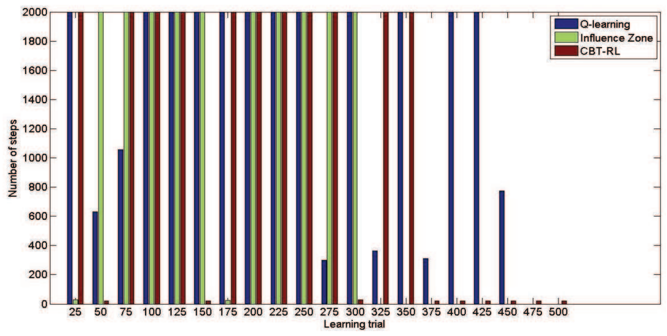

Fig. 10. Number of steps to the goal using the learned policy

VI. Conclusion

In this paper, we propose CBT-RL, a topology-based enhancement method for the original Q-learning algorithm for mobile robot navigation. The method adopts the Instantaneous Topological Map (ITM) model for constructing topological representation of the agent's environment. The neighborhood relationships information encoded in the incrementally-built ITM is used to spread the value function updates over the observed trajectories and to generate a curiosity used to optimize the action selection of the RL agent. We compared our proposed algorithm to the Q-learning and Influence Zone in static and dynamic environments. The simulation results support our claim that CBT-RL has higher learning performance than those algorithms and can adapt quickly in response to any stochastic changes in the environment topology. We demonstrated in this study the utility of the topological maps for facilitating value function updates and optimizing the exploration strategy in discrete state space. It would be interesting to investigate the potential of the algorithm in continuous state or continuous action spaces.

Acknowledgment

This research is supported by UMRG Grant (RP003D-13ICT) from University of Malaya.

References

[1] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin,* vol. 2, no. 4, pp. 160-163, 1991.

[2] 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.

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

[4] C. G. C. H. Watkins, "Learning from delayed rewards," King's College, Cambridge, 1989.

[5] M. Wiering and J. Schmidhuber, "Fast Online Q(λ)," *Machine Learning,* vol. 33, no. 1, pp. 105-115, 1998.

[6] J. Peng and R. J. Williams, "Incremental Multi-Step Q-Learning," *Machine Learning,* vol. 22, no. 1-3, pp. 283-290, 1996.

[7] M. Zeller, R. Sharma and K. Schulten, "Motion planning of a pneumatic robot using a neural network," *Control Systems, IEEE,* vol. 17, no. 3, pp. 89 - 98, 1997.

[8] L. Busoniu, R. Babuska, B. De Schutter and D. Ernst, Reinforcement learning and dynamic programming using function approximators, New York: CRC Press, 2010.

[9] A. P. S. Braga and A. F. R. Araújo, "A topological reinforcement learning agent for navigation," *Neural Computing & Application,* no. 12, p. 220–236, 2003.

[10] A. P. S. Braga and A. F. R. Araújo, "Influence zones: A strategy to enhance reinforcement learning," *Neurocomputing,* vol. 70, no. 1-3, pp. 21-34, 2006.

[11] R. S. Sutton and A. G. Barto, Introduction to reinforcement learning, Cambridge, MA: MIT Press / Bradford Books, 1998.

[12] L. Kaelbling, M. Littman and A. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research,* vol. 4, pp. 237-285, 1996.

[13] E. Remolina and B. Kuipers, "Towards a general theory of topological maps," *Artificial Intelligence,* vol. 152, no. 1, p. 47–104, 2004.

[14] S. Thrun and A. Buckenz, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI*, Portland, Oregon, 1996.

[15] T. Kohonen, Self-Organization and Associative Memory, New York: Springer Berlin Heidelberg, 1989.

[16] T. Kohonen, Self-organizing maps, New York: Springer Berlin Heidelberg , 2001.

[17] T. Martinetz and K. Schulten, "A neural-gas network learns topologies," in *Artificial Neural Networks*, Amsterdam, 1991, pp. 397-402.

[18] B. Fritzke, "A Growing Neural Gas Network Learns Topolgies," *Advances in Neural Information Processing Systems,* 1995.

[19] J. Jockusch and H. Ritter, "An Instantaneous Topological Mapping Model for Correlated Stimuli," in *Proceedings of the IJCNN'99*, Washington, DC, 1999.

[20] A. Y. Ng and M. Jordan, "PEGASUS: A policy search method for large MDPs and POMDPs," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, San Francisco, CA, USA, 2000.