Neural Comput & Applic (2015) 26:1939–1954
DOI 10.1007/s00521-015-1861-8

ORIGINAL ARTICLE

Topological Q-learning with internally guided exploration for mobile robot navigation

Muhammad Burhan Hafez · Chu Kiong Loo

Received: 9 September 2014 / Accepted: 17 February 2015 / Published online: 28 February 2015

Abstract Improving the learning convergence of reinforcement learning (RL) in mobile robot navigation has been the interest of many recent works that have investigated different approaches to obtain knowledge from effectively and efficiently exploring the robot's environment. In RL, this knowledge is of great importance for reducing the high number of interactions required for updating the value function and to eventually find an optimal or a nearly optimal policy for the agent. In this paper, we propose a topological Q-learning (TQ-learning) algorithm that makes use of the topological ordering 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 model which we use for accelerating value function updates as well as providing a guided exploration strategy for the agent. We evaluate our algorithm against the original Q-learning and the Influence Zone algorithms in static and dynamic environments.

Keywords Reinforcement learning · Q-learning · Convergence acceleration · Topological map · Guided exploration

M. B. Hafez (✉) · C. K. Loo
Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia
e-mail: burhan.hafez@gmail.com

C. K. Loo
e-mail: ckloo.um@um.edu.my

1 Introduction

Reinforcement learning (RL) is a subfield of machine learning that enables an autonomous agent to learn to perform a task in initially unknown environment through experience. It directs the agent to continuously taking actions that maximize the received rewards from the environment. This sequential decision making is strongly related to the problem of goal-oriented mobile robot navigation, in which the aim is to find an optimal path between two states in the environment while avoiding obstacles. The path planning algorithms cannot be employed to solve such a problem since they require a model of the environment as an input, which is inapplicable here because the agent has no prior knowledge of its world. Unlike the supervised learning method, the RL agent optimizes its behavior without external supervision. Instead, it uses its collected experiences of trial and error, learning from the results of its actions. Accordingly, the goal of the agent is to learn an optimal policy (sequence of actions that maximize the received rewards) through exploration, which requires enormous interactions with the environment. It also becomes very difficult in large state spaces and in online applications as the number of the possible policies grows exponentially with the number of states in the environment.

Throughout the literature, different approaches have addressed the problem of accelerating RL convergence from many different perspectives. Some of these are model-based [1–3] and others are model-free [4–6]. 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 [7]. Since it was introduced, many methods have been proposed to improve and speed up Q-learning,

using the eligibility traces [6, 8], using the generalization ability of the supervised learning algorithms [9–13] or using a hierarchical approach [14, 15]. A different line of research has been on studying the spatial pattern of the agent's movement in the environment [16–18]. A good example of this is the work by Braga and Araújo [16]. In this study, they apply the concept of latent learning which is the learning that occurs in the absence of reward signals from the environment. The proposed method involves building a topological representation of the environment while exploring it until a reward is received on reaching a goal state. After which, the knowledge encoded in the topological map is exploited to update the value function. In their second study [17], the authors have modified their previous approach to allow for incremental value function update instead of delaying the update until reaching a goal state. An important modification has been to restrict the update after each interaction to a group of states, called by the authors "the influence zone," whose states have lower value estimate than that of the current state. However, although most of these methods achieve a considerable learning performance, they give little consideration to the exploration strategy of the RL agent, which is a significant component of the learning process. In fact, modern RL algorithms can effectively exploit the available experience data, but they typically depend on naïve exploration methods and do not direct the RL agent to collect experiences which will be beneficial for learning the desired task.

Effective exploration is still one of the challenging research problems in RL, and the trade-off between exploration and exploitation in RL has long been a subject of interest in several studies. For instance, a simple strategy ε-greedy selects an action randomly with a probability of ε and according to a learned policy with a probability of $1 - \varepsilon$ [19]. A more effective method is the exploration bonus proposed by Sutton, which assigns each state-action pair a bonus inversely proportionate to the number of times the pair has been visited [20]. The bonus is used in computing the state-action value estimate. Accordingly, the resulted policy encourages the agent to take actions that lead to parts of the environment the agent has much uncertainty about. Luciw et al. [21] used a combination of perceptual and cognitive models to derive an intrinsic reward that acts as a curiosity signal and is used to direct the exploration. Therefore, instead of being driven solely by external rewards, that might be absent, the agent will be driven by its own curiosity. However, the study employs a basic vector quantization for building an internal state representation, without benefitting from the topological relationships already exist in the environment. Another interesting study provided a theoretical framework for designing an intrinsic reward function, IRF, based on maximizing information gain called by authors as "predictive information" [22]. This study makes a strict assumption that any IRF is independent of the agent's task, which excludes task-dependent information that might be useful for optimal exploration.

In light of the above, it becomes clear that there is a considerable lack of information in the use of the topological characteristics of the agent's environment for guiding the exploration to achieve fast learning convergence. Thus, we propose a new RL algorithm that uses a topological model of the environment to accelerate learning an optimal/nearly optimal policy through optimizing the exploration strategy of the RL agent by making it more guided and informed. To achieve such an optimal exploration, the algorithm provides the agent with the capability to learn how to effectively explore its environment rather than depending on randomized exploration. The proposed algorithm will use an internal reward (as opposite to the external reward received from the environment) derived from the change in the state's value estimates and the change in the environment topological model, in order to guide the exploration.

The paper is organized as follows: Sect. 2 gives background information of the original Q-learning and the Influence Zone algorithms. The use of the Instantaneous Topological Map (ITM) to model the agent's environment is presented in Sect. 3. The proposed algorithm is then demonstrated in Sect. 4. Comparisons and simulation results are shown in Sect. 5. Section 6 provides explanations for the results and a general discussion of the different aspects of our approach. We conclude the paper in Sect. 7 by summarizing the main concepts of our approach and its contribution.

2 Temporal difference learning, Q-learning and the Influence Zone

The 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: $S \times A \times S \rightarrow [0, 1]$ is a state-transition distribution, R: $S \times A \times \mathrm{S} \rightarrow \Re$ 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; π: $S \rightarrow A$. The value of each state is given by a value function, as follows:

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

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

Since RL agent does not have a full knowledge of the underlying MDP, many algorithms suggest to build an approximate model for the MDP, and these algorithms are called model-based. On the other hand, model-free algorithms do not estimate the MDP but rather estimate the state values directly by using the collected samples that result from taking actions according to a fixed policy π. Each sample has the following form:

$$\text{Sample}_i = r + \gamma V^{\pi}(s_i') \tag{2}$$

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:

$$V^{\pi}(s) = \frac{1}{n}\sum_i \text{sample}_i \tag{3}$$

where n is the number of the collected samples from state s.

Instead of holding all samples to be averaged when a new sample is collected, an exponentially weighted average is used. This average is updated on observing a new sample, as follows:

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(\text{sample} - V^{\pi}(s)) \tag{4}$$

where α is a weighting term (the learning rate); $0 \leq \alpha \leq 1$. This equation is called temporal difference learning (TD learning) [19] because it learns the new value estimate of state s based on the difference between the new sample and the current value estimate.

Using the TD learning, Watkins proposed a method to assign values to state-action pairs and named it incremental dynamic programming [5]. The method was later called Q-learning, since it learns the state-action values (Q-values). The state-action value function of the Q-learning agent is updated as follows:

$$Q(s,a) = Q(s,a) + \alpha(r + V(s') - Q(s,a)) \tag{5}$$

The value of the next state s', $V(s')$, is computed by taking the maximum over all possible actions of the state-action values (the Q-values); $V(s') = \max_{a'} Q(s', a')$. Because this value update is applied independently of any policy, the Q-learning is considered an off-policy TD learning algorithm.

In the Influence Zone algorithm and after the agent takes an action a in state s, it observes a new state s_{t+1} and a reward $r(s_{t+1})$. Then the agent computes the temporal difference (Error_TD) for all the states s that are topological neighbors to the state s_{t+1}, as follows:

$$\text{Error\_TD} = r(s_{t+1}) + \gamma V(s_{t+1}) - Q(s, a_{ss_{t+1}}) \tag{6}$$

If the Error_TD for any of the states is above some threshold Φ, then the agent updates the corresponding Q-value of that state only, and the Error_TD is accumulated in θ. Only when θ is big enough (above some threshold θ_0) does the agent update the Q-values of the states in the further neighborhoods with respect to the state s_{t+1}, with the condition that the values of these states are smaller than that of the node s_{t+1}.

3 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 input space. The information provided by the topology learning methods in terms of neighborhood relationships is highly desirable for autonomous navigation applications [23, 24].

Kohonen's self-organizing feature map (SOFM) was the first algorithm to learn a mapping from an input space to an output space represented by a network of neurons, utilizing competitive learning [25, 26]. The neural gas (NG) proposed by Martinetz and Schulten overcomes the SOFM deficiency of using a fixed and predetermined network topology [27]. 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) [28]. 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 [12], GNG remains inadequate for modeling the agent environment. This is because the GNG is based on the assumption that the stimuli (input data), such as agent locations in our study, are statistically uncorrelated, which is untrue assumption especially in robotics applications where the stimuli are generated along continuous trajectories. This makes it unsuitable in such applications. Moreover, node creation in GNG is done every fixed number of time steps instead of being initiated instantly once a new stimulus is observed which is necessary when the agent is exploring its environment.

The approach that overcomes the above-mentioned drawbacks of GNG and constructs a topological map from correlated stimuli is the ITM [29]. Instead of assigning a sequence of inputs to 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 [29], 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 toward 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 whether the edge between n and m is non-Delaunay, i.e., check whether the node s is located inside the circle defined by the diameter nm. 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. Figure 1 shows an example of an ITM network.

In the next section, we describe how we use ITM for facilitating the value function updates and for providing a guided exploration for the RL agent.

4 The TQ-learning

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 newly observed state $V(s')$, as shown in Eq. (5). This means that this update will also contribute in turns to the value estimate of all the previously observed states on the trajectory linking the start state to the current state s. However, Q-learning performs only a single update in each interaction with the environment. This update contributes later to the proceeding states over the subsequent experiences,[1] causing slow convergence rate. To overcome this limitation, a topological map of the environment is built using the ITM model discussed earlier. This is because the ITM map captures the neighborhood relations among the environment states,[2] allowing us to propagate any update backward to all the proceeding states in every interaction with the environment.

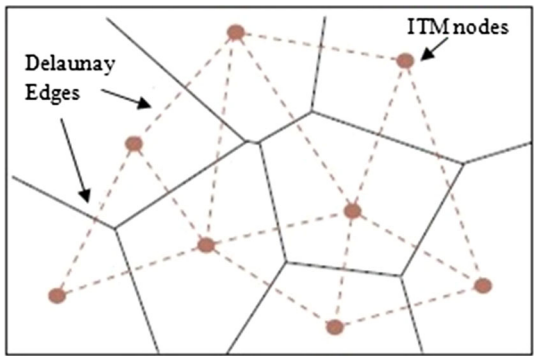


Fig. 1 ITM network

4.1 Task learning and exploration optimization

Our algorithm consists of two successive phases: task learning and exploration optimization. In the task learning phase, the ITM model is used to build a topological representation of the environment for accelerating the value function updates. In the 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.

4.1.1 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. 2, 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 backward to their neighbors and so on. However, in contrast to the Influence Zone method [17] 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

[1] By "subsequent experiences", we mean subsequent visits by the agent to those proceeding states.

[2] The environment states are represented by nodes in the ITM map as will be discussed later.

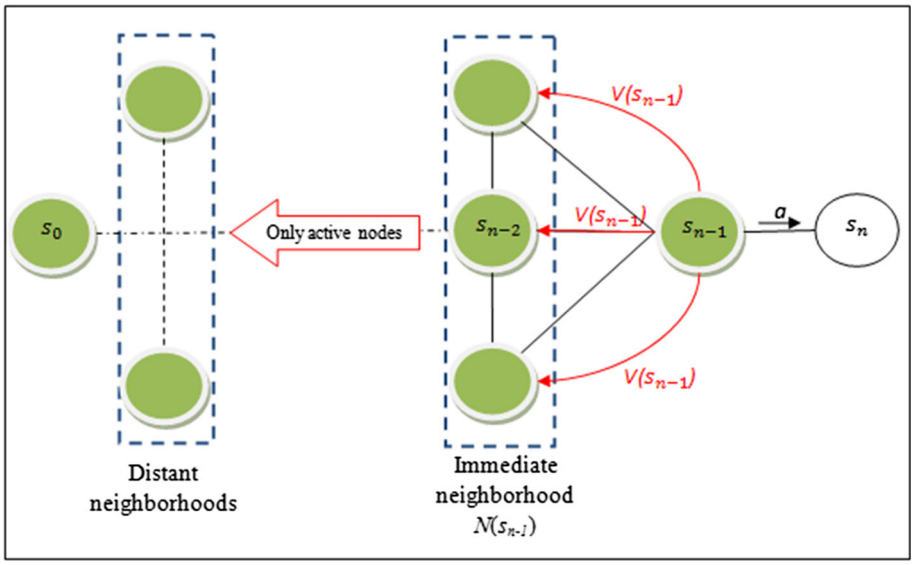


Fig. 2 Backward propagation of state value estimate through topological edges after an interaction with the environment

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.

4.1.2 *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 [30], as follows:

$$V(\pi) = E_{s_0 \sim D}[V^{\pi}(s_0)] \tag{7}$$

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. (7) as:

$$V(\pi) = \sum_{s_0 \in S} D(s_0) V^{\pi}(s_0) \tag{8}$$

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:

$$r_{\text{int}} = V(\pi') - V(\pi) + \text{per}_{\text{err}} + \tau \tag{9}$$

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; $\text{per}_{\text{err}} = |\xi - \text{Nearest}(\xi)|$. $\tau < 0$ is a small punishment used to punish (negatively reward) any action that leads to a negligible change in policy, so that such actions are discouraged to help accelerate the learning. Using Eq. (8), we can rewrite Eq. (9):

$$r_{\text{int}} = \sum_{s_0 \in S} D(s_0)\left[V^{\pi'}(s_0) - V^{\pi}(s_0)\right] + \text{per}_{\text{err}} + \tau \qquad (10)$$

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. (10). Accordingly, the internal reward function is as follows:

$$r_{\text{int}}(t) = \sum_{s_0 \in S}[V_t(s_0) - V_{t-1}(s_0)] + \text{per}_{\text{err}} + \tau \qquad (11)$$

We can interpret this internal reward as consisting of two key components: the policy value improvement (the change in the learned policy) and the perception improvement (the change in the external state); $r_{\text{int}}(t) =$ Policy value improvement + Perception improvement.

The first component encourages the agent to take actions that lead to an improvement in the learned policy (rewarding the improvement 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 a nearly optimal policy is learned. This exploration strategy is optimized in the sense that it only guides the agent to areas where the update to state value estimate is expected to be significant rather than to areas where the value estimates already converged (any further update is negligible).

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* (Q_{task}). 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* ($Q_{\text{exploration}}$).

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.

4.2 TQ-learning agent architecture

Figure 3 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.

In each interaction with its environment at a state s, the agent observes a new state s' and receives an external reward. A new node is then created in the topological map which corresponds to the newly observed state (if such a node is not present), and the map is updated accordingly by performing the adaptation steps of the ITM algorithm detailed in Sect. 3. The perception error per_{err} is computed by taking the difference between the reference vector of the recently added node and the reference vector of its closest node in the topological map. Next, the immediate neighborhood, defined by the nodes $N(s)$, and the distant neighborhoods of the new node are identified. After that, the value estimate of the current state s is updated using the task-dependant Q function. This update is propagated backward neighborhood by neighborhood until all active nodes of these neighborhoods update their value estimates. Afterward, the difference between the old and the new value estimates of these nodes together with the perception error are combined to generate the internal reward. This is followed by updating the task-independent Q function (the exploration function). Finally, the agent takes the next action according to the policy derived from the current exploration function instead of a greedy or randomized action selection, and the process is repeated.

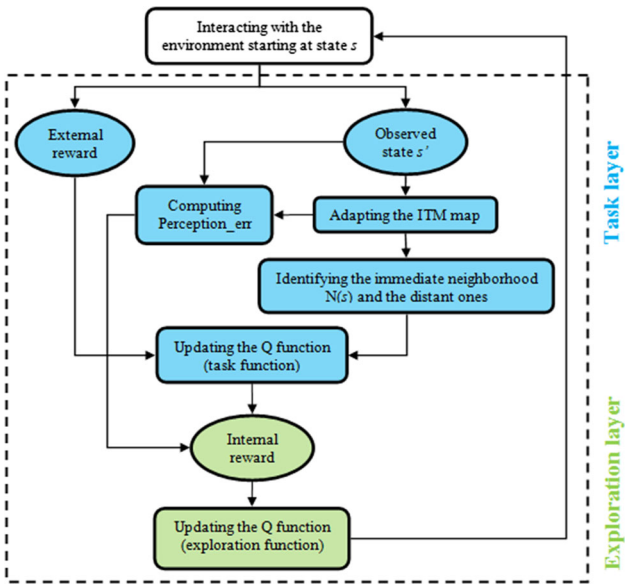


Fig. 3 The architecture of TQ-learning agent

Algorithm 1 TQ-Learning algorithm

1: **While** stopping_cond ≠ true **do**
2: $s \leftarrow s_0$: s_0 is the start state
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' and o'.active ← True
7: **if** $o' \notin Map$ **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: TD_val ← $r + \gamma V_{task}(s')$ - $Q_{task}(s,a)$
13 $Q_{task}(s,a) \leftarrow Q_{task}(s,a) + \alpha$ TD_val
14: accumulator ← accumulator + $V_{task}^{new}(s)$ - $V_{task}^{old}(s)$
15: Neighborhood ← {}
16: **if** |TD_val|> ω **then**
17: **for** every $s_i \in N(s)$ **do**
18: s_i.identified ← True
19: Neighborhood ← Neighborhood ∪ $\{s_i\}$
20: $Q_{task}(s_i,a_i) \leftarrow Q_{task}(s_i,a_i) + \alpha\,(r_{s_i \to s} + \gamma V_{task}(s) - Q_{task}(s_i,a_i))$:
$a_i \leftarrow arg\,max_{a \in A} \overrightarrow{(a)} \cdot \overrightarrow{(w_s - w_{s_i})}$
21: accumulator ← accumulator + $V_{task}^{new}(s_i)$ - $V_{task}^{old}(s_i)$
22: **end for**
23: Neighborhood ← getNextNeighborhood (Neighborhood)
24: **While** Neighborhood ≠ ∅ **do**
25: **for** every $n_i \in$ Neighborhood **do**
26: n_i.identified ← True
27: $maxNode$ = maxNode(n_i): returns the neighboring node with maximum value (with respect to the node n_i).
28: $Q_{task}(n_i,a_i) \leftarrow Q_{task}(n_i,a_i) + \alpha\,(r_{s_i \to s} + \gamma V_{task}(maxNode) - Q_{task}(n_i,a_i))$:
$a_i \leftarrow arg\,max_{a \in A} \overrightarrow{(a)} \cdot \overrightarrow{(w_{maxNode} - w_{n_i})}$
29: accumulator ← accumulator + $V_{task}^{new}(n_i)$ - $V_{task}^{old}(n_i)$
30: **end for**
31: Neighborhood ← getNextNeighborhood (Neighborhood)
32: **end while**
33: internal_r ← accumulator + $Per_{err} + \tau$
34: $Q_{exp}(s,a) \leftarrow Q_{exp}(s,a) + \alpha$ (internal_r + $\gamma V_{exp}(s') - Q_{exp}(s,a)$)
35: **end if**
36: $s \leftarrow s'$
37: **end while**
38: **end while**
39: **End**

Algorithm 2 Next Topological Neighborhood Identification

1: **procedure** getNextNeighborhood (N)
2: nextN ← {}
3: **for** every $s_i \in$ N **do**
4: **for** every $z_i \in N(s_i)$ **do**
5: **if** z_i. identified = False **and** z_i. active = True **then**
6: nextN ← nextN ∪ $\{z_i\}$
7: **end if**
8: **end for**
9: **end for**
10: **return** nextN
11: **end procedure**

As we can see, the purpose of the "getNextNeighborhood" is to retrieve the nodes of the next adjacent neighborhood of some neighborhood N. These nodes are checked to be active nodes (have been traversed in the current learning trial) and identified nodes (their value estimates have not been updated in the current learning trial). This resulting neighborhood is then sent to the main algorithm in order to update the value estimates of its nodes, as shown in lines 23 and 31 of the TQ-learning algorithm.

As the TQ-learning algorithm shows, after each update to a state value by the task-dependant Q function, we save the difference between the old and the new values in an accumulator in order to be used later in producing the internal reward, as shown in lines 14, 21 and 29 of the algorithm. Line 33 shows the internal reward term internal_r which corresponds to Eq. (11). This internal reward is then used in updating our task-independent Q function (the exploration function) $Q_{\exp}(s,a)$ with respect to the current

state s and current action a. This function is used to guide the exploration of the agent (deciding on which action to select) as shown in line 4. The action a_i in lines 20 and 28 is chosen to be the action that gives the maximum dot product with the vector $\overrightarrow{s_i s}$ and the vector $\overrightarrow{n_i \max Node}$, respectively. For example, the selected action in line 28 is the one that leads to the neighboring node with the maximum value estimate.

Table 1 gives a description of the global parameters of the TQ-learning algorithm.

5 Simulation 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 with increasing levels of complexity.

The RL agent selects one of the possible actions that correspond to the eight compass directions shown in Fig. 4.

Table 1 Global parameters of TQ-learning algorithm

	Description
Stopping_cond	The stopping condition can be either a maximum number of learning trials allowed for 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. If it is equal or less than the threshold, the algorithm terminates
Map	It is the topological map incrementally built using the ITM algorithm
Threshold ω	The purpose of this threshold is to reduce the number of updates per learning trial. For instance, if the TD_val (the value of the temporal difference) falls under this threshold, it means that the difference between the current and predicted state value estimates is too small (negligible) to be propagated back to other neighboring nodes

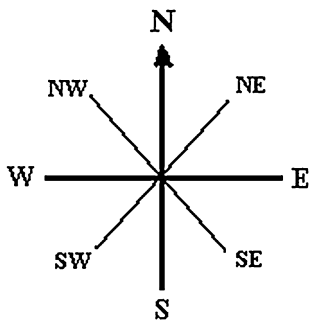


Fig. 4 RL agent's possible actions

Table 2 The parameter values of the algorithms

Algorithm	Parameters
Q-learning	$\gamma = 0.8$, $\alpha = 0.5$, $\varepsilon = 0.3$
Influence Zone	$\gamma = 0.8$, $\alpha = 0.5$, $\varepsilon = 0.3$, $e_{\max} = 0.5$, $\Phi = 10^{-16}$, $\theta_0 = 0.5$
TQ-learning	Task-dependant Q function: $\gamma = 0.8$, $\alpha = 0.5$
	Task-independent (exploration) Q function: $\gamma = 0.99$, $\alpha = 0.2$ $e_{\max} = 0.5$, $\omega = 0.1$, $\tau = -0.009$

The states of the agent environment are the two-dimensional vectors specifying locations in the grid world in which the agent navigates.

The external reward received from the environment after taking an action is defined as follows.

$$r = \begin{cases} -1 & \text{when the action leads to obstacle/border,} \\ +1 & \text{when the action leads to a gloal state,} \\ 0 & \text{otherwise.} \end{cases}$$

The parameters of the TQ-learning algorithm were determined experimentally by performing multiple simulations and fixed to the values given in Table 2 in order to allow for comparisons between the three algorithms in different experiments. Table 2 shows the parameters of the three algorithms along with their associated values which were used in the experiments conducted.

γ is the discount factor, α is the learning rate, ε is the parameter of the ε-greedy exploration method,[3] $e_{\max}$ is the node insertion threshold of the ITM algorithm, Φ and θ_0 are the temporal difference thresholds for updating node values of the immediate and distant neighborhoods, respectively, ω is the temporal difference threshold for updating the node values of the subsequent neighborhoods of a current state, and τ is the punishment term of the internal reward equation. The initial Q-values in the algorithms are set to zero. The algorithms were implemented in Java 1.7, and the simulation experiments were performed on a Core I3 CPU 2.13 GHz platform with 2 GB RAM.

Through preliminary simulations in a 20×20 grid-world environment, some of the TQ-learning parameters were found to have a strong influence on the simulation outcome, and others were found to have relatively insignificant influence. For example, using small values for the ω, threshold allows for more value function updates that lead to fast convergence. Similarly, increasing the punishment term τ speeds up the learning by discouraging the agent from taking actions that do not improve the leaned policy because they will result in small or negative internal reward. As for the explorative setting, different

[3] In ε-greedy exploration, the agent selects the next action randomly with a probability of ε and based on the learned policy with a probability of $1 - \varepsilon$.

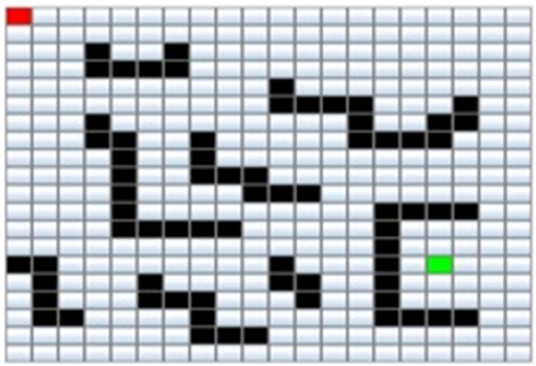


Fig. 5 Static environment

values for the discount factor γ and the learning rate α of the exploration Q function were evaluated and found to slightly affect the learning performance. However, a small learning rate causes slow learning. The best results were achieved with $\gamma \approx 1$ and $\alpha = 0.5$.

We consider two performance measures:

1. The learning progress: repeatedly calculated every specific number of learning trials by testing the agent on its leaned policy (with respect to the task function). During the tests, we suspend the learning, and the agent is given 200 steps to reach the goal. This measure is represented by the number of steps to the goal.
2. The policy value: computed using Eq. (7) in every learning trial, and it is the value of the policy generated by the current task function (task-dependant Q function). It represents the extent to which the state values have been updated.

5.1 Static environment

Here we apply the three algorithms to the static grid-world environment of size 20×20 shown in Fig. 5. 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 on its learned policy after each learning trial by letting the agent act according to the policy derived from the learned task function. The means of 30 experiments are shown in Fig. 6. The number of steps to reach the goal in the Q-learning is 190 on average as shown in Fig. 6a. Influence Zone's agent takes five learning trials to converge to a suboptimal policy, reaching the goal with 27 steps. TQ-learning, on the other hand, needs only two learning trials to learn an optimal policy with 24 steps to the goal.

Figure 6b shows how the agent's external policy evolves with learning in the three algorithms. While the Q-learning appears to require long time to converge to the optimal policy value (the time goes beyond 100 learning trials), Influence Zone learns high policy value faster. However, Influence Zone converged very slowly after 30 learning trials, unable to learn higher policy value. Conversely, TQ-learning was the only algorithm that showed an exponential growth of policy value in the first ten learning trials, finishing the learning process with a policy value of 35.74, whereas the Q-learning and Influence Zone finished the learning with low policy values (18.20 and 23.47, respectively). This clearly shows the impact of the exploration optimization part of the TQ-learning.

5.2 Dynamic environment

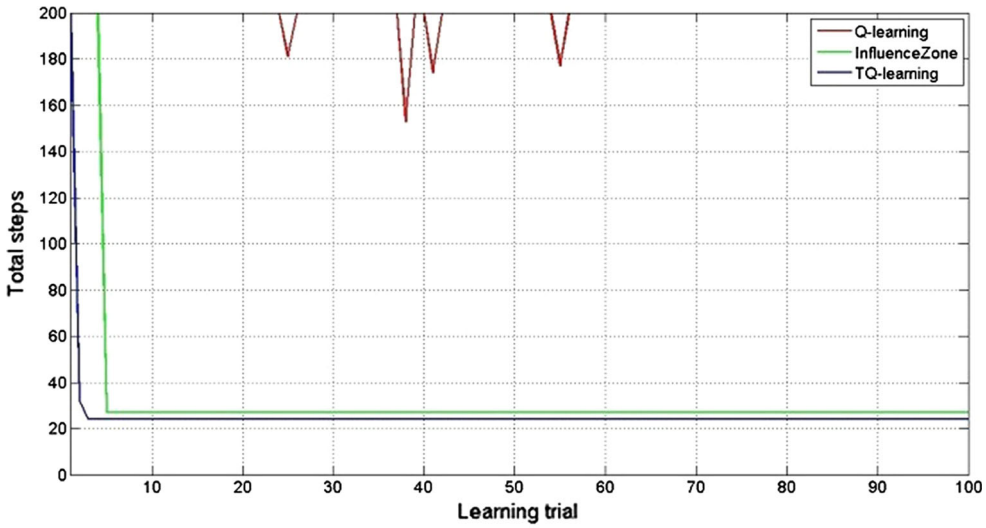
To see how our TQ-learning algorithm reacts to stochastic changes in the agent's environment (from the perspective of the agent), we set up a four-room configuration shown in Fig. 7 where the agent has to learn a path from the start state to the goal state. After 199 learning trials, the short path to the goal state becomes blocked with new obstacles.

Figure 8 shows the change in the environment topology during the learning process, where the red nodes of the topological map represent the free states of the environment. The nodes corresponding to the free states that have been changed into obstacles have been removed when that change was detected during the learning process.

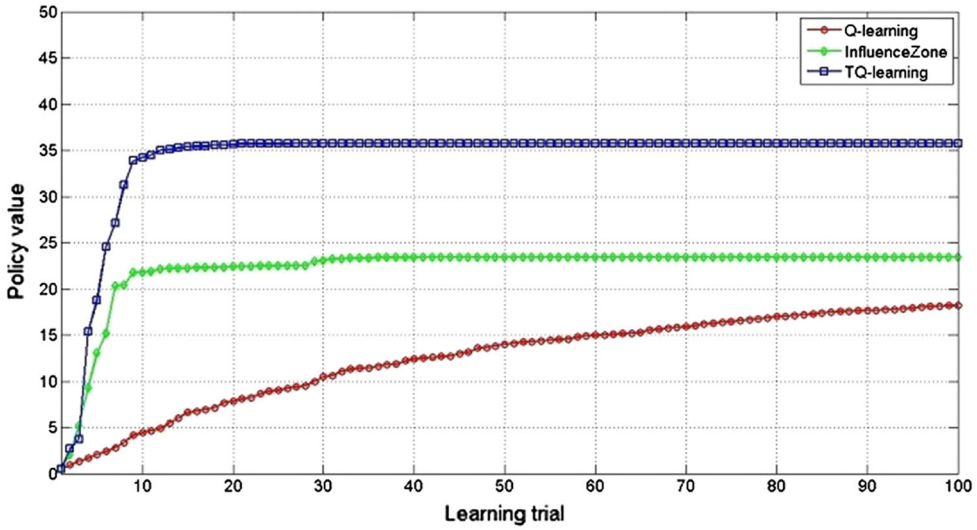
In this experiment, we test the agent on its learned policy every 25 learning trials, with the agent given up to 200 steps to reach the goal. The number of steps the agent takes to the goal has risen in the three algorithms after the change occurred at the learning trial 200 as shown in Fig. 9a. But while the Q-learning and the Influence Zone require many learning trials to learn another new trajectory, TQ-learning adapts more quickly to such a stochastic change in the environment topology, requiring much less trials and converging to an optimal policy with 21 steps to the goal at the end of the learning process.

Regarding policy value, TQ-learning was able to reach a policy value of 65.51 within the first nine learning trials, while Q-learning and Influence Zone were below 40 in policy value as clearly shown in Fig. 9b. We can detect from the figure that the algorithms showed a slight decrease in policy value when the change occurred at the trial 200, which is mainly because of the states that became obstacles with zero values. At the end of the learning process, TQ-learning attained the highest policy value of 68.67, whereas Q-learning and Influence Zone attained 30.37 and 41.51, respectively.

In the last experiment on dynamic environments, we set the number of learning trials to 200. Every 20 learning trials, the environment changes stochastically to one of the three configurations shown in Fig. 10, with the number of



(a) The learning progress (number of steps to the goal)



(b) Policy value (with respect to the task function)

Fig. 6 Comparison result of Q-leaning, Influence Zone and TQ-learning

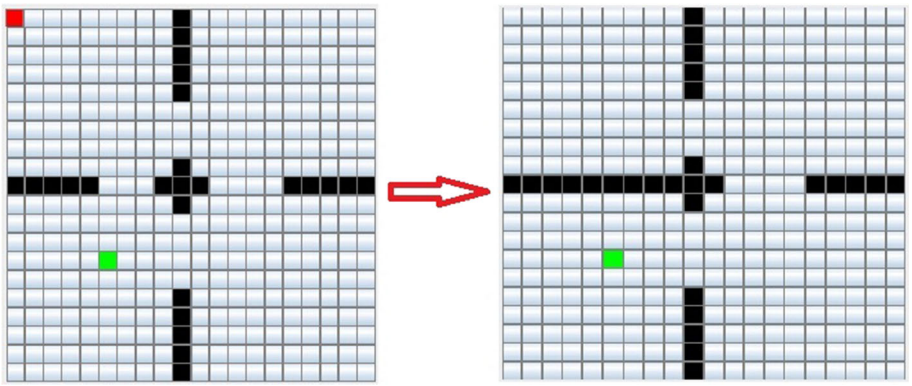


Fig. 7 Dynamic 20 × 20 environment (one downward path blocked stochastically)

the selected configuration drawn randomly from a uniform distribution over the integer range [1, 3]. In this challenging experiment, the position of the goal is changing randomly along with its surrounding obstacles, as Fig. 10 illustrates. We test the RL agent on its learned policy every ten learning trials.

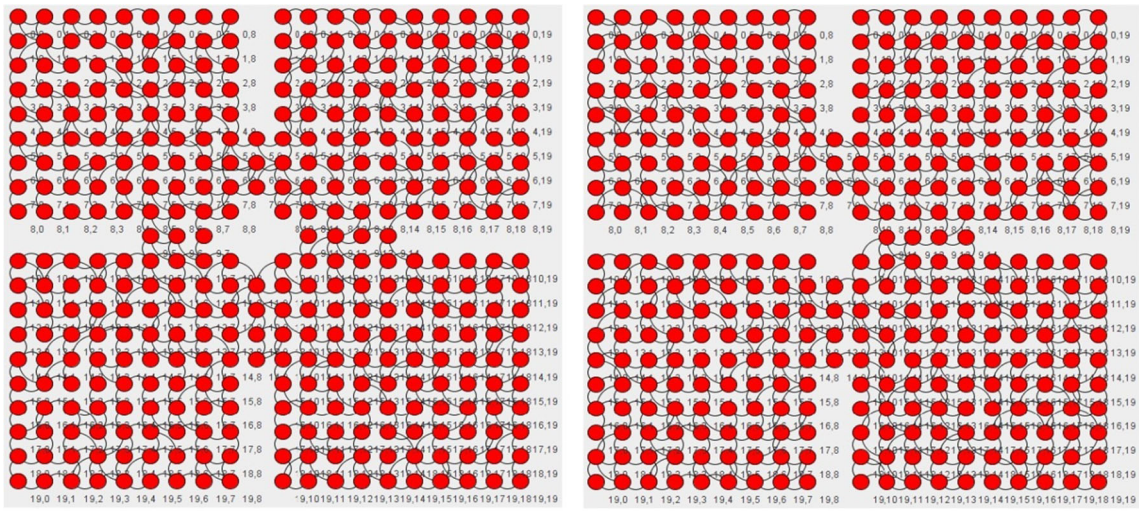


Fig. 8 The ITM map: **a** before the change occurs, **b** after the change occurs

Influence Zone algorithm failed to adapt to this challenging stochastic topology change as shown in Fig. 11a. The agent in the algorithm could not continue the learning process, permanently taking actions that lead to some previous goal position at the trial 40 without being able to change its policy to discover the new goal position. Q-learning also showed poor learning progress, taking more than 200 steps to reach a new goal position.

In contrast to Q-learning and Influence Zone, TQ-learning algorithm was able to adapt quickly to this stochastic change in the goal position and its topological neighborhood, learning an optimal policy that leads to the goal position of the final selected configuration within only 22 steps by the end of the learning process as can be easily observed in Fig. 11a.

Figure 11b shows that the policy value of the task function in TQ-learning continued to increase despite the frequent change in the main task of the agent due to the goal position's stochastic change. Unlike TQ-learning algorithm, Q-learning was heavily affected by the stochastic change in the goal position, with the policy value decreasing to around 0.7 after each change. Overall, the policy value in the TQ-learning was much higher than that of the Q-learning in all the learning trials. TQ-learning and Q-learning completed the learning process with a policy value of 58.46 and 5.94, respectively, whereas Influence Zone was not able to complete the learning for the reason discussed earlier.

Table 3 shows the computational and time costs of the three algorithms along with the policy value results in all the conducted experiments.

The computational cost is represented by the number of value function updates (averaged over the executed learning trials) and referred to in the table as N. For representing the time cost and because the three algorithms converge to different policy values as illustrated in Figs. 6, 9 and 11, we display the time taken by each algorithm to converge to a policy value of 30, referred to as CT (convergence time) together with the execution time ET.

The table shows four measures N, ET, CT and $V(\pi)$ representing the computational cost, the execution time measured in seconds, the convergence time measured in seconds and the policy value at the end of the learning process, respectively. Env1, Env2 and Env3 refer to the environments depicted in Figs. 5, 7 and 10 respectively.

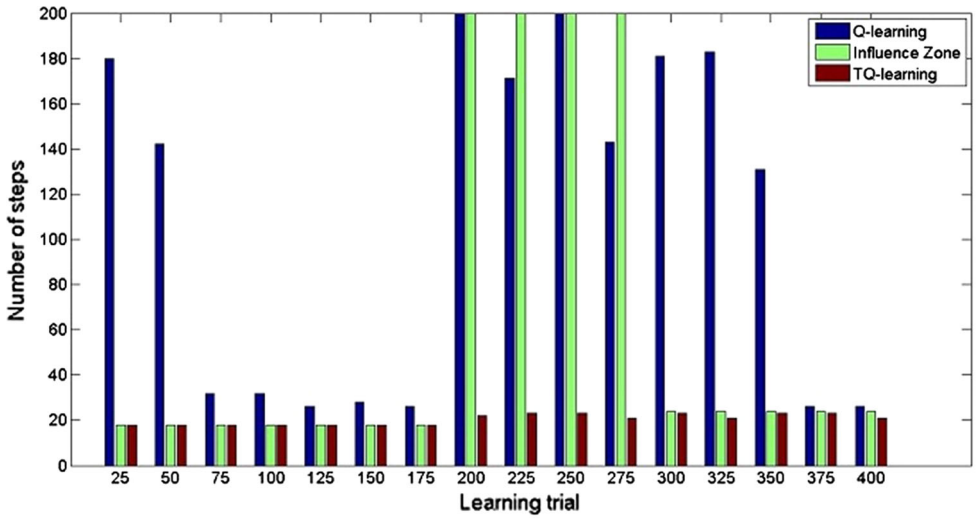
The infinity sign in the table means the algorithm requires time close to infinity to converge to a policy value of 30. The results of the Influence Zone in Env3 cannot be measured because the Influence Zone failed to continue the learning process in such a dynamic environment as discussed earlier. The table reconfirms that TQ-learning converges faster than the other two algorithms and reaches higher policy values at the end of the learning, but with a relatively long run time. The table also shows that the TQ-learning tends to take more value function updates than the other two algorithms.

6 Discussion

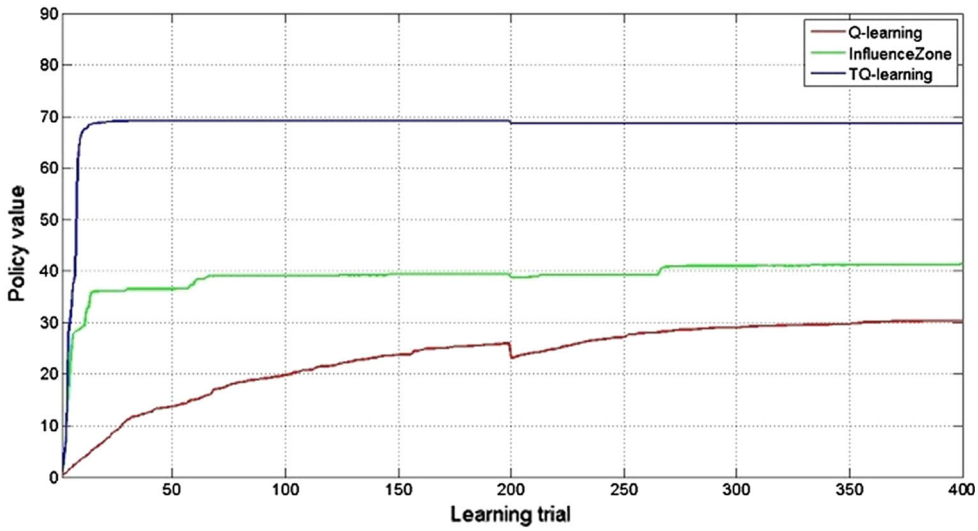
Here we suggest a set of interpretations for the behavior of our proposed algorithm in all the experiments conducted.

6.1 The performance in the static environment

As we have seen in Fig. 6a, TQ-learning algorithm converged to the optimal policy with just two learning trials on average, faster than Q-learning and Influence Zone



(a) The learning progress (number of steps to the goal)



(b) Policy value (with respect to the task function)

Fig. 9 Comparison result of Q-leaning, Influence Zone and TQ-learning in the dynamic environment of four-room configuration

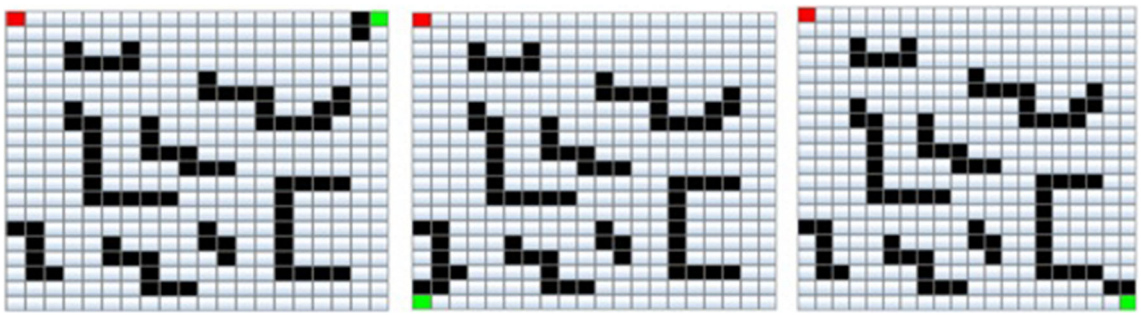


Fig. 10 Stochastically changing 20 × 20 environment (goal and its surrounding obstacles change between the three configurations)

algorithms. One of the reasons for this is that TQ-learning takes advantage of the incrementally built topological map of the environment to do multiple value function updates in the current state and its topological neighbors instead of a single update like in the case of the Q-learning algorithm. In addition, and most important, the exploration of TQ-

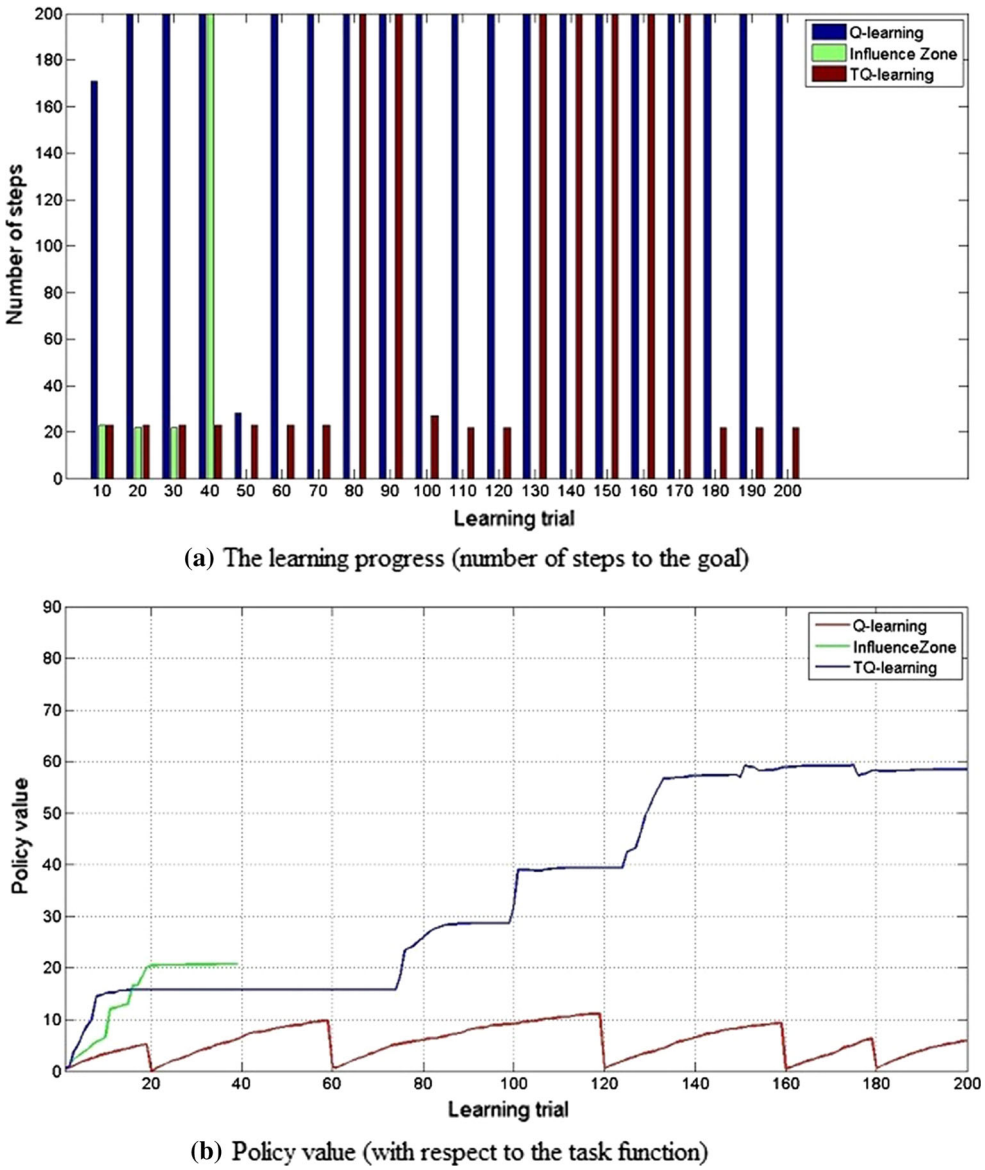


Fig. 11 Comparison result of Q-leaning, Influence Zone and TQ-learning in the dynamic environment of changing goal

Table 3 Summary of the policy value performance

	Env1				Env2				Env3			
	N	*ET*	*CT*	$V(\pi)$	*N*	*ET*	*CT*	$V(\pi)$	*N*	*ET*	*CT*	$V(\pi)$
Q-learning	570	0.70	∞	18.20	211	1.91	1.80	30.37	6472	10.73	∞	5.94
Influence Zone	12,431	4.22	∞	23.47	4874	4.52	2.60	41.51	–	–	–	–
TQ-learning	17,180	17.60	3.92	35.74	12,125	5.32	1.38	68.67	19,433	18.76	14.86	58.46

learning agent during the learning process is driven by how much the next action improves the learned policy, which is represented by our proposed internal reward, instead of taking the next action randomly. This guided exploration makes the topological map of the TQ-learning agent grows in coverage, adding more nodes than the map of the Influence Zone agent. And since having more nodes in the topological map means the agent has more information about its environment, the agent can discover the optimal solution (shortest trajectory) faster than the Influence Zone agent.

It has been found that the Influence Zone algorithm generates on average 266 nodes in the topological map by the end of the learning process, while the TQ-learning

generates 330 nodes, which is the number of all the free states in the considered static environment, within just the first eight learning trials.

The performance of TQ-learning in terms of policy value was significantly higher than that of Q-learning and Influence Zone, as we saw in Fig. 6b. This is a direct consequence of the way the TQ-learning agent takes action in its environment. For instance, Q-learning and Influence Zone use the ε-greedy action selection, which causes the agent to take, with high probability, the next action based on the current Q-values of the task function. This means that once the agent finds a path to the goal, it will mostly traverse the same path next time repeatedly, updating only the values of the states along that path. On the other hand, TQ-learning uses its exploration optimization strategy that encourages the actions that improve the learned policy. Thus if the agent learns one path to the goal, and the values of the states in that path have converged (no further improvement can be done) after being visited a number of times, the agent will try another path whose states' values have not been improved. Consequently, much more states have their values updated than in the Q-learning and Influence Zone, causing high policy value.

6.2 The performance in the dynamic environment

TQ-learning has shown a quick adaptation to a stochastic change in the topology of the environment as illustrated in Fig. 9. While other algorithms require many trials to learn a new path to the goal after the closing of the first one, TQ-learning directly learned a new path.

To account for the TQ-learning fast adaptation, let us first consider the behavior of the other two algorithms. When the Q-learning agent encounters an obstacle which was previously a free state, it receives a negative reward and it will keep taking the action that leads to that new obstacle many times until the Q-value associated with that action becomes low enough to make the agent try another action. Therefore, lots of trials are needed to find a new solution (path). As for the Influence Zone agent, when it encounters this change, it will not take the action that leads to the obstacle again, because it can remove the node associated with the obstacle from its topological map. However, since the ε-greedy exploration of the Influence Zone directs the agent, with high probability, to execute the first learned trajectory, the topological map is not developed enough (few edges and nodes) at the time of the change. That is why Influence Zone agent cannot directly find another trajectory to the goal.

In contrast, the topological map of the TQ-learning agent is more developed at the change time, having more nodes and edges that can be traversed to find a new path mainly due to the TQ-learning's exploration optimization.

A remarkable performance has been achieved by the TQ-learning in the challenging dynamic environment of changing goal (Fig. 10). Despite the repeated stochastic change in the goal position, TQ-learning was the only algorithm to adapt to this change and learn an optimal policy as shown in Fig. 11. In this experimental setting, Q-learning was not able to discover every new goal position in a short time. This is because in order for Q-learning to change its learned policy (to find the new goal position), it has to take many random steps in the environment. This was apparent in Fig. 11a when it took more than 200 steps to find every new goal position.

Likewise, Influence Zone showed a great deficiency in that it was not able to continue the learning process right after the first change in the goal position. The reason for this is that when the agent visited a previous goal position, the Q-values of the neighboring nodes, according to the ε-greedy exploration, can only direct the agent to this same position since the topological map is not fully developed in this stage. Thus, the agent entered an infinite loop of taking actions that lead to the same position.

On the other hand, the main reason for the successful behavior of the TQ-learning in this environment is that it is capable of rapidly changing its exploration policy (using its exploration optimization strategy) to seek experiences that have high potential of increasing the state value estimates. Accordingly, it was successfully able to keep track of every new goal position.

Regarding policy value in the two experiments on the dynamic environments, we noticed that the TQ-learning obtained much higher policy value than the number obtained by Q-learning and Influence Zone, as shown in both Figs. 9b, 11b. The reason for this performance is exactly as was given when discussing the policy value performance in the static environment.

It should be noted that employing the ITM map in our proposed algorithm has offered a set of advantages that contributed to the successful results detailed previously. Most important advantage is the speed in constructing the topological map where node creation occurs swiftly once a new state is observed, something that would be difficult if we relied on other topology learning methods. Another advantage is that any agent exploring its surrounding can record the topological information in an ITM and then use it to navigate from one position (node in the ITM) to another and simultaneously propagate the state value estimates backward to other topological neighborhoods. This subsequently enabled us to compute our internal reward value as in Eq. (11) and facilitated our proposed strategy

for guided exploration. Besides, ITM has few parameters and calculations when compared to other unsupervised learning techniques. However, ITM has a disadvantage in that it cannot be directly applied to some high-dimensional data such as raw pixel data or video data that is highly prone to the presence of noise, and different solutions might be adopted in such situations.

Finally, it has been noticed that the execution time of TQ-learning algorithm is relatively longer than that of the Influence Zone with the same computing resources, mainly because of the additional task-independent Q function (the exploration function), which was introduced to guide and optimize the exploration strategy of the agent, and its update process. However, the high convergence rate of the algorithm compensates for this increase in run time as shown in Table 3.

7 Conclusion

In this paper, we have proposed TQ-learning, a topology-based RL algorithm for mobile robot navigation. The algorithm adopts the ITM model for constructing a topological representation of the agent's environment. The neighborhood relationships information and the node values of the incrementally built ITM are used to spread the value function updates over the observed states and to generate an internal reward, respectively. The internal reward is used to optimize the action selection mechanism of the RL agent. We compared the proposed algorithm to the original Q-learning and Influence Zone algorithms in static and dynamic environments. The simulation results indicate that TQ-learning has higher learning performance than those algorithms and can adapt quickly in response to any stochastic changes in the environment topology.

TQ-learning has two advantages over the existing topological method (Influence Zone). First, in TQ-learning, the exploration is independent of the task, which enables the agent to discover and adapt efficiently to any changes in the environment topology, especially the change in the goal position, which suggests the application of the algorithm in problems with sequential tasks. Second, TQ-learning has higher policy value as a result of its exploration strategy which is driven by how much the state values are evolving, leading to a faster convergence. As a second contribution of this work, we provided a better understanding of how the information encoded in the environment's topological map can be used to speed up the value function updates as well as to optimize the exploration strategy of the RL agent.

We have demonstrated in this study the utility of the topological maps for enhancing RL for mobile robot navigation in discrete state space. It would be interesting to investigate the potential of the algorithm in continuous state or action spaces.

Acknowledgments This research is supported by High Impact Research MoE Grant UM.C/625/1/HIR/MoE/FCSIT/10 from the Ministry of Education Malaysia.

References

1. Sutton RS (1991) Dyna, an integrated architecture for learning, planning, and reacting. ACM SIGART Bull 2(4):160–163
2. Moore AW, Atkeson CG (1993) Prioritized sweeping: reinforcement learning with less data and less time. Mach Learn 13(1):103–130
3. Hwang KS, Jiang WC, Chen YJ (2012) Tree-based Dyna-Q agent. The 2012 IEEE/ASME international conference on advanced intelligent mechatronics, Kaohsiung, Taiwan
4. Sutton RS (1988) Learning to predict by the methods of temporal differences. Mach Learn 3(1):9–44
5. Watkins CGCH (1989) Learning from delayed rewards. King's College, Cambridge
6. Wiering M, Schmidhuber J (1998) Fast online Q(λ). Mach Learn 33(1):105–115
7. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
8. Peng J, Williams RJ (1996) Incremental multi-step Q-learning. Mach Learn 22(1–3):283–290
9. Touzet CF (1997) Neural reinforcement learning for behaviour synthesis. Rob Auton Syst 22(3–4):251–281
10. Zeller M, Sharma R, Schulten K (1997) Motion planning of a pneumatic robot using a neural network. IEEE Control Syst 17(3):89–98
11. Busoniu L, Babuska R, Schutter BD, Ernst D (2010) Reinforcement learning and dynamic programming using function approximators. CRC Press, New York
12. Millán JDR, Posenato D, Dedieu E (2002) Continuous-action Q-learning. Mach Learn 49(2–3):247–265
13. Munos R, Szepesvári C (2008) Finite-time bounds for fitted value iteration. J Mach Learn Res 1:623–665
14. Dietterich TG (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition. Artificial intelligence research
15. Takase N, Kubota N, Baba N (2012) Multi-scale Q-learning of a mobile robot in dynamic environments. SCIS-ISIS, Kobe
16. Braga APS, Araújo AFR (2003) A topological reinforcement learning agent for navigation. Neural Comput Appl 12:220–236
17. Braga APS, Araújo AFR (2006) Influence Zones: a strategy to enhance reinforcement learning. Neurocomputing 70(1–3):21–34
18. Dai P, Strehl AL, J. Goldsmith (2008) Expediting RL by using graphical structures. Proceedings of the seventh international joint conference on autonomous agents and multiagent systems
19. Sutton RS, Barto AG (1998) Introduction to reinforcement learning. MIT Press/Bradford Books, Cambridge
20. Sutton RS (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. Proceedings of the seventh international conference on machine learning
21. Luciw M, Graziano V, Ring M, Schmidhuber J (2011) Artificial curiosity with planning for autonomous perceptual and cognitive development. Development and learning (ICDL), Frankfurt
22. Zahedi K, Martius G, Ay N (2013) Linear combination of one-step predictive information with an external reward in an episodic policy gradient setting: a critical analysis. Front Psychol. doi:10.3389/fpsyg.2013.00801

23. Remolina E, Kuipers B (2004) Towards a general theory of topological maps. Artif Intell 152(1):47–104
24. Thrun S, Buckenz A (1996) Integrating grid-based and topological maps for mobile robot navigation. Proceedings of the thirteenth national conference on artificial intelligence AAAI, Portland, Oregon
25. Kohonen T (1989) Self-organization and associative memory. Springer, Berlin
26. Kohonen T (2001) Self-organizing maps. Springer, Berlin
27. Martinetz T, Schulten K (1991) A neural-gas network learns topologies. Artif Neural Netw, Amsterdam, pp 397–402
28. Fritzke B (1995) A growing neural gas network learns topologies. Adv Neural Inf Process Syst 7:625–632
29. Jockusch J, Ritter H (1999) An Instantaneous Topological Mapping Model for correlated stimuli. Proceedings of the IJCNN'99, Washington, DC
30. Ng AY, Jordan M (2000) PEGASUS: a policy search method for large MDPs and POMDPs. Proceedings of the sixteenth conference on uncertainty in artificial intelligence, San Francisco