Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Neurocomputing 70 (2006) 21–34
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Influence zones: A strategy to enhance reinforcement learning

Arthur Plínio de S. Braga[a], Aluízio F.R. Araújo[b,*]

[a]Departamento de Engenharia Elétrica, Universidade Federal do Ceará, Campus do Pici-Bl.705, Caixa Postal 6001, Fortaleza, CE, 60455-760, Brazil
[b]Departamento de Sistemas da Computação, Universidade Federal de Pernambuco, Av. Prof. Luís Freire, s/n, Cidade Universitária, Recife, PE 50740-540, Brazil

Available online 30 August 2006

Abstract

Reinforcement Learning (RL) aims to learn through direct experimentation how to solve decision-making problems. RL algorithms often have their practical applications restricted to small or medium size problems—mainly because of their strategies for value function estimation demanding very high number of interactions. To overcome this difficulty, we propose to enhance RL performance by updating several state (or state–action) values at each interaction. Therefore, the influence zone algorithm, an improvement over the topological RL agent (TRLA) strategy, allows to reduce the number of requested interactions. Such a reduction is based on the topological-preserving characteristic of the mapping between states (or state–action pairs) and value estimates. The comparison of the influence zone approach with seven other RL algorithms suggests that the proposed algorithm is among the fastest to estimate the value function and that it takes less value function updatings to perform such an estimation. The influence zone algorithm also presents a remarkable flexibility in adapting its policy to changes of the input space topology.

Keywords: Reinforcement learning; Self-organizing map; Instantaneous topological map; Learning acceleration

1. Introduction

Reinforcement learning (RL) [1,9,10,23,29] has become one of the most active research areas in artificial intelligence [29] in which the vast possibilities of RL applications for real world problems is one of the main motivations for the research. However, current RL algorithms often have their actual application restricted to toy or medium-size problems because of their standard strategies for value function estimation often characterized by computational requirements growing exponentially with the number of state variables [29]. Currently, nearly all RL algorithms are based on temporal-difference (TD) methods [23,29] to estimate the value function by backward propagation of the returns and value function estimates to associate each state, action–state pair, with a prediction of future consequences [23]. Nevertheless, TD methods also involve high number of iterations to estimate the value function. Hence, this pitfall is an open research topic in which the main approaches to reduce the number of iterations and accelerate RL algorithms include eligibility traces [21,25,32], generalization strategies [18], [27,30] and model-based alternatives [20,28].

Some recent papers [3,7,18,27,30] employed combinations of the mentioned approaches to enhance RL performance through a strategy to accelerate the learning. The topological RL agent (TRLA) [3] is one of these works in which self-organizing map (SOM) network [13] properties are used to estimate the value function improving the propagation of real returns and value function estimates not only in time, as typical TD methods, but also considering the topological connections between the states.

The influence zone algorithm, defined in Section 5, is an improvement of the algorithm TRLA [3]. This new option has two main original characteristics in comparison with TRLA to be pointed out: first of all, the influence zone algorithm does not update all states, or action–state pairs, in a considered topological neighborhood. The new algorithm updates states or action–state pairs with their

*Corresponding author. Tel.: +55 81 21268430x4321; fax: +55 81 21268438.
E-mail addresses: arthurp@icmc.usp.br (A.P.S Braga), aluizioa@cin.ufpe.br (A.F.R Araújo).

0925-2312/$ - see front matter
doi:10.1016/j.neucom.2006.07.010

values effectively affected (the influence zone) by the change in the value of the current state (or action–state pair). This novelty reduces the number of updatings in comparison with TRLA, and it points out to a RL algorithm faster than TRLA. Secondly, the influence zone algorithm is an incremental version of TRLA in regard to its value function update strategy: depending on an error threshold, the new algorithm performs one-step or n-step value function updatings. These two new features permit the influence zone algorithm to reduce the number of updatings and to be employed to situations that the TRLA did not respond suitably, such as a changeable environment (Section 6.3).

Simulation tests on a navigation task show that the influence zone algorithm often converges faster by demanding in average less value function updatings than six of the most important RL algorithms and the TRLA [3].

This paper is organized as follows: Section 2 presents an overview on how to accelerate RL by spreading the TD error. The influence zone concept is presented in Section 3. The essential knowledge on the self-organizing algorithm used to generate a topological map in the implementation of the influence zone is provided in Section 4. Section 5 presents the algorithm to implement influence zone followed by simulations comparing it with other RL algorithms presented in Section 6. A general discussion on the obtained results and final conclusions are shown in Section 7.

2. Spreading the TD error

Several RL algorithms [18,20,21,25,27,28,30,32] adopt mechanisms to adjust not only the immediate state value estimate, but also the value estimate of a set of states per interaction. Eq. (1) exemplifies the general update rule of the value function in these algorithms[1]:

$$V(s) := V(s) + \alpha H(s)[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \quad \forall s \in S_n. \tag{1}$$

This general update rule is similar to its one-step version [29]. However, it spreads the TD error, $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, calculated in the transition between states s_t and s_{t+1}, over all states within a subset of the state space, $S_n \subseteq S$. The spreading function $H(s)$ is used to adjust value estimates of all states belonging to S_n instead of only the previous state value estimate. There are different views to the spreading function $H(s)$. In the eligibility trace approach, $H(s)$ is the eligibility trace of a state s representing the degree to which each state has been visited in the recent past [9,29]. In [16], the function $H(s)$ is denoted as transitional proximity and it is calculated through a table that stores all learned state transitions. Touzet [30] approximates $H(s)$ using a SOM [13] in which each state s is represented by a node (n_s) and S_n is the set of nodes connected to n_s, at each interaction. In this case, only the value estimate of each node immediately connected to the current node is adjusted. In generalization approaches [9,18,27,29], the spreading function is associated with the aggregation of similar states. The RL agent–environment model used in the Dyna approach [28] works like the spreading function. Ribeiro [22], in his QS-learning, computes $H(s)$ as an Euclidean distance-dependent function of state s_t and all other states in S_n.

From the mentioned above, one can see that $H(s)$ implementations somehow consider a distance measure between the current state and all states in S_n. However, two main limitations can be observed in the cited implementations of $H(s)$: (i) the necessary calculations generally demand a significant computational effort (such as in eligibility trace approaches [21,25,32], and in transitional proximity [16]; and (ii) the $H(s)$ function is only considered within an immediate neighborhood of the current state s, restricting S_n to a small set of states at time t (as in [22,30]). Regarding the mentioned points, the second option reduces the computational effort at expenses of a worse performance in the value function estimation in comparison with eligibility trace algorithms.

2.1. Applying SOM for spreading the TD error

Recent works report on the use of SOM to accelerate RL by spreading the TD error [3,7,18,27,30]. In RL, the capacity to preserve the topology of the input space [16] allows learning over a compact representation of the input space, where the transitions between regions of this space can be preserved. In other words, solutions of a Markov decision process (MDP) [1] can be obtained from a smaller MDP in which the negative role of the curse of the dimensionality [1] and the curse of modeling [2] are attenuated.

Concerning capabilities and limitations of a number of models combining SOM and RL, two distinct features are considered to the algorithm proposed in this paper:

- *The construction of a topological map*: models that use constructive topological maps [7,18] tend to learn more accurately in comparison with RL models based on fixed maps [27,30], as the original Kohonen algorithm [13,24], because such dynamic maps can construct better representations even in complex environments [4,8].
- *The way to use the topological map*: topological relationships can be used to propagate the value update along the state or state–action pair space [27,30]; however, some works just consider the SOM as a tool to produce a compact representation [7,18].

Taking the previous discussion as a starting point, the following premises are considered for a new agent: (i) maps with variable structure are more precise to represent transitions of the input space states and (ii) neighborhood relations encoded in the topological map is an information

[1]The reasoning for the action-value function is straightforward.

to facilitate value function updates of a significant number of states. These two premises will be used in Section 5 to implement the concept to be introduced in the next section.

3. The influence zone

In an RL agent, state transitions, following a greedy policy π, are generated through reinforcements obtained by agent–environment interactions [9,29] as sketched in Fig. 1. The considered past states have their values related with the current state value as follows:

$$V^{\pi}(s_{t-n}) = E\left[r_{t-n+1} + \gamma r_{t-n+2} + \cdots + \gamma^{n-1} r_t + \gamma^{n} V^{\pi}(s_t)\right], \tag{2}$$

where $V^{\pi}(s)$ is the state-value of state s for a greedy policy π, and $E[\cdot]$ denotes the expected value. For the estimation

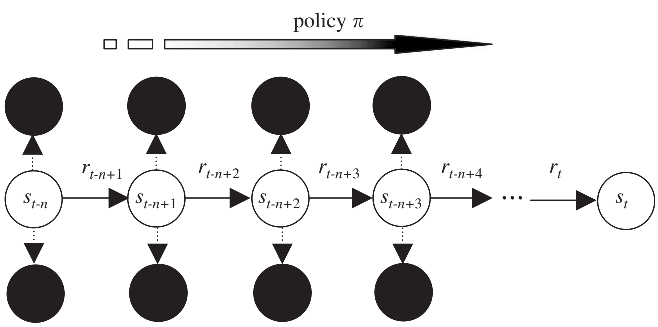

Fig. 1. State transitions for an RL agent with policy π. White circles represent states in time sequence, and dark circles represent close states that are not visited in sequence (based on [23]).

of the value function, Eq. (2) explicitly shows that a change in $V^{\pi}(s_t)$ affects the expected values of s_{t-n}, s_{t-n+1}, $s_{t-n+2}, \ldots, s_{t-1}$. Thus, if such a set of affected states is known for each s_t, the value function updatings can be restricted to this set. This restriction accelerates the estimation of the value function.

However, it is not a simple task to define a set of states to be influenced by a change in $V^{\pi}(s_t)$. The close states in Fig. 1 illustrate that more than one trajectory can reach s_t through a given policy π. Ribeiro [23] argues that beyond temporal sequence dependence, Eq. (2), there is an structural dependence in which states spatially close to s_{t-n}, s_{t-n+1}, $s_{t-n+2}, \ldots, s_{t-1}$ can have also their values affected by changes in $V^{\pi}(s_t)$. The figure below illustrates how a change in one state can structurally affect the value function shape.

Comparing the environment sketched in Fig. 2a with those described in Figs. 2b and c one can see that the differences consist only in one obstacle state. Even though, such a single change, from a free state to a state occupied by an obstacle, affected the value function of the states around the obstacle, evidencing the structural dependency of those states with the changed state. Thus, to discover this set of states, from now onwards denoted as the *influence zone* of s_t, $T(s_t)$, which has their values computed by Eq. (2), one can use topological neighborhood in the input space S constructed through a topological map. Topological maps can represent structural and temporal dependencies between states in $S \subset \Re^{n}$, $n \geqslant 1$, through its connectivity information that encodes relationships of similarity in the input space.

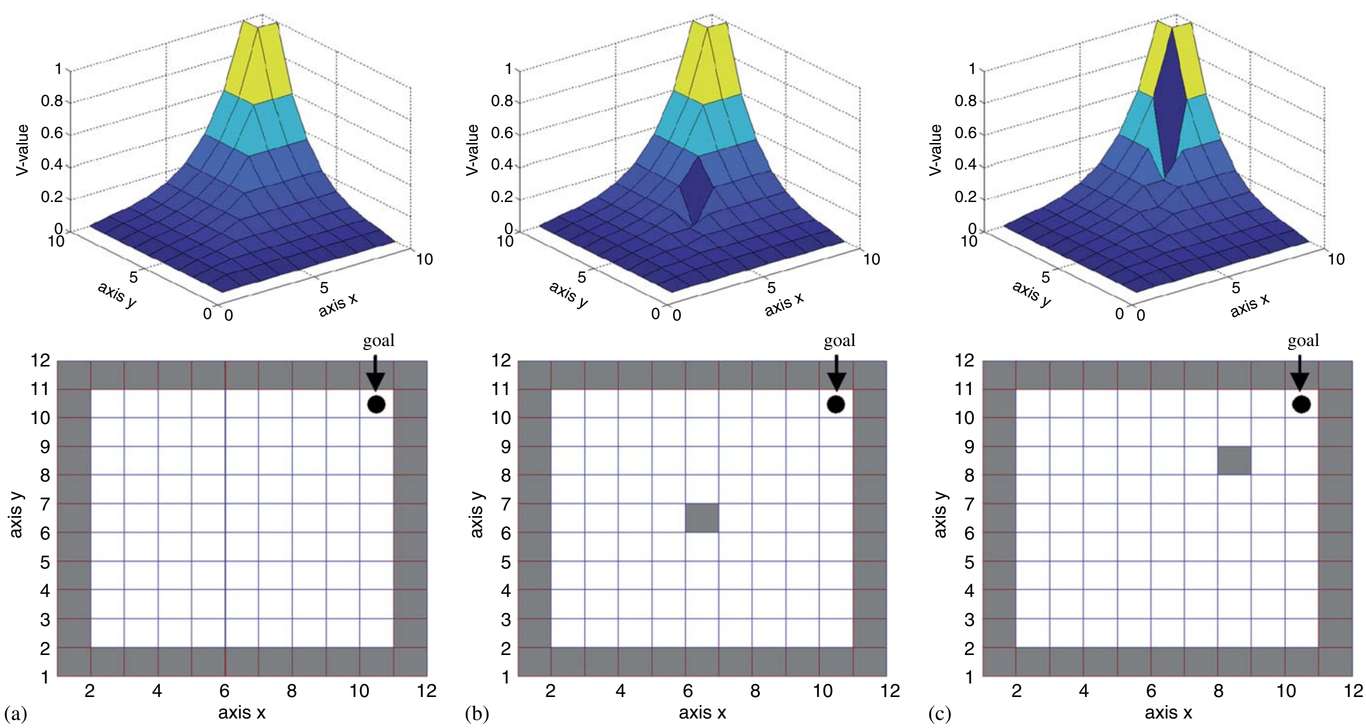

Fig. 2. The value function for: (a) a simple 12×12 grid environment, and the changes in the value function shape with the change of one state (b–c).

The topological map capability of representing an input space in $\Re^n$ [4,5,11,13,14] is particularly important to extend the application of influence zone concept to n-dimensional RL problems ($n>2$). However, in order to label an algorithm derived from the influence zone concept as recommendable to improve RL efficiency, two targets must be reached: (i) this strategy should reach a performance, in the value function estimations, at least close to eligibility traces [21,25,32] and (ii) the influence zone should execute less updatings than the considered RL models, by restricting the updatings to subsets of S (as in [22,30]).

The determination of $T(s_t)$ considers the following points: (i) two adjacent spatial states have a tendency to be also adjacent in the value space, this was stated by McCallum [16], who argued that in some sense an RL algorithm basically learns a mapping (V: $S \rightarrow \Re$ or Q: S, $A \rightarrow \Re$) that preserves the topology; and (ii) Since states in the influence zone of s_t are taken to the current state by a greedy policy, their values should be smaller than the current state value, $\forall s \in T(s_t)$, $V(s) < V(s_t)$. The first point is common to other RL and SOM combinations [3,7,18,27,30], whereas the second point reduces the number of updatings. As an example, the TRLA [3], an initial version of the algorithm described in Section 5, updates the value function estimate along all topological neighborhoods. However, given s_t, the influence zone updates only those states that satisfy $\forall s \in T(s_t)$, $V(s) < V(s_t)$. An algorithm to use influence zone to estimate the value function is described in Section 5; however, the adopted map for the simulations performed is shown next.

4. The instantaneous topological map

Kohonen proposed the SOM consisting of an array of M neurons, in which each particular node n has a weight vector $\mathbf{w}_n \in S$ associated to it, and it is linked to other nodes through pre-established connections. For being pre-determined, the edges of SOM may impair learn of the topological structure of the stimuli space which is initially unknown. Moreover, pre-determined connections can distort the information of complex environments containing obstacles of unknown shape and position.

Two very representative endeavors to construct more accurate topological maps include the works of Martinetz and Schulten [14] and the one proposed by Fritzke [4,5]. Martinetz and Schulten proposed a competitive Hebbian learning model in which learning considers the creation of connections between nodes. This kind of learning employs a winner-take-all method to create a new edge between the two nearest nodes that respond to a given stimulus $\xi \in S$. Fritzke improved the topology representing networks by proposing a mechanism to include and delete nodes in the topological map allowing the topological map to adapt itself to slow changing topologies.

The instantaneous topological map (ITM) [8] is a more recent model that preserves the ability to insert and exclude nodes, and builds an accurate description of an environment. Such flexibility is obtained through a technique very used in graphics computing for object and environment description: the Delaunay triangulation [6]. Furthermore, the ITM is a fast-learning method, potentially suitable to on-line applications. As proposed in [8], the ITM employs Delaunay triangulations regarding the positions of nodes and edges. The ITM can be described as follows:

1. *Determination of the first and second winner*: considering a stimulus ξ provided by the environment, determine the nearest and the second-nearest nodes, n and s, respectively,
$$n = \arg\min_i \|\xi - \mathbf{w}_i\|, \quad s = \arg\min_{j, j \neq n} \|\xi - \mathbf{w}_j\|, \tag{3}$$
where i, j, n and $s \in M$.
2. *Updating of the weight vectors*: for the winner node n, update the components of the weight vector towards the stimulus ξ considering a learning step $\in$:
$$\Delta \mathbf{w}_n = \in (\xi - \mathbf{w}_n). \tag{4}$$
3. *Creations and deletions of connections and exclusion of disconnected nodes*: considering the first and the second winner, link them through an edge $\overline{ns}$, provided that it does not exist. After such an edge creation, the presence of non-Delaunay edges must be avoided. Hence, for each node $m \in N(n)$, where $N(n)$ is the set nodes connected to n. Find any non-Delaunay edge $\overline{nm}$ determined by the criterion (5), and remove $\overline{nm}$.
$$\forall m \in N(n) : \text{If}(\mathbf{w}_n - \mathbf{w}_s) \cdot (\mathbf{w}_m - \mathbf{w}_s) < 0, \tag{5}$$
where $\mathbf{w}_s$, $\mathbf{w}_n$ and $\mathbf{w}_m$ are the weight vectors associated with nodes s, n and m. If any edge is deleted, exclude any remaining node with links.
4. *Creation and deletion of nodes due to necessity of accuracy*: If the stimulus ξ satisfies the following criteria:
$$(\mathbf{w}_n - \xi) \cdot (\mathbf{w}_s - \xi) > 0 \text{ and } \|\mathbf{w}_n - \xi\| > e_{\max}, \tag{6}$$
where $e_{\max}$ is a given distance, then insert a new node y, with $\mathbf{w}_y = \xi$, and connected with n by a new valid Delaunay edge $\overline{ny}$. On the other hand, if
$$\|\mathbf{w}_n - \mathbf{w}_s\| < 0.5 e_{\max}, \tag{7}$$
remove node s.

In Delaunay triangulation, each triangle can be associated with a circle [6]. Inside this circle can not exist any of the points used as vertices for the triangulation, otherwise there is a non-Delaunay edge that should be deleted. The geometric relations in Eqs. (5–7) guarantees Delaunay edges. Next section presents how ITM will be used to implement the influence zone algorithm.

5. The influence zone algorithm

Despite similarities with some SOM-based RL agents like [3,25,29], the concept of *influence zone* proposed in Section 3 aims to select states, or action–state pairs, in a topological neighborhood to have their values updated.

Thus, similarly to the TRLA [3], some attributes are included in each node of the topological map to establish the topological neighborhoods, and to store the estimate of the value function (V and Q). All attributes are described in Fig. 3. Two new attributes (Q and r) are added to the four original attributes used in TRLA (w, *edges*, V, and e), to implement the algorithm of Fig. 4. The attribute Q is a vector in which each element is associated with a neighbor node. The selection of which nodes will have their Q and V attributes updated follows the satisfaction of $V(s) < V(s_t)$, where s_t is the current state. Differently from the TRLA algorithm [3], where the value function is only updated in the occurrence of a feedback signal (reward) and all states in a topological neighborhood have their values updated, the influence zone algorithm is an incremental version of TRLA, characterized through:

(i) Depending on the Error_TD, $r(s_{t+1}) + \gamma V(s_{t+1}) - Q(s, a_{ss_{t+1}})$, the new algorithm performs an one-step or an n-step updating in the value function (see Fig. 4). The current strategy starts the value function updating in less iterations since it depends on both the error and the goal occurrence.

(ii) When a n-step updating of the value function occurs, the new algorithm (Fig. 4) defines the influence zone of the current state s_t, $T(s_t)$, and only the states in $T(s_t)$ have their values updated.

Fig. 4 describes the *influence zone algorithm* in two levels:

- *Level* 1: Updating of a value function at each state transition whenever the Error_TD overcomes a threshold ϕ and such a change, similarly to [30], is limited to one transition (one-step update) of the topological neighborhood.
- *Level* 2: In order to reduce the number of updatings, it is triggered only when the error accumulator, θ, is higher than a given threshold θ_0. In this level occurs an n-step updating in the computed influence zone followed by a reset operation for the error accumulator (Line 2.1), thus when the loop in Line 2.2 is concluded and the algorithm returns to *Level* 1. Line 2.2 of the algorithm summarizes how the influence zone is considered: given a state s_{t+1}, the states belonged to influence zone of s_{t+1}

Attributes	Description of the attribute	Type of data
W	reference vector contents	vector
edges	list of neighbor nodes	vector
V	max value function associated to state ***w***	scalar
Q	state-action value function associated to state ***w*** and each edge	vector
r	return signal when in state ***w***	scalar
e	indicates if this node is in the current considered topological neighborhood	binary

Fig. 3. Attributes of the nodes.

Influence zone algorithm

1-step update	1. For $\forall s \in N^1(s_{t+1})$, 1.1. $Error\_TD = r(s_{t+1}) + \gamma V(s_{t+1}) - Q(s, a_{ss_{t+1}})$ 1.2. If $\lvert Error\_TD \rvert > \phi$, 1.2.1. $Q(s, a_{ss_{t+1}}) = Q(s, a_{ss_{t+1}}) + \alpha . Error\_TD$ 1.2.2. $V(s) = \max_a Q(s,a)$ 1.2.3. $\theta = \theta + \lvert Error\_TD \rvert$
n-step update	2. If $\theta > \theta_o$, 2.1. $\theta = 0$ 2.2. For $\forall s \in N^k(s_{t+1})$, $k = 2,3,4,\ldots,N_{viz}$, $V(s) < V(s_{t+1})$ and $r(s) = 0$, 2.2.1. $s_b = \arg\max_{b \in N^1(s)} V(b)$ 2.2.2. $Error\_TD'' = r(s_b) + \gamma V(s_b) - Q(s, a_{ss_b})$ 2.2.3. $Q(s, a_{ss_b}) = Q(s, a_{ss_b}) + \alpha . Error\_TD''$ 2.2.4. $V(s) = \max_a Q(s,a)$

Fig. 4. The two levels of influence zone algorithm.

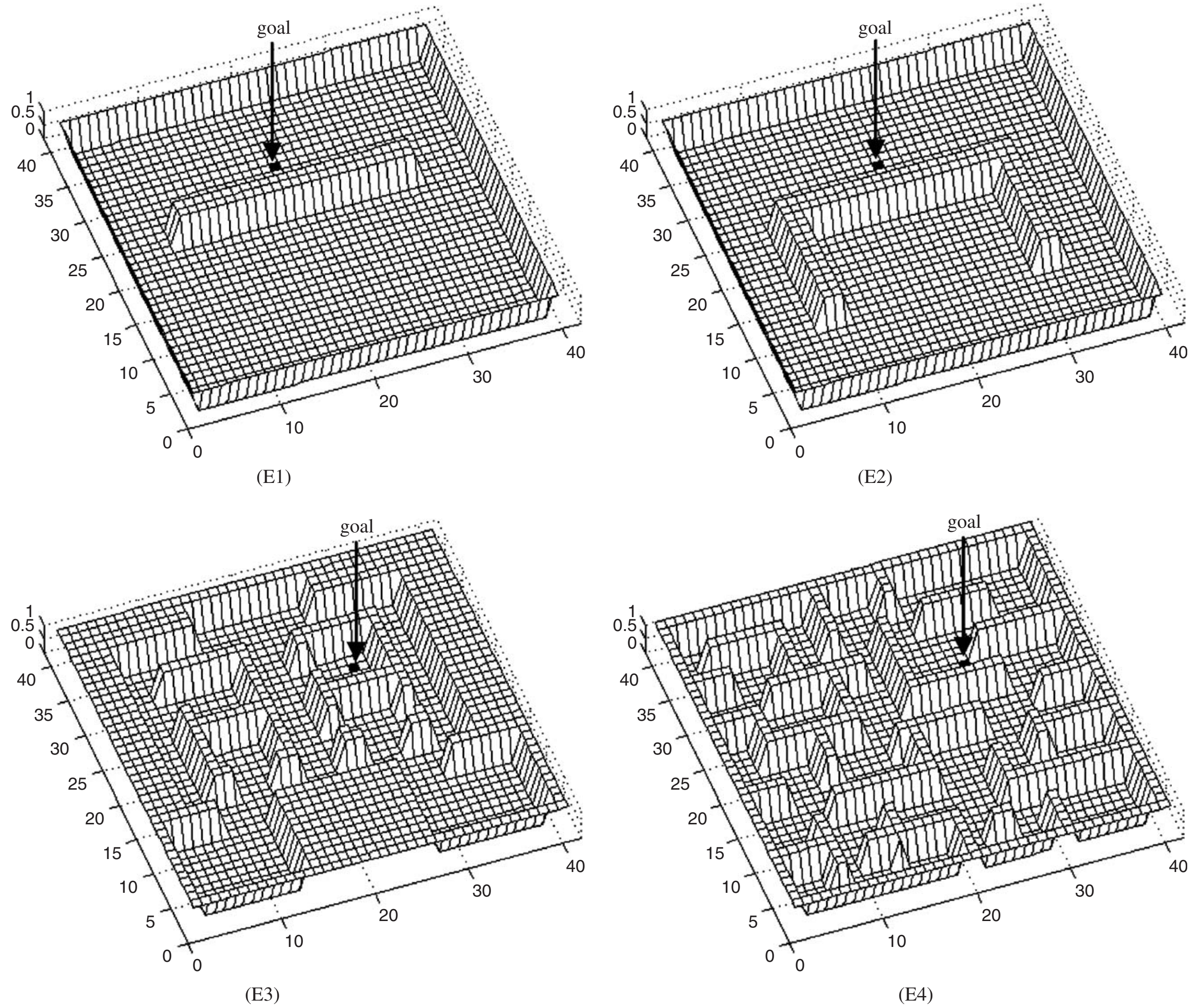

Fig. 5. Different configurations used to test the influence zone algorithm.

are found among the k topological neighborhoods of s_{t+1}, $N^k(s_{t+1})$, and must satisfy the condition $V(s) < V(s_{t+1})$ (Section 3). The ITM is used to learn $N^k(s_{t+1})$.

Identically to the TRLA, for all states in a Voronoi cell[2] ([13,14]) of the topological map, the action selection of the *influence zone algorithm* aims to reach the neighbor node with the highest value function[3] (V attribute). There is a fixed number of possible actions represented by vectors (n^3, where n is the input space dimension, and there are three possible values for each element of the vectors associated with the actions: -1, 0, 1) at each Voronoi cell/node and, since the topological map could be an irregular structure. The policy π chooses the action that takes the agent to the closest position to the neighbor node with highest V attribute. This is done by first computing the vector $\mathbf{v}_d$ which comes from the current node, $\mathbf{s}$, to the weight vector associated with the neighbor node with the highest value function, $\mathbf{w}_{\text{greedy}}$, through the following expression:

$$\mathbf{v}_d = \mathbf{w}_{\text{greedy}} - \mathbf{s}, \tag{8}$$

the action is selected computing which vector associated with an action is the closest one to $\mathbf{v}_d$. This is achieved by finding the action-vector which has the max scalar product with $\mathbf{v}_d$:

$$\pi(s) = \underset{i \in A(s)}{\arg\max}\{\mathbf{v}_d \cdot \mathbf{v}_i\}, \tag{9}$$

[2]All states in a Voronoi cell are represented by a node in the topological map.

[3]In case of multiple neighbor nodes with the same highest value, one of them is randomly chosen.

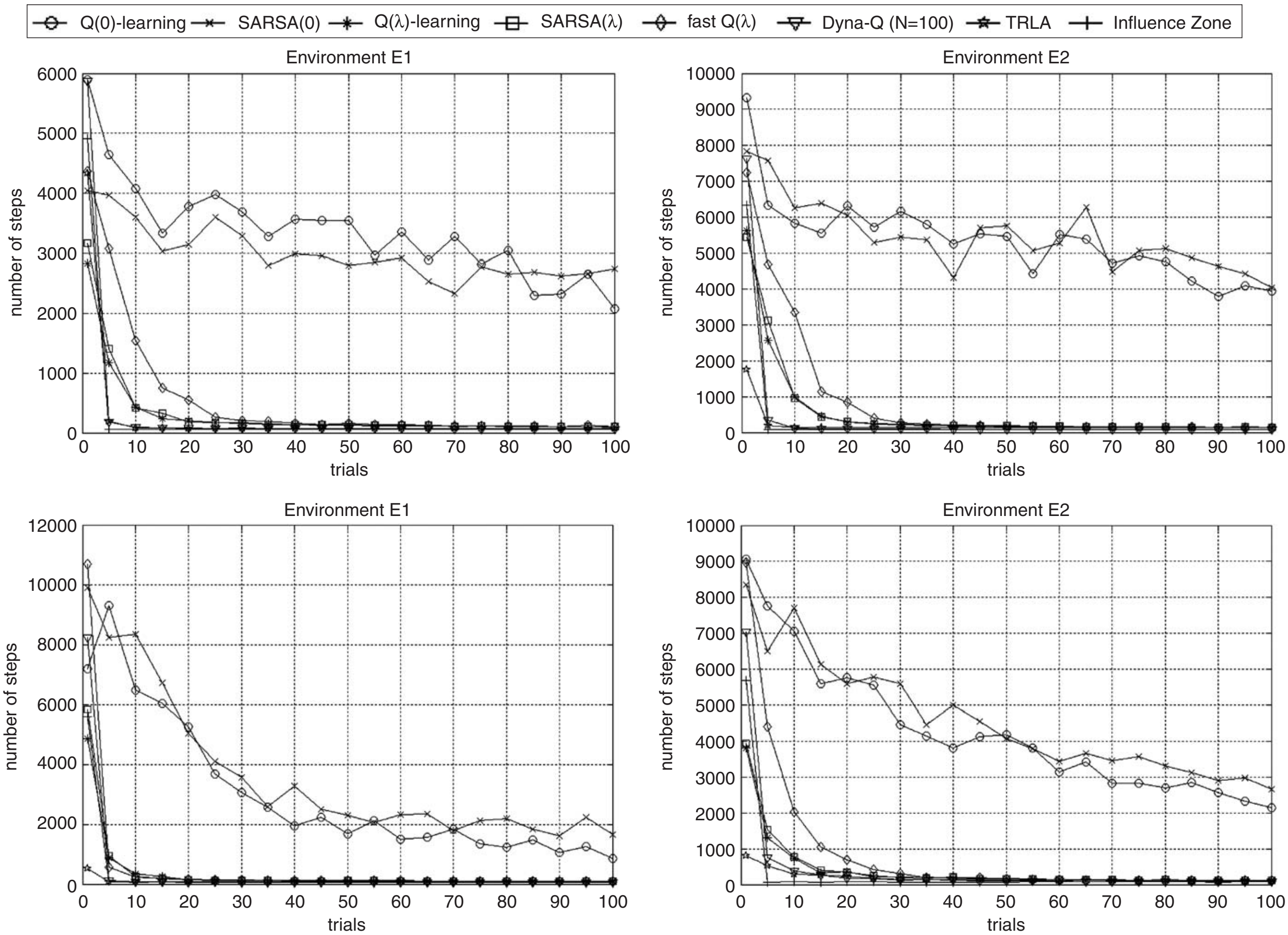

Fig. 6. Average number of steps, connecting initial and final points, in each trial, in environments of Fig. 5 for the eight RL algorithms.

Table 1
ANOVA tables for data presented in Fig. 6 (level of significance 0.05)

	Source	SS	Df	MS	F	p-value
E1	Columns	198098.7	5	39619.7	42.19	0
	Error	445077.8	474	939		
	Total	643176.5	479			
E2	Columns	331312.8	5	66262.6	52.85	0
	Error	594254.3	474	1253.7		
	Total	925567.1	479			
E3	Columns	135521.2	5	27104.2	32.02	0
	Error	401178.8	474	846.4		
	Total	536700	479			
E4	Columns	110658.6	5	22131.7	30.9	0
	Error	339457.8	474	716.2		
	Total	450116.5	479			

where $A(s)$ is the set of all possible actions from state $\boldsymbol{s}$, vectors $\mathbf{v}_i$ are associated with each action i from $A(s)$ representing the expected state transition.

6. Experiments and results

The influence zone algorithm was tested to solve a complex kind of task: autonomous navigation problem in an initially unknown world [3,17]. This problem comprises an agent that has to move within its environment, avoiding obstacles, to reach an initially unknown goal state. The navigation problem is an instance of a goal-directed RL problem (GDRLP) [12] in which an agent aims to learn a path, between an initial state and a goal state, through RL strategies. Following the initial learning, such a path can be changed to improve the initial solution, to obtain an optimal, or sub-optimal, policy.

All following experiments were simulated, using routines developed by the authors in the MATLAB®. For this 2-D navigation problem, eight possible actions for the RL agent were considered. They are represented by the following vectors: $\mathbf{v}_1 = (-1,1)$; $\mathbf{v}_2 = (0,1)$; $\mathbf{v}_3 = (1,1)$; $\mathbf{v}_4 = (1,0)$; $\mathbf{v}_5 = (1,-1)$; $\mathbf{v}_6 = (0,-1)$; $\mathbf{v}_7 = (-1,-1)$; $\mathbf{v}_8 = (0,-1)$

Four environment configurations were considered to simulate different levels of complexity for the navigation

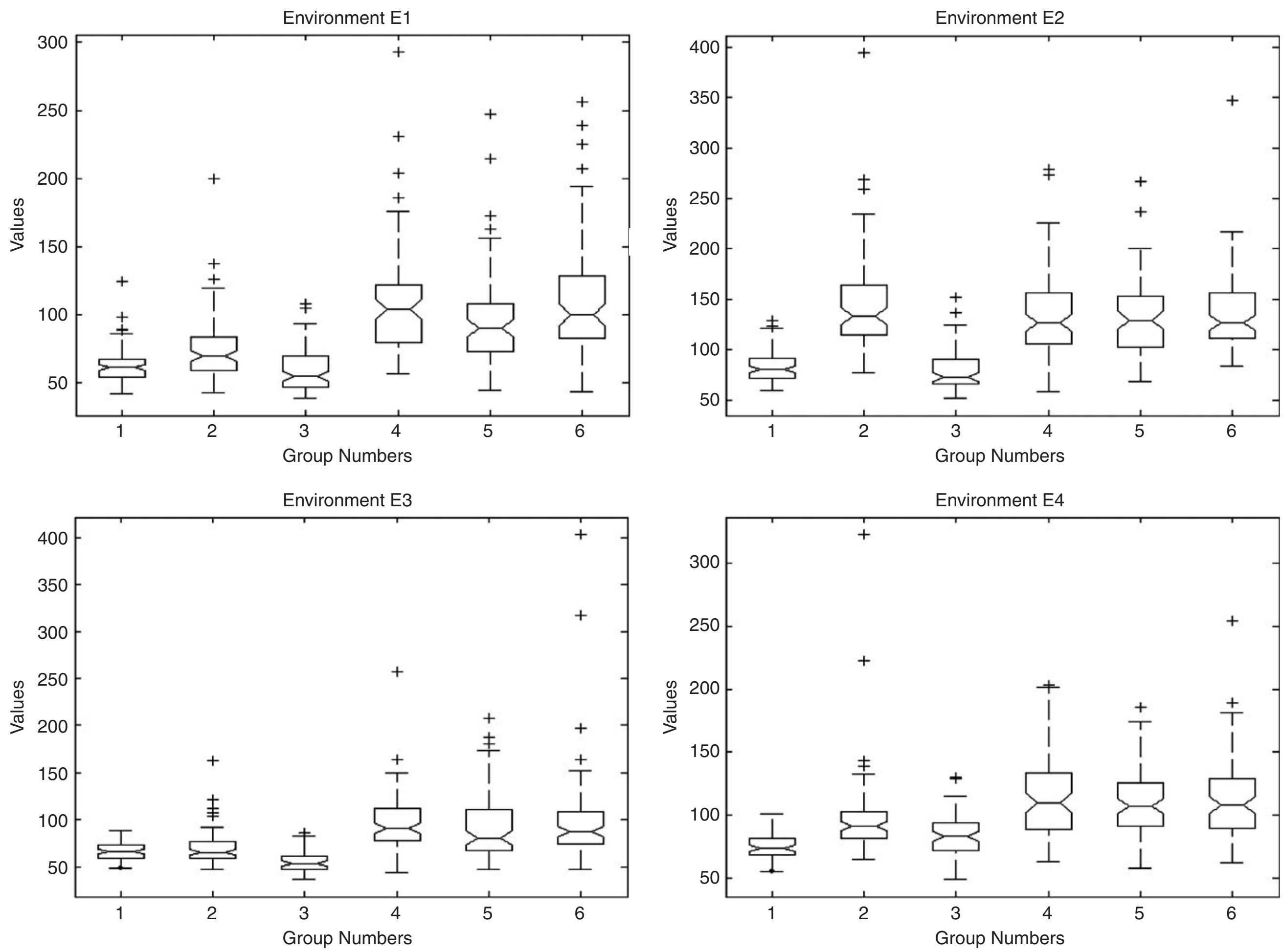

Fig. 7. Notched box plots of trajectory length generated by the six considered algorithms/groups for the four environments. The algorithms are identified as Group 1—influence zone algorithm; Group 2—TRLA; Group 3—Dyna-Q; Group 4—fast $Q(\lambda)$-learning; Group 5—$Q(\lambda)$-learning; Group 6—SARSA(λ).

Table 2
Estimated difference in groups means with a 95% confidence interval

Groups		Environments			
		E1	E2	E3	E4
1	2	[−24.9; −11.2; 2.6]	[−77.7; −61.7; −45.8]	[−17.2; −4.1; 9.0]	[−34.2; −22.1; −10.0]
1	**3**	**[−9.6; 4.2; 17.9]**	**[−12.1; 3.9; 19.9]**	**[−1.5; 11.6; 24.7]**	**[−19.7; −7.6; 4.4]**

Table 3
Average number of updatings for the last 50 trials of the results in Fig. 6

	Environments			
	E1	E2	E3	E4
Q-learning	3136.3	5055.8	4825.9	4987.2
SARSA	2304.2	4460.6	1143.4	2319.1
$Q(\lambda)$	4619.8	6419.7	4347.5	4816.6
SARSA(λ)	1907.0	2482.1	1768.2	2016.6
Fast $Q(\lambda)$	849.6	1207.2	787.5	883.6
Dyna-Q	5678.1	7282.3	5510.6	6215.4
TRLA	826.0	826.5	561.8	652.7
Influence zone	1413.5	1367.0	480.7	883.3

tasks (Fig. 5). An environment may have three kinds of states: (i) *free states* are those the agent can visit; (ii) *obstacle states* are prohibited positions of the environment for the agent and (iii) *the goal state* is the state to be reached by the agent.

The performance of the influence zone algorithm for autonomous navigation is tested and the results are compared with those obtained with 7 RL algorithms (see Appendix A for the parameters of each algorithm): (i) Q-learning [31]; (ii) SARSA [25]; (iii) $Q(\lambda)$-learning [21]; (iv) SARSA(λ) [25]; (v) Dyna-Q [28]; (vi) fast $Q(\lambda)$-learning [32]; and (vii) TRLA [3]. The reward function [29] of the

Table 4
ANOVA tables for data summarized in Table 3 (level of significance 0.05)

	Source	SS	Df	MS	F	p-value
E1	Columns	8.70246e+006	2	4351229.3	7.02	0.0019
	Error	3.53512e+007	395	620197		
	Total	4.40537e+007	397			
E2	Columns	4.4524e+006	2	2226200.6	8.51	0.006
	Error	1.49124e+007	395	261620.9		
	Total	1.93648e+007	397			
E3	Columns	2.3676e+006	2	1183799.0	34.08	1.837e–010
	Error	1.97983e+006	395	34733.9		
	Total	4.34743e+006	397			
E4	Columns	736626.9	2	368313.5	4.88	0.0111
	Error	4303004.0	395	75491.3		
	Total	5039630.9	397			

RL agents is

$$r_{t+1} = \begin{cases} 1 & \text{if } s_{t+1} \text{ is the goal state,} \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Two performance criteria were used to compare the RL algorithms. The *Criterion* 1 (C1) considers the length (number of steps) of the trajectory from a given point to the goal in each trial. The *Criterion* 2 (C2) computes the total number of value function updatings in each trial.[4]

6.1. Criterion 1 (C1)

Fig. 6 shows the learning performance of each RL agent as function of the trials by showing the number of steps of the generated trajectories.

The data sketched in Fig. 6 represents the average of 80 experiments for each environment. From such curves, there is a clear difference between the trajectories generated by Q-learning and SARSA, and the trajectories generated by other RL algorithms. Hence, the performance of the influence zone algorithm is analyzed only in comparison with the TRLA, Dyna-Q, fast $Q(\lambda)$-learning, $Q(\lambda)$ and SARSA(λ). For a statistical analysis of such data, ANOVA and the extended Tukey test [15,19,26] were carried out. The purpose of one-way analysis of variance (ANOVA) is to verify whether data from several groups have a common mean. That is, in the current case, to determine whether the algorithms are actually different, according to the C1. The *null hypothesis* in ANOVA is that all means are equal; this hypothesis is rejected if the p-value [15,19,26] has a very small value. Table 1 shows the p-values computed in the ANOVA table for each environment.

[4]Operations involved in the construction of the structure used to calculate the value functions updating were not included in this computation.

All p-values in Table 1 are null, then at least one of those algorithms generated trajectories with a different mean from the other ones.

The notched box plots (Fig. 7) allow to identify two clusters: (i) Groups 1-3 (Influence Zone, TRLA and Dyna-Q), having shorter trajectories than (ii) Groups 4–6 (fast $Q(\lambda)$-learning, $Q(\lambda)$-learning, SARSA(λ)). However, to specify with a level of significance which algorithms are similar, and which ones have the smallest trajectories, a multiple comparison test is necessary. In this case, the Tukey's significant-difference (TSD) test [15,19,26] was used. The Turkey test is the equivalent of t-test for multiple comparisons and consists in the computation of confidence intervals for the means of a pair of groups. If the value 0 (zero) is not within the computed interval then the means are different. Table 2 presents the 95% confidence interval for the estimated difference in means of the Groups 1–3, the short-trajectory cluster in Fig. 7.

The three values associated with each environment include the (i) lower limit; (ii) the estimated mean, and (iii) the upper limit of the difference between data from the groups being compared. Hence, if an interval contains 0 (zero), then the two groups under comparison are not considered different. Based on the interval contents, the means of trajectory lengths generated by influence zone algorithm and Dyna-Q are not statistically different (marked row in Table 2). Furthermore, these two algorithms produced the shortest trajectories.

6.2. Criterion 2 (C2)

The procedure adopted to analyze the performance of the considered RL algorithms for C1 is followed in this subsection for C2. Then, the data was collected in 80 experiments (100 trials each) for each algorithm in the four considered environments.

Table 3 summarizes the average number of value function updatings performed by each algorithm, and these data is used to indicate the candidates to the best performance according to C2. Table 3 shows Q-learning, SARSA, Dyna-Q, SARSA(λ) and $Q(\lambda)$-learning exhibit an undesirable behavior: they do execute a high number of updating in all four environments.

Hence, the performance of the influence zone algorithm is compared with the algorithms that presented lower number of updatings: TRLA and fast $Q(\lambda)$-learning. ANOVA, used in Table 4, shows the existence of significant differences between the means of the number of updatings with very small values for p-value. As a hypothesis test, ANOVA adopted a level of significance (α-level), equal to 0.05 in the current case. Then (since p-value <0.05) there is, at least, one of the RL algorithms (Influence Zone, TRLA or fast $Q(\lambda)$-learning) statistically distinct. In order to find statistical differences among the tested algorithms, the notched box plots shown in Fig. 8 shows the medians of updatings for the three considered models. These graphs suggest that the TRLA and the fast $Q(\lambda)$-learning have the

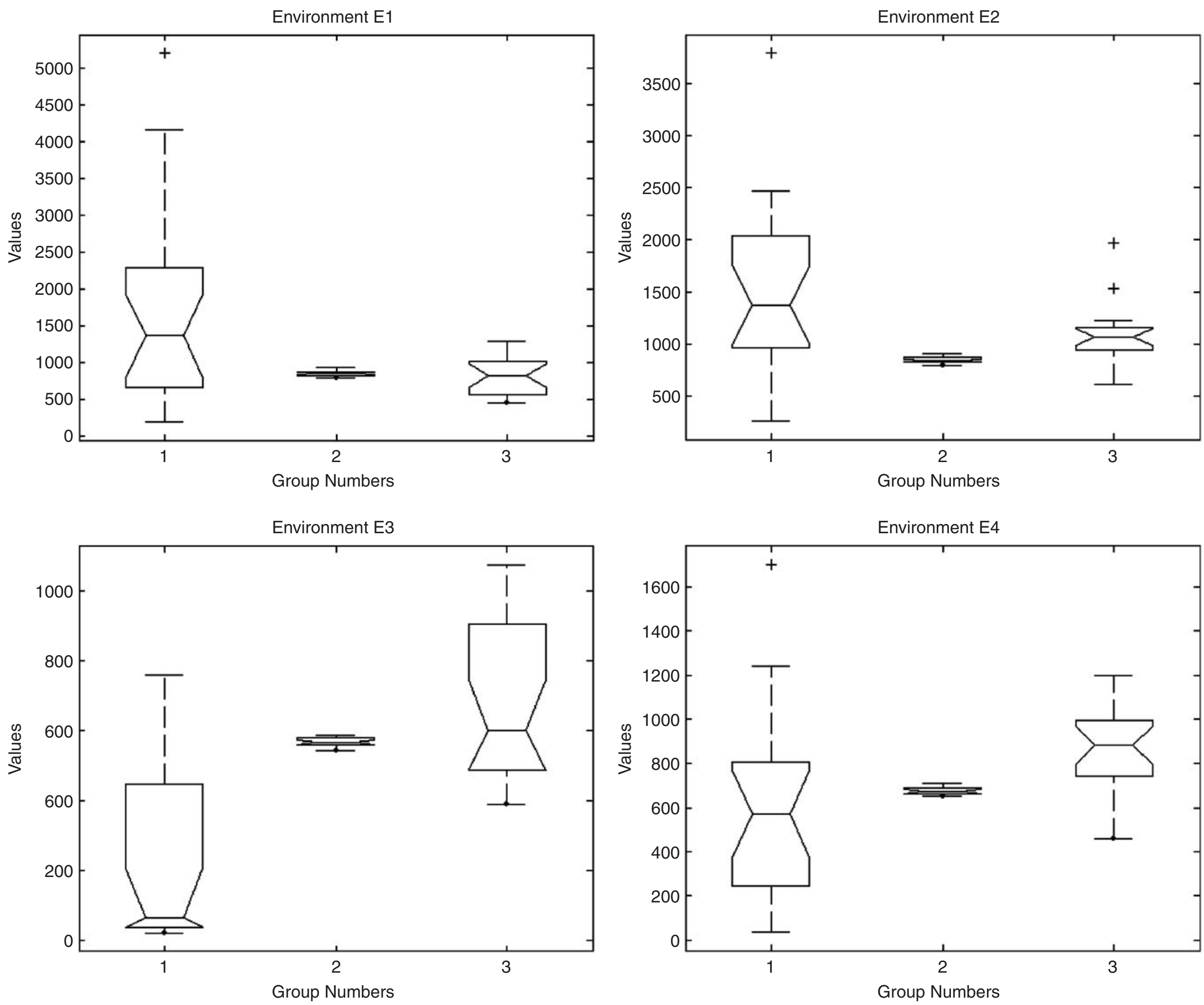

Fig. 8. Notched box plots of the number of updatings of the value function computed by the five considered algorithms/groups in the four environments. The algorithms are identified as Group 1—influence zone algorithm; Group 2—TRLA; Group 3—fast $Q(\lambda)$-learning.

Table 5
Estimated difference in groups means with a 95% confidence interval

Groups		Environments			
		E1	E2	E3	E4
1	2	[187; 786; 1385]	[272; 661; 1050]	[−486; −344; −203]	[−301; −92 ; 117]
1	3	[229; 828; 1428]	[22; 411; 800]	[−612; −470; −328]	[−476; −267; −58]
2	3	[−557; 42; 642]	[−639; −250; 140]	[−268; −126; 16]	[−384; −175; 34]

lowest number of updatings in environments E1 and E2, and that the influence zone algorithm has the lowest number of updatings in environments E3 and E4.

Tables 5 and 6 show the confidence intervals computed by the Tukey test to validate the above initial hypothesis about the performance of the influence zone algorithm in comparison with the TRLA and the fast $Q(\lambda)$-learning in their number of value function updatings. Table 5 points out that the influence zone algorithm has different number of updating in all cases, except for the TRLA in E4, whereas the TRLA and the fast $Q(\lambda)$-learning have the same performance. Table 5 together with Fig. 8 suggests that the influence zone algorithm has better performance in more complex environments (E3 and E4).

6.3. Change in environment structure

The four room configuration of Fig. 9 were used to test the RL agents performance in a changeable environment in which the reward function [29] is given as follows:

$$r_{t+1} = \begin{cases} +1 & \text{if } s_{t+1} \text{ is the goal state,} \\ -1 & \text{if } s_{t+1} \text{ is an obstacle,} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

To verify the comparative behavior of the influence zone approach to respond to modifications in the four rooms, 30 experiments were executed for each algorithm in the environment of Fig. 9 where the rooms have all internal doors open during the first 99 trials, and a closed door from trial 100 to trial 200. This closed door blocks the shortest path between initial and goal states, then a new trajectory should be learned. In the first stage (trials 1–99): the influence zone algorithm, Dyna-Q, fast $Q(\lambda)$-learning, $Q(\lambda)$-learning and SARSA(λ) approaches generated a cluster of shortest trajectories (in average, 28.83; 37.53; 39.04; 43.60 and 41.74, respectively), whereas the one-step algorithms, Q-learning and SARSA, generated the worst results (in average, 89.71 and 83.78, respectively). In this experiment, the TRLA was not considered since its original work [3] does not consider a multi-reward function as in the expression (11).

Table 6
ANOVA tables for data presented in Fig. 10b (level of significance 0.05)

Source	SS	Df	MS	F	p-value
Columns	1.64385e+009	2	8.21927e+008	235.49	0
Error	3.03651e+008	87	3.49025e+006		
Total	1.94751e+009	89			

Fig. 10 shows the behavior of the simulated algorithms for the criteria C1 (Fig. 10a) and C2 (Fig. 10b). The comparative behaviors during trials 1–99 are very similar, for both criteria, to those reported in Sections 6.1 and 6.2. Hence, the behaviors after the change in environment topology are the focus of this subsection.

From Fig. 10a, it is clear the difficulty of most of the RL algorithms to learn new trajectories in the presence of the new obstacle. In this second stage, only the influence zone and the Dyna-Q effectively adapt themselves reaching the target, not expressing a tendency to maintain the trajectories learned in first stage. Even though, the superiority of the influence zone algorithm against Dyna-Q is easily detectable, in terms of numbers of steps, the first one produced trajectories within the range [55.89$\pm$6.71] at trial 200, while Dyna-Q generated trajectories within the range [2043.0$\pm$4046.6] at the same trial.

The Dyna-Q employed its model of the world, i.e., learning a new value function involves random selection of states in which the agent avoids the closed door in the environment, causing slow convergence to the new value function (in average, 339860.0 value function updatings, for the last 50 trials). The influence zone update strategy is more efficient because of the topological map: after closing the door, the influence zone restricts the updating of the value function to the set of states reaching the closed door. Thus, a lower number of states had their values updated: 525.39 value function updatings, in average, for the 50 last trials.

Fig. 10b also shows Q-learning and SARSA compute a small number of value function updatings.

The p-value of Table 5 is evidence that the three means of the influence zone algorithm, Q-learning and SARSA are not the same. This is also shown in Fig. 11, where it is easy to see such differences.

The average number of updatings with influence zone is 356.1, while Q-learning and SARSA performs 8381.0 and 10000.0, respectively. The presented results show a remarkable flexibility of the influence zone algorithm.

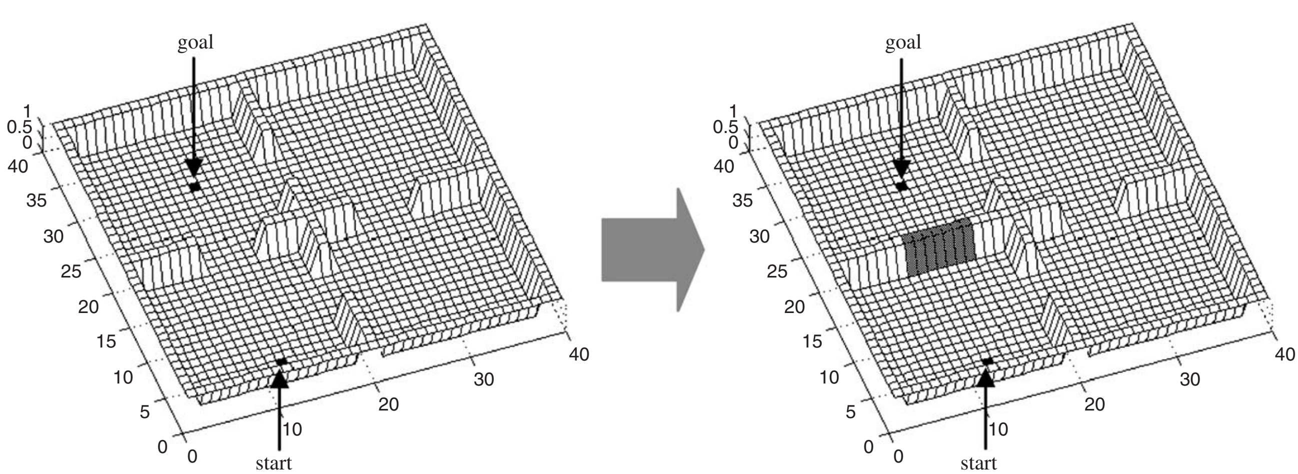

Fig. 9. Experimental sketch for analysis of the performance of RL models in the environment with change in the topology.

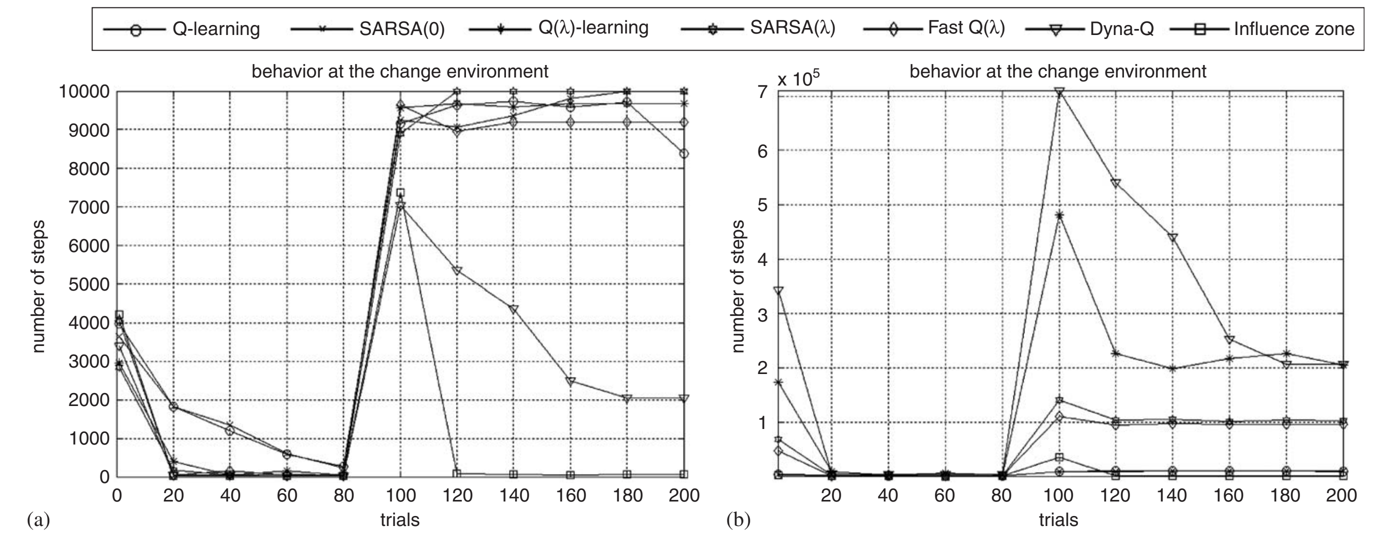

Fig. 10. (a) Number of steps, connecting initial and final points, in each trial, in the four room environment, (b) Number of value function updates, in each trial, during the training.

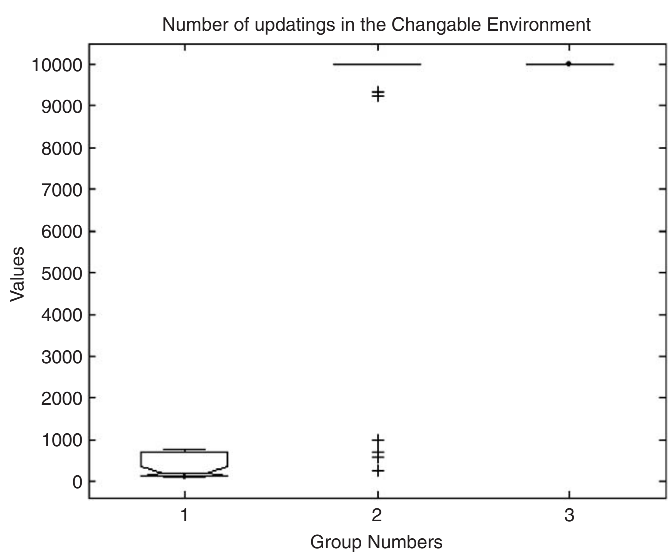

Fig. 11. Notched box plots of the number of updatings of the value function computed by the three considered algorithms/groups in the changeable environment of Fig. 10 after the door is closed. The algorithms are identified as Group 1—influence zone algorithm; Group 2—Q-learning; Group 3—SARSA.

7. Conclusions

The main contribution of this work is to propose an efficient strategy to accelerate RL by adopting the concept of influence zone to update the value function. Differently from the TRLA [3], that as influence zone algorithm uses a topological map to update the value function, the current algorithm is an improvement capable to work with more complex reward functions as the expression (11) since this strategy permits to restrict the updatings only to the set of states that have their value correlated with the value of the current state. Such a concept allows choosing a number of states to have their values updated at each state transition. The learned value function is an estimation of the optimal value function (Section 6), at the explored area of the environment, where states in the same topological neighborhood are considered as having a similar value function. This hypothesis is consistent with the temporal credit assignment performed by standard RL algorithms since states in the same topological neighborhood of a goal state might have a similar number of state transitions to reach such a goal.

The obtained results assessed the efficiency of the proposed value function update strategy. This simple mechanism allows similar or better learning acceleration than that observed in RL algorithms as the Dyna-Q and the fast $Q(\lambda)$-learning (Section 6). Such a performance suggests that the proposed algorithm is eligible to be applied in on-line problems since it reduced significantly the trajectory length since the first trial.

In order to label an algorithm as recommendable to improve RL efficiency, two performance criteria are considered: (i) the capacity to reach near optimal solution (in our case, the trajectories) in few trials, and (ii) the necessity of reduced number of updatings to diminish computing time. Section 6.1 shows the means of trajectory lengths generated by influence zone algorithm are not statistically different from the Dyna-Q, overcoming the other six RL-simulated algorithms. Furthermore, these results suggest an improvement of the influence zone algorithm in comparison with the TRLA. The influence zone algorithm is clustered with the TRLA and the fast $Q(\lambda)$-learning as those with less updatings. The results suggest that the proposed model tends to do better than the other two in more complex environments (Section 6.2).

Remarkable distinct performances were observed when the environment configuration changed (Section 6.3). In RL agents, performance degradation can occur due to environment changes when the policy can become unsuitable to such a new condition. Sutton and Barto (Chapter 9 in [29]) see this general problem as a version of the exploration–exploitation conflict: the agent must explore the environment to learn its changes, without disrupting the invariant knowledge previously acquired. The results plotted in Fig. 10, and discussed in Section 6.3, suggest the influence zone as a possible heuristic to reduce this conflict with a practical strategy: the prior knowledge represented by the topological map is used to determine the states affected by the changes, and only their values are updated.

Despite its promising first results, the influence zone implementation is very dependent on how precise is the environment topological representation by the SOM, and the algorithm can be improved by adopting more efficient search strategies to determine the topological neighborhoods. Inadequate connections between nodes or inaccurate representation of the goal state can deteriorate its performance. The potential of the proposed algorithm suggests future works with problems in dimensions higher than two. A 3D problem as in [11] could be an interesting test for the influence zone algorithm. Besides, the authors hope to investigate the applications of the current algorithm and its future variations to partially observable problems [9], a very active field in RL.

Acknowledgments

The authors would like to thank Dr. Jeremy Wyatt, University of Birmingham, UK, for his help regarding the experiments in the changeable environment (Section 6.3). This work was supported by FAPESP Grant No. 98/12700-5.

Appendix A

Algorithms' parameters used in simulations are shown in Table A.1.

Table A.1
Algorithms' Parameters used in Simulations

Algorithms	Parameters
$Q(0)$-learning	$\alpha = 0.5$; $\gamma = 0.8$; $\varepsilon = 0.3$
SARSA(0)	$\alpha = 0.5$; $\gamma = 0.8$; $\varepsilon = 0.3$
$Q(\lambda)$-learning	$\alpha = 0.5$; $\gamma = 0.8$; $\varepsilon = 0.3$; $\lambda = 0.7$; $\varepsilon_H = 10^{-16}$
SARSA(λ)	$\alpha = 0.5$; $\gamma = 0.8$; $\varepsilon = 0.3$; $\lambda = 0.7$; $\varepsilon_H = 10^{-16}$
Fast $Q(\lambda)$	$\alpha = 0.5$; $\gamma = 0.8$; $\varepsilon = 0.3$; $\lambda = 0.7$; $\varepsilon_m = 10^{-16}$
Dyna-Q	$\alpha = 0.5$; $\gamma = 0.8$; $\varepsilon = 0.3$; $N = 100$
TRLA	$\gamma = 0.8$; $\varepsilon = 0.3$; $e_{max} = 0.5$; $\epsilon = 10^{-4}$
Influence zone	$\alpha = 0.5$; $\gamma = 0.8$; $\varepsilon = 0.3$; $e_{max} = 0.5$; $\epsilon = 10^{-4}$; $\phi = 10^{-16}$; $\theta_o = 0.5$

α = reinforcement learning rate; γ = discount factor; ε = ε-greedy policy parameter; λ = lambda; ε_H = threshold to control the H list inclusion; ε_m = threshold to control the backups in *fast* $Q(\lambda)$; e_{max} = maximum radius on ITM; ϵ = self-organizing learning rate; N = number of repetitions; ϕ = threshold for Error_TD; θ_0 = threshold for cumulative Error_TD.

References

[1] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[2] D.P. Bertsekas, J.N. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, Belmont, MA, 1996.

[3] A.P.S. Braga, A.F.R. Araújo, A topological reinforcement learning agent for navigation, Neural Comput. Appl. 12 (3-4) (2003) 220–236.

[4] B. Fritzke, A growing neural gas network learns topologies, Adv. Neural Inform. Process. Syst. 7 (1995) 625–632.

[5] B. Fritzke, Unsupervised ontogenetic networks, in: E. Fiesler, R. Beale (Eds.), Handbook of Neural Computation, Institute of Physics Publishing and Oxford University Press, 1996.

[6] P.L. George, Automatic Mesh Generation—Application to Finite Element Methods, Wiley, New York, 1991.

[7] A. Großmann, Continual learning for mobile robots, Ph.D. thesis, School of Computer Science, The University of Birmingham, Birmingham, UK, 2001.

[8] J. Jockusch, H. Ritter, An instantaneous topological mapping model for correlated stimuli. Proceedings of the IJCNN'99, 1999, pp. 445.

[9] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, J. Artif. Intell. Res. 4 (1996) 237–285.

[10] M.H. Kalos, P.A. Whhitlock, Monte Carlo Methods, Wiley, New York, 1986.

[11] M.Y. Kim, H. Cho, Three-dimensional map building for mobile robot navigation environments using a self-organizing neural network, J. Robot. Syst. 21 (6) (2004) 323–343.

[12] S. Koenig, R.G. Simmons, The effect of representation and knowledge on goal-directed exploration with reinforcement learning algorithms, Mach. Learn. 22 (1996) 227–250.

[13] T. Kohonen, Self-Organization and Associative Memory, Springer, Heidelberg, 1984.

[14] T. Martinetz, K. Schulten, Topology representing networks, Neural Networks 7 (3) (1994) 507–522.

[15] R.L. Mason, R.F. Gunst, J.L. Hess, Statistical Design and Analysis of Experiments, Wiley, New York, 1989.

[16] R.A. McCallum, Using transitional proximity for faster reinforcement learning, Proceedings of the Ninth International Conference on Machine Learning, 1992, pp. 316–321.

[17] J. Milán, R. del, Rapid, safe, and incremental learning of navigation strategies, IEEE Trans. Syst., Man, Cybernet. 26 (1996) 408–420.

[18] J. Millán, R. del, D. Posenato, E. Dedieu, Continuous-action Q-learning, Mach. Learn. 49 (2002) 247–265.

[19] D.C. Montgomery, Design and Analysis of Experiments, Wiley, New York, 1984.

[20] A.W. Moore, C.G. Atkeson, Prioritized sweeping: reinforcement learning with less data and less time, Mach. Learn. 13 (1993) 103–130.

[21] J. Peng, R.J. Williams, Incremental multi-step Q-learning, Mach. Learn. 22 (1996) 283–290.

[22] C.H.C. Ribeiro, Aspects of the behaviour of a learning agent in control tasks, Ph.D. thesis, Imperial College of Science, Technology and Medicine, University of London, 1998.

[23] C.H.C. Ribeiro, Reinforcement learning agents, Artif. Intell. Rev. 17 (2002) 223–250.

[24] H. Ritter, T. Martinetz, K. Schulten, Neural Computation and Self-Organizing Maps—An Introduction, Addison-Wesley Publishing Company, Reading, MA, 1992.

[25] G.A. Rummery, Problem solving with reinforcement learning, Ph.D. thesis, Cambridge University, 1995.

[26] W.C. Schefler, Statistics—Concepts and Applications, The Benjamin/Cummings Publishing Company, Inc., 1988.

[27] A.J. Smith, Applications of the self-organising map to reinforcement learning, Neural Networks 15 (8–9) (2002) 1107–1124.

[28] R.S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, SIGART Bull. 2 (1991) 160–163 ACM Press.

[29] R.S. Sutton, A. Barto, Introduction to Reinforcement Learning, MIT Press/Bradford Books, Cambridge, MA, 1998.

[30] C. Touzet, Neural reinforcement learning for behaviour synthesis, Robot. Autonom. Syst. 22 (3–4) (1997) 251–281.
[31] C.J.C.H. Watkins, Learning from delayed rewards, Ph.D. thesis, King's College, Cambridge, 1989.
[32] M. Wiering, J. Schimidhuber, Fast online $Q(\lambda)$, Mach. Learn. 33 (1998) 105–115.

Arthur P.S. Braga received his B.S. degree in Electrical Engineering from Federal University of Ceará in 1995. The M.S. and Ph.D. degrees in Electrical Engineering from the University of São Paulo in 1998 and 2004, respectively. His Ph.D. thesis concerned a methodology for the learning acceleration of reinforcement learning algorithms. His research interests include theory and applications in the areas of reinforcement learning, neural networks and robotics. Currently, Dr. Braga is developing his post-doctorate in the automation of manufacturing processes through artificial intelligence techniques.

Aluízio F.R. Araújo was born in Recife, Pernambuco, Brazil, in 1957. He received his B.S. degree in Electrical Engineering from Federal University of Pernambuco in 1980, the M.S. degree in Electrical Engineering from State University of Campinas in 1988, and the Ph.D. in Computer Science and Artificial Intelligence from the University of Sussex in 1994. He worked in São Francisco Hidroelectrical Company for 5 years and in 1994 he became an Assistant Professor at University of São Paulo, where in 1998 he was promoted to Adjunct Professor. Currently, Dr Araújo is lecturer in Federal University of Pernambuco. His research interests are in the areas of neural networks, machine learning, robotics, dynamic systems theory, and cognitive science. He has published more than 80 papers in national and international journals and conferences.